Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Ingeniería en Electrónica Industrial y Automática

Desarrollo de sistema de asistente por voz para control de un robot móvil

Director: Dr. Eduardo Zalama Casanova

Codirector: Raúl Gómez Ramos

Alumno: José Gabriel Nieto Ramos

Curso: 2023-2024

Agradecimientos

A mi padre, por ser mi referente siempre desde pequeño, ayudarme a encauzarme en mi camino. Por ser mi fuerza lógica hasta ahora y siempre.

A mi madre, por ser quien más ha soportado, quién más me ha motivado durante este camino.

A mis amigos, por ayudarme a ver las cosas con perspectiva, no centrándome en el problema que tengo frente a mí.

A mi pareja, por estar ahí, por aguantar mis locuras, por animarme y motivarme para dar lo mejor de mí siempre. También por intentar comprender siempre lo que hago, aunque no lo entienda del todo.

A mis directores, gracias por el apoyo, la ayuda y la paciencia durante este proyecto.

Índice

Lista de figuras

Lista de tablas

Glosario

Dentro de esta memoria se usarán las siguientes palabras técnicas y claves, abreviaturas o anglicismos, las cuales se explicarán a continuación:

- *ASK*: abreviatura del servicio de Amazon "*Alexa Skill Kit*", con el cual se pueden crear las skills customizadas que el usuario desee para el asistente.
- *ASR*: abreviatura de "*Audio Speech Recognition*" o Reconocimiento Automático de Voz, es un software que permite a un ordenador convertir las palabras que pronuncia un ser humando en texto.
- AWS: abreviatura de "*Amazon Web Service*", son el conjunto de servicios que proporciona Amazon a sus usuarios más allá de pedir productos.
- *IA*: abreviatura de Inteligencia Artificial.
- *IoT*: abreviatura de "*Internet of Things*", describe la agrupación e interconexión de dispositivos y objetos dentro de una red (ya sean sensores, actuadores, ordenadores...).
- *M2M*: abreviatura de "*Machine to Machine*", concepto utilizado para describir la comunicación entre dos o más máquinas.
- *MQTT*: abreviatura de "*Message Queuing Telemetry Transport*", consiste en un protocolo de mensajería ligero utilizado en mensajería *M2M*.
- *NLU*: abreviatura de "*Natural Language Understanding*", es una forma de inteligencia artificial que se encarga de usar el aprendizaje automático para mejorar la comprensión del lenguaje humano.
- *QoS*: abreviatura de "*Quality of Service*", es un concepto de MQTT referido al tipo de envíos de datos y sus confirmaciones.
- *SDK*: abreviatura de "*Skill Device Kit*", son el conjunto de herramientas que un programador de skills de Alexa utiliza para desarrollar cualquier habilidad.
- *SO*: abreviatura de Sistema Operativo.
- *TTS:* abreviatura de "*Text to Speech*", concepto referido a la transcripción de oraciones habladas a texto.
- *UI*: abreviatura de "*User Interface*" o interfaz de usuario.

- *Alexa:* asistente de voz de Amazon.
- *Alexa Developer Console*: página web que proporciona Amazon para poder utilizar sus servicios AWS como programador y desarrollador.
- *ANDIN*: nombre de un proyecto desarrollado en la Fundación CARTIF centrado en crear un andador inteligente.
- *Broker*: servidor de mensajería MQTT que comunica subscriptores con la información publicada por los clientes.
- *INOTEC*: nombre de un proyecto desarrollado en la Fundación CARTIF centrado en crear un inodoro inteligente.

- *Intent*: función programada de la *skill* o habilidad de Alexa donde se definen las frases de activación y las acciones correspondientes.
- *Machine learning:* disciplina de la *IA* basada en dotar a los ordenadores la capacidad de análisis de datos y creación de predicciones a gran escala mediante el uso de algoritmos.
- *Python:* lenguaje de programación de alto nivel que se empleará para programar la *skill.*
- *Skill*: habilidad que dota de funcionalidad a una aplicación de Alexa donde se definen las frases que entenderá, que acciones realizará o que contestará.
- *SkillBuilder*: compilador de la *skill* donde podemos indicar el orden de prioridad entre las *intents*.
- *Slot*: campo variable dentro de las oraciones de activación de los *intents* que permite definir solo una frase y aportar un abanico de opciones usando el mismo comando, además de que pueden usarse en el código para crear casos.
- *Uterances*: son las diferentes frases de activación para accionar un *intent*.

Resumen

A medida que la tecnología ha ido evolucionando, la humanidad ha ido buscando maneras más cómodas de interactuar con la tecnología, acercándose a una forma lo más similar a nuestra manera de comunicarnos. Aquí es donde entran los sistemas de reconocimiento de voz.

El ser humado ha desarrollado el habla como una forma de comunicación compleja, por lo que, desde hace años, se ha intentado utilizar la tecnología para lograr reconocerlo. Estos sistemas de reconocimiento han ido evolucionando desde captar únicamente fonemas hasta comprender frases complejas, permitiéndonos ir un poco más allá y crear asistentes de voz que satisfagan las necesidades de los usuarios tan solo con hablar.

Otra razón para el desarrollo de los asistentes de voz es simplificar la interacción con la tecnología del entorno, pero sobre todo para poder llegar a un mayor número de usuarios, como pueden ser personas que no saben leer o escribir, mayores o con alguna discapacidad física, los cuales se ven muy beneficiados a la hora de interaccionar con una interfaz de voz, dada su mínima curva de aprendizaje.

A lo largo de este proyecto, se expondrá el desarrollo de una habilidad o "*skill*" del asistente de Amazon, Alexa. Se aprovecharán los servicios de comprensión de palabras y frases del asistente y se creará un modelo de respuesta apropiado, donde se realicen las acciones deseadas y se entregue al usuario una frase indicando si se ha ejecutado o no la orden.

Esta *skill* ha sido creada para controlar un robot móvil en un hogar con el objetivo de mejorar la calidad de vida de las personas que lo habitan, permitiendo usar instrucciones verbales para que realice diferentes funciones. Hay que recalcar que esta *skill* también se ha extendido para controlar un ecosistema domótico, compuesto por el robot a controlar, sensores y otros actuadores o dispositivos inteligentes. Otra ventaja de las skills de Alexa es que, siempre que se disponga de un dispositivo (ya sea un Alexa o de la propia aplicación móvil del asistente), se puede descargar en el equipo y emplearla directamente en el hogar.

Además, gracias al uso del protocolo de mensajería *MQTT*, se puede lograr el uso de numerosos dispositivos, ya sean sensores, actuadores o dispositivos inteligentes, con los que se puede facilitar la vida de los usuarios que viven dentro de la casa e incluso la interacción con el sistema desde el exterior.

Entonces, en este trabajo se abarcarán: el origen de los reconocimientos y los asistentes de voz, se desglosará el funcionamiento de Alexa, además de crear una *skill* o habilidad que cumpla con las especificaciones iniciales y las que se vayan incluyendo durante la ejecución del proyecto, se crearán circuitos de mecatrónica con funcionalidad *MQTT* y se programarán los microcontroladores de los mismos para actuar en consecuencia de las órdenes que se le manden a Alexa. Algunos ejemplos de lo último serán: un andador y un inodoro inteligentes desarrollados en la Fundación CARTIF.

1.- Introducción

La domotización ha sido un concepto que, con el paso de los años, ha ido avanzando considerablemente, para interconectar todos los elementos de nuestro hogar, ya sea el móvil, el ordenador, sensores y luces. Algo que también ha evolucionado son las maneras de interaccionar sobre la misma, ya sea usando botones físicos, aplicaciones web, aplicaciones móviles o, como en nuestro caso, aplicaciones de voz.

Las aplicaciones o asistentes de voz han ido tomando fuerza en los últimos años (algunos ejemplos de trabajos sobre ellas son [23], [33] o [35]), acompañados por la evolución de la capacidad de comprensión de lo que el usuario está diciendo, lo cual es de mucha utilidad para realizar casi cualquier tarea, desde realizar una búsqueda hasta pedir que se apague una luz. Una pequeña muestra de su popularidad es que, al escuchar las palabras “Siri” o “Alexa”, casi todo el mundo sabe lo que son o los ha utilizado alguna vez.

1.1.- Marco del proyecto

El proyecto se ha enfocado en crear un servicio de voz propio para un uso muy específico, como es la asistencia en un hogar, pero para lograrlo, se ha tenido que profundizar en los conocimientos de programación y la creación de *skills*.

Algunos de los usos son poder hacer peticiones a actuadores, lectura de sensores o notificaciones al usuario.

Una de las razones principales por las que se ha decidido crear aplicación para una interfaz de voz es que es muy sencilla de utilizar, puesto que no requiere de conocimientos de informática elevados, sino que, solo con conocer las frases adecuadas, el usuario puede hacer lo que desee, cosa que, para las personas mayores es muy útil porque les suele costar entender el funcionamiento de algo nuevo y para una persona con movilidad reducida le facilita cualquier tarea sin necesidad de desplazarse.

Para realizar el objetivo, se cuenta con diferentes dispositivos inteligentes y un sistema domótico, los cuales son:

- Un robot capaz de moverse por la casa, ya sea de forma autónoma o por una petición de usuario a través de la skill, aportando diferentes servicios, como videoconferencias, hablar con el usuario, jugar con él, realizar un seguimiento de una persona, etc.
- Un andador motorizado al que se le podrá dar órdenes para que realice desplazamientos y giros.
- Un inodoro inteligente con el que, además de utilizar un mando para controlarlo, se podrá usar la voz para activar y desactivar las diferentes funciones, cambiar entre usuarios o activar ciclos de limpieza.
- Control de sensorización, pudiendo conocer la lectura de un sensor concreto en una habitación específica.

- Control de iluminación y actuadores, es decir, realizar acciones de encendido y apagado o de activación y desactivación de diferentes equipos domóticos que pertenezcan al sistema.

De una manera indirecta, este proyecto también está orientado para introducir a las personas mayores a adaptarse y emplear nuevas tecnologías, puesto que es un colectivo muy heterogéneo en el que existen casos donde es muy difícil que comprendan ciertas cosas tecnológicas, pero también existen excepciones que a veces se desenvuelven mucho mejor de lo esperado.

Al utilizar tecnología de internet de las cosas (IoT), también se facilita la conexión de una infinidad de elementos, siendo posible controlar muchos equipos y dispositivos y, cumpliendo el principal objetivo, ayudar a las personas en su día a día.

1.2.- Objetivos del trabajo

El objetivo principal es desarrollar una *skill* funcional para controlar un entorno domótico orientado a la asistencia de personas mayores o con problemas de movilidad. Se busca centralizar con una interfaz de usuario (*UI* o *User Interface*) de voz, en nuestro caso Alexa, para que el usuario sea capaz de realizar peticiones de todo tipo, pedir la temperatura en una sala, activar luces u ordenar a un robot asistente que realice una acción determinada.

Para desarrollar el proyecto, se deben completar las siguientes tareas:

- Estudio del progreso de las tecnologías de reconocimiento de voz.
- Comprensión de los dispositivos y la arquitectura de Alexa.
- Aprender la forma de crear una skill ofrecida por la consola de desarrollo de Alexa tanto del *front-end* como del *back-end*, cumpliendo una serie de requisitos y en paralelo a un sistema ya existente.
- Estudio del protocolo *Message Queuing Telemetry Transport* (MQTT) para su implementación en la skill así como en los efectores finales y los sensores.
- Realización de pruebas y evaluación de los resultados obtenidos.
- Estudio y análisis económico del proyecto.

1.3.- Estructura del documento

Dentro de este documento, se han dividido en capítulos los diferentes bloques importantes del proyecto:

En el capítulo segundo se expone la evolución a lo largo de la historia de los asistentes de voz actuales, explicando algunas de sus dificultades.

En el capítulo tercero se pondrá en contexto sobre las tecnologías empleadas durante el proyecto (Alexa, microcontroladores, MQTT, etc.).

En el capítulo cuarto, dedicado a la implementación y desarrollo, se explica la evolución de la aplicación del asistente, así como otros aspectos del proyecto como pueden ser las diferentes funciones, montajes de circuitos mecatrónicos, etc.

En el capítulo quinto se hace un análisis de resultados obtenidos de la aplicación desarrollada.

En el capítulo sexto se realiza un estudio económico y se analiza la viabilidad económica del proyecto.

Finalmente, el capítulo 7 se dedica a las conclusiones y futuras líneas de desarrollo del proyecto.

2.- Asistentes de voz

2.1.- Evolución de las interfaces de voz

La interfaz de usuario se puede realizar mediante diferentes tipos de dispositivos como pueden ser una pantalla o un teclado de un ordenador hasta tecnologías más complejas, como pueden ser aquellas basadas en los movimientos, como el que se basa en la Xbox Kinect. Como se puede observar en la figura 1, en los últimos 150 años ha existido una evolución exponencial de esta tecnología.

Figura 1. Cronograma de la evolución de las interfaces de voz. [26]

Inicialmente, el objetivo del reconocimiento de voz se centró en ser capaz de reconocer fonemas, puesto que es el componente más básico del habla. Se tardó mucho tiempo en lograr un reconocimiento de voz al uso, puesto que lo que existía antes de los años 70 no eran más que grabadoras, sin capacidad de interpretación.

Algunos ejemplos de esto son Thomas Edison con su máquina de dictado (siglo XIX), Bell Labs con *Audrey* (década de 1950) o IBM con *Shoebox* (1962).

Ya en la década de los 70, en la Universidad Carnegie Mellon en Pittsburgh (Pensilvania) se desarrolló *Harpy*, con la ayuda del Departamento de Defensa de los Estados Unidos y la Agencia de Proyectos de Investigación Avanzada de Defensa. *Harpy* podía entender unas 1 000 palabras, como el vocabulario de un niño de 3 años aproximadamente. Tras estos avances, se profundizó a lo largo de esa década en la comprensión de palabras aisladas y se buscó el reconocimiento de voz de cualquier locutor.

IBM desarrolló *Tangora* en los años 80, capaz de comprender la voz de cualquier usuario y reconocer unas 20 000 palabras y alguna oración, pero aún era necesario hablar despacio, de forma clara y sin ruido. Ya iniciados los 90, se podría decir que todos los sistemas se basaban en comparar los datos con plantillas, grabando con valores numéricos las señales de sonido y guardándolas, lo cual hacía que el habla del usuario tuviera que ser claro y despacio, aumentando las posibilidades de que se reconocieran los sonidos.

El primer reconocimiento de voz continuo aparece en 1 997, es decir, que no había que pararse entre palabras y era capaz de entender unas 100 palabras y transformarlas en algo comprensible.

Con la llegada del nuevo siglo, la gestión de reconocimiento de voz se comienza a adaptar a las aplicaciones móviles y webs, provocando un avance importante de la tecnología. Uno de los grandes avances fue el aprendizaje automático, donde Google utilizó los datos en la nube para lograr una mejor precisión de los algoritmos de aprendizaje automático, lo cual ha causado que, en la actualidad, el asistente de Google este implementado en un 50% de los teléfonos inteligentes.

En el año 2 011, Apple lanzó un asistente virtual que usaba IA llamado Siri, aportando cierta humanidad al reconocimiento de voz. A partir de aquí, han ido apareciendo varios competidores como Cortana de Microsoft o Alexa de Amazon, creando la actual lucha por ver quien logra anteponerse a los demás.

El progreso es importante, como ya sabemos, empezando por lo más básico como los fonemas, pasando por la comprensión de frases y llegando a lo existente hoy, donde las máquinas pueden entender al locutor casi en su totalidad.

Muchos de los asistentes de voz, en sus inicios, fueron pensados para que existieran únicamente en los *smartphones*, pero a medida que han pasado los años se han descentralizado gracias a la tecnología *IoT*, empleada para centralizar todos los dispositivos de una red, lo cual ha provocado que dichos asistentes existan en otros dispositivos y sean capaces de controlar otros objetos *IoT*.

En la actualidad y con ayuda de las nuevas IA's, es más que probable que el avance de los asistentes de voz se vea mejorado con respecto a su manera de conversar con el usuario, haciendo aún más humanas sus interacciones.

2.2.- Aplicaciones de voz frente a aplicaciones móviles convencionales

Desde hace tiempo, existen aplicaciones móviles para realizar consultas, asistencias, pedir citas, las cuales se analizan por seres humanos, pero actualmente están siendo sustituidas parcialmente por máquinas o asistentes de voz con el objetivo de simplificar dichas interacciones, pero por esa misma razón, aún no pueden sustituir a un ser humano de manera total y, por ello, muchas personas siguen prefiriendo interactuar con un ser humano antes que con una máquina.

A medida que avancen los asistentes de voz, muchas de las aplicaciones móviles serán sustituidas, gracias al '*Natural Language Processing Technology*' (NLP), lo que permite a los asistentes comprender el contexto de las palabras utilizando aprendizaje automático.

Una de las grandes diferencias entre ambas reside en la realización de una interfaz cómoda para el usuario. Un programador de aplicaciones móviles tiene que tener primero los conocimientos suficientes para crear completamente la aplicación, lo que provoca que el

usuario deba acostumbrarse a la aplicación y adaptarse a navegar por ella, lo que hace que todo este tipo de aplicaciones sigan un patrón. Mientras tanto, en las *UI's* de voz se busca aprender a interactuar con los humanos, lo cual hace que los asistentes busquen ser tan cómodos que cualquiera fuese capaz de usarlos sin saber siquiera su utilidad. Esto hace que, para mejorar dichas interacciones, deban guardar sus conocimientos sobre temas ya tratados por usuarios para poder tener conversaciones fluidas con otros en el futuro.

A la hora de comparar los recursos utilizados para crear dichas aplicaciones, también existen diferencias importantes, sobre todo cuando se habla del *front-end* o la parte que el usuario ve de la *skill* y con la que interactúa. Mientras que las aplicaciones móviles necesitan una inversión costosa y compleja respecto a su diseño, por otro lado, en una aplicación de voz no es necesario una *UI* complicada, lo cual abarata su coste, ahorrando tiempo y dificultad a la hora de crearlo (sin embargo, la dificultad recae en la comprensión del habla normal del usuario).

Ambos tipos de aplicaciones deberían poseer funcionalidades útiles y eficaces, pero no es así. Se puede ver en la actualidad como los jóvenes invierten mucho tiempo en sus smartphones, lo cual indica en parte que otro objetivo de las aplicaciones móviles es mantener al usuario utilizándola, cosa que las aplicaciones de voz, al ofrecer los resultados esperados inmediatamente al usuario sin información adicional, no hacen.

Otro punto relevante viene asociado a la obtención y la manera de almacenar las aplicaciones. Las aplicaciones móviles deben ser descargadas en el dispositivo, lo cual a veces lleva tiempo, y ocupan el espacio del mismo, quizás haciendo inaccesible para el usuario toda la información que requiera. Por otro lado, el acceso a las aplicaciones de voz se realiza por la nube, sin ocupar espacio en nuestro dispositivo y pudiendo acceder siempre que se quiera (y siempre que se disponga de conexión Wifi), lo cual facilita mucho su uso al usuario.

Uno de los problemas actuales de los asistentes de voz es su capacidad de comprender las intenciones del usuario para anteponerse a este, es aquí donde las *IA's* entran en juego, permitiendo crear contenido personalizado y del gusto del usuario.

2.3.- Áreas de aplicación de los asistentes de voz

En la actualidad, la orientación que tienen los asistentes de voz es, en mayor medida, para los usuarios particulares, haciendo sencillo su día a día ya sea programando una alarma, controlando la domótica del hogar, narrando las noticias, etc.

Figura 2. Respuestas más comunes empleadas por Alexa. [38]

Como se puede observar en la Figura 2, referida a los usos de Alexa en 2017, principalmente se usaba como un reproductor y altavoz, pero también respondía a muchas cuestiones de los usuarios, además del control de dispositivos e incluso compras.

Aparte de todas estas funciones, también son muy útiles para personas con movilidad reducida, con discapacidades visuales, niños pequeños o personas que no saben leer correctamente o incluso adultos ocupados.

2.4.- Mensajería *IoT*

En 1982, una máquina modificada de la empresa CocaCola se convirtió en el primer dispositivo conectado a Internet, apareciendo tras de sí distintos artículos académicos y, ya en 1 999, tomó fuerza el concepto de Internet de las Cosas. A partir de aquí, se incorporaron sensores y actuadores al *IoT* para aportarle "inteligencia".

Con la entrada del nuevo milenio, se pudo ver el incremento de dispositivos conectados a red, llegando a superar a la población mundial antes del 2010 (aproximadamente 12 500 millones de dispositivos inteligentes). Desde este punto, su crecimiento ha seguido siendo exponencial y, gracias a la aparición del 5G, ha podido avanzar aún más, mejorando las velocidades de las conexiones inalámbricas anteriores.

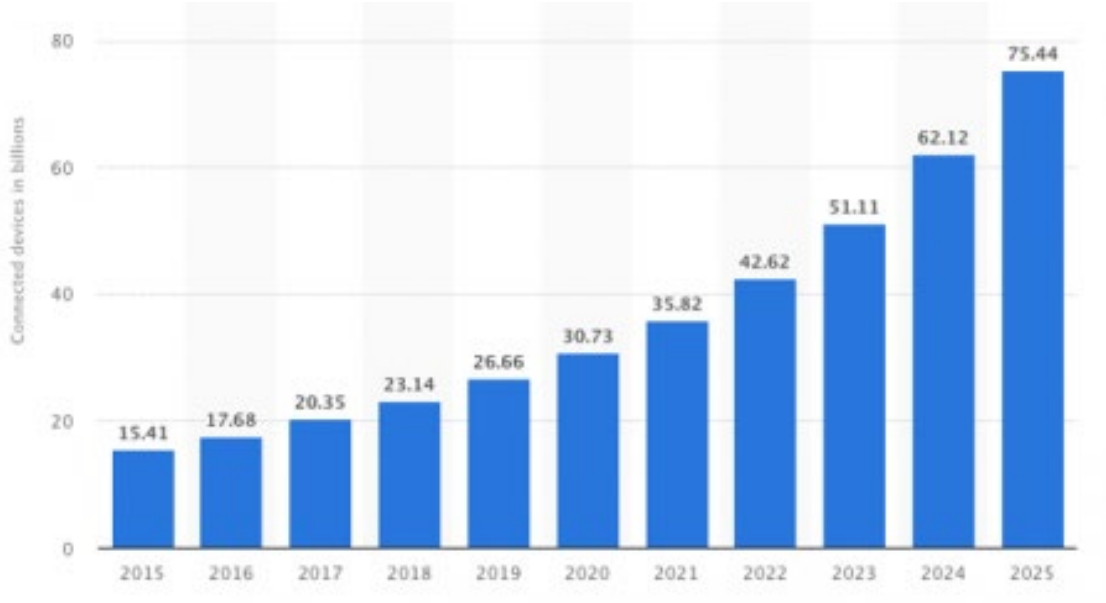

Figura 3. Número de dispositivos *IoT* existentes (en billones). [9]

A medida que han pasado los años, se puede ver que el número de dispositivos con conectividad a internet se ha ido disparando exponencialmente, estimándose que se disparará en los años futuros, como se observa en la gráfica de la figura 3.

2.5.- Áreas de aplicación del MQTT

Principalmente, el protocolo MQTT se emplea en el control de dispositivos *IoT*, puesto a su sencillez y su ligereza.

Si se navega un poco por internet, se puede encontrar su uso no solo en industria, sino que también es empleado en hogares o en pequeños proyectos. Algunos ejemplos de lo anterior serían los enlaces [8], [12], [16], [19], [22] o [36], pudiéndose ver la variedad de usos del MQTT como pueden ser comunicación de sensores entre microcontroladores o incluso el control de un dron.

3.- Análisis previo

3.1.- Definición del problema

La skill ha sido desarrollada inicialmente para el **control de un robot**, pero posteriormente se ha ido adaptando para convertirse en una **solución centralizada de control domótico**, gracias a diferentes proyectos de la fundación CARTIF como son los proyectos ANDIN e INOTEC (los cuales desarrollan un andador y un inodoro inteligentes respectivamente). Como bien se entiende en el título de este trabajo, se buscaba inicialmente controlar un robot para que, mediante comandos de voz, realizase tareas, pero posteriormente se han ido incluyendo varias funcionalidades adicionales.

El principal objetivo es hacer una UI de voz cómoda para el usuario y que sea capaz de controlar un robot, aunque también se empleará para controlar los dispositivos inteligentes conectados a red de la casa.

3.2.- Requisitos de la aplicación

Dentro de los requisitos de la skill, se busca poder controlar lo siguiente: un robot y/o actuador móvil, un andador y un inodoro inteligentes y la sensorización y la iluminación de las diferentes habitaciones del hogar. Para lograr esto, se deben crear *intents*, donde se implementan las frases para activar una función de la skill concreta y donde se ejecutarán partes del código, en el apartado 3.6 se explicará más detalladamente, junto al funcionamiento de Alexa.

Control del robot

Con una serie de *intents*, se administran diferentes órdenes para controlar un robot conectado por MQTT, las cuales son pedir al robot que se desplace a una habitación en concreto, que se desplace una distancia en una dirección pedida, que detenga la acción que está realizando actualmente y que active o desactive las funciones del mismo (ya sea hablar o hacer una videoconferencia).

Control del andador

Un *intent* dedicado a controlar los movimientos de un andador motorizado no sensorizado, indicándole hacia donde desea el usuario que se desplace, como ir hacia adelante, hacia atrás o que gire hacia uno de los lados (cuánto gire y cuánto se desplace se especificará dentro de la programación del propio microcontrolador).

Control del inodoro

Se ha creado un *intent* para controlar un inodoro inteligente, dando funciones como activar el ventilador de secado, el chorro limpiador, modificar la temperatura de la taza o del agua e incluso inclinarlo para ayudar al usuario a levantarse.

Sensorización

Con este *intent* se pedirá la información a un sensor específico de la casa, y como se dispone de algún sensor que nos aporta dos datos, también se puede especificar cuál de los dos desea el usuario.

Iluminación

Este *intent* actualmente está pensado para controlar las luces inteligentes de la casa, pudiéndole indicar la luz de una habitación en concreto, pero podría ser utilizado también para aquellos actuadores que solo tengan las acciones de encender y apagar o todo-nada.

Emergencia

Este último *intent*, permite al usuario enviar un mensaje de emergencia a la persona denominada como contacto de emergencia o llamando a los servicios sanitarios directamente.

3.3.- Asistentes de voz y sus plataformas

En la actualidad, existen numerosos asistentes de voz, pero los más utilizados son: Alexa, Cortana, Google Assistant y Siri (figura 4). Aquí se observarán las diferencias entre estos cuatro asistentes y porque se ha elegido Amazon Alexa como asistente del proyecto.

Figura 4. Iconos de los 4 asistentes de voz más usados. [9]

INFORMACIÓN	**Alexa**	**Cortana**	**Google Assistant**	**Siri**
Compañía	Amazon (2014)	Microsoft (2014)	Google (2016)	Apple (2011)
Frase de activación	'Alexa, …'	'Hola Cortana, …'	'Ok Google, …'	'Siri, …'
S.O.	Android, iOS	Android, iOS, Windows	Android, iOS	Cualquier dispositivo de Apple
Dispositivos	Familia Amazon Echo	Harma Kardon Invoke	Google Home	HomePod
Navegador de búsquedas	Bing	Bing	Google	Bing, Google
FUNCIONES				
Reproducir música	Sí	Sí	Sí	Sí
Llamadas	Sí	Sí	Sí	Sí
Mensajes	Sí	Sí	Sí	Sí
Imágenes	Sí		Sí	Sí
Crear recordatorios y alarmas	Sí	No	Sí	Sí
Noticias/Tiempo	Sí	Sí	Sí	Sí
Calendario	Sí	Sí	Sí	Sí
Información	Sí	Sí	Sí	Sí

Tabla 1. Comparación entre los 4 asistentes de voz más extendidos en la actualidad.

Como se puede ver en la tabla 1, los cuatro asistentes cubren similares funcionalidades aunque finalmente se ha elegido Amazon Alexa.

Una de las razones principales es la amplia gama de dispositivos que se pueden manejar y controlar, puesto que se pueden usar muchas marcas sin ningún tipo de problema. También cuenta con muchos servicios diferentes con los que permiten trabajar con domótica, como son NodeRed o HomeAssistant e implementar varias skills a la vez.

Otra razón muy importante, guarda relación con la manera de trabajar y desarrollar una skill única, puesto que existe la posibilidad de desarrollar en varios lenguajes conocidos, utilizar múltiples servicios de Amazón Web Services (AWS), para adaptar dispositivos IoT o crear conexiones con robots, o incluso utilizar servicios privados, ya sean servidores, equipos, etc. Todo esto facilita la tarea de desarrollar aplicaciones, ventaja que toma frente a sus competidores.

3.4.- El entorno de Amazon Alexa

El asistente de voz de Amazon es algo más que un simple asistente, sino que permite la coexistencia de diversos tipos de clientes y de desarrolladores.

Dentro de todo este entorno, existen cuatro "tipos": Amazon (proveedor de Alexa y que aporta parte de los servicios), los desarrolladores (dedicados a programar skills y que usan *Alexa Skill Kit* (*ASK*) de Amazon para crearlas), los fabricantes (crean los dispositivos físicos que Alexa controlará) y, por último, los clientes (usarán los servicios aportados por Alexa, algunas skills de los desarrolladores junto a los dispositivos de diferentes fabricantes).

3.5.- Arquitectura Alexa

Amazon Alexa se conforma por los dispositivos donde interacciona el usuario, el software del propio asistente en la nube, los diferentes servicios AWS (figura 5) a los que puede acceder y las distintas skills creadas por Amazon, empresas privadas o particulares.

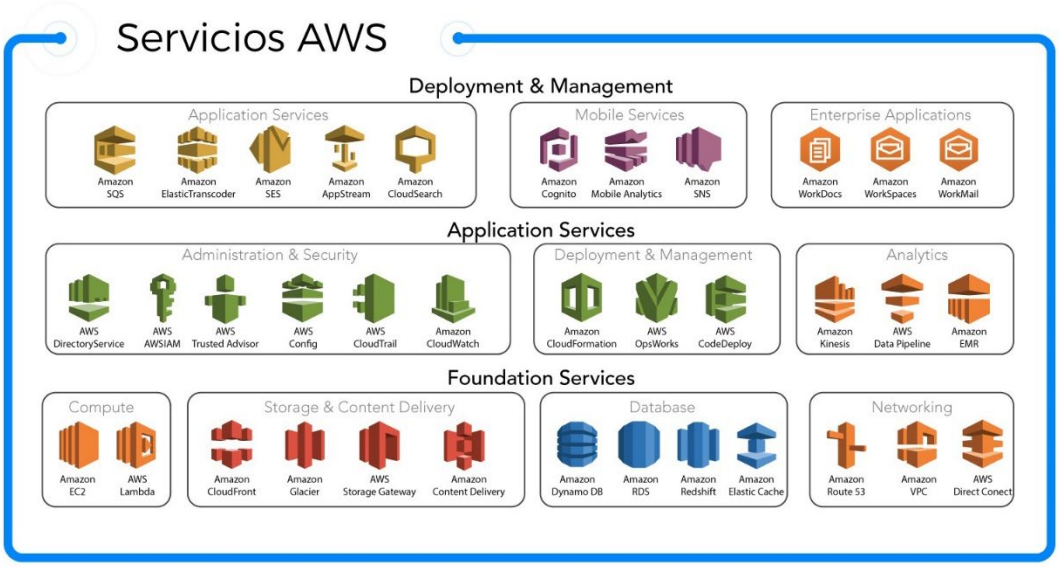

Figura 5. Esquema completo de los servicios de AWS. [46]

3.5.1.- Dispositivos

Alexa cuenta con varios dispositivos para albergarse, la familia Echo, entre los cuales están el Dot, el Plus, el Pop, el Show y el Studio, se pueden ver en la figura 6. Estos dispositivos no son más que un altavoz, un micrófono y un microcontrolador para enviar y recibir datos vía Wifi.

Para la realización de este proyecto, se empleará el **Alexa Echo Dot de 3ª generación**, debido a que es económico, no son necesarios los sensores de proximidad ni la pantalla de sus sucesores ni se necesitan características especiales.

Figura 6. Presentación de la familia Echo. [25]

3.5.2.- Entorno

Cuando se habla de entorno, se hace referencia al sistema operativo de Alexa. Lo que hace peculiar a este asistente es el potencial que posee en el reconocimiento por voz y la detección de peticiones.

El funcionamiento de una petición sigue el siguiente orden:

1. Se realiza una petición, capturada por el dispositivo y enviada a la nube.
2. El reconocimiento de Voz (ASR) de Alexa se encarga de capturar las palabras, transcribiendo la voz en texto.
3. El NLU, en inglés "*Natural Language Understanding*", detecta el significado de las palabras detectadas, orientando su respuesta hacia las intenciones del usuario.
4. La skill se pone en funcionamiento para obtener el resultado a devolver.
5. El TTS o "*Text To Speech*" es el servicio encargado de crear un audio desde un texto.
6. Se envía el audio al dispositivo y se reproduce.

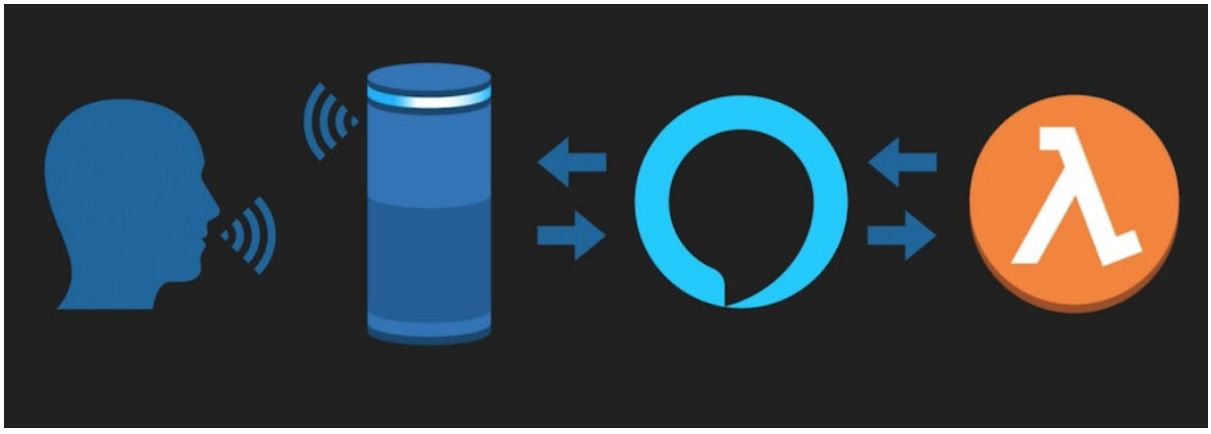

Figura 7. Rutas de interacción entre las partes de la skill. [5]

Con el esquema de la figura 7, se entiende que realmente la estructura de Alexa consta de los servicios de Amazon y de las conexiones del dispositivo junto a una serie de complementos o skills que aportan muchas funcionalidades al asistente y que son creadas por personas o empresas para cubrir alguna tarea concreta. Explicando un poco más detalladamente la figura anterior, se puede ver, de izquierda a derecha, al usuario comunicándose con el dispositivo que alberga Alexa, que es el siguiente dibujo, el tercero representa la skill y, por último, el servicio AWS Lambda, donde existe la programación de la skill, el acceso a otras funcionalidades de AWS, etc.

3.5.3.- Alexa Skill

Como ya se ha explicado, las *skills* aportan funcionalidades concretas, existiendo miles en la actualidad. Sin embargo, si Alexa dispusiera de todas ellas sin ningún tipo de control, podría ser catastrófico, por lo que, para poder instalar una *skill* que quiera el usuario, este debe descargarla desde el mercado de Alexa en la aplicación, añadiendo solo las funcionalidades deseadas.

Una skill consta de dos partes: interfaz y servicio. La primera se encarga de la descripción de la skill, qué funciones o *intents* dispone la misma y su integración en Alexa, mientras que la segunda administra peticiones y realiza toda la lógica para devolver un resultado.

Algo que es muy importante y que ya se ha expuesto, es la naturalidad y la fluidez de la conversación entre el usuario y la *skill*, lo cual hace que, durante el desarrollo de respuestas y de comprensión de peticiones, se busque crear varias opciones diferentes de respuestas o frases para que no sea una conversación cerrada, pero también se busca que las respuestas sean claras y rápidas, para no abrumar en exceso al usuario.

Amazon proporciona algunas *skills* que ya van integradas en el dispositivo, como son la reproducción de música con Spotify o Apple Music, realizar bromas, contarnos el tiempo meteorológico de alguna ciudad concreta, etc., pero los usuarios pueden crear sus propias skills gracias a *ASK*.

3.6.- Funcionamiento de Alexa

Antes de nada, se debe comprender bien que es lo que hace Alexa, puesto que los usuarios, al comprar un Echo o un Echo Dot, no compran un asistente integrado en dicho dispositivo, sino que lo que se esta comprando es en realidad un dispositivo IoT capad de escuchar y mandar a la nube de Amazon el audio recibido, como se ve en la figura 8.

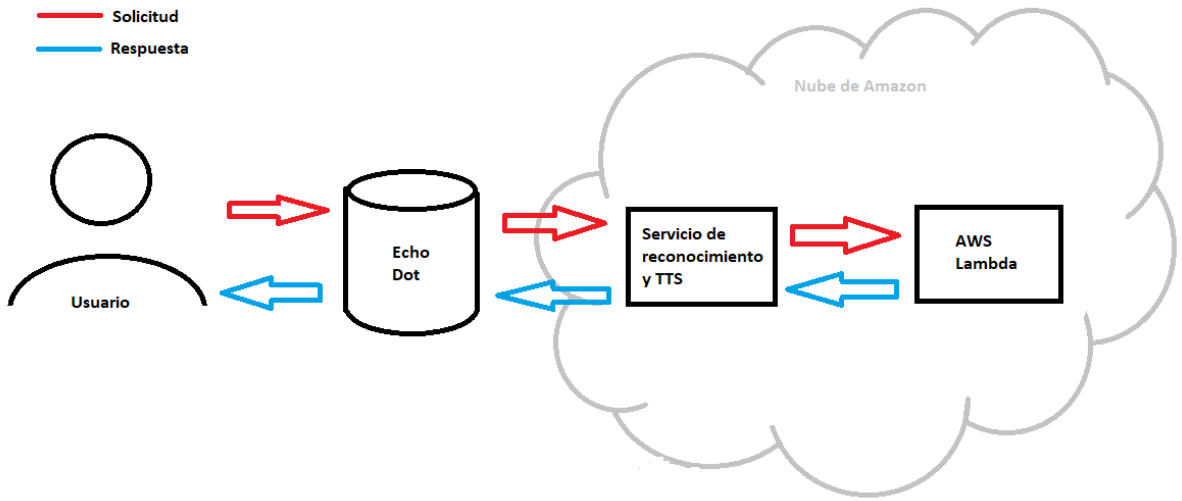

Figura 8. Rutas de interacción entre las partes de la skill, especificando lo que es tangible y lo que existe en la nube.

Para crear una skill, lo primero que se debe comprender es la estructura de llamadas del asistente, donde se llamará a Alexa, usará el nombre de invocación de nuestra skill y lo que se desea que haga. Se puede ver un ejemplo claro y sencillo (no propio de la skill diseñada en este proyecto) en la figura 9.

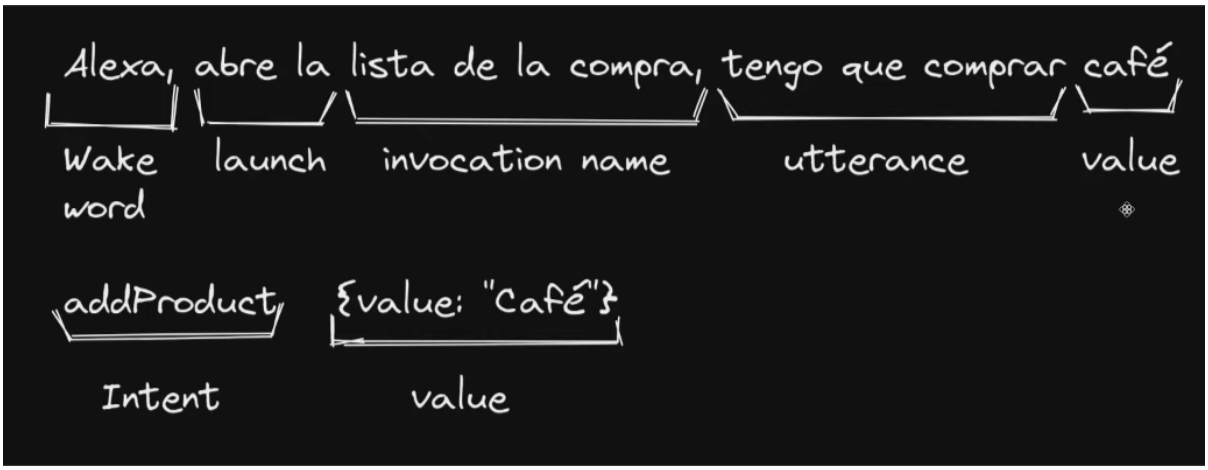

Figura 9. Ejemplo de comando de voz para utilizar un intent de una skill.

Otra cosa que se debe saber es la estructura de la programación de los *intents*, puesto que cada uno es una clase, permitiendo almacenar funcionalidad e información. Dentro de la misma, se definen al menos las dos siguientes funciones: "*can_handle*" (función propia de la clase para retornar el valor del *LaunchRequest* o del encargado de disparar la solicitud de activación de algún intent) y "*handle*" (función donde se define todo lo que el intent hará una vez se dispare su uso y que retorna la respuesta que Alexa nos dirá). Con el objetivo de crear

un código robusto, dentro de la función "*handle*", se han utilizado excepciones, para así poder saber si ha sucedido un error y que Alexa lo notifique.

Una vez ha sido comprendido todo lo anterior, se puede pasar a introducirse con los *intents*, que son los elementos donde definir lo que puede hacer una skill. Dentro de los mismos, se definen las **frases** con las que entenderá Alexa que el usuario quiere usar dicho *intent* y con los **slots o campos variables** se puede dar más funcionalidad a una misma frase, pudiendo sustituir dicho campo por varias palabras y facilitar en la programación la tarea de optimización de código. Con dichos *slots*, se han definido campos de direcciones en la casa, actuadores, sensores, variable a medir..., todo para lograr hacer una experiencia más natural y cómoda al usuario en cada interacción con la skill.

3.7.- Sistema de comunicación MQTT

Esta skill utiliza MQTT (*Message Queuing Telemetry Transport*) para la comunicación con los actuadores, los sensores, Alexa y servidores, debido a la facilidad y velocidad de los mensajes de dicho protocolo por la red.

MQTT es un protocolo de mensajería ligero basado en un modelo publicador/subscriptor, donde los publicadores o clientes mandan información a un "tema" o tópico específico y los subscriptores reciben solo la información de los temas que les interesan. Es un protocolo muy utilizado en *IoT* y sistemas de comunicación *M2M* (Máquina a Máquina) gracias a su simplicidad y eficiencia.

Para su correcto funcionamiento, se debe tener un *broker* que actúe de servidor de la mensajería, encargado de recibir y enviar información entre clientes, actuando como un nivel de aplicación en la arquitectura de redes, como se puede ver en la figura 10.

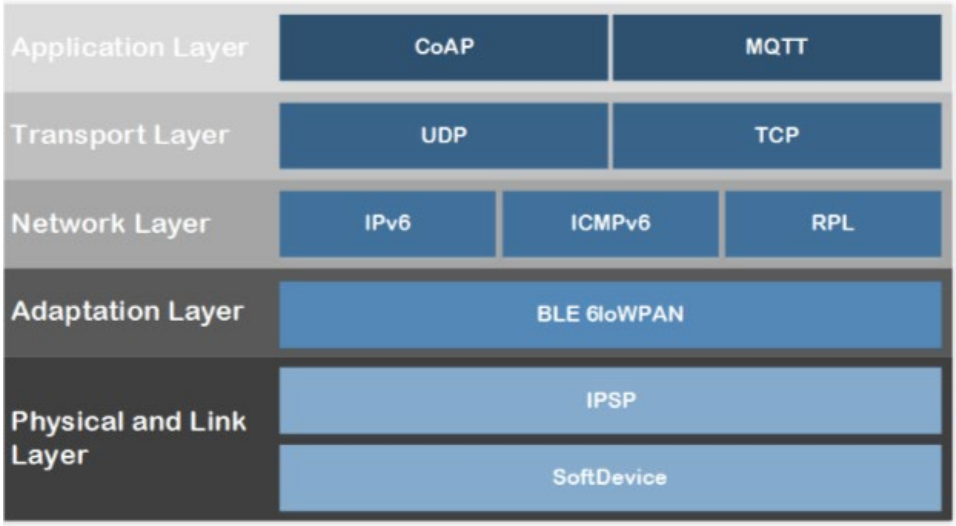

Figura 10. Esquema de niveles de red de comunicaciones. [7]

Una de las principales ventajas de este sistema de mensajería reside en que la forma de conectar los clientes y los subscriptores con el *broker* es intangible, o sea que no se necesitan conexiones físicas para su realización, sino que se realiza vía Wifi (figura 11).

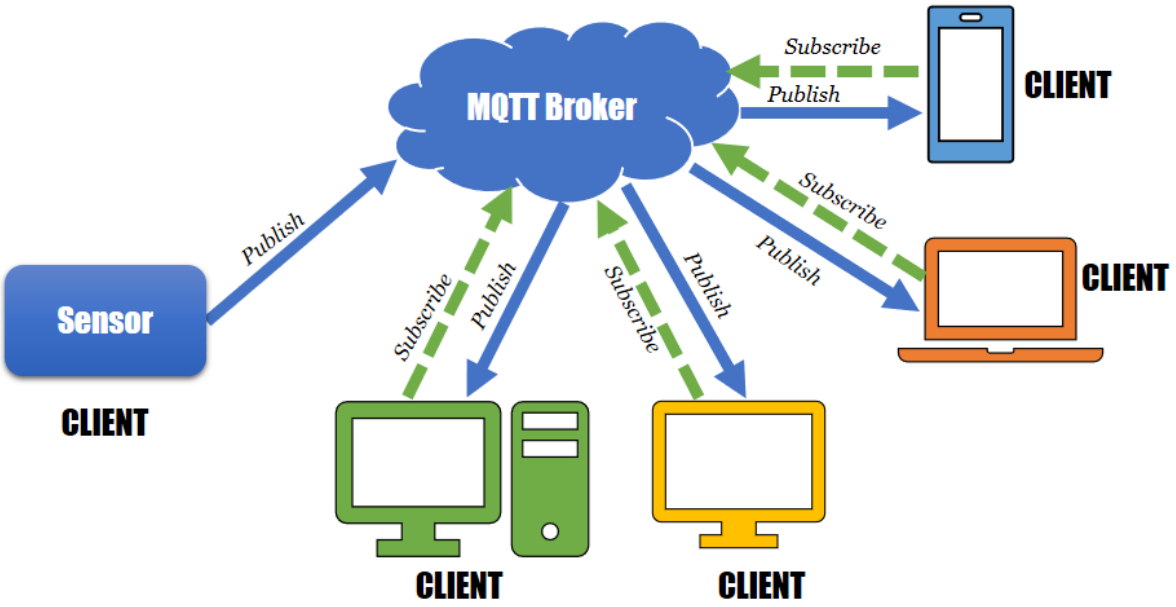

Figura 11. Visión global de un entorno MQTT. [1]

Dentro del *MQTT*, hay diferentes conceptos que se deben conocer como los 3 niveles de QoS (calidad de servicio), los temas/tópicos o los mensajes retenidos.

Los niveles de QoS distinguen tres casos, desde el 0 al 2, siendo el menos fiable el nivel 0 y el que más el nivel 2. El QoS 0 se envía un mensaje al subscritor solo una vez, sin confirmación de recepción (una entrega como máximo), el QoS 1 añade una espera del *broker* de confirmación de recepción del mensaje por parte del subscritor y, si no la recibe, vuelve a enviar el mensaje (entregado al menos una vez) y el QoS 2 realiza 4 pasos para garantizar que el mensaje ha sido recibido y que haya sido recibido una única vez (entrega una única vez). Para nuestro caso, solo será empleado el QoS 0.

Por otro lado, **los temas o tópicos** son estructuras como directorios (se ramifican usando ”/“) y tienen como objetivo simplificar y facilitar la selección de direcciones de tránsitos de datos.

Existe un parámetro a la hora de enviar mensajes para retenerlos, esto significa que el *broker* los puede almacenar si tienes marcado dicho parámetro, incluso en un largo periodo de tiempo. Esto último nos interesa para actualizar la información de sensores, puesto que puede interesarnos solo que envíe información cada vez que su valor varíe en un incremento o cada cierta cantidad de tiempo, logrando que su información se conserve en el *broker* y pudiendo acceder a ella en cualquier momento.

3.8.- Elección de microcontrolador

Se busca necesariamente que disponga de conectividad Wifi, por lo que, una solución buena y económica es el microcontrolador ESP Wroom 32 de la marca ESPRESSIF Systems. Este microcontrolador posee un doble núcleo Tensilica Xtensa LX6 que permite llegar a frecuencias de trabajo de 240GHz y otro coprocesador de bajo consumo. También posee Wifi 802.11 b/g/n, bluetooth v4.2 BR/EDR y BLE, una memoria ROM de 384KB, una SRAM de 520KB, una SRAM en RTC de 16KB y una PSRAM de 8MB, posee pines PWM, SPI, I2S e I2C, CAN, UART y muchas otras funcionalidades.

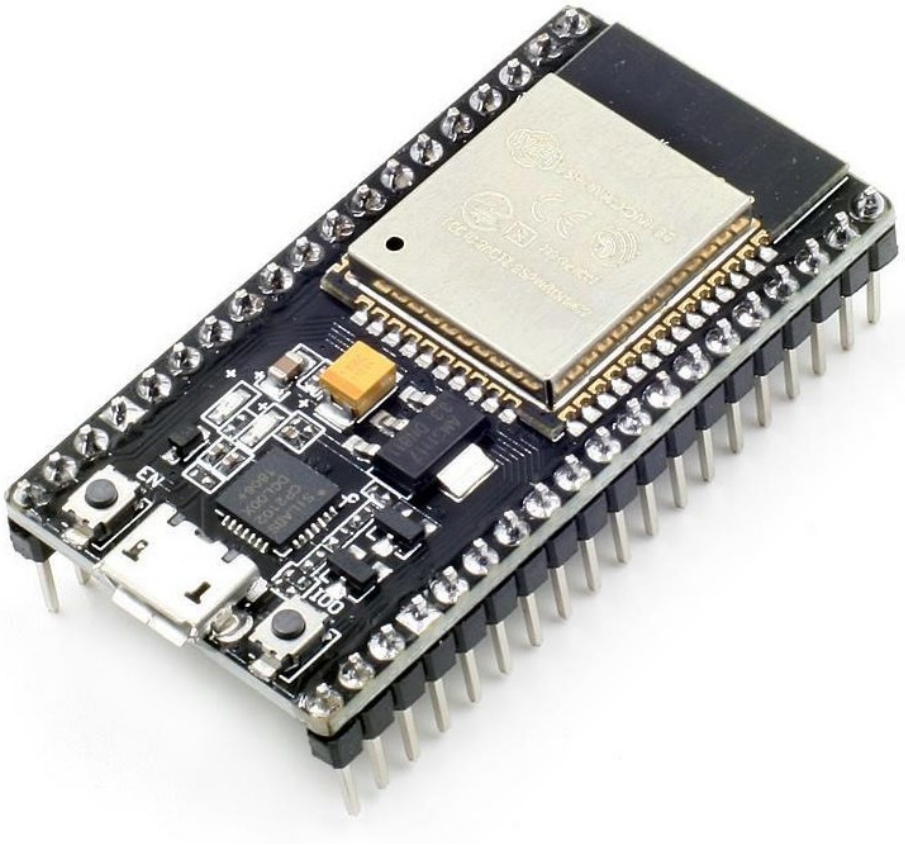

Figura 12. Microcontrolador ESP32.

Este microcontrolador aporta mucha versatilidad y ha tomado mucha fuerza en el mundo de los dispositivos *IoT* gracias a su conectividad y a su reducido tamaño. Aunque exista el ESP32 (figura 12), también puede usarse el chip en placas creadas por particulares o por empresas, ocupando poco espacio en proyectos. Algunos ejemplos son el andador inteligente o el inodoro inteligente desarrollados en la fundación CARTIF y que también pueden ser controlados por la skill.

Características	**ESP-32**
Procesador	Tensilica Xtensa LX6 32 bit Dual Core a 160MHz
Memoria RAM	520kB
Memoria Flash	<16MB
ROM	448kB
Alimentación	[2.2, 3.6] V
Rango de temperaturas	[-40, 125] ºC
Consumo de corriente	80 mA (promedio) 225 mA (máximo)
Consumo en modo Deep-sleep	2.5uA (RTC + memoria RTC)
Cooprocesador	ULP (consumo inferior a 150uA)
WiFi	802.11 b/g/n
Bluetooth	V4.2 BR/EDR y BLE
UART	3 puertos
I2C	2 interfaces
SPI	4 interfaces
GPIO (utilizables)	11 pines
PWM	16 canales
ADC	2 (<18 canales) (12 bits)
ADC preamplificador	Sí (bajo ruido hasta 60dB)
DAC	2 canales de 8 bits
I2S	2 interfaces
CAN 2.0	1 bus
Ethernet	10/100 Mbps MAC
Sensor de temperatura	Sí
Sensor de efecto Hall	Sí
Infrarrojos	Sí
Timers	Sí
Encriptación	AES, SHA, RSA, ECC
RNG	Sí
Encriptación flash	Sí
Secure Boot	Sí

Tabla 2. Características Hardware del ESP 32.

4.- Implementación y desarrollo

Cuando se crea una skill, se debe cumplimentar una serie de datos, como cuál es su nombre, la región local y el idioma, el uso de la misma (existen modelos ya creados para algunos usos), el lenguaje de programación (albergado en Node.js, Python o en uno propio) y si se quiere crear una skill desde un ejemplo de prueba como "*Hello World*" o aprovechar una skill ya existente.

A continuación, se irán exponiendo los diferentes apartados a culminar para la generación de una skill (sin personalizar) utilizando ASK:

Tipo de skill

Las herramientas de ASK nos permiten trabajar sobre alguna de las siguientes plantillas (figura 13), puesto que facilitan el trabajo a la hora de comenzar a desarrollar ciertas skills.

- *Custom Skill*: nos permite crear una *skill* personalizada, configurando los elementos del modelo de interacción, como son los *intents*, los slots y las *uterances* o frases de activación, y que se desarrolla casi en su totalidad en la página web de *Amazon Developer*. Esta es la **opción escogida** para el desarrollo de nuestra *skill*.
- *Flash Briefing Skill*: es un tipo de *skill* preparada para noticias y es capaz de leer un artículo o de reproducir un audio. El problema de esta *skill* es que se necesita actualizar constantemente para que no quede anticuada. Nos permite introducir el contenido en dos formatos: texto y audio.
- *Music Skill*: ya se ha comentado antes que se puede lanzar una orden para reproducir música desde una aplicación concreta.
- *Smart Home Skill*: una skill preparada para control y monitorización de dispositivos en la nube. En esta opción solo se puede interaccionar en el control de dispositivos dentro de una nube concreta, pero solo comandos de dicha aplicación, no lo que el usuario quiera. Esto último fue la causa por la cual se eligió una *custom skill* como tipo para albergar la aplicación.
- *Vídeo Skill*: este tipo de *skills* son, con diferencia, las más sencillas de usar, puesto que permiten al usuario localizar y aprender de un vídeo sin invocar una *skill* concreta.

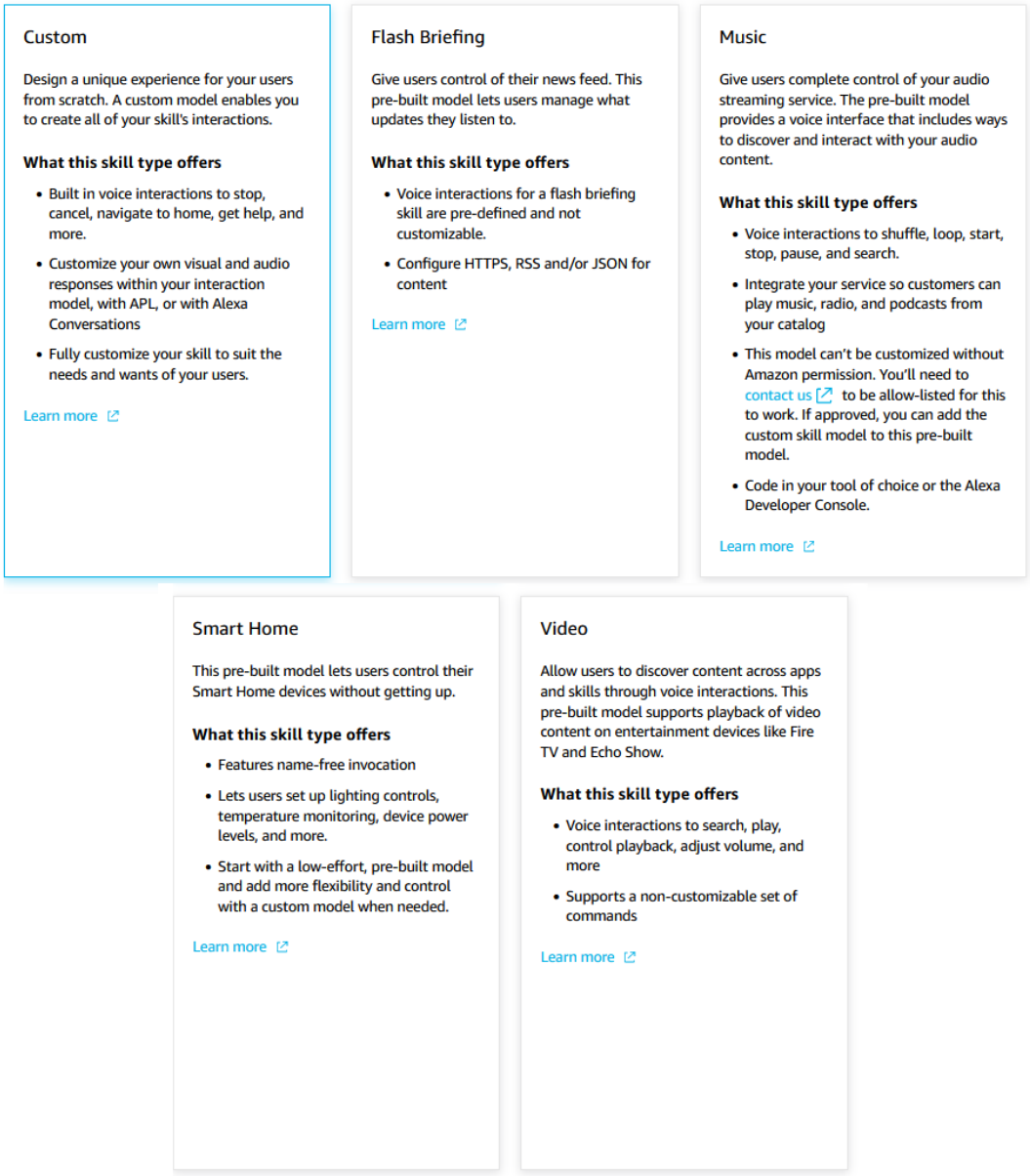

Figura 13. Tipos de skills que se pueden crear con SDK. [3]

Lenguaje de programación

Dentro de las opciones disponibles por *ASK*, se puede programar una skill con Node.js, **Python** (**el lenguaje elegido** por ser sencillo y constar de muchas librerías, en el apartado **4.3.1** se explica más detalladamente por qué ha sido elegido) o uno propio (pudiendo almacenar la interfaz y la lógica de la aplicación en un servidor externo a Amazon).

La diferencia entre los dos primeros y el último reside en el uso de la aplicación, puesto que los primeros necesitan usar los servicios de Amazon con ciertos límites que, de ser superados, comenzarán a costar dinero (figura 14).

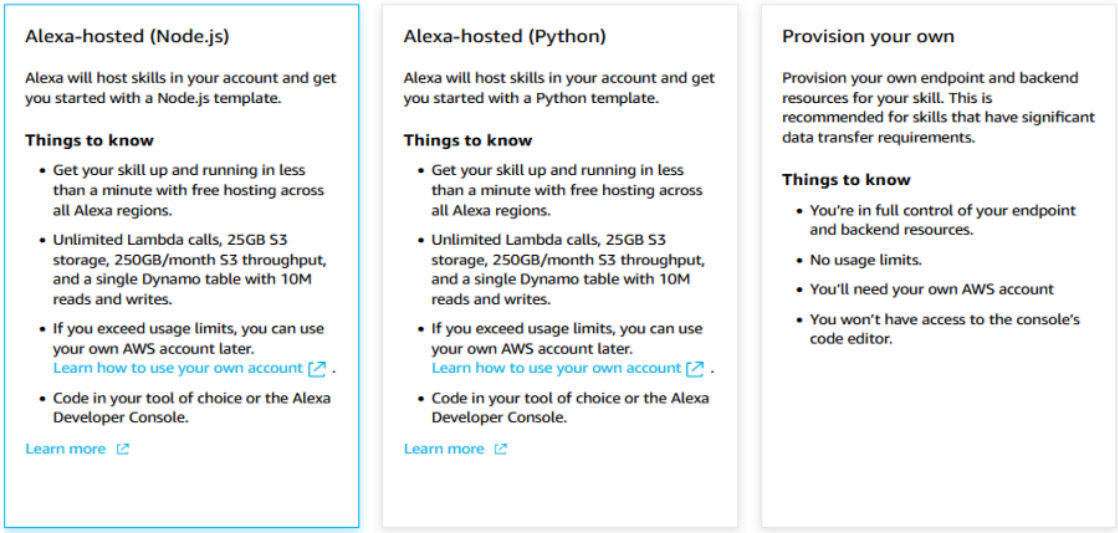

Figura 14. Selección del lenguaje de programación y del almacenaje de nuestra skill, así como las diferencias entre ellos. [3]

Skill

En este último apartado, se puede elegir hacer una *skill* de cero o aprovechar algunos modelos de *skill* ya creados para poder modificarlos posteriormente (todos los ejemplos que dispone aparecen en la figuta 15). Esta *skill* partirá de cero.

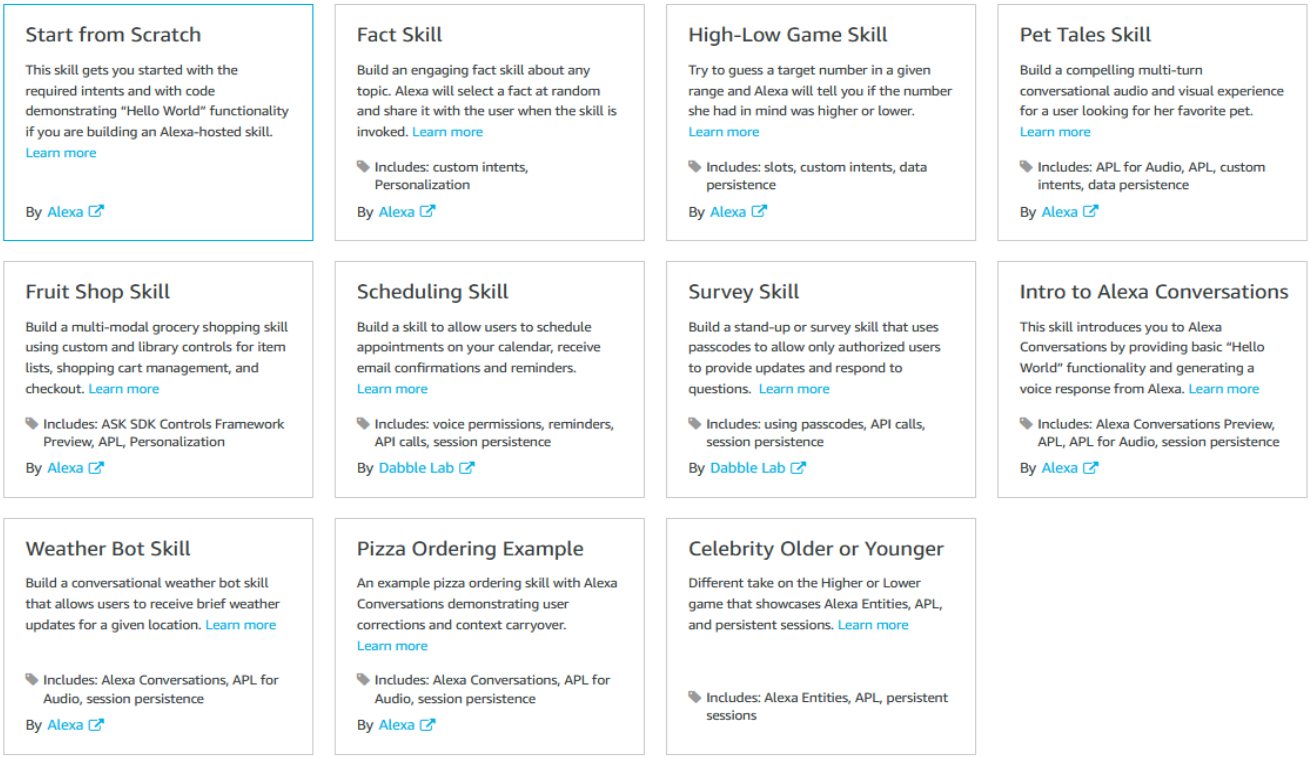

Figura 15. Ejemplos de skill ya preconfiguradas. [3]

4.1.- Descripción de la skill

Las *skills* tienen dos partes, una parte servidor, la cual será en su totalidad en el *SDK* (aunque se podrían incluir servicios de *AWS* como *Lambda*) y una cliente, que determina el modelo de interacción.

4.2.- Front-end

Esta es la parte que enfrentan los usuarios a la hora de utilizar una *skill*, que se desarrollará en el *Alexa Developer Console* (figura 16). Esta herramienta web proporcionada por Amazon nos permite crear tanto el *front* como el *back-end* (donde se programa la aplicación), además de permitir al desarrollador testear las actualizaciones que se vayan realizando desde la misma web.

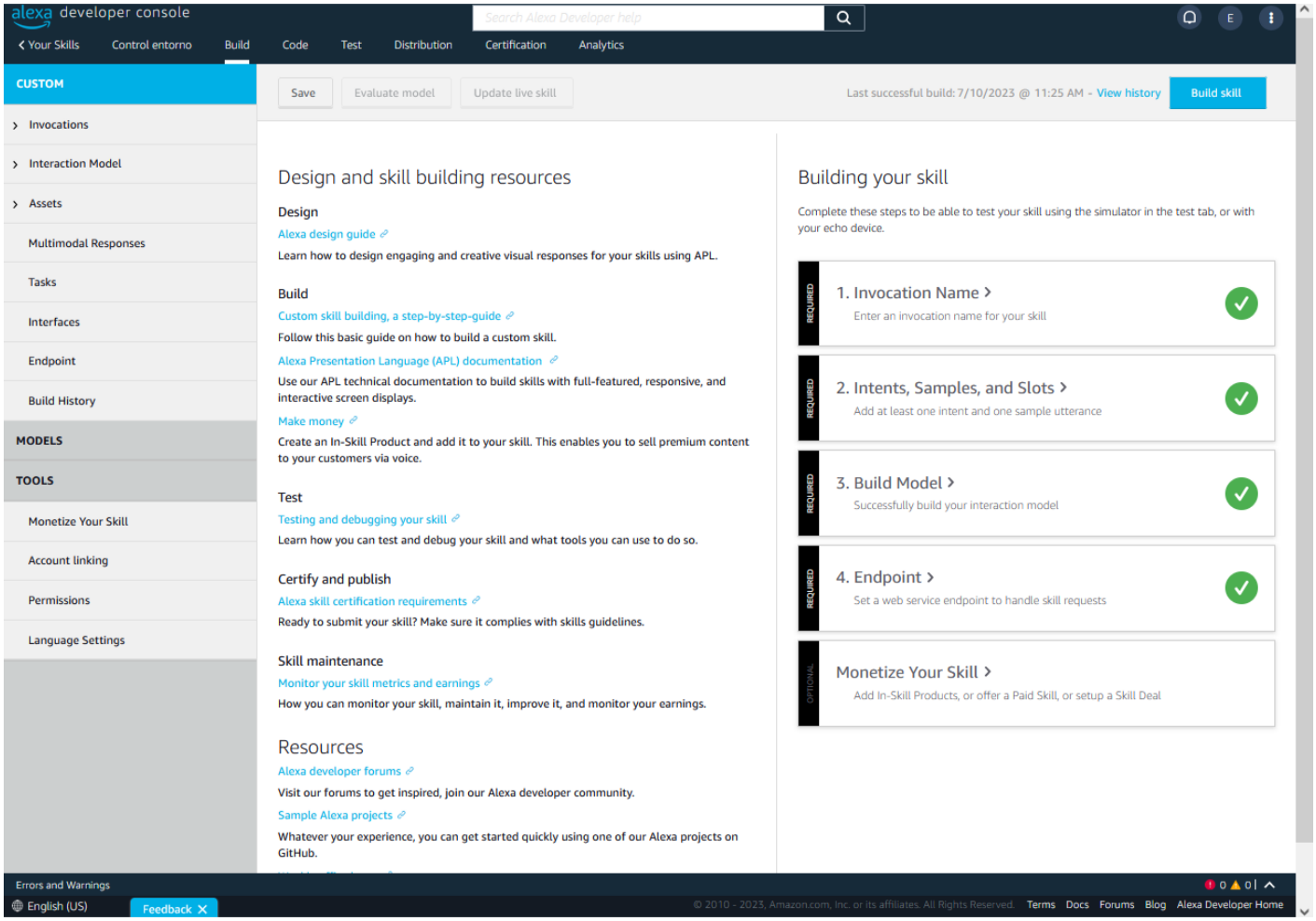

Figura 16. Visión global del entorno de desarrollo de skills de SDK. [3]

4.2.1.- Invocación

Aquí se indica la palabra o frases de activación de la *skill* (figura 17). En este caso, para iniciar la skill, el usuario deberá decir: "*Alexa, abre* ***control entorno***.", y ya podrá utilizar todas las funciones de la *skill*.

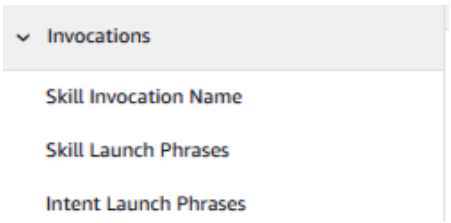

Figura 17. Definición de invocaciones. [3]

4.2.2.- Modelo de interacción

En este desplegable serán definidos las funciones o *intents* de la skill desarrollada (figura 18). Estos *intents* permiten definir las frases con las que serán activados.

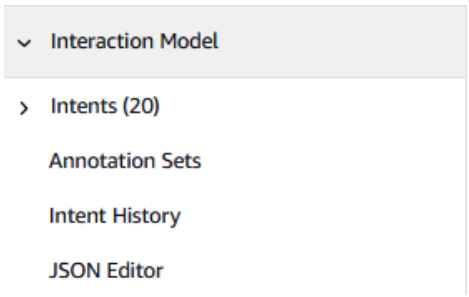

Figura 18. Componentes del modelo de interacción. [3]

La manera en la que serán creados los *intents* en el *front-end* es como se observa en la Figura 19, incluyendo un nombre al *intent* e incluyendo frases con las que se activará dicha función (también se pueden emplear slots, que serán explican más adelante). Lo que se debe hacer aquí es abrir un amplio abanico de frases para que cualquier usuario pueda usar los *intents* de manera natural para ellos, tampoco se debe ser redundantes, pero debe disponer de tanta libertad como se desee para crear infinidad de frases.

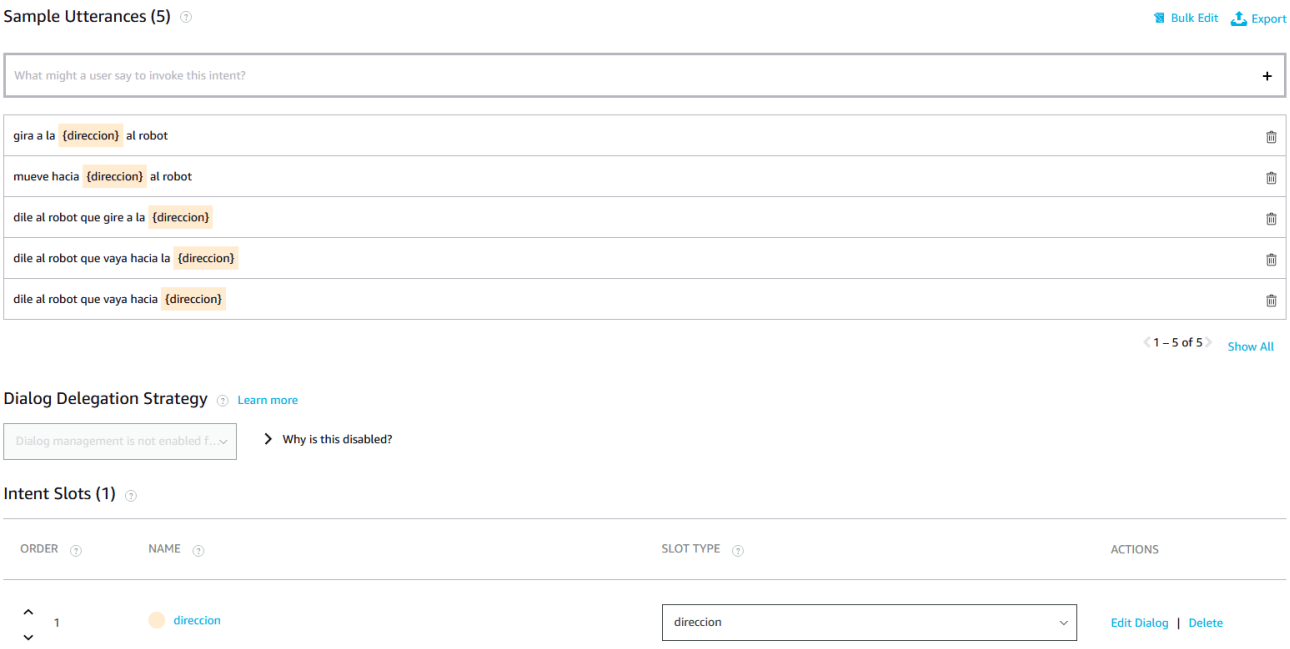

Figura 19. Ejemplo de las frases de activación del intent MoveTo. [3]

A continuación, se puede ver que la *skill* desarrollada consta de 20 intents en la Figura 20, de los cuales uno es un ejemplo para comprender el funcionamiento y la programación básica de los *intents* (*HelloWorldIntent*), 14 *intents* están dedicados a satisfacer las funciones propias de la skill para cumplir los requisitos y al final existen 5 *intents* obligatorios para el funcionamiento básico de la skill que nos aporta el *SDK* (Skill Developer Kit).

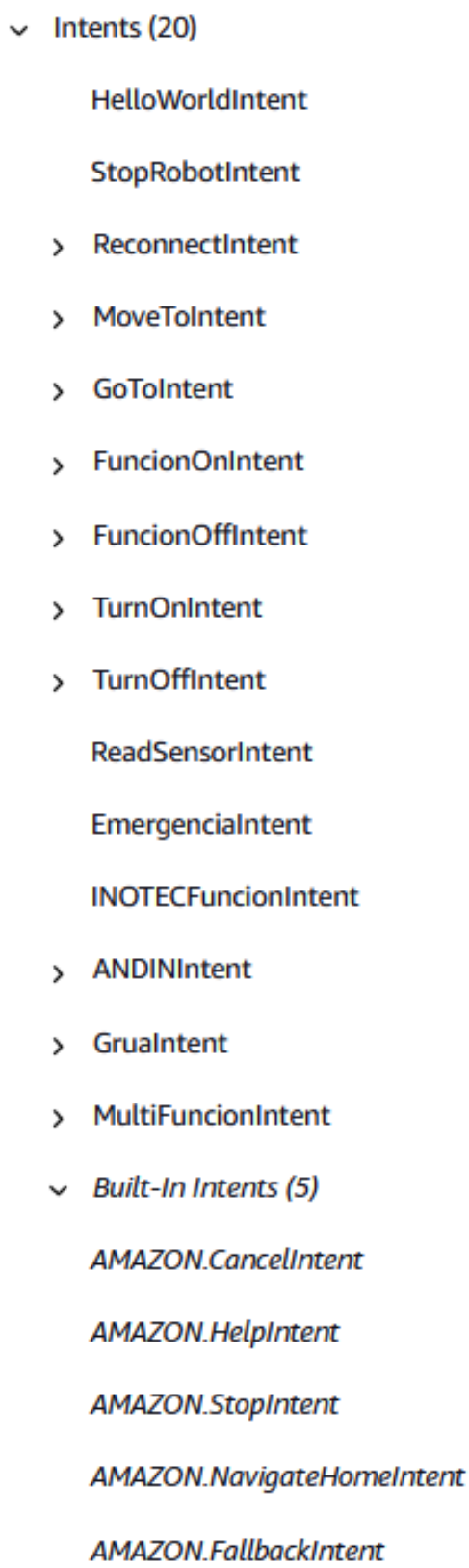

Figura 20. Todos los intents de la skill desarrollada. [3]

4.2.3.- Slots

Estos elementos se utilizan para crear *campos*, es decir, crear una variable que puede tener varios valores (p.e.: el slot *direccion* puede tener los valores *izquierda*, *derecha*, *adelante* y *atrás*). Su principal función es facilitar la creación de frases de activación sin ser redundantes y también poder crear en un mismo *intent* opciones según el valor de las mismas. Se pueden ver los utilizados en la figura 21.

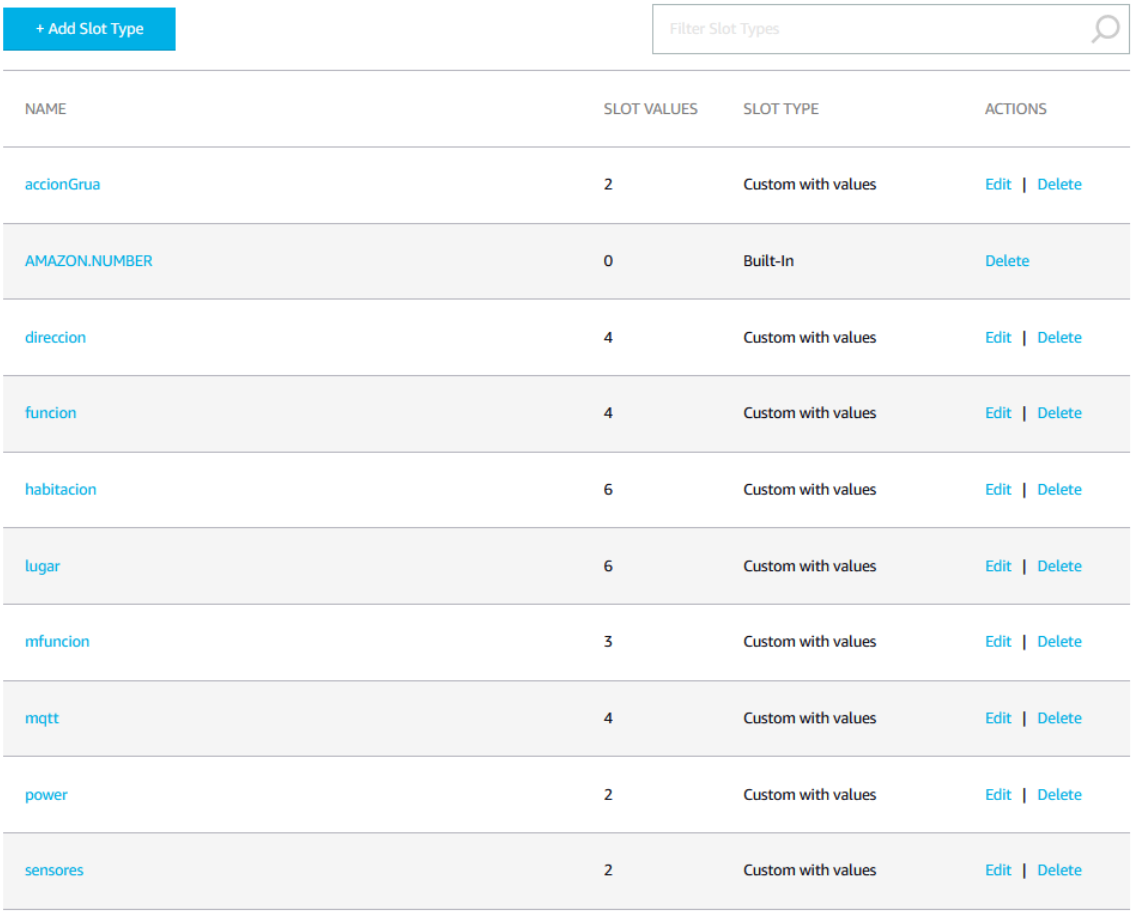

Figura 21. Slots empleados para el funcionamiento de la skill. [3]

4.3.- Back-end

Aquí es donde reside toda la lógica de la *skill* y la parte compleja de la misma, puesto que se debe buscar funcionalidad a la vez que sencillez. Con esto último, hacemos referencia a que la *skill* necesita tener una cantidad apropiada de *intents* para lograr cumplir todos los objetivos pedidos, pero sin crear dos iguales para evitar redundancias o colisiones entre sí (aquí entran en importancia los *slots*).

4.3.1.- Lenguaje de programación

El lenguaje en nuestro caso es **Python**. El principal motivo es por su sencillez y por el conocimiento previo del que disponía ya con anterioridad, aunque me ha servido para profundizar mucho en el mismo, además de aprender el uso de varias librerías.

Ventajas

- Lenguaje de alto nivel.
- Polivalente y de paradigmas.
- Alto número de bibliotecas y frameworks.
- Portabilidad.
- Gratis y de código abierto.
- Sencillo.
- Comunidad fuerte.

Desventajas

- Lenguaje interpretado que lo hace más lento que los lenguajes compilados como C++.
- Alto consumo de memoria.

Usando este lenguaje, el archivo *lambda_function.py* se forma de la siguiente manera: un encabezado donde serán definidas las librerías, funciones de acceso libre para los *intents* y variables fijas, la programación de los diferentes *intents* y, por último, el *SkillBuilder*. En los dos siguientes subapartados se tratará todo esto en profundidad.

4.3.2.- Librerías empleadas

Se han empleado las librerías *logging* y múltiples dependencias de *ask_sdk_core*, propias de Amazon Alexa para el correcto funcionamiento de los servicios Alexa, junto a la librería *random* para calcular número aleatorios y la librería *paho* para la comunicación *MQTT* con el *Broker* (ver figura 22).

```
# Este código es una skill para transmitir mensajes MQTT a un broker para controlar un robot y el entorno de un hogar domotizado

import logging
import ask_sdk_core.utils as ask_utils

from ask_sdk_core.skill_builder       import SkillBuilder
from ask_sdk_core.dispatch_components import AbstractRequestHandler
from ask_sdk_core.dispatch_components import AbstractExceptionHandler
from ask_sdk_core.handler_input       import HandlerInput

from ask_sdk_model import Response


# ===============================================================================================================================================


## Librerías para obtener números random y conexiones con servicios MQTT:

import random

import paho.mqtt.client  as mqtt_client


## Librería para acceder a información de APIs en formato JSON:

import requests
import json
```

Figura 22. Código 1: Importación de librerías. [3]

Paho es una librería creada por la fundación *ECLIPSE* para centralizar las funciones de MQTT, con la cual permite conectarse al broker, así como crearuna conexión con Alexa como un publicador y un subscriptor de información. Algunas de las funciones son las siguientes:

- Importación de la librería: para simplificar el uso de la librería, se utilizará con el nombre de "*mqtt_client*", puesto que el nombre de la dependencia es demasiado largo.

  import paho.mqtt.client as mqtt_client

- Conexión al broker: con los siguientes comandos se puede realizar la conexión, habiendo definido una id de cliente, disponiendo del certificado para poder pasar la seguridad TLS y un usuario y una contraseña de acceso, además de después conectarnos al broker indicando la dirección y el puerto.

  client = mqtt_client.Client(client_id)

  client.tls_set(ca_certs = RUTA_CERTIFICADO)

  client.username_pw_set(username, password)

  client.connect(broker_address, port)

- Desconexión del broker: lo que hace esta función es cerrar la conexión de un cliente para no poder seguir enviando o escuchando un tópico (sería necesario reconectar al broker si se desea volver a enviar o recibir información).

client.disconnect()

- Publicador: con esta función se logra mandar un mensaje a un tópico específico, además de indicar el QoS deseado y si se quiere que el broker retenga la información.

```
client.publish(topic, "Alexa conectado", qos = 0, retain = False)
```

- Subscriptor: con la siguiente función se logra capturar un mensaje de un tópico al que el cliente se ha subscrito.

```
def subscribe(client):
    def on_message(client, userdata, msg):
        value = str(msg.payload)
        client.publish(topic, value, qos = 0, retain = False)
    client.subscribe(topicData)
    client.publish(topicPeticion, "temperature", qos = 0, retain = False)

    client.on_message = on_message
```

Por otro lado, JavaScript Object Notation **(JSON)** es una librería que se emplea para poder trabajar con el formato JSON y, en este caso, es compatible con los mensajes de MQTT. Usar un mensaje JSON nos permite enviar mucha información con una sola variable.

- Importación de la librería: se hace para poder acceder a las funciones de la librería.

```
import json
```

- Formato JSON: aquí está un ejemplo de cómo sería una variable JSON.

```
obj = {
            "IdOrden": "1057206722",
            "Orden": "Hablar",
            "Campo": {
                "Frase": "Buenos días, Pepe",
                "Volumen":""
                }
          }
```

- Codificación: con cualquiera de las dos siguientes funciones se logra convertir diccionarios al formato JSON, los cuales funcionan de una manera similar.

```
json_data = response.json()
```

```
json_msg = json.dumps(obj)
```

- Acceso a un valor concreto: se puede indicar con este comando que campo de datos se desea conocer de la variable JSON.

  data = json_data[jd] *# siendo jd un campo de un objeto JSON (p.e.: "Frase" de obj)*

Finalmente, se está empleando **Request** para acceder a la información de una API que, utilizando JSON, se pueden acceder a numerosas variables solo con una dirección.

- Importación de la librería: se impondrá la versión 2.25.0 de la librería en el archivo "*requirements.py*".

  import requests

- Acceso a información de una API: esta será la única función que se usará de la librería, puesto que está pensada para acceder a datos de los sensores, los cuales están expuestos en una API con un formato JSON, cuya dirección corresponde con la variable *url*.

  response = requests.get(url)

4.3.3.- Descripción del código

Encabezado

Aquí se definirán las librerías explicadas en el apartado anterior, las variables referidas a los tópicos, los certificados de seguridad para conectar la skill al broker MQTT e id del cliente para conectar la skill al broker MQTT y algunas funciones dedicadas a conectarse al broker MQTT, a subscribirse a un tópico MQTT, acceder a datos de una API.

Intents

- *LauncherRequestHandler*

Este intent es el encargado de que, según se inicie la skill, Alexa conteste al usuario que se ha iniciado dicha skill y es donde se realiza la conexión con el broker, saltando un error de conexión si no lograse conectarse y, por ello, no inicializar la skill.

- *ReconectIntentHandler*

La finalidad de este intent es refrescar la conexión con el broker, ya sea por algún problema al enviar información o al recibirla.

- *StopIntentHandler*

Primer intent de control del robot con el que se manda un mensaje al MQTT para finalizar una acción que esté realizando actualmente.

- *GoToIntentHandler*

Intent para mandar órdenes MQTT al broker sobre posiciones en la casa concretas y hacer que el robot vaya a dicha habitación. Aquí se emplea el slot "lugar" para que, con un mismo intent, pudiendo enviar diferentes posiciones como puede ser "salita", "baño", "cocina", etc.

- *MoveToIntentHandler*

Es muy similar al intent anterior, con la ligera diferencia de que aquí se usará el slot "direccion" para indicar que el robot avance hacia adelante o hacia atrás o que gire.

- *MFuncionIntentHandler*

Aquí se describirán diferentes acciones múltiples que será capaz de ejecutar el robot, como puede ser hablar y moverse o hacer una videoconferencia, hablar y moverse. Resumiendo, este intent está dedicado para combinar las funciones de las que dispone el robot.

- *FunctionOnIntentHandler* y *FunctionOffIntentHandler*

Ambos intents se emplean para activar y desactivar funciones del robot, ya sea el lidar, la cámara, algunas funciones como mapeo, etc. Estos handlers se podrían haber reducido a uno, pero complicaría mucho el momento de crear las frases para el mismo, puesto que se tendría que distinguir entre los verbos de activar y de desactivar, este motivo ha hecho que se llegue a la solución de crear dos intents separados.

- *TurnOnIntentHandler* y *TurnOffIntentHandler*

Estos dos intents son utilizados para encender o apagar dispositivos, actualmente se usan para controlar unas luces, pero se puede extrapolar a cualquier dispositivo. El motivo de que haya dos intents es el mismo que el explicado en los intents FunctionOnIntentHandler y FunctionOffIntentHandler.

- *ReadSensorIntentHandler*

Con este intent se conecta la aplicación de Alexa con una API donde están albergados los datos de los sensores. En esta dirección, si se pusiera en el navegador, lo que se obtendría es un JSON que alberga la información de la temperatura, la humedad y la luminosidad de diferentes habitaciones.

- INOTECFuncionIntentHandler

Se ha creado un intent para controlar un inodoro inteligente, dando funciones como activar el ventilador, el chorro limpiador, modificar la temperatura de la taza, mover el inodoro o incluso inclinarlo para ayudar al usuario a levantarse.

- ANDINIntentHandler

Aquí controlaremos, usando comandos de voz, un andador motorizado. Se emplearán órdenes sencillas como ir hacia adelante o hacia atrás, girar, etc., puesto que el andador no dispone de sensorización y autonomía suficiente para realizar órdenes complejas.

- *EmergenciaIntentHandler*

Este intent está destinado a notificar al servidor de alguna emergencia por parte del usuario, ya sea por una situación para avisar a emergencias, para llamar a algún contacto, etc.

- *HelpIntentHandler*

Este intent hace que Alexa nos pueda ayudar con alguna cuestión si el usuario no sabe muy bien la orden o lo que desea hacer con la skill.

- *CancelOrStopIntentHandler*

Esta función hace que se finalice o detenga la ejecución de cualquier otra de la skill.

- *FallbackIntentHandler*

Lo que se hace aquí es que, si Alexa no ha comprendido bien lo que se ha dicho, solicita al usuario que vuelva a pedir la activación de la función que se intenta utilizar.

- *SessionEndedRequestHandler*

Al usar esta función, se fuerza que la skill se cierre y no poder volver a utilizar ninguna de sus funciones hasta que el usuario vuelva a activar mediante el uso del LauncherRequestHandler.

- *IntentReflectorHandler*

Se usa para el modelo de interacción en el test y debug, pero simplemente repite el intent invocado.

- *CatchAllExceptionHandler*

Recoge todos los errores de sintaxis o de rutas. Si se recibe un error como que el handler no existe o no se encuentra, se deberá comprobar que el mismo handler que se está llamando este declarado en el *SkillBuilder* o si la frase utilizada esté bien declarada.

SkillBuilder

Esta parte es la más importante del código, puesto que compila los intents e integra la prioridad entre ellos a la hora de hacer una búsqueda sobre cuál se ha activado. El objeto *SkillBuilder* funciona como un punto de acceso para nuestra skill, redirigiendo las peticiones y respuestas a los respectivos handlers. Asegúrate de que los nuevos handlers o interceptores que has definido, estén incluidos y por encima de *IntentReflectorHandler*, puesto que el orden importa (se procesan de arriba a abajo), cosa que se puede ver en la figura 23.

```
# El objeto SkillBuilder funciona como un punto de acceso para nuestra skill, redirigiendo las peticiones y
# respuestas a los respectivos handlers. Asegurate de que los nuevos handlers o interceptors que has definido,
# estén incluidos y por encima de IntentReflectorHandler, puesto que el orden importa (se procesan de arriba a abajo).

sb = SkillBuilder()


sb.add_request_handler(LaunchRequestHandler())

# ====================================================

sb.add_request_handler(ReconnectIntentHandler())
sb.add_request_handler(StopRobotIntentHandler())
sb.add_request_handler(GoToIntentHandler())
sb.add_request_handler(MoveToIntentHandler())
sb.add_request_handler(MultiFuncionIntentHandler())
sb.add_request_handler(FuncionOnIntentHandler())
sb.add_request_handler(FuncionOffIntentHandler())

sb.add_request_handler(TurnOnIntentHandler())
sb.add_request_handler(TurnOffIntentHandler())
sb.add_request_handler(ReadSensorIntentHandler())

sb.add_request_handler(INOTECFuncionIntentHandler())
sb.add_request_handler(ANDINIntentHandler())
sb.add_request_handler(GruaIntentHandler())

sb.add_request_handler(EmergenciaIntentHandler())

# ====================================================

sb.add_request_handler(HelloWorldIntentHandler())
sb.add_request_handler(HelpIntentHandler())
sb.add_request_handler(CancelOrStopIntentHandler())
sb.add_request_handler(FallbackIntentHandler())
sb.add_request_handler(SessionEndedRequestHandler())

sb.add_request_handler(IntentReflectorHandler())           # IntentReflectorHandler debe ser la última, antes del
sb.add_exception_handler(CatchAllExceptionHandler())       # detector de excepciones.


lambda_handler = sb.lambda_handler()
```

Figura 23. Código 2: Definición de los handlers en el SkillBuilder. [3]

4.4.- MQTT

Aquí se expondrá el resto de las partes que conformar el sistema aparte de la skill, ya sea el servidor de mensajería, algunos actuadores, los sensores conectados y los temas de publicación.

4.4.1.- Broker

Actúa como puente de conexión entre publicadores y subscriptores, almacenando información mandada a través del mismo, es decir, es un servidor donde se transmite la información y donde también puede conservarse. Un ejemplo sencillo es la figura 24.

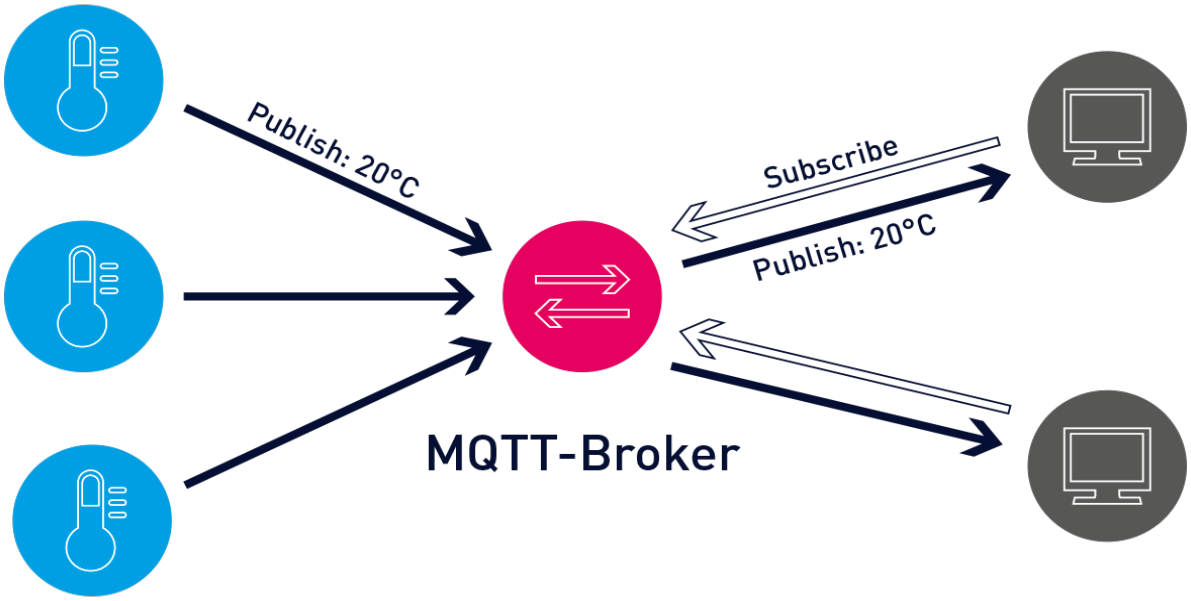

Figura 24. Esquema del ecosistema MQTT y conexión entre sensores y equipos. [41]

Existen dos tipos de brokers, un tipo es **local**, el cual consiste en que el servidor será un equipo propio (como podría ser una Raspberry Pi o un PC) y otra es utilizar un broker dispuesto por otras empresas (como Eclipse, Mosquitto, etc.) o **uno propio**, siendo el caso que se utilizará en este proyecto. Aunque en este caso, existen dos brokers comunicados entre sí: uno de ellos local (al que se conectarán los dispositivos inteligentes y al cual se accede estando en la red de la casa, no de forma remota) y otro existente en la red (al que se conecta Alexa). Este último dispone de mayor seguridad debido a que está más expuesto a ciberataques, necesitando certificados de acceso.

4.4.2.- Actuadores

Los actuadores que controlará la skill desarrollada serán los siguientes dispositivos:

- Robot: dispositivo con sensorización capaz de ejecutar órdenes complejas y de funcionar de manera autónoma que dispone de diferentes funcionalidades y que puede analizar datos en tiempo real (en la figura 25 se observa el robot empleado).

Figura 25. Robot Temi.

- Andador: elemento motorizado y domotizado por un proyecto de CARTIF el cual se busca aportar funcionalidades por voz para lograr desplazarlo y realizar aproximaciones hacia sitios concretos, puesto que no cuenta con ningún tipo de sensorización de entorno (en la figura 26 se muestra el dispositivo).

Figura 26. Proyecto ANDIN.

- Inodoro: elemento domotizado por un proyecto de CARTIF que busca aportar comandos por voz para activar las diferentes funcionalidades del mismo: elevar o bajar la taza, activar el ventilador de secado, calentar o enfriar la taza o activar el chorro limpiador (en la figura 27 se ve el inodoro inteligente).

Figura 27. Proyecto INOTEC.

Microcontroladores ESP-32

Todos los microcontroladores ESP-32 estarán programados en C, usando las librerías de *PubSubClient* y *WiFiClientSecure* para la comunicación con el broker. Todos ellos harán inicialmente una conexión a la red Wifi, posteriormente al broker, enviando un mensaje al tópico de avisos para comprobar su conexión al mismo y, por último, permanecerá a la escucha del tópico correspondiente para operar en función de lo que reciba.

La finalidad del uso de estos microcontroladores es controlar el andador y el inodoro, encargándose del control de motores, sensores, etc. Estos dispositivos son los que nos permiten apodarles como inteligentes.

Aunque vayan a tener objetivos diferentes, hay partes que son comunes entre todos ellos, como pueden ser las funciones de conectarse al Wifi, conectarse al broker, publicar en un tópico y subscribirse a uno (junto con el análisis de la información recibida).

4.4.3.- Sensores

Actualmente el sistema está conectado a tres sensores como los de la Figura 28, los cuales ofrecen datos de temperatura, humedad y luminosidad.

Figura 28. Sensor de temperatura, humedad y luminosidad.

Aunque también se podrían usar sensores junto a microcontroladores u otros dispositivos inteligentes.

4.4.4.- Tópicos

Se han definido diferentes tópicos a los cuales se conectará el asistente Alexa tanto para enviar información, como para recibirla:

- "*avisos*": a este tema se publicará la información desde Alexa una vez se conecta y una vez se desconecta para informar al personal encargado de controlar el *broker* y pueda ver si la conexión se realiza de forma correcta.
- "*emergencia*": dentro de este tema, se enviarán mensajes para notificar el requerimiento de ayuda por si el usuario se ha caído y no es capaz de levantarse, etc.

- " VIVIENDA/21/ROBOT/input/": Alexa envía información a este tópico para dar las órdenes o los comandos al robot en formato JSON. Existen varios subtópicos que derivan de este para desempeñar varias acciones: moverse hacia unas coordenadas en concreto, desplazarse hasta un punto en el mapa ya asignado, hablar, realizar videoconferencias, etc.
- "*VIVIENDA/1/Luz1/set*": a este tópico está conectado una de las luces de prueba, mediante el cual se puede hacer que se encienda o se apague.
- "*VIVIENDA/1/Luz2/set*": igual que el tópico anterior, pero dedicado a controlar otra luz diferente.
- "*peticion_alexa/VIVIENDA/1/THRaul*": tópico donde Alexa publica la intención de actualizar la información de uno de los sensores, para que después este envíe la nueva información actualizada a la API.
- "*ANDIN/*": aquí se publicarán las órdenes para controlar ANDIN, pudiendo ejecutar órdenes como moverse hacia adelante o atrás o girar/pivotar hacia la derecha o a la izquierda. Existirán varios subtópicos para indicar direcciones y distancias.
- "*INDOTEC/*": en este tópico se publicarán los estados de las diferentes funciones del retrete, ya sea cambiar una temperatura, la posición, el usuario, etc.

5.- Evaluación y pruebas

Este proceso es crítico durante cualquier proyecto, puesto que es donde se comprueba el correcto funcionamiento y la validación de los requisitos.

Durante el desarrollo de la skill, a medida que se iban incluyendo nuevos dispositivos, se han ido realizando múltiples pruebas siguiendo una metodología de desarrollo en espiral, refiriéndonos a que, cada vez que en la aplicación se incluía algo nuevo, se volvía a exponer a las pruebas anteriores ya realizadas y las nuevas, verificando el correcto funcionamiento de todas las implementaciones.

Algunas de las pruebas han consistido en:

- Comprobación del modelo de interacción: se testeaba la correcta respuesta ante diferentes peticiones de usuarios. Estas pruebas consistían en comprobar si las frases empleadas para activar cada uno de los *intent* funcionaban de forma correcta, así como las respuestas del asistente una vez realiza la petición o cuando falla. Se han realizado las pruebas cada vez que se creaba una modificación en el modelo de interacción de un nuevo *intent*, para así comprobar que las respuestas funcionan correctamente, no existen frases clonadas entre *intents* o las frases sean demasiado mecánicas o poco naturales.

- Comprobación del envío de mensajes MQTT: para ello, se utilizaba la herramienta MQTT Explorer para poder observar que la skill mandaba mensajes por el *broker*. Para realizar estas pruebas, la skill posee un elemento bandera que, una vez se inicia y se conecta al bróker, nos notifica la conexión del mismo y envía un mensaje MQTT al bróker diciendo que Alexa está conectado. Esta bandera siempre ha estado habilitada, por lo que, siempre que la skill se ha utilizado (ya sea para pruebas o para verificaciones de funcionamiento), se ha podido verificar su conexión al bróker y su capacidad de enviar mensajes MQTT.

- Comprobación de las funciones del robot: como desplazamiento entre habitaciones, conexiones de videollamadas con varios usuarios e interacción por comandos vocales. Para comprobar las funciones, se ha colocado el robot en diferentes zonas del mapa de pruebas para ver si se desplaza entre los diferentes puntos, así como enviar frases concretas o probar a realizar videollamadas con diferentes contactos que tiene asignados para realizar pruebas. Aquí entra un factor importante, que a medida que se estaba creando la skill, se estaba programando la aplicación Android del robot, por lo que algunos de los fallos son debido a incompatibilidades de programación referidos a los tópicos y a los mensajes a enviar.

- Comprobación de las funciones del andador y del inodoro inteligentes: como son aproximación y giros del andador y de las funciones propias del inodoro (como son las funciones de secado, lavado, activación del ciclo de limpieza cambio de usuario o temperaturas de taza y agua). Las pruebas del andador se han realizado en una sala con obstáculos, tales como mesas, sillas, puertas…, para comprobar como es de complicado

manejarlo con la voz para aproximarse a un usuario. Por otro lado, para el inodoro se ha comprobado que es capaz de capturar correctamente los mensajes MQTT para ejecutar el control propio. Los subscriptores de ambos dispositivos están programados e implementados en ESP32s y programados en lenguaje C con el IDE de Arduino.

Como incluir un vídeo para mostrar la secuencia de funcionamiento es complicado, se explicarán por orden los eventos en el orden que suceden:

1) Se abre la *skill*: "Alexa, abre control entorno."
2) Una vez iniciado, se pide la orden o la función deseada (por ejemplo: "Alexa, dile a ANDIN que vaya hacia adelante.").
3) La skill envía un mensaje MQTT al *broker* que será escuchado por el dispositivo y Alexa nos notifica si ha habido éxito al enviarlo.
4) El dispositivo captura el mensaje y actúa en consecuencia.

- Comprobación de la conexión con la API: verificando que la información de los sensores se actualiza correctamente y que la *skill* puede usarla sin problemas. El cliente simplemente renueva la medida del sensor y la publica en la API mediante un código Python.

- Comprobación del control de iluminación: la activación y desactivación de la misma en diferentes salas. Para probar estas funciones, se creó un subscriptor con un ESP32 (la cual dispone de in led interno) y la skill se encargaba de encender y apagar el led. También dispone, por programación del subscriptor, de acciones de cambio de estado sin necesidad de decir el estado que deseamos.

Finalmente y tras múltiples pruebas con diferentes baterías formada por personas a las cuales se les ha explicado el funcionamiento de la skill en función del avance que poseyera, se ha realizado un cálculo estadístico con un resultado del **81,50%** de interacciones ejecutadas de forma correcta. Para entrar más en detalle de este resultado, se dispone de la tabla 3:

Experimento	Peso (%)	Sujetos	Nº de interacciones totales	Positivas	Negativas	Sobre 100%	Sobre su peso
Modelo de interacción	15	3	60	47	13	78,333333	11,75
Envío de MQTT	15	3	60	53	7	88,333333	13,25
Robot	30	8	160	136	24	85	25,50
ANDIN	15	3	45	34	11	75,555556	11,33
INOTEC	15	3	45	38	7	84,444444	12,67
API	5	2	20	12	8	60	3,00
Iluminación	5	4	20	16	4	80	4,00
						TOTAL	**81,50**

Tabla 3. Cálculo de fiabilidad de la skill.

Explicando brevemente la tabla 3, se observa el peso asignado a cada uno de los experimentos. El número de sujetos no sigue ninguna distribución, puesto que dependía de la disponibilidad de ellos, pero el número de interacciones es asignado en función del peso y del número de participantes, obteniendo así el resultado parcial y total de fiabilidad de la *skill*.

Ha sido una tarea un tanto compleja puesto que la interfaz de programación no tenía una terminal donde mostrar errores, entonces si algo fallaba o estaba mal (ya fuera error de sintaxis, una mala sangría, etc.), el simulador no ejecutaba más que lo que se observa en la figura 29.

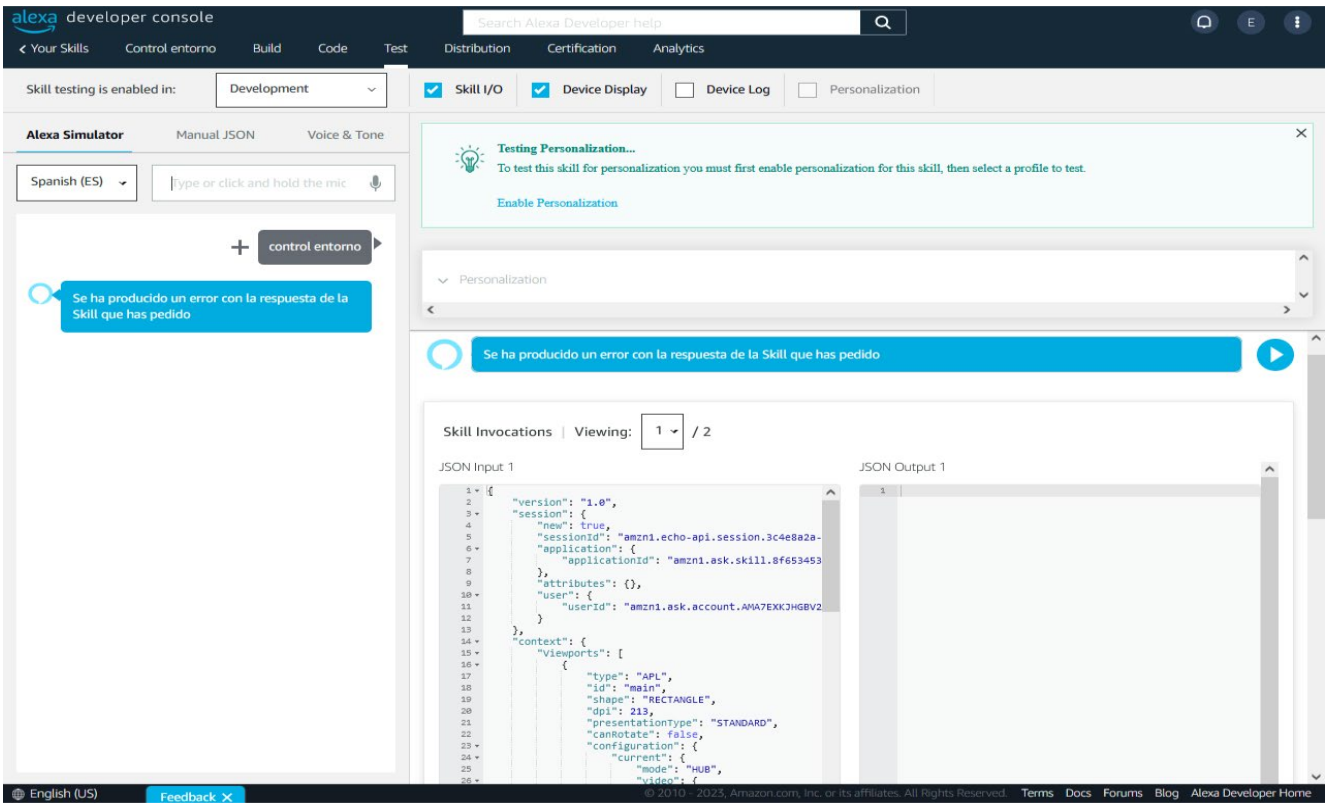

Figura 29. Entorno de pruebas del SDK. [3]

Esto hacia un poco compleja la tarea de detección de errores dentro del código, aunque la solución alcanzada después fue un acierto para solventarlo: pasar el código a *Visual Studio Code*. Aquí solo se producirían como error las funciones referenciadas a las librerías de Amazon, pero todos aquellos correspondientes a la sintaxis o a las funciones de las otras librerías de Python podrían ser corregirdos.

Una vez ya se dispone de la *skill* funcional, toca comprobar si realmente la *skill* se comunica con el *broker*. Para ello, se usará un programa llamado ***MQTT Explorer*** (figura 30), donde se conecta como un cliente más a un *broker* para ver el tráfico de mensajes dentro de todos los tópicos:

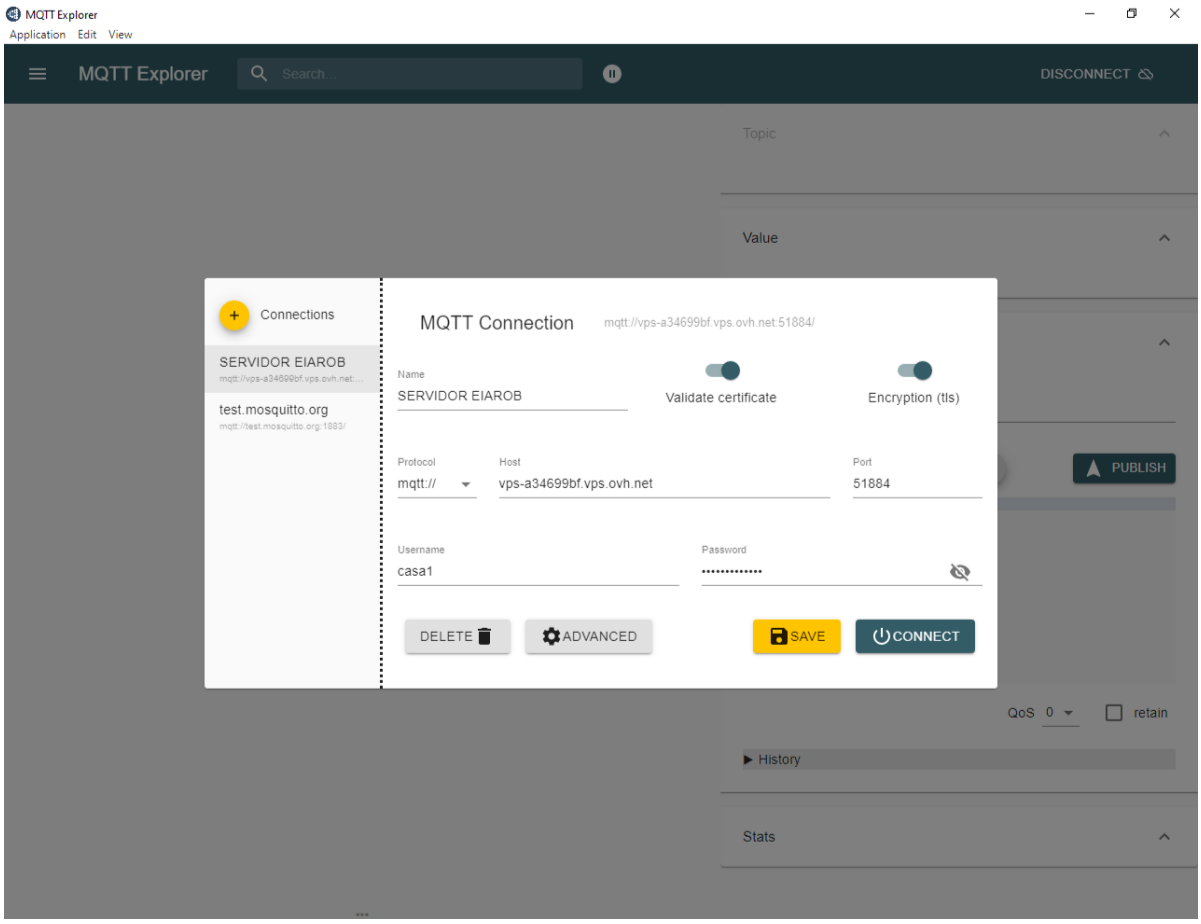

Figura 30. Acceso al *Broker* desde la aplicación MQTT Explorer. [39]

Con esta aplicación también se pueden enviar mensajes a algún tópico concreto para testear el funcionamiento interno del *broker* o para ver si un subscriptor recibe los mansajes.

Ya habiendo logrado las dos partes anteriores, hay que probar los actuadores con sus respectivas programaciones. Se ha hecho de forma directa, conectándonos al *broker* con los mismos ESP-32, enviando un mensaje de que se ha conectado al *broker* y, posteriormente, enviando mensajes por Alexa o por el propio MQTT Explorer.

6.- Estudio económico

6.1.- Introducción

En el presente capítulo se realizará un estudio del coste económico que supone la realización del proyecto desarrollado.

Hasta este momento se ha estudiado la viabilidad de este proyecto desde un punto de vista técnico, es decir, si cumple o no el objetivo marcado desde el comienzo. Habiendo concluido este punto, queda pendiente comprobar la viabilidad económica del mismo.

En los siguientes apartados se estudiarán por separado costes directos e indirectos y finalmente se mostrarán los costes totales del proyecto.

A la hora de comenzar un proyecto, lo primero que se debe realizar es una planificación temporal del mismo, incluso antes de diseñar, programar o idear algo. Para ello, se ha basado en el siguiente índice, aunque se ha seguido un modelo de trabajo cíclico para verificar errores y añadir nuevas especificaciones:

1. Formación y documentación: este apartado es clave a la hora de desarrollar cualquier proyecto y, por ello, uno de los que más tiempo requiere si no se es un experto en la materia del proyecto. Varias cosas que también afectan son el hecho de no existir una documentación clara por parte de Amazon sobre el uso de otras librerías de MQTT que no tengan que ver con sus servicios propios y otra a la continua actualización de las especificaciones, aumentando más el alcance de la *skill*. Además, en este apartado se incluirán todas las horas de investigación para comprender bien el funcionamiento de Alexa y de sus skills.

2. Estudio del problema: en este apartado se ha incluido todo el diseño sobre el funcionamiento de las diferentes funciones de la *skill*, creando diferentes croquis y diagramas de flujos para comprender bien que es lo que se desea hacer en cada caso y con cada función de la *skill*. También se incluye la ideación de los modelos conversacionales, todas las frases que pueda usar el usuario para activar una función concreta y todas las respuestas de Alexa, para lograr así una interacción fluida y natural.

3. Desarrollo de la aplicación: aquí se deben distinguir dos partes, lo referido a la *skill* y el *broker* y todo el hardware creado para controlar los actuadores.

    - Sobre la *skill*: hay que analizar las cosas que se piden de cada funcionalidad, sabiendo bien que se quiere hacer y bajo que comandos.

- Sobre el hardware: se han desarrollado parte de los circuitos y las programaciones de los microcontroladores ESP32 para actuar ante publicaciones en diferentes tópicos.

4. Puesta a punto del sistema (detección de errores): otro apartado clave dentro de los proyectos, comprobar que lo que se ha desarrollado funciona y cumple completamente con lo especificado en el primer punto.

5. Elaboración de la documentación: aquí se archivan las horas de documentaciones y de la creación de esta memoria del proyecto.

	Febrero	Marzo	Abril	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre
Formación y documentación											
Estudio del problema											
Desarrollo de la aplicación											
Puesta a punto del sistema											
Elaboración de documentación											

Tabla 4. Diagrama de Gantt del proyecto.

6.2.- Recursos empleados

A continuación, se muestra un resumen de los recursos hardware y software empleados en el desarrollo de la aplicación. Es importante tener en cuenta únicamente la amortización del material durante el período de tiempo que ha sido utilizado en el proyecto, para calcular el coste real.

Software:

- Sistema operativo: Windows 10.

- Softwares de programación: Amazon Web Service, Arduino IDE y Visual Studio Code.

Hardware:

- Un ordenador portátil: Asus Zenbook 13.
- Un ordenador de sobremesa: HP Compaq 6200 Pro Microtower.
- Un Alexa Echo Dot 3ª generación.
- Andador ANDIN.
- Inodoro INOTEC.
- Circuitos con ESP-32.

Material ofimático:

- Libros de consulta.
- Otros consumibles.

6.3.- Costes directos

En cuanto a los costes directos, se evalúan:

- Coste del personal.
- Cortes amortizables de programas y equipos.
- Coste de materiales directos empleados.

6.3.1.- Costes del personal

La realización del presente proyecto ha sido llevada a cabo por un ingeniero encargado del diseño y puesta a punto del sistema y aplicaciones correspondientes.

Se calcula el coste anual para un ingeniero para posteriormente adecuarlo al número de horas trabajadas por el mismo. Este coste anual incluye:

- Sueldo bruto anual, así como los posibles incentivos por su trabajo.
- Cotización a la Seguridad Social, que es un 35% del sueldo bruto.

Teniendo en cuenta esto, el coste anual del ingeniero será:

COSTE ANUAL	
Sueldo bruto más incentivos	32 744,44 €
Seguridad Social (35% sueldo bruto)	11 453,56 €
Coste total	**44 178,00 €**

Tabla 5. Coste anual del personal.

Se calcula a continuación una estimación de los días efectivos trabajados al año:

DÍAS EFECTIVOS POR AÑO	
Año medio	365,25 días
Sábados y Domingos	-104,36 días
Días de vacaciones efectivos	-20,00 días
Días festivos reconocidos	-15,00 días
Días perdidos estimados	-5,00 días
Total días efectivos estimados	**220,89 días**

Tabla 6. Días efectivos por año.

Conociendo ya el número total de días efectivos de trabajo, y que la jornada laboral es de 8 horas, obtenemos el total de horas efectivas de trabajo:

220,89 días/año x 8 horas/día = 1 767,12 horas/año

El coste por hora de un ingeniero se calcula como la división del sueldo anual entre las horas efectivas trabajadas al año:

Coste hora = 44 178[€/año] / 1 767,12[h/año] = **25 €/hora**

En la siguiente tabla (Tabla 5) se muestra una distribución temporal aproximada del trabajo empleado por el ingeniero para el desarrollo del proyecto:

DISTRIBUCIÓN TEMPORAL DE TRABAJO	
Formación y documentación	175 horas
Estudio del problema	75 horas
Desarrollo de la aplicación	200 horas
Puesta a punto del sistema	150 horas
Elaboración de la documentación	80 horas
Total de horas empleadas	**680 horas**

Tabla 7. Distribución temporal de trabajo.

El coste personal directo es calculado como la multiplicación de las horas y el coste efectivo de una hora de trabajo del ingeniero:

680 horas x 25 €/hora = 17000 €

COSTE PERSONAL DIRECTO	**17 000 €**

6.3.2- Costes de amortización de equipos y programas

Para el cálculo de estos costes se debe realizar previamente la inversión total y calcular la amortización lineal correspondiente según los criterios aconsejados por la ley. En este apartado estudiaremos tanto los costes de la amortización del material de oficina como los costes de amortización de los equipos. Por suerte, los softwares empleados son gratuitos.

Se estima como tiempo de amortización un periodo de **3 años**, ya que es el considerado como vida útil de dicho material. De forma que al calcular el coste hay que multiplicar por un factor del [p.ej. 0.33 para 3 años] los precios mostrados:

MATERIAL	IMPORTE (aprox.)	AMORTIZACIÓN (33,3%)
S.O. Windows 10	129,99 € * 2 Ord. = 159,98 €	53,33 €
Ordenador Asus Zenbook	800 €	266,67 €
Ordenador HP	300 €	100 €
Paquete ofimático Office 365	149 € * 2 = 298 €	99,33 €
Alexa Echo Dot 3	45 €	15 €
Andador	150 €	50 €
Inodoro	200 €	66,67 €
Circuito ESP-32	30 €	10 €
Sistema domótico	1 000 €	333,33 €
TOTAL MATERIAL	**2 982,99 €**	**994,33 €**

Tabla 8. Amortización del material empleado.

Hay que apuntar que los programas informáticos no incluidos anteriormente son de libre distribución y su coste, por tanto, es cero.

El coste final por hora de utilización del material es calculado mediante la división de la amortización anual entre el número de horas de uso en dichos equipos.

Coste final/hora = 994,33[€/año] / 1767,12[horas/año] ≃ **0,56 €/hora**

Se considera el tiempo de uso como el tiempo total necesario para la realización del proyecto, por ser necesario en las etapas de análisis, diseño, programación y documentación. Por tanto, el coste de amortización de material será:

425 horas x 0,56 €/hora = 239,13 €

COSTE DE AMORTIZACIÓN DE MATERIAL DE OFICINA	**239,13 €**

6.3.3.- Costes derivados de otros materiales

Los costes recogidos a continuación se denominan consumibles, e incluyen, por ejemplo, libros de consulta, papel de impresora, fotocopias, cartuchos de tinta, etc.

Este tipo de material es necesario para la realización de los diferentes trabajos, tanto en la fase de desarrollo y edición, impresión de listados, manuales e informes, almacenamiento de programas y documentos, etc.

El coste total de este material es de 180 €.

COSTE DE CONSUMIBLES	**180,00 €**

6.3.4.- Costes directos totales

De todos los costes obtenidos anteriormente concluimos que los costes directos totales son los derivados de la suma de los costes de personal, amortización de material y de consumibles. Por tanto, será:

17000 € + 239,13€ + 180,00 € = 17443,50 €

COSTES DIRECTOS	**17 419,13 €**

6.4.- Costes indirectos

Los costes indirectos son los gastos producidos por la actividad requerida para la elaboración del proyecto y que no se pueden incluir en ninguno de los gastos directos.

COSTES INDIRECTOS PARCIALES	
Dirección y servicios administrativos	150,00 €
Consumo de electricidad	180,00 €
Consumo de telefonía	25,00 €
Consumo de desplazamiento	140,00 €
Total gastos indirectos	**495,00 €**

Tabla 9. Costes indirectos.

Por tanto, los costes indirectos totales ascienden a:

COSTES INDIRECTOS	**495,00 €**

6.5.- Costes totales

Los costes totales son el resultado de sumar los gastos directos e indirectos, siendo el montante total para este proyecto:

COSTES TOTALES	
Costes directos	17 419,13 €
Costes indirectos	495,00 €
Coste total del proyecto	**17 914,13 €**

Tabla 10. Costes totales.

En conclusión, el coste total del proyecto asciende a la cantidad de:

COSTES TOTALES DEL PROYECTO	**17 914,13 €**

Conclusiones y líneas futuras

Conclusiones

Como conclusiones del proyecto, se puede decir que se ha creado una aplicación útil, cómoda y escalable a unas dimensiones concretas, aunque se ha diseñado de forma muy genérica para abarcar muchas funcionalidades, puesto que no deja de ser un primer prototipo.

Hay que recalcar que, inicialmente, la *skill* estaba centrada solo en ser compatible con un servicio *MQTT* y así controlar un robot, pero gracias a la gran utilidad de la aplicación, se ha sobredimensionado el proyecto para incluir todo un ecosistema domótico, con sensores y más dispositivos inteligentes.

La skill es una herramienta que **cumple con las especificaciones**, no solo de control y de comunicación *IoT*, sino que también cumple de manera notable los requerimientos de lenguaje fluido y natural que ha sido comentado en anteriores puntos. La manera más fiable de corroborar esta afirmación es recurrir a los resultados obtenidos en el apartado de 5 de Evaluación y Pruebas, donde se ha calculado un **81,50%** de resultados favorables.

A continuación, se desglosarán las formas en las que se han ido cumpliendo los distintos objetivos:

- Control del robot: la aplicación proporciona funciones al usuario para poder ordenar al robot que se desplace entre habitaciones, transmitir frases y para poder realizar videollamadas con contactos, además de tener funciones para emplear estas opciones de forma simultánea. El subscriptor del robot está basado en Android.
- Control del andador: la skill dispone de funciones para hacer pivotar y desplazar el andador mediante comandos de voz, también dando la opción al usuario de especificar los grados y los metros de manera exacta para lograr una aproximación del dispositivo al destino más cómoda. Este cliente esta creado con Arduino y albergado en un ESP32, dicha programación se ha hecho verificando los certificados de acceso y creando funciones para que, al recibir un mensaje MQTT en su tópico correspondiente, realice las operaciones correspondientes.
- Control del inodoro: se dispone de funciones para el control de un inodoro inteligente, pudiendo controlar la activación de funciones de limpieza como lavado o secado, así como la activación del ciclo completo, modificación de la temperatura de la taza y del agua para la limpieza, selección de usuario y control de elevadores para modificar la altura de la taza, facilitando la acción de sentarse y levantarse del usuario. El cliente está basado, al igual que para el andador, en un código Arduino implantado en un ESP32.
- Control de iluminación: se habilitan funciones para controlar la iluminación de diferentes habitaciones, siendo necesario especificar la habitación y, en caso de haber más de un dispositivo de iluminación en dicha sala, especificar el dispositivo concreto.

Estas funciones no solo son útiles para la iluminación, sino que también podrían ser útiles para actuadores y activación o desactivación de dispositivos.

- Control de sensores: los sensores están programados para ir enviando de forma cíclica información a la API, aunque estas funciones, aparte de permitir al usuario acceder a la información cuando desee, también puede actualizarla en dicho momento para disponer de una información en tiempo real.

Además, la función de emergencia implementada adicionalmente es muy útil y puede proporcionar mucho valor a la *skill*.Como **principal desventaja** de este proyecto es la posibilidad de **desconexión de la red**, puesto que se dejaría de tener conexión a los servidores de Amazon y al *broker*. Esto podría solucionarse con un *broker* local y, si el dispositivo Alexa lo permitiera, disponer del servicio de voz dentro del mismo entorno del *broker*.

Personalmente, este trabajo ha servido para profundizar los **conocimientos de programación** con C y con Python, conocer los papeles a desempeñar en un **equipo de trabajo** y descubrir la versatilidad de uno mismo dentro de este. También ha sido útil para conocer las diferentes etapas de las que consta un proyecto, así como vivir la evolución del mismo, pudiendo ver como algo con una aplicación concreta puede convertirse en la solución de otros proyectos.

Líneas futuras

Algunas cosas que quedarían pendientes para realizar en el futuro serían **buscar nuevas formas de comunicación con los sensores y crear nuevos dispositivos** con los que comunicarnos, con el objetivo de mejorar aún más las prestaciones del sistema y la experiencia de los usuarios. También se mejorará el modelo de interacción de la *skill*, con el objetivo de disponer de más frases con las que poder invocar las diferentes funciones y también para hacer que las frases ya creadas sean más breves y fluidas.

También se investigará **mejorar la comunicación con robots** para aumentar el abanico de posibilidades, como puede ser un sistema basado en *ROS* (*Robot Operative System*) (figura XX) o con cualquier robot comercial que nos permita emplear MQTT para su control, y también añadir más funciones, como pudiese ser pedir información de los mismos, realizar la búsqueda del usuario que habita la casa, etc.

Figura 31. Logotipo de ROS. [45]

Como también es obvio, otro camino óptimo para mejorar nuestra aplicación sería **mejorar los diálogos de comunicación**, haciéndolos aún más naturales para los usuarios, añadiendo nuevas respuestas para que no sean las conversaciones monótonas y/o cíclicas.

Para finalizar y en base al párrafo anterior, no se ha podido experimentar con el público al que estaría dedicada esta aplicación, por lo que en el futuro sería importante **realizar una batería de pruebas con diferentes perfiles** (ya sean personas mayores, con algún tipo de discapacidad, etc.) para conocer los puntos débiles de las interacciones de la aplicación y del modelo de interacción de la misma, con el objetivo de pedir la opinión de aquellos a los que estará dedicada la *skill*.

Bibliografía

[1] MQTT amb entorns gràfics de programació - https://www.scoop.it/topic/ipee/?&tag=Internet+de+las+cosas - Acción Lamas, Angel (18 de noviembre de 2023).

[2] MQTT y JSON en Python - https://www.youtube.com/watch?v=m3DfK3eLmWI - ahijoesu (6 de noviembre de 2023).

[3] Zona de desarrollo de Skills de Alexa - https://developer.amazon.com/alexa/console/ask - Alexa Developer Console (12 de diciembre de 2023).

[4] Alexa Skills Kit SDK for Python - https://alexa-skills-kit-python-sdk.readthedocs.io/en/latest/api/core.html - Alexa Skill Kit SDK for Python (15 de septiembre de 2023).

[5] About Alexa-Hosted Skills - https://developer.amazon.com/es-ES/docs/alexa/hosted-skills/build-a-skill-end-to-end-using-an-alexa-hosted-skill.html - Alexa Skill Kit (20 de agosto de 2023).

[6] AlexaPi - https://github.com/sammachin/AlexaPi-Dev/blob/master/src/main.py - AlexaPi (30 de marzo de 2023).

[7] Protocolo MQTT - https://aprendiendoarduino.wordpress.com/tag/mqtt-qos/ - Arduino (20 de noviembre de 2023).

[8] Alexa Voice Controlled RaspberryPi Dron With IoT AWS - https://www.instructables.com/Alexa-Voice-Controlled-Raspberry-Pi-Drone-With-IoT/ - armaanp69 (10 de abril de 2023).

[9] Internet de las cosas: cuando todo está conectado - https://www.lavanguardia.com/vida/junior-report/20190301/46752655177/internet-cosas-dispositivos-conectados-iot.html - Barchilón, Miriam (13 de octubre de 2023).

[10] JSON en Python. ¿Cómo tratar un json en Python? - https://www.youtube.com/watch?v=J5RGJ-g1yQg - Blijf (3 de noviembre de 2023).

[11] ¿Qué es MQTT? - https://www.youtube.com/watch?v=T1_w8-8Y5kc - ChepeCarlos (21 de septiembre de 2023).

[12] Ejemplo MQTT Python - https://www.youtube.com/watch?v=T362losqJys - ChepeCarlos (21 de septiembre de 2023).

[13] IoT y los asistentes virtuales - https://www.fundacionctic.org/es/actualidad/iot-y-los-asistentes-virtuales - CTICO (4 de noviembre de 2023).

[14] Configuring and Testing Alexa Skill Endpoints - https://www.youtube.com/watch?v=HCqEJ3S2mwY -Culver, Andrew (10 de junio de 2023).

[15] How to call a remote API from an Alexa Skill using Python - https://www.youtube.com/watch?v=ST8uOLbqQUo - Dabble Lab (29 de octubre de 2023).

[16] Cómo comunicar un ESP32 con una página web a través de MQTT - https://www.electrogeekshop.com/como-comunicar-un-esp32-con-una-pagina-web-a-traves-de-mqtt/ - Damián, Juan (30de septiembre de 2023).

[17] How to Enable WebSockets for Mosquitto MQTT Broker - https://cedalo.com/blog/enabling-websockets-over-mqtt-with-mosquitto/ - Domin, Bohdan (30 de septiembre de 2023).

[18] API in Amazon Alexa Skill - https://www.youtube.com/watch?v=M6jK0O1Ahqs - DSwithRS (18 de mayo de 2023).

[19] Alexa and RaspberryPi - https://dzone.com/articles/alexa-and-raspberry-pi-demo-part-2-listening-to-ex - DZone (11 de marzo de 2023).

[20] Alexa Skills - https://www.kinisoftware.com/alexa-skills/ - Engelmo Moriche, Joaquín (2 de septiembre de 2023).

[21] Amazon Alexa Development - https://www.youtube.com/watch?v=QkbXjknPoXc - freeCodeCamp.org (26 de agosto de 2023).

[22] TLS connection with client certificate validation issue between MQTT bróker and ESP32 client - https://github.com/espressif/arduino-esp32/issues/5021 - Gal, Norbert (30 de septiembre de 2023).

[23] Skill de alexa para monitorización de personas mayores - https://ruc.udc.es/dspace/bitstream/handle/2183/31975/GarciaPineiro_Fabian_TFG_2022.pdf?sequence=2&isAllowed=y – García Pineiro, Fabian (29 de noviembre de 2023).

[24] Alexa Reading and speaking incorrect json file - https://amazon.developer.forums.answerhub.com/questions/244662/alexa-reading-and-speaking-incorrect-json-file.html - Henrique dos Santos, Pedro (16 de noviembre de 2023).

[25] Amazon anuncia que […] sus dispositivos Echo ya son compatibles con […] el nuevo estándar de domótica - https://www.20minutos.es/tecnologia/moviles-dispositivos/amazon-

anuncia-que-17-de-sus-dispositivos-echo-ya-son-compatibles-con-matter-el-nuevo-estandar-de-domotica-5086484/ - Higuera, Ana (2 de noviembre).

[26] La comunicación humano-máquina es hoy más natural y efectiva - https://ia-latam.com/2019/07/12/la-comunicacion-humano-maquina-es-hoy-mas-natural-y-efectiva/ - IALatam (14 de noviembre de 2023).

[27] AlexaSmartHome MQTT Bridge - https://github.com/ai91/AlexaSmartHome.MQTT.bridge - Ibragimoff, Anar (6 de octubre de 2023).

[28] Protocolo MQTT: conceptos que debes saber - https://www.youtube.com/watch?v=-fQzbyLqsMc - INNOVA DOMOTICS (4 de octubre de 2023).

[29] Sending a MQTT message in an Alexa Skill - https://stackoverflow.com/questions/76399961/sending-a-mqtt-message-in-an-alexa-skill - jaster (25 de septiembre de 2023).

[30] Build an Alexa controlled robot with AWS RoboMaker - https://aws.amazon.com/es/blogs/robotics/build-alexa-controlled-robot/ - Jiang , Marty (15 de mayo de 2023).

[31] How to make HTTP Request from an Alexa Skill to Get Data from an External API using Python 3 - https://dev.to/ajot/how-to-make-http-requests-from-an-alexa-skill-to-get-data-from-an-external-api-using-python-3-45in - Jotwani, Amit (2 de noviembre de 2023).

[32] Cómo crear tu primera Skill en Amazon Alexa - https://www.youtube.com/watch?v=LAN-BJs6zrw - KeepCoding (7 de septiembre de 2023).

[33] Uso de MQTT para control de dispositivos de IoT - https://riunet.upv.es/bitstream/handle/10251/173841/Lliso%20-%20Uso%20de%20MQTT%20para%20el%20control%20de%20dispositivos%20de%20IoT.pdf?sequence=1 - Lliso Cosin, Alejandro (29 de junio de 2023).

[34] ¿Cómo empezar con Alexa Skills? Charla técnica de AWS - https://www.youtube.com/watch?v=8E0Aq7UpYrQ - Marcía y la nube (21 de septiembre de 2023).

[35] Skill de Alexa para el estudio - https://riull.ull.es/xmlui/bitstream/handle/915/20610/Skill%20de%20Alexa%20para%20el%20estudio.pdf?sequence=1&isAllowed=y – Martín Armas, Héctor (2 de diciembre de 2023).

[36] Using MQTT with Robots (and Home Automation) - https://www.youtube.com/watch?v=4UIUC1YU_Sk - McAleer, Kevin (27 de septiembre de 2023).

[37] ESP32 MQTT Publish Subscribe with Arduino IDE - https://microcontrollerslab.com/esp32-mqtt-publish-subscribe-arduino-ide/ - MicrocontrollersLab (22 de septiembre de 2023).

[38] Casi el 70% de los estadounidenses usa los altavoces inteligentes para escuchar música - https://es.statista.com/grafico/12515/casi-el-70-de-los-estadounidenses-usa-los-altavoces-inteligentes-para-escuchar-musica/ - Moreno, Guadalupe (Statista) (1 de noviembre de 2023).

[39] Explorador de brokers MQTT - https://mqtt-explorer.com/ - MQTT Explorer (7 de diciembre de 2023).

[40] How to make an Alexa Skill with Python - https://www.youtube.com/watch?v=6pPGHR_YwE0 - Mr. Rigden (15 de noviembre de 2023).

[41] ¿Qué es MQTT? - https://www.paessler.com/es/it-explained/mqtt - Paessler (1 de septiembre de 2023).

[42] Control RaspberryPi GPIO With Amazon Echo and Python - https://www.instructables.com/Control-Raspberry-Pi-GPIO-With-Amazon-Echo-and-Pyt/ - PatrickD126 (20 de abril de 2023).

[43] Python json5.dumps() Examples - https://www.programcreek.com/python/example/125378/json5.dumps - ProgramCreek (10 de noviembre de 2023).

[44] Dumping Python object to JSON using json.dumps() - https://reqbin.com/code/python/pbokf3iz/python-json-dumps-example - ReqBin (20 de noviembre de 2023).

[45] Robot Operative System - https://www.ros.org/ - ROS (5 de enero de 2024).

[46] Cloud Concepts And Technology - https://mexicanpentester.com/2020/07/19/aws-certified-cloud-practitioner-2020-resumen-en-espanol/ - Sanchez Marchand, Ricardo (7 de noviembre de 2023).

[47] How to Use the Paho MQTT Client in Python with Examples - https://cedalo.com/blog/configuring-paho-mqtt-python-client-with-examples/ - Schiffler, Andreas (10 de octubre de 2023).

[48] Connecting ESP32 to a MQTT Broker with SSL Certificate - https://forum.arduino.cc/t/need-of-a-guidance-of-securely-connecting-a-esp32-to-a-mqtt-broker-with-ssl-certificate-with-a-static-ip/996519 - Sidsrihari (7 de septiembre de 2023).

[49] Paho Python MQTT Client Subscribe With Examples - http://www.steves-internet-guide.com/subscribing-topics-mqtt-client/ - Steve's Internet Guide (2 de octubre de 2023).

[50] Voice Controlled Smart Home - https://www.instructables.com/Voice-Controlled-Smart-Home/ - taifur (10 de junio de 2023).

[51] How to use ESP32 MQTTS with MQTTS Mosquitto Broker (TLS/SSL) - http://www.iotsharing.com/2017/08/how-to-use-esp32-mqtts-with-mqtts-mosquitto-broker-tls-ssl.html - Tech It Yourself (28 de agosto de 2023).

[52] Python: Publisching messages to MQTT topic - https://techtutorialsx.com/2017/04/14/python-publishing-messages-to-mqtt-topic/ - techtutorialsx (19 de octubre de 2023).

[53] El internet de las cosas: su evolución en los últimos años - https://www.tokioschool.com/noticias/internet-de-las-cosas-evolucion/ - TokioSchool (17 de agosto de 2023).

[54] Diseñando Experiencias basadas en la Voz con Amazon Alexa - https://www.youtube.com/watch?v=uTZgEhy-hdQ - Viscuso, German (20 de octubre de 2023).

[55] Desarrollando Skills Alexa con AWS Lambda y node.js - https://www.todojs.com/desarrollando-skills-alexa-con-aws-lambda-y-node-js-por-german-viscuso/ - Viscuso, German (20 de octubre de 2023).

[56] Perfil de GitHub de Germán Viscuso - https://github.com/germanviscuso?tab=repositories&q=alexa&type=&language=&sort= - Viscuso, German (20 de octubre de 2023).

[57] Alexa MQTT Skill - https://github.com/awilhelmer/alexa-mqtt-skill/blob/master/README.md - Wilhelmer, Alexander (15 de octubre de 2023).

[58] Problems with Paho MQTT client's library implementation of an Alexa Skill to change the value of a topic of Mosquitto Broker - https://stackoverflow.com/questions/76067563/problems-with-paho-mqtt-clients-library-implementation-of-an-alexa-skill-to-cha - Wolowizard (7 de octubre de 2023).

Anejos

1) Código Python en AWS: *lambda_function.py*

a) *LaunchRequest*

```python
# LaunchRequestHandler se dispara solo si se pide activar nuestra skill.
class LaunchRequestHandler(AbstractRequestHandler):
    """Handler for Skill Launch"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool

        return ask_utils.is_request_type("LaunchRequest")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response

        try:
            client = connect_mqtt()
            speak_output = "Bienvenido, con esta skill puedes controlar tu entorno. Prueba a controlar tu robot o diferentes elementos del entorno."
            return (
                handler_input.response_builder
                    .speak(speak_output)
                    .ask(speak_output)
                    .response
            )

        except Exception as e:
            logging.error("Error al conectarse con el broker MQTT: {}".format(str(e)))
            speak_output = "Ha ocurrido un error al conectarse con el broker MQTT."
            return (
                handler_input.response_builder
                    .speak(speak_output)
                    .response
            )
```

b) *ReconnectIntent*

```python
# Aquí tendremos nuestros handlers par las diferentes órdenes/intents uqe controlaran el robot.
# Mandamos señales por MQTT al broker para despues convertirlo al formato deseado que comprenda
# el robot con el que trabajemos (para no tener que estar actualizando la skill todo el rato):

nameSkill = "control entorno"


## Comenzamos a definir todos los intents con los que controlaremos al robot:


# ReconnectIntent para enviar por vía MQTT un topic
# para reiniciar la conexión con el broker MQTT.

# Sample Utterances ({mqtt} = mqtt, broker, broker mqtt, mosquito):
# ->reconecta {mqtt}
# ->reconecta con {mqtt}
# ->reconecta con el {mqtt}
# ->reconecta a {mqtt}
# ->reconecta al {mqtt}
class ReconnectIntentHandler(AbstractRequestHandler):
    """Handler para ReconnectIntent"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool

        return ask_utils.is_intent_name("ReconnectIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response

        # Desconectamos primero y después reconectamos
        client.disconnect()

        try:
            client = connect_mqtt()
            speak_output = "Se ha reconectado al broker MQTT."

        except Exception as e:
            logging.error("Error al reconectarse con el broker MQTT: {}".format(str(e)))
            speak_output = "Ha ocurrido un error al reconectarse con el broker MQTT."

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(nameSkill)
                .response
        )
```

c) StopRobotIntent

```
# StopRobotIntent para enviar por vía MQTT un topic
# para que el robot detenga cualquier movimiento.

# Sample Utterances:
# ->deten al robot
# ->para al robot
# ->dile al robot que se detenga
# ->dile al robot que se pare
# ->haz que el robot se detenga
# ->haz que el robot se pare
class StopRobotIntentHandler(AbstractRequestHandler):
    """Handler para StopRobotIntent"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool

        return ask_utils.is_intent_name("StopRobotIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        try:
            client.publish(topicRobot, "stop", qos = 0, retain = False)
            speak_output = "Se ha publicado un mensaje para parar."

        except Exception as e:
            logging.error("Error al publicar el mensaje en el broker MQTT: {}".format(str(e)))
            speak_output = "Ha ocurrido un error al publicar el mensaje en el broker MQTT para detenerse."

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(nameSkill)
                .response
        )
```

d) *GoToIntent*

```
# GoToIntent para enviar por vía MQTT un
# topic para que el robot vaya a un lugar.

# Sample Utterances ({lugar} = home, lugar/posición 1, ..., lugar/posición 5):
# ->haz que el robot vaya a {lugar}
# ->haz que el robot vaya a la {lugar}
# ->haz que el robot vaya al {lugar}
# ->haz que el robot vuelva a {lugar}
# ->haz que el robot vuelva a la {lugar}
# ->haz que el robot vuelva al {lugar}
# ->dile al robot que vaya a {lugar}
# ->dile al robot que vaya a la {lugar}
# ->dile al robot que vaya al {lugar}
# ->dile al robot que vuelva a {lugar}
# ->dile al robot que vuelva a la {lugar}
# ->dile al robot que vuelva al {lugar}
class GoToIntentHandler(AbstractRequestHandler):
    """Handler para GoToIntent"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("GoToIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response

        intent_lugar = ask_utils.request_util.get_slot_value(handler_input, "lugar")

        # Publicamos un mensaje sobre el destino deseado:
        # (realizado de forma genérica puesto que en dos casas no hay el mismo número de habitaciones)
        try:
            speak_output = "Se ha publicado un mensaje para ir a " + intent_lugar + "."


            if (intent_lugar == "home"):
                obj = {
                        "location": "home base",
                        "speedLevel": None
                    }

                json_msg = json.dumps(obj)

                client.publish(topicRobot_mov, json_msg,   qos = 0, retain = False)



            elif (intent_lugar == "trastero"):
                obj = {
                        "location": "trastero",
                        "speedLevel": None
                    }

                json_msg = json.dumps(obj)

                client.publish(topicRobot_mov, json_msg,   qos = 0, retain = False)

            elif (intent_lugar == "salita"):
                obj = {
                        "location": "salita",
                        "speedLevel": None
                    }

                json_msg = json.dumps(obj)

                client.publish(topicRobot_mov, json_msg,   qos = 0, retain = False)

            elif (intent_lugar == "raúl"):
                obj = {
                        "location": "raúl",
                        "speedLevel": None
                    }

                json_msg = json.dumps(obj)

                client.publish(topicRobot_mov, json_msg,   qos = 0, retain = False)

            elif (intent_lugar == "cocina"):
                obj = {
                        "location": "cocina",
                        "speedLevel": None
                    }

                json_msg = json.dumps(obj)

                client.publish(topicRobot_mov, json_msg,   qos = 0, retain = False)

            elif (intent_lugar == "sofá"):
                obj = {
                        "location": "sofá",
                        "speedLevel": None
                    }

                json_msg = json.dumps(obj)

                client.publish(topicRobot_mov, json_msg,   qos = 0, retain = False)
```

```
        elif (intent_lugar == "pasillo"):
            obj = {
                    "location": "pasillo",
                    "speedLevel": None
                }

            json_msg = json.dumps(obj)

            client.publish(topicRobot_mov, json_msg,   qos = 0, retain = False)

        elif (intent_lugar == "baño"):
            obj = {
                    "location": "baño",
                    "speedLevel": None
                }

            json_msg = json.dumps(obj)

            client.publish("VIVIENDA/21/ROBOT/input/movement/move_dest", json_msg,   qos = 0, retain = False)

        elif (intent_lugar == "aseo"):
            obj = {
                    "location": "aseo",
                    "speedLevel": None
                }

            json_msg = json.dumps(obj)

            client.publish(topicRobot_mov, json_msg,   qos = 0, retain = False)

        else:
            speak_output = "No existe dicho lugar."


    except Exception as e:
        logging.error("Error al publicar el mensaje en el broker MQTT: {}".format(str(e)))
        speak_output = "Ha ocurrido un error al publicar el mensaje en el broker MQTT para ir a " + intent_lugar + "."

    return (
        handler_input.response_builder
            .speak(speak_output)
            .ask(nameSkill)
            .response
    )
```

e) MoveToIntent

```
# MoveToIntent para enviar por vía MQTT un
# topic para que el robot se mueva en una dirección.

# Sample Utterances ({direccion} = adelante, atrás, derecha, izquierda):
# ->dile al robot que vaya hacia {direccion}
# ->dile al robot que vaya hacia la {direccion}
# ->dile al robot que gire a la {direccion}
# ->mueve hacia {direccion} al robot
# ->gira a la {direccion} al robot
class MoveToIntentHandler(AbstractRequestHandler):
    """Handler para MoveToIntent"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("MoveToIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response

        intent_direccion = ask_utils.request_util.get_slot_value(handler_input, "direccion")
        speak_output = "Se ha publicado un mensaje para desplazarse."

        # Conecta al broker MQTT y publica un mensaje
        try:
            if (intent_direccion == "adelante"):
                speak_output = "Se ha publicado un mensaje para desplazarse hacia adelante."
                client.publish(topicRobot, "adelante", qos = 0, retain = False)

            elif (intent_direccion == "atrás"):
                speak_output = "Se ha publicado un mensaje para desplazarse hacia adelante."
                client.publish(topicRobot, "atrás", qos = 0, retain = False)

            elif (intent_direccion == "derecha"):
                speak_output = "Se ha publicado un mensaje para girar hacia la derecha."
                client.publish(topicRobot, "derecha", qos = 0, retain = False)

            elif (intent_direccion == "izquierda"):
                speak_output = "Se ha publicado un mensaje para girar hacia la izquierda."
                client.publish(topicRobot, "izquierda", qos = 0, retain = False)

            else:
                speak_output = "No existe dicha dirección. Pruebe con adelante, atrás, derecha o izquierda."

        except Exception as e:
            logging.error("Error al publicar el mensaje en el broker MQTT: {}".format(str(e)))
            speak_output = "Ha ocurrido un error al publicar el mensaje en el broker MQTT para realizar ese desplazamiento"

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(nameSkill)
                .response
        )
```

f) FuncionOnIntent

```
# FuncionOnIntent para enviar por vía MQTT un
# topic para que el robot active una función.

# Sample Utterances:
# ->{funcion}
# ->haz que el robot active {funcion}
# ->haz que el robot active la {funcion}
# ->haz que el robot active el {funcion}
# ->activa {funcion} del robot
# ->activa la {funcion} del robot
# ->activa el {funcion} del robot
# ->enciende {funcion} del robot
# ->enciende la {funcion} del robot
# ->enciende el {funcion} del robot
class FuncionOnIntentHandler(AbstractRequestHandler):
    """Handler para FuncionOnIntent"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("FuncionOnIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response

        intent_funcion = ask_utils.request_util.get_slot_value(handler_input, "funcion")

        # Conecta al broker MQTT y publica un mensaje
        try:
            speak_output = "Se ha publicado un mensaje para activar " + intent_funcion + "."

            if (intent_funcion == "cámara"):
                client.publish(topicRobot, "cam on",   qos = 0, retain = False)

            elif (intent_funcion == "mapeo"):
                client.publish(topicRobot, "map on",   qos = 0, retain = False)

            elif (intent_funcion == "lidar"):
                client.publish(topicRobot, "lidar on", qos = 0, retain = False)

            elif (intent_funcion == "seguimiento"):
                client.publish(topicRobot, "seg on",   qos = 0, retain = False)

            else:
                speak_output = "No existe dicha función, pruebe con cámara, mapeo o lidar."


        except Exception as e:
            logging.error("Error al publicar el mensaje en el broker MQTT: {}".format(str(e)))
            speak_output = "Ha ocurrido un error al publicar el mensaje en el broker MQTT para activar " + intent_funcion + "."

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(nameSkill)
                .response
        )
```

g) FuncionOffIntent

```
# FuncionOffIntent para enviar por vía MQTT un
# topic para que el robot desactive una función.

# Sample Utterances:
# ->haz que el robot desactive {funcion}
# ->haz que el robot desactive la {funcion}
# ->haz que el robot desactive el {funcion}
# ->desactiva {funcion} del robot
# ->desactiva la {funcion} del robot
# ->desactiva el {funcion} del robot
# ->apaga {funcion} del robot
# ->apaga la {funcion} del robot
# ->apaga el {funcion} del robot
class FuncionOffIntentHandler(AbstractRequestHandler):
    """Handler para FuncionOffIntent"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("FuncionOffIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response

        intent_funcion = ask_utils.request_util.get_slot_value(handler_input, "funcion")

        # Conecta al broker MQTT y publica un mensaje
        try:
            speak_output = "Se ha publicado un mensaje para desactivar " + intent_funcion + "."

            if (intent_funcion == "cámara"):
                client.publish(topicRobot, "cam off",   qos = 0, retain = False)

            elif (intent_funcion == "mapeo"):
                client.publish(topicRobot, "map off",   qos = 0, retain = False)

            elif (intent_funcion == "lidar"):
                client.publish(topicRobot, "lidar off", qos = 0, retain = False)

            elif (intent_funcion == "seguimiento"):
                client.publish(topicRobot, "seg off",   qos = 0, retain = False)

            else:
                speak_output = "No existe dicha función, pruebe con cámara, mapeo o lidar."


        except Exception as e:
            logging.error("Error al publicar el mensaje en el broker MQTT: {}".format(str(e)))
            speak_output = "Ha ocurrido un error al publicar el mensaje en el broker MQTT para desactivar " + intent_funcion + "."

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(nameSkill)
                .response
        )
```

h) TurnOnIntent

```
# TurnOnIntent para enviar por vía MQTT un
# topic para que se encienda/apague la luz.

# Sample Utterances:
# ->enciende {object}
# ->enciende la {object}
# ->enciende el {object}
class TurnOnIntentHandler(AbstractRequestHandler):
    """Handler para TurnOnIntent"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("TurnOnIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response

        intent_power = ask_utils.request_util.get_slot_value(handler_input, "object")
        #intent_room = ask_utils.request_util.get_slot_value(handler_input, "habitacion")

        # Conecta al broker MQTT y publica un mensaje
        try:
            speak_output = "Se ha publicado un mensaje para encender " + intent_power + "."

            if (intent_power == "luz"):
                """
                speak_output = "Se ha publicado un mensaje para encender la luz de " + intent_room + "."

                if (intent_room == "salón"):
                    client.publish(topicLuz1, "ON", qos = 0, retain = False)

                elif (intent_room == "cocina"):
                    client.publish(topicLuz2, "ON", qos = 0, retain = False)

                else:
                    speak_output = "No existe esa luz."
                """
                client.publish(topicLuz, "ON",   qos = 0, retain = False)

            else:
                speak_output = "No existe " + intent_power + "."


        except Exception as e:
            logging.error("Error al publicar el mensaje en el broker MQTT: {}".format(str(e)))
            speak_output = "Ha ocurrido un error al publicar el mensaje en el broker MQTT para encender " + intent_power + "."

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(nameSkill)
                .response
        )
```

i) TurnOffIntent

```
    # TurnOffIntent para enviar por vía MQTT un
    # topic para que se encienda/apague la luz.

# Sample Utterances:
# ->apaga {object}
# ->apaga la {object}
# ->apaga el {object}
class TurnOffIntentHandler(AbstractRequestHandler):
    """Handler para TurnOnIntent"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("TurnOffIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response

        intent_power = ask_utils.request_util.get_slot_value(handler_input, "object")

        # Conecta al broker MQTT y publica un mensaje
        try:
            speak_output = "Se ha publicado un mensaje para apagar " + intent_power + "."

            if (intent_power == "luz"):
                """
                speak_output = "Se ha publicado un mensaje para apagar la luz de " + intent_room + "."

                if (intent_room == "salón"):
                    client.publish(topicLuz1, "OFF", qos = 0, retain = False)

                elif (intent_room == "cocina"):
                    client.publish(topicLuz2, "OFF", qos = 0, retain = False)

                else:
                    speak_output = "No existe esa luz."
                """
                client.publish(topicLuz, "OFF",   qos = 0, retain = False)

            else:
                speak_output = "No existe " + intent_power + "."


        except Exception as e:
            logging.error("Error al publicar el mensaje en el broker MQTT: {}".format(str(e)))
            speak_output = "Ha ocurrido un error al publicar el mensaje en el broker MQTT para apagar " + intent_power + "."

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(nameSkill)
                .response
        )
```

j) ReadSensorIntent

```
# ReadSensorIntent recibe por vía MQTT un
# topic con información de sensores.

# Sample Utterances:
# ->dime el valor de temperatura
# ->temperatura
class ReadSensorIntentHandler(AbstractRequestHandler):
    """Handler para ReadSensorIntent"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("ReadSensorIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response

        try:
            #client.publish(topicPeticion, "temperature")
            subscribe(client)
            medida = value

            #intent_sensor = ask_utils.request_util.get_slot_value(handler_input, "sensor")
            #intent_room = ask_utils.request_util.get_slot_value(handler_input, "habitacion")


            if (intent_sensor == "temperatura" or intent_sensor == "termómetro"):
                speak_output = "El valor de la temperatura es " + medida + " en " + intent_room + "."
                if (intent_room == "salón"):
                # acceso a la información
                    speak_output = "El valor de la temperatura es " + medida + " en " + intent_room + "."
                elif (intent_room == "habitación"):
                # acceso a la información
                    speak_output = "El valor de la temperatura es " + medida + " en " + intent_room + "."
                elif (intent_room == "cocina"):
                # acceso a la información
                    speak_output = "El valor de la temperatura es " + medida + " en " + intent_room + "."
                else:
                    speak_output = "No existe dicho sensor en esa sala."

            elif (intent_sensor == "humedad"):
                speak_output = "El valor de la humedad es " + medida + " en " + intent_room + "."
                if (intent_room == "salón"):
                # acceso a la información
                    speak_output = "El valor de la humedad es " + medida + " en " + intent_room + "."
                elif (intent_room == "habitación"):
                # acceso a la información
                    speak_output = "El valor de la humedad es " + medida + " en " + intent_room + "."
                elif (intent_room == "cocina"):
                # acceso a la información
                    speak_output = "El valor de la humedad es " + medida + " en " + intent_room + "."
                else:
                    speak_output = "No existe dicho sensor en esa sala."
            else:
                speak_output = "No existe dicho sensor."

            #speak_output = "El valor de la temperatura es " + medida + "."


        except Exception as e:
            logging.error("Error al publicar el mensaje en el broker MQTT: {}".format(str(e)))
            speak_output = "Ha ocurrido un error al publicar el mensaje en el broker MQTT para leer."

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(nameSkill)
                .response
        )
```

k) INOTECFuncionIntent

```
# Intenthandlers diseñados para controlar el INOTEC:

# TurnOnIntent para enviar por vía MQTT un
# topic para que se encienda/apague la luz.

# Sample Utterances:
# ->enciende {funcionINOTEC}
# ->activa {funcionINOTEC}
class INOTECFuncionIntentHandler(AbstractRequestHandler):
    """Handler para TurnOnIntent"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("INOTECFuncionIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response

        #intent_function = ask_utils.request_util.get_slot_value(handler_input, "funcionINOTEC")

        # Conecta al broker MQTT y publica un mensaje
        try:
            speak_output = "Se ha publicado un mensaje para activar el secado."
            client.publish(topicINOTEC, "secado", qos = 0, retain = False)

        except Exception as e:
            logging.error("Error al publicar el mensaje en el broker MQTT: {}".format(str(e)))
            speak_output = "Ha ocurrido un error al publicar el mensaje en el broker MQTT para activar el secado."

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(nameSkill)
                .response
        )
```

l) ANDINIntent

```
# ANDINIntent para enviar por vía MQTT un
# topic para que el robot se mueva en una dirección.

# Sample Utterances ({direccion} = adelante, atrás, derecha, izquierda):
# ->dile a andin que vaya hacia {direccion}
# ->dile a andin que gire a la {direccion}
class ANDINIntentHandler(AbstractRequestHandler):
    """Handler para MoveToIntent"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("ANDINIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response

        intent_direccion = ask_utils.request_util.get_slot_value(handler_input, "direccion")
        speak_output = "Se ha publicado un mensaje para desplazar al ANDIN."

        # Conecta al broker MQTT y publica un mensaje
        try:
            if (intent_direccion == "adelante"):
                speak_output = "Se ha publicado un mensaje para desplazar ANDIN hacia adelante."
                client.publish(topicANDIN, "adelante", qos = 0, retain = False)

            elif (intent_direccion == "atrás"):
                speak_output = "Se ha publicado un mensaje para desplazar ANDIN hacia adelante."
                client.publish(topicANDIN, "atrás", qos = 0, retain = False)

            elif (intent_direccion == "derecha"):
                speak_output = "Se ha publicado un mensaje para girar ANDIN hacia la derecha."
                client.publish(topicANDIN, "derecha", qos = 0, retain = False)

            elif (intent_direccion == "izquierda"):
                speak_output = "Se ha publicado un mensaje para girar ANDIN hacia la izquierda."
                client.publish(topicANDIN, "izquierda", qos = 0, retain = False)

            else:
                speak_output = "No existe dicha dirección. Pruebe con adelante, atrás, derecha o izquierda."

        except Exception as e:
            logging.error("Error al publicar el mensaje en el broker MQTT: {}".format(str(e)))
            speak_output = "Ha ocurrido un error al publicar el mensaje en el broker MQTT para realizar ese desplazamiento"

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(nameSkill)
                .response
        )
```

m) EmergenciaIntent

```
# EmergenciaIntentHandler envía por vía MQTT un
# mensaje en caso de que el usuario requiera de
# ayuda extrema.

# Sample Utterances:
# ->ayuda/ emergencia
# ->activar ayuda/emergencia
class EmergenciaIntentHandler(AbstractRequestHandler):
    """Handler para ReadSensorIntent"""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool
        return ask_utils.is_intent_name("EmergenciaIntent")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response

        try:
            client.publish(topicEmer, "SOS")

            speak_output = "Se ha mandado un MQTT para notificar su petición."


        except Exception as e:
            logging.error("Error al publicar el mensaje en el broker MQTT: {}".format(str(e)))
            speak_output = "Ha ocurrido un error al publicar el mensaje en el broker MQTT para pedir ayuda."

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(nameSkill)
                .response
        )
```

2) Código C en Arduino IDE para ESP-32

a) Función **w*ifiInit***: para iniciar la conexión a una red wifi.

```
// Función para conectarse a Wifi
void wifiInit() {
    Serial.print("Conectándose a ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    unsigned long ini_temp= millis();

    while (WiFi.status() != WL_CONNECTED)
    {
        digitalWrite (LED, HIGH);
        delay(5000);
        digitalWrite (LED, LOW);
        delay(5000);
        Serial.print("...");

        if (millis() - ini_temp > 100000)
        {
            ESP.restart();
        }
    }
```

b) Función ***callback***: cuando se esta subscritos a un tópico, nos permite detectar si se ha recibido un mensaje.

```
void callback(char* topic, byte* message, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String messageTemp;

  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
  }
  Serial.println();

  if (String(topic) == topic) {
    Serial.print("Changing output to ");
    if(messageTemp == "on"){
      Serial.println("on");
      digitalWrite(LED, HIGH);
    }
    else if(messageTemp == "off"){
      Serial.println("off");
      digitalWrite(LED, LOW);
    }
  }
}
```

c) Función ***reconnect***: para reconectarnos al broker MQTT si se ha perdido la conexión.

```
// Función conexión a MQTT
void reconnect() {
  while (!mqttClient.connected()) {
    Serial.print("Intentando conectarse MQTT...");

    if (mqttClient.connect("python-mqtt-110")) {
      Serial.println("Conectado");
      mqttClient.publish(topic, "INOTEC conectado");
      //Elisa----------------
      mqttClient.subscribe("INOTEC");
      //--------------------
      //mqttClient.subscribe("Entrada/01");

    } else {
      Serial.print("Fallo, rc = ");
      Serial.print(mqttClient.state());
      Serial.println(". intentar de nuevo en 5 segundos");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
```