PRÓLOGO

¿Qué es un chatbot? ¿Dónde se puede encontrar uno? ¿Cada vez hay más? ¿Son más inteligentes que las personas? Este trabajo es parte de los requisitos de la graduación para el grado en Ingeniería Informática de la Universidad Rey Juan Carlos I. El tiempo empleado comprende desde el mes de octubre del año 2017 hasta el mes de junio del 2018.

El trabajo de fin de grado tiene como título “MetaChatBot: ChatBot para crear ChatBots”. La investigación sobre el funcionamiento de un chatbot se ha realizado, en su mayor parte, en diversas páginas webs y ha sido complicada por temas de poco tiempo al día y nuevas tecnologías, pero se ha podido sobrellevar gracias a la disposición del profesor Francisco Domínguez Mateos para atender las dudas y planteando nuevos retos para la mejora del proyecto, por lo tanto, doy las gracias a mi tutor por la disposición, buena comunicación y todos los aportes que realizó para mejorar la implementación del proyecto.

Con este trabajo se pretende entender un poco más sobre el funcionamiento de la estructura que tiene un chatbot, el cómo funciona, cómo aprende a dar respuestas, cómo se puede construir uno…

La idea de este proyecto es del profesor Francisco Domínguez Mateos, tutor del trabajo que usará este proyecto para abrir camino a la mejora de otros proyectos que tiene en mano.

AGRADECIMIENTOS

Después de unos intensos nueve meses de trabajo por fin ha llegado el día en el que veo los frutos de mi curiosidad por la programación, investigación…

He podido aprender nuevas tecnologías, a dar soluciones a problemas que se planteaban a medida que realizaba el proyecto, a ver que siempre hay una forma de resolverlos, aunque no siempre de la forma que se esperaba ni la más óptima.

Primero, me gustaría a gradecer a mi tutor por los consejos que me ha dado y la paciencia que tuvo durante estos meses ya que no siempre he podido cumplir los hitos que nos planteábamos por temas de mis prácticas externas, por buscar tiempo para poder resolver mis dudas y plantearme nuevos desafíos para que yo pueda conseguir realizarlos y así mejorar el proyecto que teníamos en mano.

Agradecer a mi familia por el apoyo que me dieron siempre, aconsejándome a no desistir cuando me encontraba bloqueado, por las conversaciones ajenas al trabajo que me ayudaron a entender más cosas sobre mi futuro.

Muchas gracias…

RESUMEN

El MetaChatbot se encarga de guardar los datos que el usuario vaya indicando al sistema en una estructura JSON, que es usada como “esqueleto” de un chatbot, donde se guardan los elementos necesarios para su posterior uso. El MetaChatbot es capaz de interaccionar con el usuario mediante teclado/pantalla o voz/altavoces debido a la programación modular que se ha empleado para poder tener flexibilidad.   
A demás, el MetaChatbot es capaz de ejecutar los chatbtos que se hayan creado para su prueba.

Otra parte del sistema es el programa de procesamiento, CProcessor, que se encarga de generar todas las carpetas, ficheros y código para cada chatbot.   
Esto se logra gracias a la estructura JSON que se ha explicado anteriormente.  
Esta parte del sistema da, al chatbot procesado, lo necesario para su ejecución y correcto funcionamiento reduciendo la intervención del usuario a la hora de programarlo para que sólo tenga que editar las acciones de su chatbot en los ficheros que se crean automáticamente al procesarlo.

Por último, el programa implementado de SolveError se usa para mejorar cualquier chatbot generado por/para el sistema. Este programa lee un fichero de errores que se genera a la hora de utilizar el MetaChatbot, dicho fichero se encuentran en todos los chatbots.  
Tras seleccionar el chatbot que se quiere mejorar, el usuario mediante instrucciones puede resolver aquellas peticiones que no se hayan reconocido utilizándolo, reduciendo así un poco más la intervención directa del usuario sobre la estructura del chatbot.

El conjunto de estas partes es el resultado de este proyecto… un sistema capaz de crear la estructura de un chatbot, generar el código, ficheros, carpetas para el chatbot, realizar ejecuciones de prueba y poder editar la estructura introduciendo las sentencias que no se hayan reconocido utilizando un chatbot auxiliar.

ÍNDICE

[CAPÍTULO 1: INTRODUCCIÓN 5](#_Toc524260198)

[1.1 MOTIVACIÓN 7](#_Toc524260199)

[1.2 OBJETIVOS 8](#_Toc524260200)

[1.3 METODOLOGÍA 9](#_Toc524260201)

[1.4 ESTRUCTURA 9](#_Toc524260202)

[1.5 CONVENSIONES 10](#_Toc524260203)

[CAPÍTULO 2: ESTADO DEL ARTE 11](#_Toc524260204)

[2.1 CRÍTICA DEL ESTADO DEL ARTE 15](#_Toc524260205)

[2.2 PROPUESTA 15](#_Toc524260206)

[CAPÍTULO 3: ANÁLISIS DEL PROBLEMA 18](#_Toc524260207)

[3.1 ANÁLISIS DE REQUISITOS 18](#_Toc524260208)

[3.1.1 REQUISITOS FUNCIONALES 18](#_Toc524260209)

[3.1.2 REQUISITOS NO FUNCIONALES 20](#_Toc524260210)

[3.2 ANÁLISIS DE SOLUCIONES 20](#_Toc524260211)

[3.3 SOLUCIONES PROPUESTAS 23](#_Toc524260212)

[CAPÍTULO 4: ANÁLISIS DE LAS HERRAMIENTAS 30](#_Toc524260213)

[4.1 ENTORNO DE PROGRAMACIÓN 30](#_Toc524260214)

[4.2 LENGUAJE DE PROGRAMACIÓN 30](#_Toc524260215)

[4.3 LIBRERÍAS 31](#_Toc524260216)

[4.4 COSTES DEL SISTEMA 32](#_Toc524260217)

[4.5 HERRAMIENTAS DE INTELIGENCIA ARTIFICIAL 32](#_Toc524260218)

[CAPÍTULO 5: DISEÑO DE LA SOLUCIÓN 34](#_Toc524260219)

[5.1 CLASES 35](#_Toc524260220)

[5.2 INTERACCIONES ENTRE CLASES 36](#_Toc524260221)

[5.4 ESTRUCTURA JSON 39](#_Toc524260222)

[5.5 CÓDIGO RELEVANTE 40](#_Toc524260223)

[5.6 FUNCIONAMINETO 42](#_Toc524260224)

[CAPÍTULO 6: RESULTADOS 43](#_Toc524260225)

[CAPÍTULO 7: CONCLUSIONES 47](#_Toc524260226)

[CAPÍTULO 8: TRABAJO FUTURO 48](#_Toc524260227)

[Bibliografía 50](#_Toc524260228)

# CAPÍTULO 1: INTRODUCCIÓN

*“Los bots están empezando a dominar el mundo… Pero si aún no te has enterado solo tienes que hablar con tu móvil (y olvidarte de las app)”* (**HAJ-SALEH** 2017)

Para entender la ida general que este trabajo traba, se empezará definiendo conceptos claves.

¿Qué es un bot?

Según la definición de la web elEconomista.es*[[1]](#footnote-1)*

*“Un bot es un software de inteligencia artificial diseñado para realizar una serie de tareas por su cuenta y sin la ayuda del ser humano como hacer una reserva en un restaurante, marcar una fecha en el calendario o recoger y mostrar información a los usuarios. El modelo más frecuente es el del chatbot, un robot capaz de simular una conversación con una persona y por ello cada vez están más presentes en las aplicaciones de mensajería.”* (**elEconomista** 2016)

Un bot no es más que un programa que realiza las acciones que un usuario le pide sin tener una simulación de comunicación por parte del bot a la hora de interaccionar con el usuario.  
Estas acciones pueden ser programadas para que el bot las realice repetidamente cada cierto tiempo o cuando detecta alguna clave que desencadene sus acciones librando a los usuarios de las tareas que son repetitivas y tediosas.  
Tal y como comenta **Alberto Haj-Saleh** en su artículo “QUÉ SON EXACTAMENTE LOS BOTS, Y CÓMO FUNCIONAN” en la revista GQ, un ejemplo de bot es la herramienta más usada de la historia, GOOGLE; en este caso es usado para la consulta de información.  
  
Desde un principio, cuando se quería buscar información de habitaciones de hotel, se tenía que comparar las páginas webs de muchos hoteles del lugar de destino, ver su disponibilidad, características, ventajas, desventajas, etc. Esto hace unos años ha cambiado gracias a las aplicaciones que tienen las empresas de hoteles para ofrecer sus servicios.   
Aunque con las aplicaciones se redujo el trabajo de la búsqueda de habitaciones teniendo toda la información en el móvil, el trabajo de buscar una habitación para hospedarse hasta unos años seguía siendo repetitivo y tedioso. El usuario se tenía que darse de alta en el aplicativo, rellenar formularios, aceptar condiciones de uso, vincular cuentas de correo, contacto, cuanta bancaria para el pago, etc.

Ahora las cosas han cambiado, gracias a que estas tareas tediosas las pueden hacer los bots. Buscan los hoteles con los requisitos que se le indican, comparando precios para mostrar los más asequibles, incluso se le puede decir todo hablando.

Haciendo uso de los bots, ya programados para determinadas tareas como el ejemplo de una reserva de hotel, a través de servidores de mensajería (FACEBOOK MESSENGER, TELEGRAM, WHATSAPP, etc.) una persona puede activar el bot indicándole lo que desea hacer para que éste haga la tarea, pudiendo pagar autenticándose a través de muchas opciones, una de ellas es usando la huella de un dedo.  
Entonces… ¿en qué se diferencia un chatbot de un bot?

Según distintas fuentes como la web Wikipedia.es[[2]](#footnote-2), un chatbot es:

*“Un*[*bot*](https://es.wikipedia.org/wiki/Bot)*de charla o bot conversacional es un*[*programa*](https://es.wikipedia.org/wiki/Programa_(computaci%C3%B3n))*que simula mantener una*[*conversación*](https://es.wikipedia.org/wiki/Chatear)*con una*[*persona*](https://es.wikipedia.org/wiki/Humano)*al proveer respuestas automáticas a entradas hechas por el usuario”* (**Wikipedia** 2018)

A diferencia de un bot, la idea de un chatbot es poder tener una comunicación casi real con una persona. Esta funcionalidad se logra gracias a distintas formas de aprendizaje por parte de las máquinas.

La inteligencia artificial es una ciencia que estudia la simulación, por parte de una máquina, del comportamiento humano. Unas de las ramas de esta ciencia de Inteligencia Artificial son el Machine Learning y el Deep Learning, ambas permiten a los chatbots poder predecir las peticiones de los usuarios, mejorar si se equivocan, aprender de los propios usuarios y conocer sus preferencias, etc.

Hoy en día se puede interaccionar con los bots, aunque no sea muy evidente. Los bots han estado presentes hace más de cinco décadas y con el paso de los años han ido evolucionando volviéndose más inteligentes pudiendo funcionar como asistentes personales teniendo una interacción muy similar a la de los humanos.

**Ramón Muñoz** del periódico El País, menciona que en el año 2017 se alcanzó los 5000 millones de usuarios de teléfonos móviles lo que supone que más de la mitad de la población mundial puede interaccionar con un bot y/o chatbot desde sus smartphones.  
Esto explica el por qué grandes empresas hace años hayan optado por desarrollar y/o mejorar los bots para, así, poder llegar a un mayor número de usuarios.

Según **Alberto Haj-Saleh**, el auge de los bots está haciendo que los usuarios no descarguen tantas aplicaciones como se ha estado haciendo, optando más por los asistentes personales tales como CORTANA, ALEXA, SIRI, etc.

Ahora bien, ¿cómo funciona un chatbot?

Internamente un chatbot se basa en una estructura con las posibles peticiones del usuario y posibles respuestas dándoles un valor en función de la petición de entrada.  
El chatbot puede estar implementado con Inteligencia Artificial para que pueda realizar un mejor reconocimiento de las peticiones y así dar una respuesta más adecuada, además de que puede aprender según vaya teniendo más interacción con usuarios.

Aunque ya existen distintas herramientas para crear chatbots, éstas sirven para crear chatbots específicos y en muchas de ellas dan como respuesta opciones múltiples perdiendo la finalidad de una chatbot, por lo que este proyecto tiene como uno de sus objetivos conseguir un entorno donde se pueda crear chatbots que simulen una conversación con el usuario mediante texto o voz, una característica que no se encuentran en muchas de estas aplicaciones a la hora de crear chatbots.

Pero… ¿Qué es un Meta Chatbot?

Con todo lo explicado anteriormente se llega a la idea de este trabajo, un Meta Chatbot es un chatbot que genera otros chatbots, es decir, un entorno inteligente que genera otros entornos inteligentes para su posterior edición y mejora por parte del usuario, manteniendo una comunicación con el usuario similar a una persona.

Pudiendo crear chatbots para distintas áreas usando una interacción por teclado o por voz.

**\*\*(Si es inteligente, no sería capaz de aprender por sí sólo)**

Para conseguir desarrollarlo, se ha usado una pequeña red neuronal y con el apoyo de la herramienta Keras se ha podido generar un modelo de la misma.

Este trabajo se apoya en el artículo publicado en chatbotsmagazine.es [[3]](#footnote-3) por el autor **gk\_**[[4]](#footnote-4) en el que menciona:

*“In conversations,****context is king!****We’ll build a chatbot framework using Tensorflow and add some context handling to show how this can be approached.”* (**gk\_** 2017)

Su idea ha sido modificada para este trabajo, usando otros caminos para su realización.

## 1.1 MOTIVACIÓN

¿Cómo funciona un Meta Chatbot? ¿Qué tan complejo es el algoritmo? ¿Cómo puede responder a lo que el usuario quiere? ¿Qué son las neuronas del chatbot? ¿Cómo funciona Deep Learning y Machine Learning?

En la actualidad, aunque la interacción con los chatbots sea continua al usar los smartphones, aún los usuarios no son conscientes del potencial y el papel que pueden llegar a tener en nuestras vidas.

La idea de que un programa pueda reconocer nuestro lenguaje, respondernos a lo que queremos e incluso anticiparse a nuestras necesidades debido al aprendizaje de nuestras acciones, fueron uno de los puntos por lo que este proyecto empezó.

Partiendo del conocimiento de que un chatbot se usa para facilitar la interacción del usuario con otros medios y en la obtención de información, se planteó la pregunta: ¿ya existe “algo” que pueda permitir crear todos los chatbots que uno quiera? Tras una breve investigación para encontrar algo similar, los resultados fueron que hay aplicaciones gratuitas como, CHATFUEL, CLIENGO, LEADAKI, TELEGRAM, FACEBOOK MESSENGER, donde el propio usuario puede programar el chatbot que desee pero con un tema específico, enfocados al marketing, para las plataformas de mensajería en el que se usará el chatbot y con una interacción mediante texto. Para algunas de estas opciones, como TELEGRAM, es necesario tener conocimiento de programación.

También hay webs de pago como clustaar.es, 1MillonBot.es, AdaChatbots.es, oct8ne.es, etc., dedicadas a crear chatbots para un área en concreto como, por ejemplo, banca, administración pública, asistente personal, etc., que son programados según la necesitad del cliente.

Tras esta búsqueda, el objetivo de conseguir un programa que pudiese crear chatbots para cualquier ámbito, no sólo para el área de marketing, con una interacción por voz y que genere sus códigos, es cuanto menos motivador e interesante.

## 1.2 OBJETIVOS

Este proyecto se inició con la idea de conseguir un programa que los usuarios puedan utilizar para crear sus propios chatbots sea cual sea el área que deseen. Estos usuarios tendrán que tener conocimientos, ya sean mínimos o conocimientos amplios, de lo que es un chatbot y sobre la estructura de este programa ya que es necesario poder entender cómo funciona un programa para poder sacar todo el provecho que éste ofrece.

Dentro de los objetivos se encuentran:

1. Poder crear cualquier chatbots.
2. La interacción con el sistema podrá ser de forma escrita o hablada, algo que no todos los programas ofrecen.
3. Todos los datos, ficheros, etc., se tendrán que generar automáticamente para quitar el mayor trabajo posible al usuario.
4. El mismo sistema podrá ejecutar los chatbots creados por el Meta Chatbot en la misma ejecución de éste.
5. El código tendrá que ser modular, es decir, que las acciones que el chatbot tendrá que ejecutar debería estar implementada en clases distintas.
6. Se podrá acceder a un chatbot auxiliar que permita mejorar todos los chatbots, incluyéndose, modificando los ficheros JSON y volviendo a generar el modelo de aquel chatbot que se haya mejorado.
7. Dar un aviso de la acción que se ha realizado.

## 1.3 METODOLOGÍA

Para realizar el código modular se tuvo que detallar qué es lo que tendría que hacer el meta chatbot en cada momento, quién ejecutaría las acciones, quién ejecutaría el meta chatbot, qué entrada/salida debería de tener le chatbot.

Con estas cuestiones planteadas, se empezó a programar un chatbot sencillo, el chatbot de “Lista de la Compra”, que sería un resultado tras usar el meta chatbot.

Una vez implementado este chatbot, se entendió mejor cómo se debería estructurar el código dividiéndolo en distintos programas principales.

Tras conseguir esta modularización del código para que los distintos programas principales trabajasen juntos y dar sentido al meta chatbot, se planteó el objetivo de poder interactuar con el chatbot de forma más dinámica haciendo que el usuario pudiese hablar con él.

Se realizó utilizando un módulo de servicio XML utilizando threading[[5]](#footnote-5) para la recepción de la petición del usuario y un nodo ROS para la emisión de la respuesta del chatbot.

Con estos dos puntos logrados, al chatbot resultante se le puede añadir distintos módulos de interacción con el usuario previamente programados para que la comunicación con el usuario sea más fluida e internacional.

Otra parte importante que se desarrolló fue un chatbot auxiliar que permitiese solucionar aquellas sentencias no reconocidas por cualquier chatbot del sistema.

Explicar lo que es servicio XML, threading,etc???

## 1.4 ESTRUCTURA

En las páginas X se hablará sobre el análisis que se hizo frente a los problemas/dificultades encontradas a lo largo del desarrollo del trabajo planteando así, los requisitos necesarios en la página X para conseguir los objetivos planteados en la página X.

Tras realizar estas tareas, se verá las distintas formas de abarcar estas dificultades para obtener un buen resultado mostrando el diseño de las soluciones en la página X.

\*\* ES NECEARIO PRESENTAR LA ESTRUCTURA BÁSICA DEL DOCUMENTO??

## 1.5 CONVENSIONES

A lo largo de la memoria se mostrará distintas tipologías a la hora de escribir:

* Nombres de páginas web, libros, revistas: Se mostrará subrayado.
* Autores o Empresas: Se mostrarán con las letras en negrita.
* Títulos de proyectos, trabajos, artículos: Se mostrarán en mayúsculas y entre comillas.
* Nombres de aplicaciones: Se mostrarán mayúsculas
* Cita de los autores o texto proveniente de páginas web: Se mostrará entre comillas y en cursiva.
* Código fuente: Se mostrará en cursiva en *Courier New* como tipo de letra y dispuesto en un párrafo.

\*\* ES NECEARIO???

# CAPÍTULO 2: ESTADO DEL ARTE

A menudo, mayoría de usuarios interacciona con chatbots sin darse cuenta.

Desde un chatbot que da información en una web de compra online hasta un chatbot que ayuda al usuario a planificar su día.

Desde páginas web como canalbots.es[[6]](#footnote-6), planetachatbot.es[[7]](#footnote-7), hacen referencia a la historia de la evolución de un bot y/o chatbot, de los cuales se presenta un breve resumen:

* En 1964, **Joseph Weizenbaum** desarrolló el primer chatbot llamado ELIZA en el MIT para IBM7094 que se desarrolló para que el bot actuase como un psiquiatra siendo el primero en ser capaz de interaccionar con el usuario en ingles sobre varios temas.

Basado en etiquetar texto y catalogar las frases, ELIZA llegaba a un punto en el que la coherencia de la conversación se le iba de las “manos”.

* En el artículo “CHATTERBOTS, TYNYMUDs, AND THE TURING TEST ENTERING THE LOEBNER PRIZE COMTETITION” de **Michael L.Mauldin**[[8]](#footnote-8)hace mención a la creación del servidor de juego TINYMUD creado por **Jim Aspnes**[[9]](#footnote-9)en 1989 en el que incluía jugadores controlados por ordenadores a los que se le denominaba bots (palabra corta de Robots). A estos bots se les conocía como Chatterbot que fueron programados basándose en ELIZA para que pudiesen interaccionar con los usuarios para que pudiesen explorar el mundo, jugar juntos, descubrir nuevas habitaciones del propio juego.

En este artículo se explica que el Chatterbot triunfó en el mundo de TINYMUD porque los jugadores creían que todos los usuarios dentro del juego eran personas, no esperaban que haya un ordenador jugando con ellos hasta incluso daban el beneficio de la duda al Chatterbot cuando cometía un grave error que una persona no haría. Esta explicación hace referencia, al conocido **Turing Test** que según la definición de Wikipedia.es

*“es una prueba de la habilidad de una máquina para exhibir un comportamiento inteligente similar al de un ser humano o indistinguible de este”* (**Wikipedia** 2018).

Si desde un principio una persona interacciona con un ordenador sin saberlo y el comportamiento del ordenador se asemeja al de una persona, la persona que está interaccionando con dicho ordenador no podría distinguir si es o no una máquina.

* En 1995 surge otro bot llamado ALICE que fue creado por un grupo de desarrolladores en el Proyecto Pandora del cual se destaca el **Dr. Richard Wallacen** quien fue el creador del dialecto XML que fue la base para extender ELIZA y generar ALICE **(**Artificial Linguistict Internet Computer Entity**).**

Según Wikipedia, el propósito de este proyecto es probar la capacidad de los agentes inteligentes de java. Si se le pregunta sobre el proyecto, ALICE responderá:

“*Soy el último resultado en la inteligencia artificial, que puede reproducir las capacidades del cerebro humano con gran velocidad y precisión*”

A.L.I.C.E

(Wikipedia 2018)

* En 1997 los ordenadores de Microsoft en una de sus herramientas de office tenían incorporado el primer agente de conversación en Windows.

Más conocido como CLIPPY era un bot con la función de ayudar a las personas a utilizar la herramienta Microsoft Office, aunque en versiones posteriores del Office 2003, CLIPPY junto son sus otras versiones como el perro, el mago y el gato desaparecieron.

* En 2011, Apple creó, junto con la empresa Nueance[[10]](#footnote-10), a SIRI que es el primer asistente virtual para un teléfono creado por Apple.

SIRI es capaz de responder a las consultas realizadas por los usuarios y podía mantener conversaciones tipo chip-chat.

Hace huso del motor de Google para responder a las consultas de usuarios que busquen algo en la web y gracias al Machine Learning, el conocimiento de SIRI aumenta según sea la cantidad de personas que la utilicen, llegando a utilizar datos de la pronunciación, significados y localidades del lenguaje para mejorar la experiencia del usuario.

* En 2011, IBMcreó un sistema inteligente llamado WATSON que ganó en el programa Jeoparty enfrentándose a los participantes con mejores resultados.

WATSON es capaz de descomponer las preguntas creando varias hipótesis de respuestas y responder con la probabilidad más alta.

Según Wikipedia, WATSON responde a las preguntas gracias a una base de datos almacenada localmente. La información contenida en esa base de datos proviene de multitud de fuentes, incluyendo enciclopedias, diccionarios, tesauros, artículos de noticias, y obras literarias, al igual que bases de datos externos, taxonomías, y ontologías (específicamente [DBpedia](https://es.wikipedia.org/wiki/DBpedia" \o "DBpedia)[[11]](#footnote-11), [WordNet](https://es.wikipedia.org/wiki/WordNet" \o "WordNet)[[12]](#footnote-12)).

* En 2014, Microsoft introdujo el asistente CORTANA a celulares, tabletas, computadores y consolas como la Xbox a través de Windows 10.

Este asistente está basado en un agente de IA del juego Halo.

Puede aprender y adaptarse gracias a la tecnología de Machine Learning aprovechando el motor de Microsoft Bing Satori que se encarga de indexar millones de datos.

* En 2014, Amazon creó ALEXA que es otro asistente virtual cuya principal característica es que se puede utilizar con el parlante inteligente llamado Amazon Echo y permite conocer información sobre el clima, productos, compras, recordatorios e inclusive video llamadas.

Existe una plataforma para desarrolladores en el que permite crear nuevas funcionalidades y/o acciones para que ALEXA las pueda reconocer y ejecutar.

* En 2016, Google crea GOOGLE ASSISTANT que es un asistente disponible en móviles y el parlante inteligente Google Home, que es capaz de incluir a otros chatbots en sus conversaciones, siendo un asistente proactivo para direcciones e información de negocios.

Entre las funcionalidades de un chatbot se encuentra reconocer la solicitud del usuario a través de un medio, poder analizarla y procesarla con la inteligencia artificial que disponga y por último dar una respuesta a través de un medio.

En el medio de recepción del chatbot puede incluir distintos canales tales como:

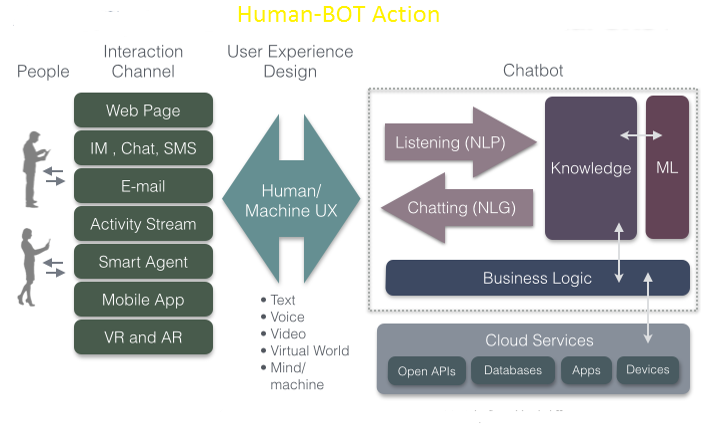
1. Una página web
2. Chat, mensajes
3. Correos
4. Otro asistente y/o agente
5. Aplicaciones

Ilustración : Esquema ChatBot

La experiencia de comunicación puede realizarse por medio de vídeos, texto, voz, mundo virtual, etc.

El chatbot procesa esta información, reconociendo el lenguaje y la vía de la comunicación para poder ejecutar las acciones necesarias, ya sean conectándose a otro medio, webs, API’s, agentes inteligentes, trazando una línea lógica de procesamiento, etc., para poder dar una respuesta concisa a la petición del usuario.

A continuación, se muestra una comparativa entre búsquedas de “bot” y “chatbot” desde 2004 hasta la actualidad. Gracias a Google Trends[[13]](#footnote-13)

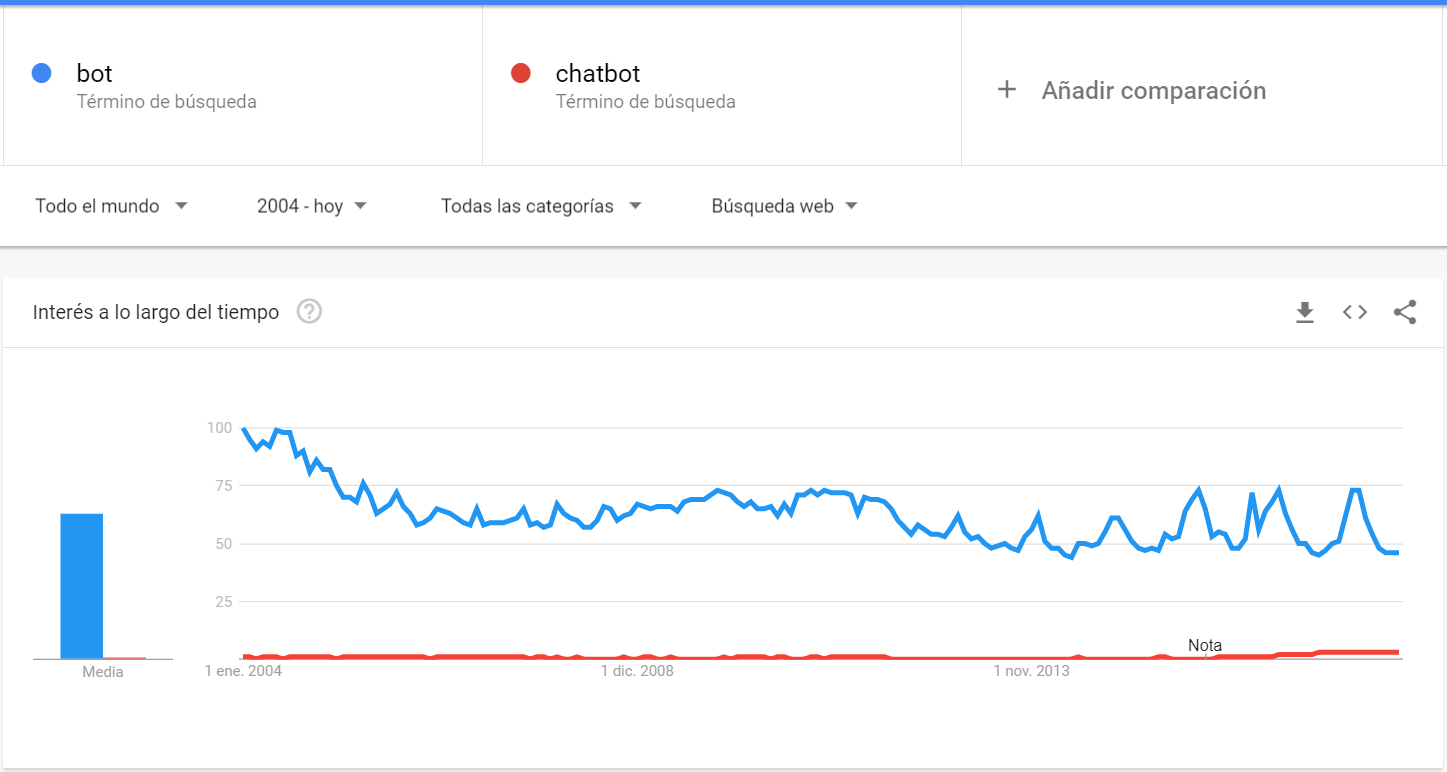
****

Ilustración : Gráfico Búsquedas en todo el Mundo

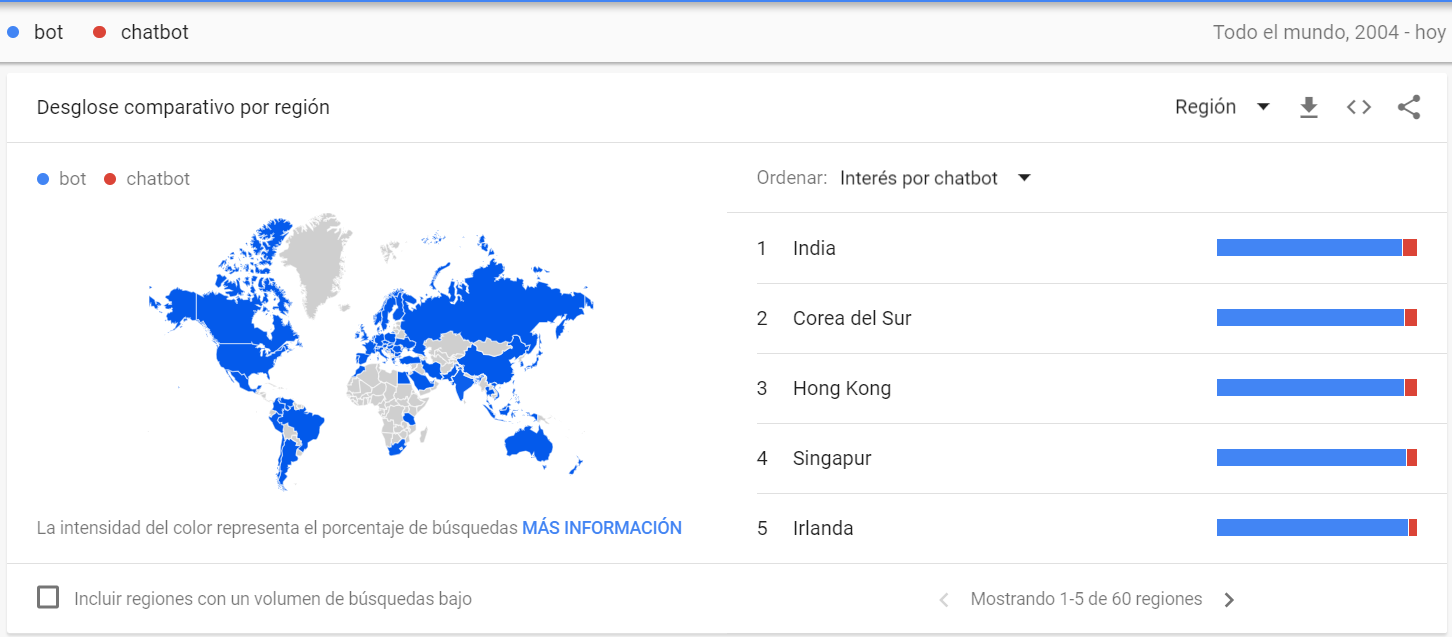
****

Ilustración : Gráfico Búsquedas - Top 5 países

## 2.1 CRÍTICA DEL ESTADO DEL ARTE

Con el paso del tiempo, la tecnología ha crecido considerablemente por lo que la industria comercial ha tenido que apoyarse en los chatbots para gestionar mejor sus negocios, ya sea desde asistentes personales a chatbots de ayuda en páginas web, para dar un mejor servicio a sus clientes.

Muchas empresas que se dedican a crear chatbots, bots, asistentes, etc., para determinadas áreas de interés, ya sea la administración pública, bancos, webs, etc., beneficiándose del poco conocimiento de la materia por parte de los usuarios.

Cabe destacar, que los chatbots para las grandes empresas son complejos, las medias y/o pequeñas empresas pueden crear sus propios chatbots si hubiese una herramienta que le permitiesen hacerlo, y no solo enfocados al ámbito del marketing, sino a muchas otras áreas y sin restricciones en la herramienta.

Este enfoque comercial de los chatbot dificulta el acceso a herramientas para poder generar los chatbots sea cual sea el área de interés, haciéndolos cada vez más restringidos a usuarios, que deben pagar para obtener un chatbot según sus necesidades o buscar herramientas alternativas que les permitan crearlos, aunque el resultado de éstos son chatbots sencillos y, además hay que tener un conocimiento de dichas aplicaciones lo que en algunos casos resulta complicado entenderlos.

En algunas herramientas, es necesario realizar un pago para poder obtener acceso a todo el material y poder realizar chatbots un poco más complejos.

**Amazon**, abriendo la plataforma para desarrolladores hace un punto de inflexión con ALEXA para poder programarla según lo que el usuario quiera, aunque dirigido para sus productos.

Convirtiéndose en punto de referencia para este trabajo, el poder programar chatbots teniendo el conocimiento básico para hacerlo y conociendo la estructura del programa, para realizar cualquier chatbot e incluso mejorarlo si éste no puede reconocer una petición.

## 2.2 PROPUESTA

Uno de los propósitos de este trabajo es conseguir que un usuario con los conocimientos básicos y que sepa utilizar el programa, pueda generar el chatbot que desee con una interacción más amigable, dirigiéndose así a la usabilidad universal con la implementación de una interacción tanto por teclado/pantalla como por habla/escucha.

Como en otras aplicaciones, este proyecto hace uso de ficheros JSON para estructurar las acciones, respuestas, peticiones del usuario; que es fácil de entender por lo que si este programa se sigue desarrollando, los usuarios podrían tener una opción de generar completamente el chatbot mediante la importación de un fichero JSON donde se tendría que indicar unas pautas para saber cómo estructurar dicho chatbot en el JSON, por ejemplo, saber dónde poner el código de la acción y con qué palabra clave, el tipo de medio a utilizar (texto, voz …), etc.

Con la ayuda de los gráficos de un estudio por parte de Statista.es[[14]](#footnote-14), se puede entender un poco en qué sentido se puede enfocar este proyecto para potenciarlo y así conseguir una mejor herramienta.

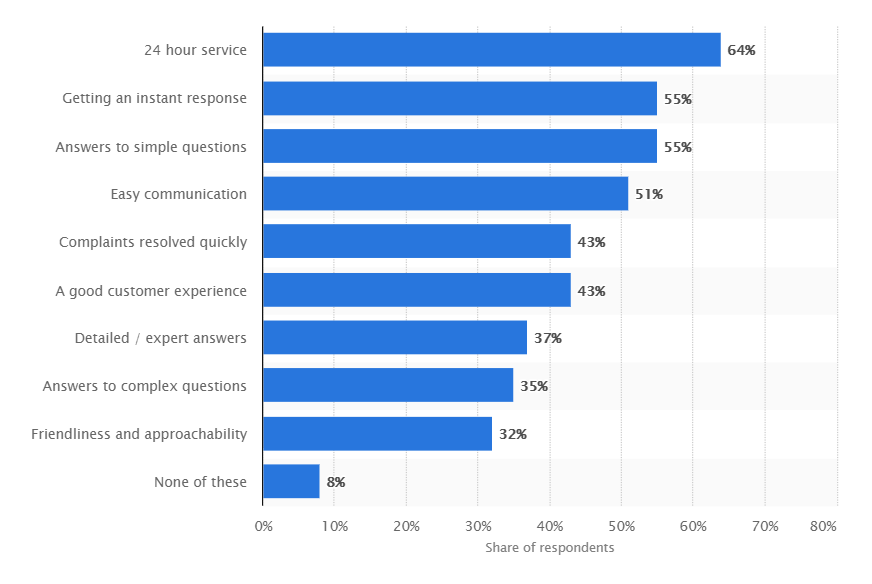
“*Si los chatbots estuvieran disponibles (y funcionaran de manera efectiva) para los servicios en línea que usas, ¿cuáles de estos beneficios esperarías disfrutar?”*

Ilustración : Encuesta a usuarios sobre los beneficios que desean

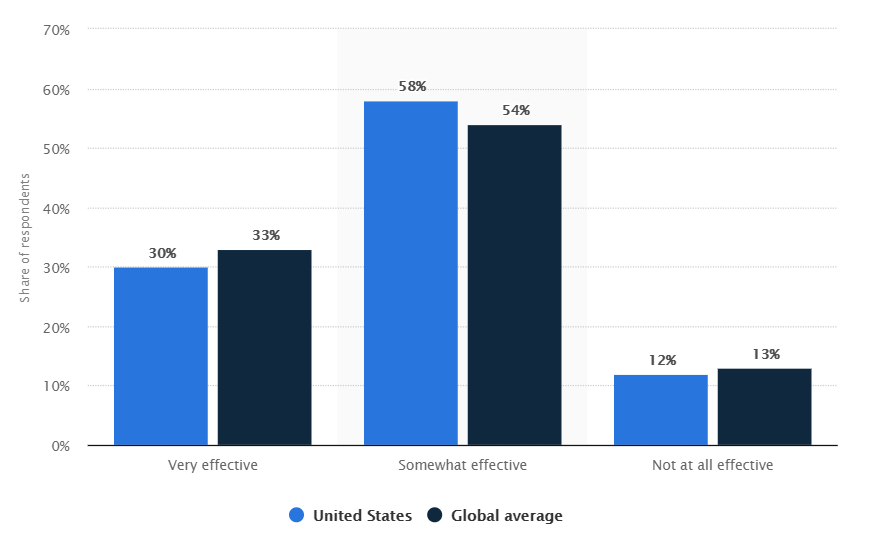
*“Si usaste un chatbot para el servicio al cliente, ¿qué tan efectivo fue el chatbot para resolver tus problemas?”*

Ilustración : Encuesta a usuarios sobre efectividad de un ChatBot

# CAPÍTULO 3: ANÁLISIS DEL PROBLEMA

A la hora de realizar los objetivos/propuestas con el fin de llevar a cabo este proyecto, se presentaron distintos problemas:

* Generar los ficheros automáticamente.
* Generar un chatbot que resuelva las sentencias no reconocidas.
* Actualizar los ficheros JSON.
* Cargar las estructuras de un chatbot desde el JSON a la estructura de la clase chatbot.
* Modularizar las clases.
* Crear los módulos de interacción teclado/pantalla y micrófono/altavoz.
* Lenguaje de programación.
* Librerías para las neuronas.

Se explicarán sus soluciones más adelante.

## 3.1 ANÁLISIS DE REQUISITOS

Para definir los requisitos necesarios para cumplir con los objetivos, se ha planteado responder a las preguntas ¿Cómo debe contestar? ¿De qué forma interaccionará el sistema con el usuario? ¿A qué usuarios está destinado? ¿Debe ser fácil de reprogramar?

### 3.1.1 REQUISITOS FUNCIONALES

* RF01: El sistema deberá dar un mensaje en el estado en el que se encuentra el usuario tras realizar una acción.
* RF02: El sistema deberá generar el modelo automáticamente al iniciar el programa.
* RF03: El sistema tendrá que iniciar, automáticamente, el programa que ejecutará los chatbots.
* RF04: El sistema podrá ejecutar cualquier chatbot.
* RF05: El sistema podrá ejecutar una interacción tanto a través de teclado/pantalla como por medio micrófono/altavoz, independencia en la forma de interactuar.
* RF06: El sistema permitirá crear el fichero Python principal para la ejecución del chatbot.
* RF07: El sistema permitirá crear los ficheros de cada acción del chatbot.
* RF08: El sistema deberá crear los ficheros JSON para cada chatbot.
* RF09: Meta Chatbot: Permitirá crear distintos Chatbots.
* RF10: Meta Chatbot: Permitirá mostrar el Chatbot actual, es decir, el Chatbot que se está editando.
* RF11: Meta Chatbot: Permitirá mostrar todos los Chatbots que se están creando y/o se han creado.
* RF12: Meta Chatbot: Cargará automáticamente todos los chatbots que se hayan creado como estructura para añadirlo a la lista de Chatbots del Meta Chatbot.
* RF13: Meta Chatbot: Permitirá crear Intenciones para cada Chatbot.
* RF14: Meta Chatbot: Mostrará la Intención actual, es decir, la Intención que se está editando.
* RF15: Meta Chatbot: Podrá cambiar la Intención entre las creadas para un Chatbot.
* RF16: Meta Chatbot: Podrá mostrar la Acción creada por cada Intención.
* RF17: Meta Chatbot: Podrá añadir, eliminar, cambiar un Chatbot.
* RF18: Meta Chatbot: Podrá añadir, eliminar, cambiar una Intención.
* RF19: Meta Chatbot: Podrá añadir, eliminar una acción.
* RF20: Meta Chatbot: Podrá generar el Chatbot actual creando sus ficheros Python y JSON.
* RF21: Meta Chatbot: Mostrará mensajes indicativos del estado de la creación del Chatbot.
* RF22: Meta Chatbot: Permitirá elegir el Chatbot que desee ejecutar.
* RF23: Meta Chatbot: Permitirá parar la ejecución del Chatbot que ha ejecutado.
* RF24: Cada Chatbot podrá guardar las peticiones de los usuarios que no se hayan entendido en un fichero JSON.
* RF25: Chatbot SolveError: Permitirá mostrar todos los Chatbots para la elección del Chatbot a resolver sus errores, incluso el Meta Chatbot y él mismo.
* RF26: Chatbot SolveError: Permitirá seleccionar el Chatbot a resolver, la petición que no se ha reconocido y una intención a la que se le quiera asociar para resolverlo.
* RF27: Chatbot SolveError: Al seleccionar el Chatbot a resolver, cargará sus intenciones y los errores automáticamente
* RF28: Chatbot SolveError: Mostrará los errores e intenciones del Chatbot a resolver.
* RF29: Chatbot SolveError: Permitirá guardar las soluciones.
* RF30: Chatbot SolveError: Permitirá mostrar las soluciones guardadas.
* RF31: Chatbot SolveError: Permitirá mostrar el último error e intención seleccionado.
* RF32: Chatbot SolveError: Resolverá los errores que se han resuelto y guardado en la lista.
* RF33: Chatbot SolveError: Editará el JSON del Chatbot que se está resolviendo para introducir los cambios.
* RF34: Chatbot SolveError: Volverá a generará el modelo del Chatbot editado.
* RF35: Chatbot SolveError: Mostrará mensajes indicativos del estado en el que se encuentra el usuario.
* RF36: Cada Chatbot podrá guardar las peticiones de los usuarios, usando una palabra clave “@Guardar”, en un fichero JSON.
* RF37: Todos los Chatbots podrán parar su ejecución.

### 3.1.2 REQUISITOS NO FUNCIONALES

* RNF01: El lenguaje usado por los Chatbots, tanto escrito como hablado, debe ser en castellano.
* RNF02: Las respuestas deben ser amenas y entendibles para cada situación.
* RNF03: Las respuestas de error deberán ser claros.
* RNF04: El tiempo de respuesta no deberá ser muy larga.

## 3.2 ANÁLISIS DE SOLUCIONES

Se han encontrado varios caminos para poder afrontar las dificultades/problemas que plantea este trabajo, entre ellas están:

* Lenguaje de programación:

Entre las numerosas opciones como Java, C#, NodeJS, etc., se optó por usar el lenguaje de Python debido a que es, para muchas webs como maestrosdelweb.es[[15]](#footnote-15), blog.aulaformativa.com[[16]](#footnote-16), muycomputer.es[[17]](#footnote-17), un lenguaje sencillo de aprender y muy flexible.

Cosas que son sencillas en Python se podrían complicar en otros lenguajes, que con unas pocas líneas de código con Python se ha logrado poder conectarse a un servicio XML para poder hacer el reconocimiento de voz, mientras que en otros lenguajes como Java se hubiese complicado debido al tipado estático de éste último.

Por ejemplo, en Python para realizar una simple impresión por pantalla del famoso “Hello, world” sólo hace falta:

*print(‘Hello, world’)*

Mientras que en JAVA hace falta invocar objetos para poder realizarlo:

*System.out.print(“Hello, world”)*

En casos más avanzados como el crear ficheros, la simplificación de Python se hace notar.

* Librerías para las neuronas.

Para la programación de las neuronas que usarán los Chatbots, se tuvieron en cuenta Tensorflow [[18]](#footnote-18)y Keras[[19]](#footnote-19), siendo el primero el que desde un inicio se utilizó por **gk\_**, dueño del artículo en el que se apoya este trabajo.

Pero después de que el proyecto estuviese avanzado, se notó una serie de incompatibilidades con las versiones del Tensorflow y la versión de Python por lo que se optó por utilizar Keras, que a su vez utiliza Tensorflow en segundo plano.

Con la librería de Keras, el entrenamiento del modelo se pudo mejorar gracias a la característica de esta librería para dificultar el entrenamiento de las neuronas y así conseguir abarcar más resultados.

Por otro lado, simplificó el método de guardar el modelo creando sólo un fichero que contiene todo, algo que no pasa con Tensorflow que guarda el modelo, los datos y las rutas de acceso en ficheros distintos.

* Generar los ficheros automáticamente:

Esta dificultad surgió a finales del proyecto, cuando se vio necesario que si el usuario generaba el Chatbot hacía falta generar toda la estructura (ficheros necesarios para el funcionamiento del chatbot creado) y no sólo el JSON.

El uso de Python como lenguaje, facilita la creación de ficheros por lo que fue sencillo implementar un método que pudiese crear los ficheros que llamaría a su vez a otros métodos que generarán la estructura correcta para el fichero en cuestión.

Pero surgió otra dificultad, el poder leer y añadir caracteres especiales al JSON lo que se resolvió con el método *“json.dump()”* de la librería JSON de Python, añadiendo ciertos parámetros para que guardase los caracteres especiales.

* Generar un chatbot que resuelva las sentencias no reconocidas:

A medida que el proyecto iba avanzando, surgió la pregunta ¿qué hará el Chatbot si no reconoce una petición?, en un principio se planteó sólo mostrando un mensaje y que luego el programador editase manualmente del JSON de dicho Chatbot, pero para hacerlo más ameno, se dio con la solución de crear otro Chatbot que se dedique sólo a resolver estos problemas, es decir, que editase los JSON de los Chatbots que tuviesen peticiones no reconocidas.

Este Chatbot editaría el JSON de aquel Chatbot que estuviese resolviendo sus errores y volviendo a general el modelo para que en la próxima ejecución del Chatbot con los errores resueltos tome en cuenta las nuevas peticiones introducidas.

* Actualizar los ficheros JSON:

Para poder actualizar los ficheros JSON, como se comentó anteriormente, Python lo convierte en algo sencillo con sólo pocas líneas y conociendo qué parámetros tocar.

Se realizó utilizando la librería JSON que aporta métodos de carga del fichero para guardarlo en una variable, el cual tendría la estructura de JSON con el que se podría acceder a cada parte de su estructura como si fuésemos a invocar un atributo de un objeto para su edición.

Para volver a cargar el JSON sólo hacía falta utilizar bien los valores de los parámetros del método de JSON para cargar los caracteres especiales.

* Cargar las estructuras de un chatbot desde el JSON a la estructura de la clase chatbot:

Para poder realizar este cambio de JSON al código de Python, sólo se tuvo que cargar la estructura del JSON en una variable para poder manipularla como si fuese un objeto.

Una vez hecho esto, sólo hacía falta encontrar los tokens del objeto JSON y vincularlo a la estructura del Chatbot e Intención.

Se recorrerá cada chatbot que haya sido creado y por cada uno se cargará su estructura JSON para luego generar la estructura que el Meta Chatbot necesita para poder manipularlo.

La dificultad estuvo en modularizar los métodos necesarios para el correcto funcionamiento de esta parte, ya que se quería conseguir un código limpio.

* Modularizar las clases:

Cada vez que se introducía nuevas funcionalidades al programa, el código se hacía más denso y pesado para mantenerlo; se tuvo que plantear bien su modularización desde casi el principio para evitar llegar a un punto en el que sea complicado hacerlo.

Por otro lado, darse cuenta de ello, ayudó a poder implementar rápido la parte de las formas de interacción, que en este caso fue por micrófono/altavoz que, si no se hubiese planteado el código de forma modular, hubiese sido costoso en tiempo introducir esta posibilidad o incluso rehacer el programa.

* Crear los módulos de interacción teclado/pantalla y micrófono/altavoz:

Para esta parte, tal y como se comenta en los anteriores puntos, su implementación no fue tan problemática gracias a la modularización con el que se fue haciendo el programa y a la flexibilidad que aporta Python.

Se abordó el problema con SimpleXMLRPCServer[[20]](#footnote-20) que hace uso del threading para comunicar el host, previamente configurado con el código, con el cliente que para este caso se hará con un móvil desde el cual el usuario podrá ejecutar un script el cual enviará la petición al puerto del thread para ejecutar el código del módulo de entrada del Chatbot que responderá ejecutando el código del módulo de salida para que se produzca una respuesta con audio.

**\*HABLAR CON EL TUTOR PARA ACLARAR LA IDEA**

## 3.3 SOLUCIONES PROPUESTAS

A continuación, se especificará las estructuras generadas más relevantes de las clases para las distintas soluciones.

Cada cual estará representado en un diagrama de clase UML con sus respectivos atributos y métodos.   
Más adelante se detallará los métodos más importantes para cada dificultad planteada, así como las conexiones entre las clases.

* En la siguiente imagen se muestran los métodos y atributos para las estructuras del ChatBot e Intención, que son usadas para crear los objetos donde se guardará los elementos que un chatbot necesita:

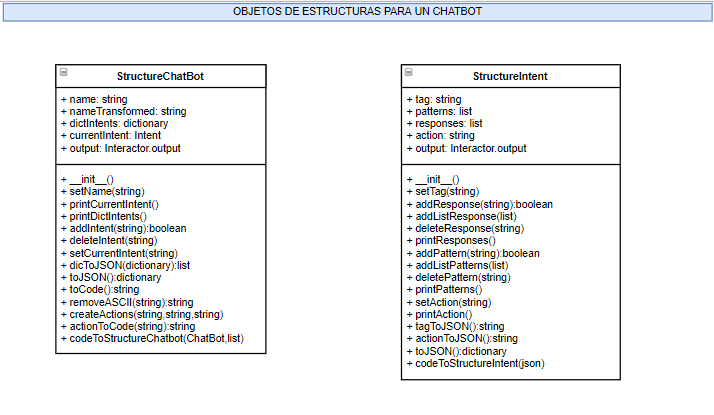


Ilustración : Estructura de un chatbot - código

* + En la estructura del Chatbot se guardarán todas las intenciones que el usuario vaya creando, pudiendo desplazarse por el diccionario de Intenciones que tenga el Chatbot que esté editando, a su vez, esta estructura guardará como atributo el nombre de aquella intención que el usuario esté editando para que el usuario pueda acceder a la Intención y editarla.

Se podrá asociar más de una intención por cada Chatbot.

* + En la estructura de la Intención se guardará las listas de los Patrones (peticiones) con las que el usuario pueda activar las acciones del Chabot y la lista de las Respuestas con las que el ChatBot podrá asociarlos a los Patrones que el usuario haya especificado.

Sólo le podrá asociar de cero a una Acción por cada Intención, de cero a N Patrones e Intenciones.

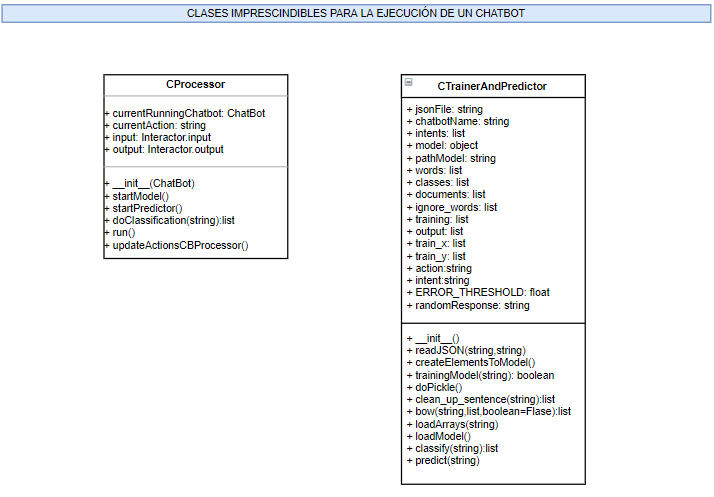
* Los siguientes diagramas hacen referencias a las clases que son las encargadas del funcionamiento de un Chatbot:

Ilustración : Métodos que ejecutan el chatbot

* + La clase CProcessor es la encargada de ejecutar el Chatbot y llamar a los métodos del objeto CTrainerAndPredictor que se encuentra en el Chatbot.
  + La clase CTrainerAndPredictor es quien creará el modelo del Chatbot para poder ejecutar la predicción de la respuesta según el patrón (petición) que el usuario haya indicado.

Para poder hacer más modular las ejecuciones de las acciones de los distintos Chatbots se implementó clases que hacen referencia si la acción conlleva una entrada por parte del usuario, es decir, si hace falta que el usuario introduzca una nueva sentencia a modo de nombre del Chatbot que va a crear o similar, o si la acción es simplemente la ejecución de un evento sin necesidad de una segunda intervención por parte del usuario.

Estas son las clases de las que todas las acciones deberán heredar e implementar los métodos, salvo el método *“checkCancellation(string)”* de la clase ActionLine que ya estará implementado y consiste en verificar si la segunda intervención del usuario no coincide con alguna palabra clave que lleve a la cancelación de la ejecución de la acción.

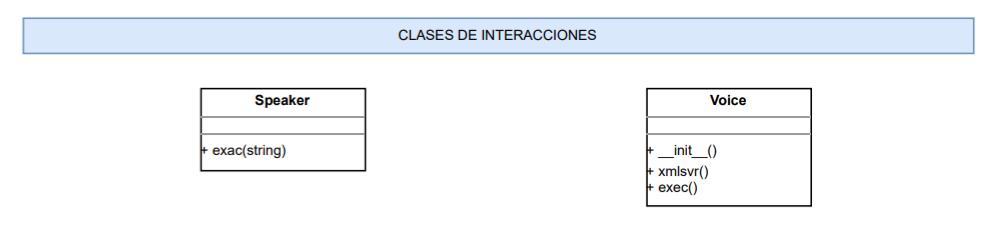
* Para lograr la interacción usando como entrada la voz y como salida un altavoz, se modularizó en dos clases, cada cual tiene implementado los métodos necesarios para su funcionamiento:

Ilustración : Clases de interacción con el usuario

* + En el caso de la clase Voice, usa el servicio XML rcp para poder interaccionar con la voz.
  + En la clase Speaker se implementa un código que llama envía la respuesta a un nodo ROS previamente programado para que pueda darle salida por el altavoz.

* La clase Interactor se encargará de establecer el tipo de interacción que los ChatBots usarán. Los ChatBots instanciarán un objeto de esta clase para poder llamar a sus atributos cuando lo necesiten.

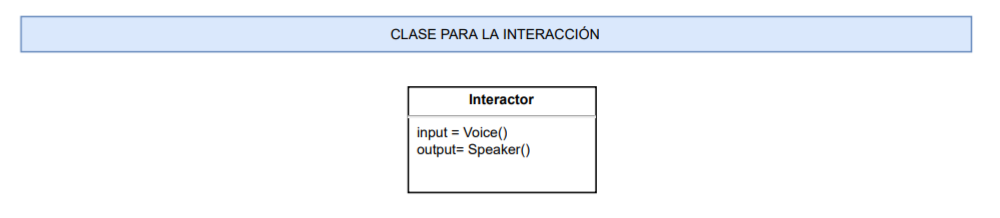


Ilustración : Clase Interactor

* Con la clase AChatBot se pretende tener la estructura básica para las instancias de los ChatBots siendo éste la clase de la cual los ChatBots heredarán.

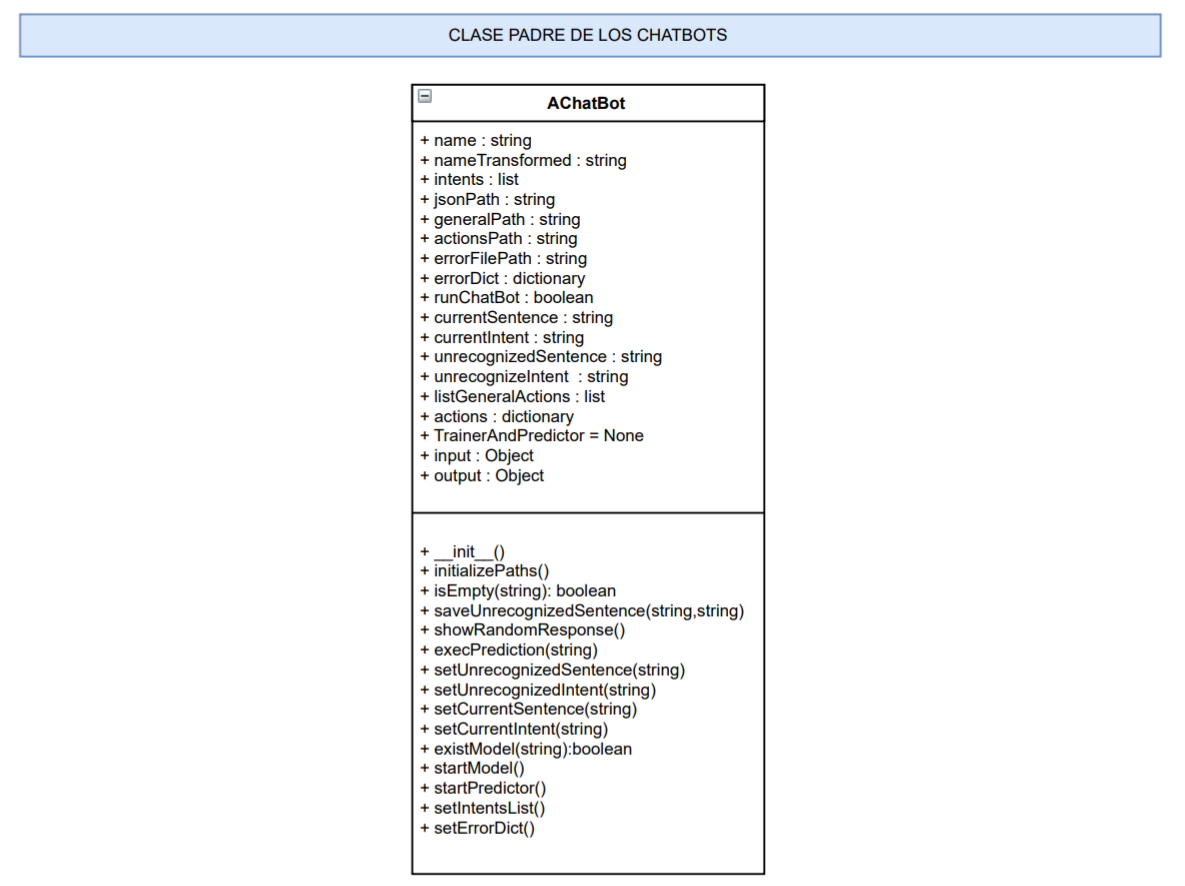


Ilustración : Estructura de ChatBot

Entre sus atributos más importantes están las rutas de los ficheros del JSON, ruta general, del fichero de errores y las acciones, la lista de acciones, errores, objeto TrainerAndPredictor.

Con esta herencia se evita tener que generar atributos, métodos comunes a todos los ChatBots que se creen.

Unos métodos importantes son:

* + initializePaths(): se encarga de obtener las rutas necesarias para guardar y cargar datos referentes a los ChatBots a lo largo de su ejecución. Cabe destacar que este método es implementado en cada ChatBot ya que el nombre del ChatBot, que es importante para conseguir las rutas, se obtiene a partir de la ubicación del fichero que ejecuta el método, así se evita pasar como parámetro el nombre del ChatBot aunque se tiene que implementar el mismo código en todos los ChatBots pero se genera automáticamente.
  + execPrediction(): este método está implementado en esta clase. Ejecuta, por una parte, la clasificación de la palabra que se le pasa como parámetro para obtener los valores de las soluciones y, por otro lado, si cumple con una serie de requisitos ejecuta la acción asociada al patrón (petición) que el usuario introdujo.
* La clase de MetaChatbot está diseñada para que pueda ejecutar otros ChatBots para su prueba sin tener que instanciar el ChatBot a ejecutar y ejecutarlo desde una instancia de la clase CPorcessor. Esta clase tendrá toda su estructura de acciones ya definidas y programadas para cumplir los objetivos del sistema, tales como el volcado automático de los ChatBots creados a la estructura de ChatBots interno del sistema, generar los nuevos ChatBots creando los ficheros necesarios, así como su correspondiente modelo.

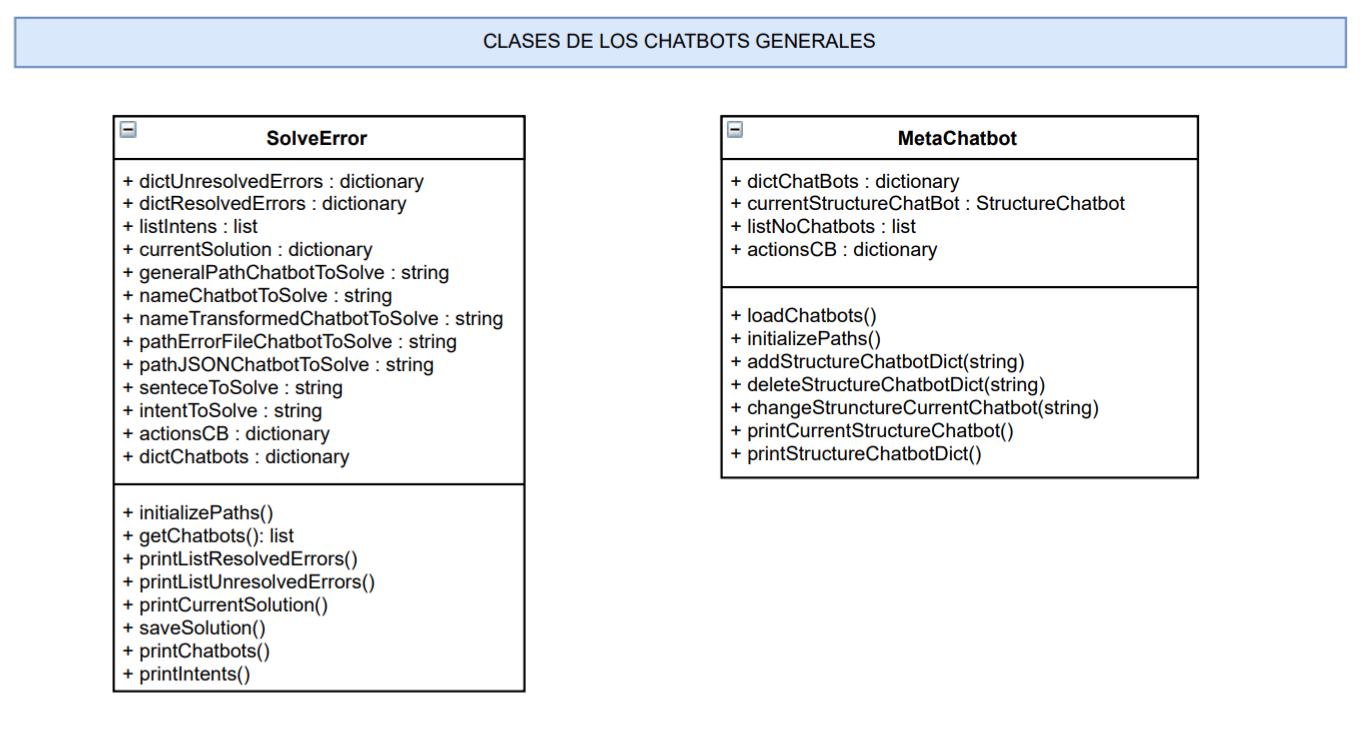


Ilustración : Estructura de Meta Chatbot y Solve Error

Los métodos más relevantes de esta clase son:

* initializePaths(): con este método, que se encuentra en todos los chatbots, establece las rutas de las carpetas, ficheros, etc., para la utilización de las mismas.
* loadChatbots(): este método se encarga de cargar todos los chatbots generados por el sistema y que se encuentren en la carpeta por defecto, para guardarlos en la estructura definida previamente para que el sistema pueda reconocerlos.
* Por otro lado, el ChatBot de SolveError será la encargada de poder mejorar cada ChatBot que tenga su fichero JSON de errores. Se ejecutará, al igual que el MetaChatbot, instanciando la clase CProcessor.  
  Podrá cargar los errores e intenciones del ChatBot que se esté resolviendo en ese momento por elección del usuario. Cuando el usuario decida procesar las soluciones introducidas al sistema, la clase SolveError tiene la capacidad de volver a generar el modelo del ChatBot mejorado para que en su próxima ejecución se pueda tener en cuenta los cambios.

Los métodos más relevantes de esta clase son:

* + initializePaths(): se comporta igual que en la clase de MetaChatBot, estableciendo los nombres de las rutas de los ficheros y carpetas para el chatbot de SolveError.
  + getChatbots(): obtiene todos los ChatBots que hayan sido creados para cargarlos en una diccionario donde se tendrá el nombre de la carpeta, que coincide con el nombre del ChatBot sin caracteres especiales (se explicará más detallado en el siguiente capítulo), y el nombre que aparece en la estructura JSON de cada ChatBot.
  + saveSolutions(): guarda todas las soluciones, que el usuario haya configurado, en un diccionario y en unas variables para saber cuál ha sido la última solución guardada.

# CAPÍTULO 4: ANÁLISIS DE LAS HERRAMIENTAS

## 4.1 ENTORNO DE PROGRAMACIÓN

El entorno seleccionado es Pycharm que presenta una interfaz sencilla.  
La versión con la que se implementó todo el sistema es “**3.6.3 |Anaconda custom (64-bit)**”

Las recomendaciones mínimas según la web de Jetbrains[[21]](#footnote-21) son:

* Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)
* 2 GB RAM mínimo / 4 GB RAM recomendado
* Python 2.4 o más actuales.
* 1024x768 resolución de pantalla

¿Qué es ANACONDA?

Según la definición de la web de Anaconda es :

Un administrador de paquetes, entornos, una distribución de Python y una colección de paquetes de código abierto. Es gratis y fácil de instalar, y ofrece soporte comunitario gratuito.

Puede instalar, eliminar o actualizar cualquier paquete Anaconda con unos pocos clics en el Navigator, o con un solo comando **conda** en Anaconda Prompt (Terminal en Linux o macOS).

Haciéndolo una herramienta versátil para poder configurar el entorno con las necesidades que el programador necesita. Se optó por esta opción por tener un respaldo comunitario a sus espaldas lo que sirve de apoyo al programador a la hora de encontrarse con dificultades, además de que facilita las instalaciones de paquetes.

## 4.2 LENGUAJE DE PROGRAMACIÓN

Las ventajas de Python frente a otros lenguajes son diversos, como la simplicidad que muestra este lenguaje para realizar tareas que en otros lenguajes resultan complicadas como se ha mencionado en otros capítulos.

Programadores como [**Luis Vallespín**](https://es.quora.com/profile/Luis-Vallespín)[[22]](#footnote-22), **Gissela Peralta**[[23]](#footnote-23) o incluso entidades como BBVA en BBVAOpen4U[[24]](#footnote-24) tienen opiniones común hacia Python sobre la facilidad de aprendizaje, un repositorio en la web, numerosos paquetes para el tratado de datos, etc.

## 4.3 LIBRERÍAS

Las librerías son códigos ya programados por otros usuarios y/o empresas que los ponen a la disposición de los demás para su uso común.

Entre las librerías más usadas e importantes para la resolución del sistema se encuentran:

* **JSON**: JSON (JavaScript Object Notation), especificado por RFC 7159 y por ECMA-404, es un formato ligero de intercambio de datos inspirado en la sintaxis literal del objeto JavaScript.  
  Con esta librería se han volcado y cargado los datos de los ficheros JSON teniendo en cuanta los caracteres especiales que tienen dichos ficheros en su estrcutura.
* **Os**: Este módulo proporciona una forma portátil de utilizar la funcionalidad dependiente del sistema operativo.   
  Con esta librería se puede leer o escribir un archivo, manipular rutas, leer todas las líneas en todos los archivos en la línea de comandos, crear archivos y directorios temporales, manejar archivos y directorios de alto nivel.  
    
  Lo que facilita las tareas de edición, creación, de ficheros así como el encontrar sus rutas.
* **Xmlrpc**: XML-RPC es un método de llamada a procedimientos remotos que utiliza XML a través de HTTP como transporte. Con él, un cliente puede invocar métodos con parámetros en un servidor remoto (el servidor recibe el nombre de un URI) y recuperar datos estructurados.  
  Esta librería es con la que se ha implementado la solución de poder ejecutar el código remotamente de un servidor para poder interaccionar con el Chatbot desde un micrófono que puede encontrarse en un móvil.
* **Threading**: Este módulo construye interfaces de subprocesamiento en la parte superior del módulo.  
  Se usa en conjunto con la librería XMLRPC para poder lograr la comunicación con el servidor.
* **Numpy**: NumPy es una extensión de [Python](https://es.wikipedia.org/wiki/Python), que le agrega mayor soporte para [vectores](https://es.wikipedia.org/wiki/Vector_(inform%C3%A1tica)) y [matrices](https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1tica)), constituyendo una [biblioteca](https://es.wikipedia.org/wiki/Biblioteca_(inform%C3%A1tica)) de funciones matemáticas de [alto nivel](https://es.wikipedia.org/wiki/Lenguaje_de_alto_nivel) para operar con esos vectores o matrices.   
  Es usado para el tratado de las matrices de resultados dadas como resultado del aprendizaje del modelo o para poder entrenarlo con las matrices de generadas al reconocer las palabras que hay en los ficheros JSON.
* **Keras**: Keras es una API de redes neuronales de alto nivel, escrita en Python y capaz de ejecutarse sobre TensorFlow, CNTK o Theano. Fue desarrollado con un enfoque en permitir la experimentación rápida.  
  Con esta librería se pudo resolver el problema de versiones que daba si se usa TensorFlow directamente teniendo un resultado similar.
* **Nltk**: Es un procesamiento del lenguaje natural con Python. Trabaja con corpora, categoriza texto, analiza estructura lingüística,etc.  
  Es usado para poder reconocer las raíces de las palabras que tienen los ficheros JSON de cada Chatbot para que el modelo pueda reconocer diferentes formas de expresiones.

## 4.4 COSTES DEL SISTEMA

**COSTES DEL SISTEMA EN SENTIDO HARDWARE O ENSENTIDO DE GASTOS PARA ADQUIRIR EL ORDENADOR, INTERNET, ETC?**

## 4.5 HERRAMIENTAS DE INTELIGENCIA ARTIFICIAL

Dentro de esta sección se tratarán de una forma más extensa de los medios que la ciencia de la Inteligencia Artificial utiliza para poder simular el comportamiento cognitivo de las personas.

Dentro de esta rama distinguiremos dos formas (información obtenida de Wikipedia):

* Deep Learning: Por definición general, es un conjunto de [algoritmos](https://es.wikipedia.org/wiki/Algoritmo) que intenta modelar abstracciones de alto nivel en datos usando arquitecturas compuestas de transformaciones no lineales múltiples.

El aprendizaje profundo es parte de un conjunto más amplio de métodos de aprendizaje automático basados en asimilar representaciones de datos.

Varias arquitecturas de aprendizaje profundo, como redes neuronales profundas, redes neuronales profundas convolucionales, y redes de creencia profundas, han sido aplicadas a campos como [visión por computador](https://es.wikipedia.org/wiki/Visi%C3%B3n_por_computador), [reconocimiento automático del habla](https://es.wikipedia.org/wiki/Reconocimiento_del_habla), y reconocimiento de señales de audio y música, y han mostrado producir resultados de vanguardia en varias tareas.

* Machine Learning: es el subcampo de las [ciencias de la computación](https://es.wikipedia.org/wiki/Ciencias_de_la_computaci%C3%B3n) y una rama de la [inteligencia artificial](https://es.wikipedia.org/wiki/Inteligencia_artificial), cuyo objetivo es desarrollar técnicas que permitan que los ordenadores aprendan. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos.

El aprendizaje automático tiene una amplia gama de aplicaciones, incluyendo [motores de búsqueda](https://es.wikipedia.org/wiki/Motores_de_b%C3%BAsqueda), diagnósticos médicos, detección de fraude en el uso de [tarjetas de crédito](https://es.wikipedia.org/wiki/Tarjetas_de_cr%C3%A9dito), análisis del [mercado de valores](https://es.wikipedia.org/wiki/Mercado_de_valores), clasificación de [secuencias de ADN](https://es.wikipedia.org/wiki/Secuencias_de_ADN), reconocimiento del habla y del [lenguaje escrito](https://es.wikipedia.org/wiki/Lenguaje_escrito), juegos y [robótica](https://es.wikipedia.org/wiki/Rob%C3%B3tica).

# CAPÍTULO 5: DISEÑO DE LA SOLUCIÓN

Para poder realizar todas las soluciones anteriormente presentadas, se ha necesitado tener una serie de configuración entre las versiones de la herramienta, librerías, lenguaje, etc., para que no haya conflictos a la hora de la ejecución y/o pruebas del programa.

A continuación, se presenta la arquitectura completa del programa:

* Clases
* Interacciones entre las clases
* Estructura de ficheros JSON
* Códigos relevantes
* Explicación del funcionamiento en ejemplos

En esta sección se mostrará todo lo referente a la estructura generada tras la realización del sistema, así como su relación.

La estructura de carpetas donde se encontrarán todas las diferentes clases será:

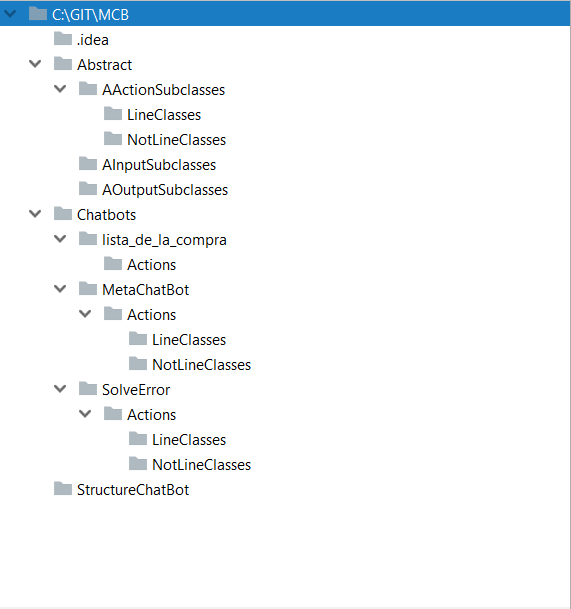


Ilustración : Arquitectura de carpetas del sistema

## 5.1 CLASES

Se presentará las clases esenciales y algunas clases de las acciones de los ChatBots:

* Acciones generales:
  + DontSaveSentence: muestra un mensaje para dar a conocer que no se ha guardado el patrón (petición).
  + FinishRunningCB: termina la ejecución de un ChatBot.
  + NotRecognizedSentence: muestra un mensaje para dar a conocer que no se ha podido reconocer el patrón (petición).
  + SaveSentence: guarda el patrón (petición) y si intención con el que se ha vinculado.
* Clases Padres
  + ActionLine: espera una intervención del usuario para introducir un valor para realizar la acción con ese valor. También actúa como clase hija heredando de **AAction**.
  + ActionNotLine: no espera una intervención del usuario y ejecuta directamente la acción. También actúa como clase hija heredando de **AAction**.
  + AAction: clase padre de los tipos de acciones.
  + AChatBot: implementa algunos métodos esenciales para el funcionamiento de los ChatBots. Es la clase padre de todos los ChatBots.
  + AInput: clase padre de los tipos de interacciones de entrada.
  + AOutput: clase padre de los tipos de interacciones de salida.
* Clases Hijas
  + Keyboard
  + Voice
  + Screen
  + Speaker
* Clases Generales
  + AInteractor: Es quien instancia las clases de interacción.
  + MetaChatBot: Es quien unifica distintas clases para el funcionamiento del sistema. También es una clase hija.  
    Las acciones más importantes, aparte de las acciones generales, son:

1. CreateChatbot: crea un objeto StructureChatbot para poder editar la estructura del ChatBot que se quiere crear.
2. StartRunningChatbot: se encarga de ejecutar un ChatBot sin parar la ejecución del Meta ChatBot.
3. BuildChatbot: se encarga de crear los ficheros necesarios para la ejecución de un ChatBots (json, acciones, modelo, etc.)

* SolveError: Es la clase que permite resolver los fallos de precisión a la hora de predecir una respuesta. Las acciones más importantes, aparte de las acciones generales, son:

1. SelectChatbot: se encarga de poder seleccionar el ChatBot a resolver y cargar sus intenciones y errores.
2. SaveSolution: permite guardar las soluciones en un diccionario.
3. ProcessSolutions: procesa las soluciones guardadas modificando el fichero JSON del ChatBot que se está resolviendo y generando de nuevo su modelo.

* CTrainerPredictor: es la clase que tiene el código para poder generar el modelo partiendo de un JSON y poder predecir una respuesta según una entrada.
* CBProcessor: esta clase se encarga de ordenar la ejecución de una ChatBot. Es decir, para la ejecución de un ChatBot se usa esta clase pasándole el ChatBot a ejecutar como parámetro.

## 5.2 INTERACCIONES ENTRE CLASES

En las siguientes imágenes se muestra las relaciones de dependencia, herencia, etc., que tienen las clases.

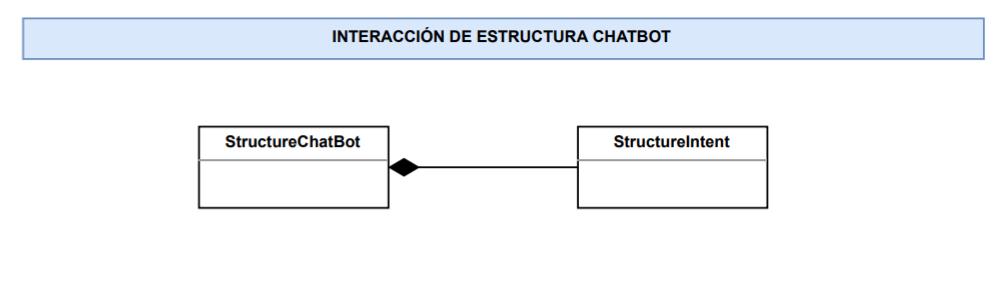
* La estructura de las intenciones depende de la estructura del chatbot, ya que sin ésta no se podría crear una intención.

Ilustración : Diagrama de clases - StructureCahtBot y StructureIntent

* Los módulos de interacción heredan de otras clases para poder realizar la encapsulación y permitir el uso de distintos módulos de interacción.

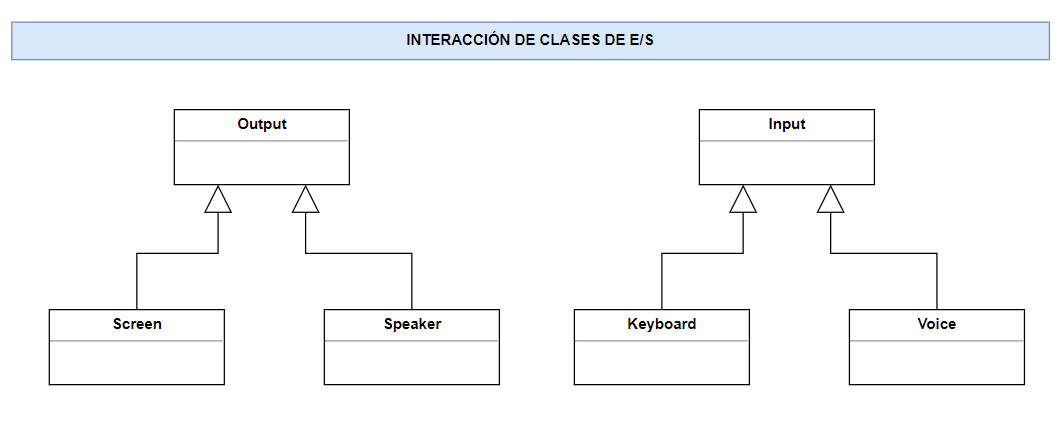


Ilustración : Diagrama de clases – Módulos de interacción

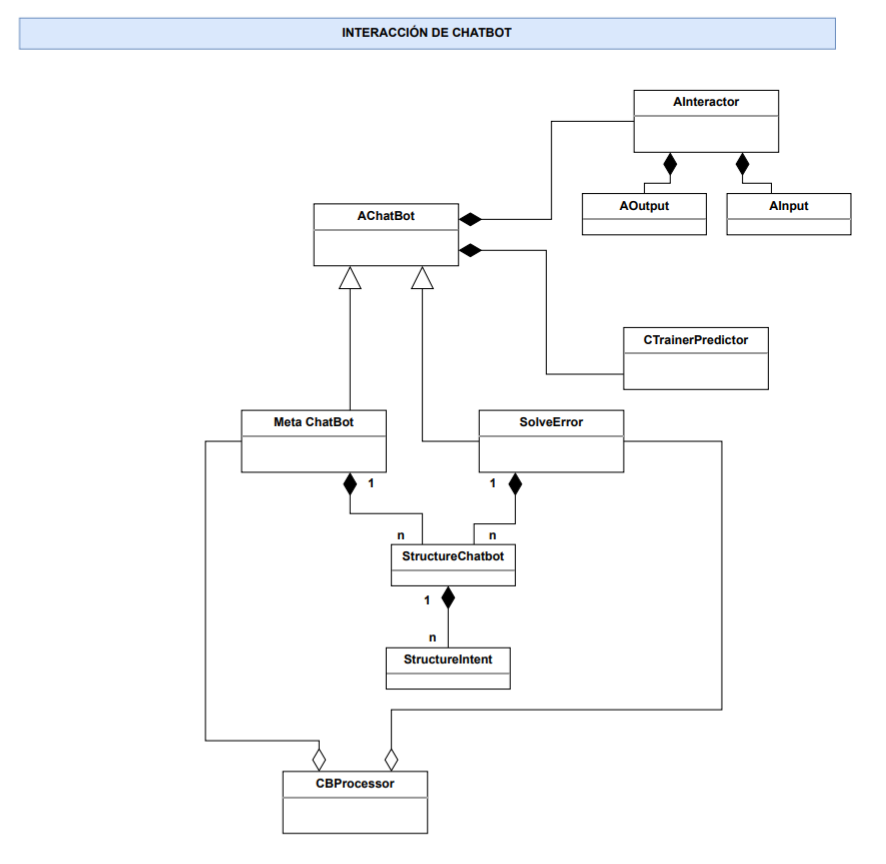
* En la siguiente imagen se muestra la interacción general del sistema completo:

Ilustración : Diagrama de clases - Clases del sistema

* A continuación, se mostrarán las acciones de los chatbots del sistema, Meta Chatbot y Solve Error.

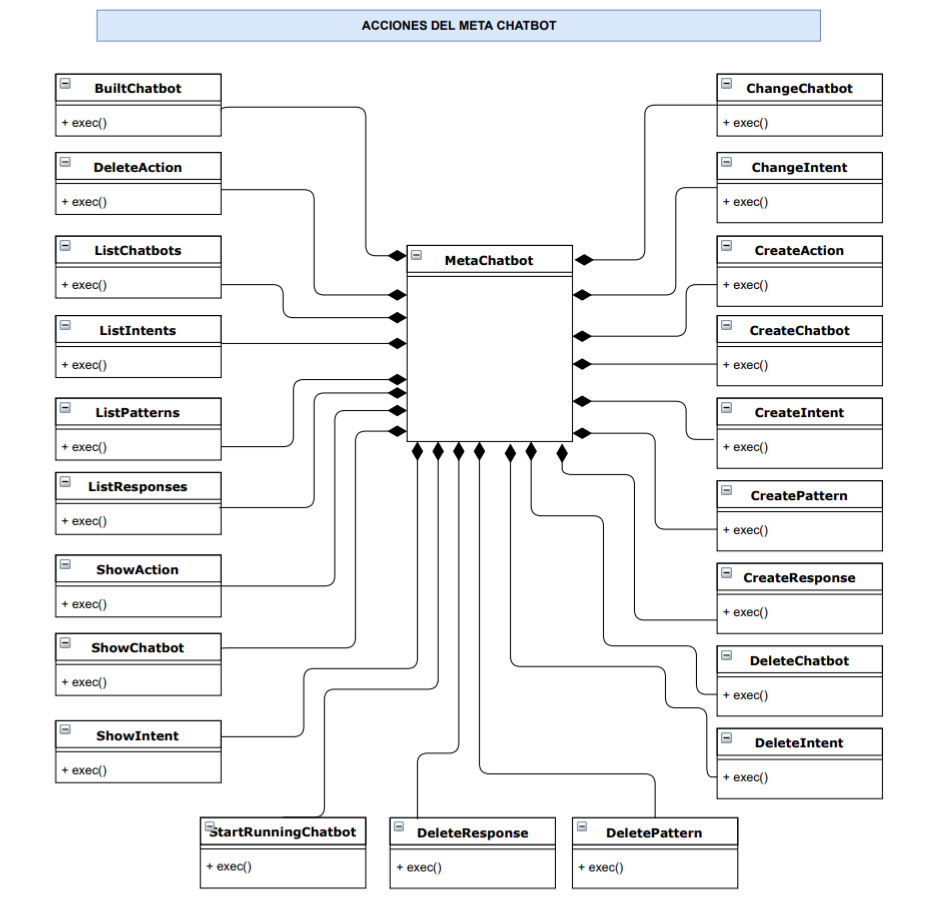


Ilustración : Diagrama de clases - Acciones del Meta Chatbot

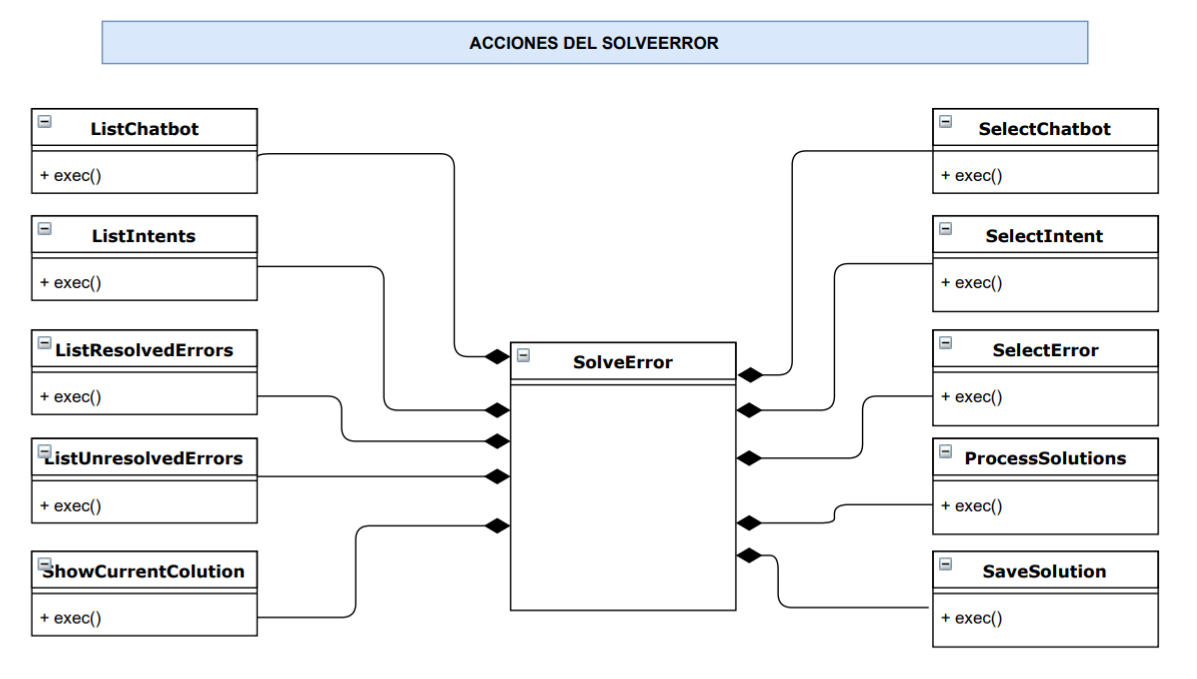


Ilustración : Diagrama de clases - Acciones del Solve Error

## 5.4 ESTRUCTURA JSON

Para la estructura de los ficheros JSON, donde se guardan el contenido de cada ChatBot, se ha tenido en cuenta la definición de **gk\_** aunque con unas ligeras modificaciones.

En la estructura que este sistema usa, no hace uso de contextos, pero sí de acciones.

La primera estructura que se encuentra en el JSON es un diccionario que contendrá, como clave, el nombre del ChatBot que se ha creado y como valor una lista de las Intenciones que contiene.

La estructura de cada Intención es un diccionario que contiene:

* “tag”: el nombre de la intención que se ha creado
* “patterns”: los patrones (peticiones) que el usuario indicará para que se ejecute una acción.
* “responses”: son las posibles respuestas del sistema frente al patrón que el usuario haya indicado.
* “action”: es la acción que se ejecutará. Cada acción está vinculado a una instancia de una clase que tendrá el código de la acción a ejecutar.

La siguiente imagen muestra la estructura JSON del Meta ChatBot:



Ilustración : Estructura de los ficheros JSON

## 5.5 CÓDIGO RELEVANTE

Dentro de esta sección, se comentarán los métodos que son importantes para el funcionamiento del sistema.

* Métodos de la clase CTrainerPredictor:
  + def predict(self, sentence): Dentro de éste método se da una respuesta acorde a la frase introducida.  
     :param sentence: Frase a reconocer para dar una respuesta.  
     :return: Respuesta según la frase introducida.
  + def classify(self,sentence): Clasifica la sentencia mostrando las posibles soluciones y su valor de coincidencia.  
     :param sentence: Frase a clasificar.  
     :return: Lista de posibles soluciones.
* Métodos de la clase CBProcessor:
  + def run(self): Método para todos los ChatBots para la ejecución de los mismos.
* Métodos de la clase StructureChatbot:
  + def toJSON(self): Convierte la estructura del chatbot a JSON.  
     :return: Devuelve una cadena de caracteres.
  + def dicToJSON(self,dicIntents): Convierte la estructura de un diccionario a json.  
     :param dicIntents: Diccionario a convertir.  
     :return: Devuelve una cadena de caracteres.
  + def removeASCII(self,name): Transforma el parámetro quitando los caracteres especiales y los espacios en blanco.  
     :param name: String que se quiere modificar.  
     :return: String modificado sin caracteres especiales.
  + def codeToStructureChatbot(self,chatbot,intents): Carga la estructura del chatbot desde su json.  
     :param chatbot: Nombre del chatbot.  
     :param intents: Lista de intenciones del chatbot.
* Métodos de la clase StructureIntent:
  + def toJSON(self): Convierte la estructura de una Intención a formato json.  
     :return: Devuelve una cadena de caracteres.
* Métodos de la clase AChatBot:
  + def execPrediction(self,sentence): Ejecuta la acción que se reconoce con el Predictor.  
     :param sentence: Sentencia que se usa para la predicción.
* Métodos de la clase Meta ChatBot:
  + def initializePaths(self): Inicializa las rutas del MetaChatbot para saber dónde está el fichero de errores, la ruta donde guardar el modelo, etc...
  + def loadChatbots(self): Carga los chatbots que se han creado con el MetaChatBot y los pone en una lista de chatbots.
* Métodos de la clase SolveError:
  + def getChatbots(self): Método que obtiene lista de todos los chatbots  
     :return: Lista de chatbots creados.

## 5.6 FUNCIONAMINETO

Con el siguiente diagrama se pretende mostrar el flujo básico que realiza al crear un ChatBot, mostrando las principales acciones a la hora de crear una chatbot.

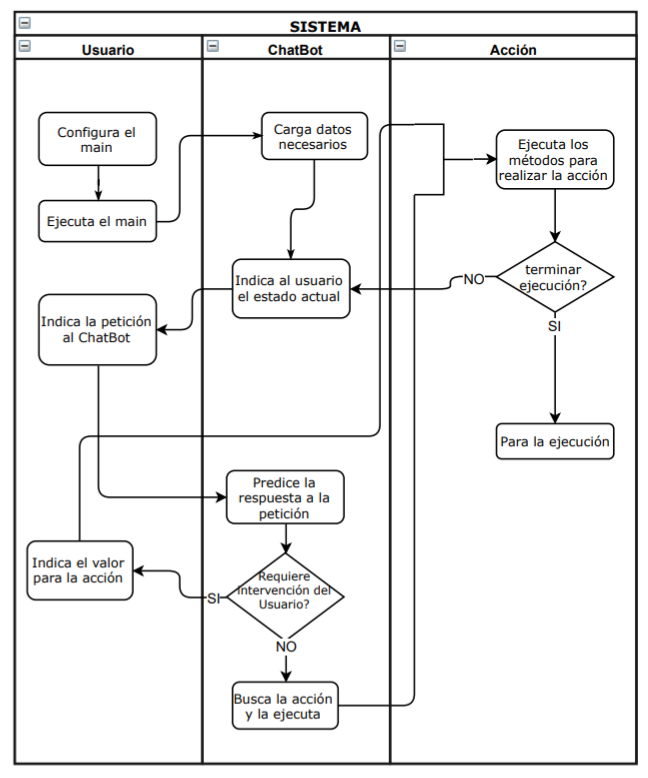


Ilustración : Diagrama de estado - Funcionamiento básico del sistema

# CAPÍTULO 6: RESULTADOS

En esta sección se planteará un ejemplo de ejecución del sistema, mostrándolo mediante capturas de pantalla, planteando el contexto actual de la ejecución.

En las imágenes se mostrará las peticiones del usuario en color verde y las respuestas del sistema en color negro.

* Crear el Chatbot “compra online”

Para la ejecución de este ejemplo, en el sistema ya existe otro Chatbot, “lista de la compra”. Se mostrará la interacción teclado/pantalla.

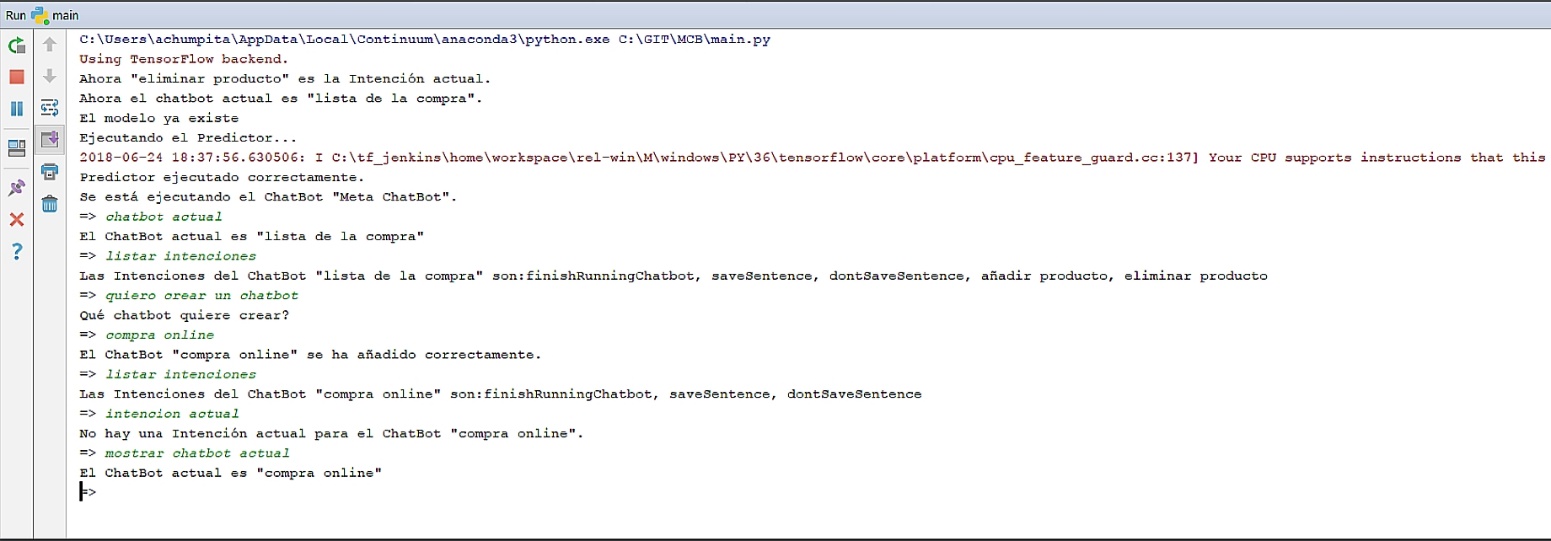


Ilustración : Ejemplo de ejecución - Crear chatbot

Como se puede ver, las acciones implementadas han sido ejecutadas gracias a que se ha podido reconocer la sintáxis del lenguaje.

* Crear una Intención, Patrones, Respuestas y Acción:

Este ejemplo es la continuación del anterior.

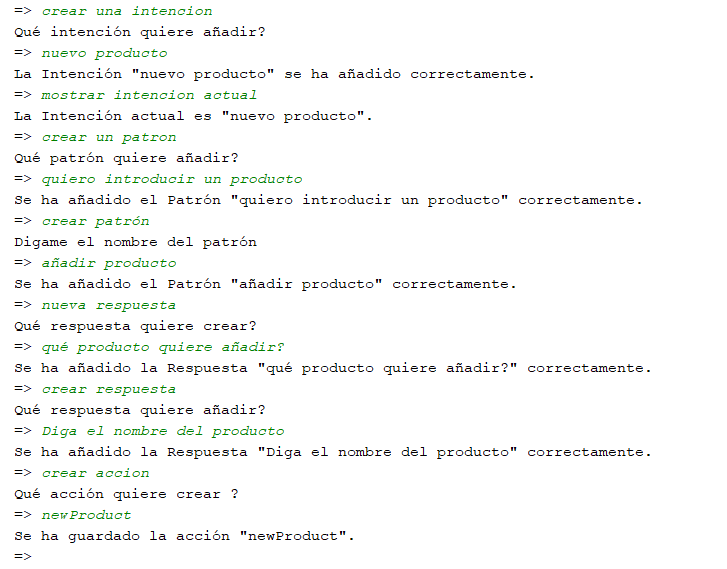


Ilustración : Ejemplo de ejecución - Crear intención, patrón, respuesta y acción

El sistema reconoce distintas formas de patrones y puede dar una respuesta al azar entre la lista de Respuestas. A su ves, puede mostrar y reconocer caracteres especiales tales como las tíldes, eñes, signos, etc.

* Generar el ChatBot

Para realizar este ejemplo, se han creado más intenciones.

En las siguientes imágenes se puede ver que el Meta ChatBot genera el modelo del ChatBot “compra online” reconociendo las raíces de los patrones.

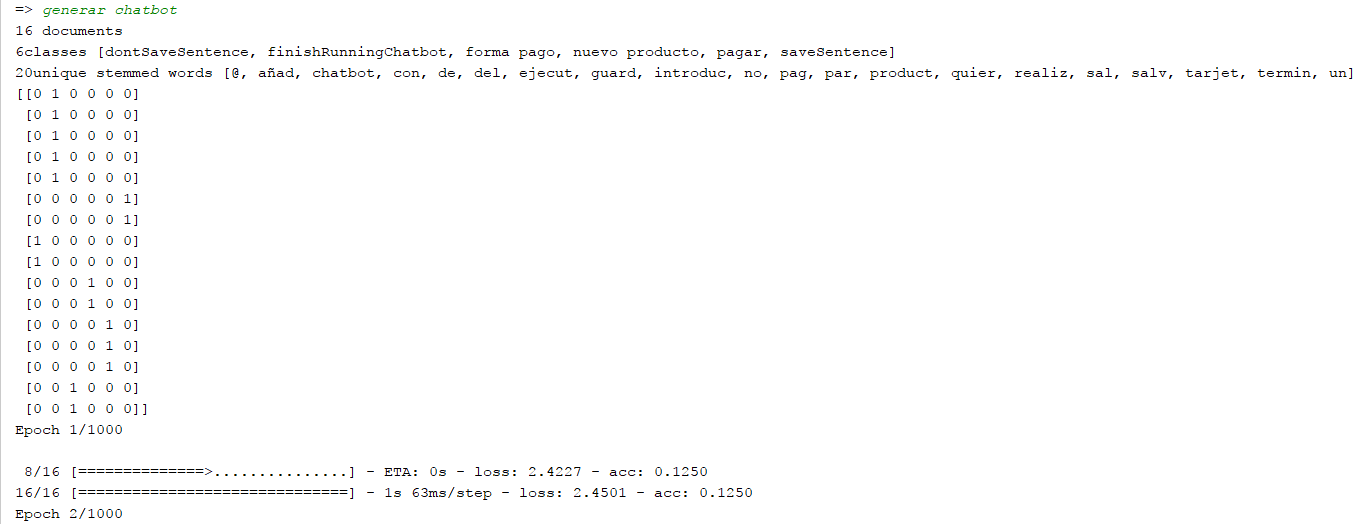


Ilustración : Ejemplo de ejecución - Generar el chatbot (1)

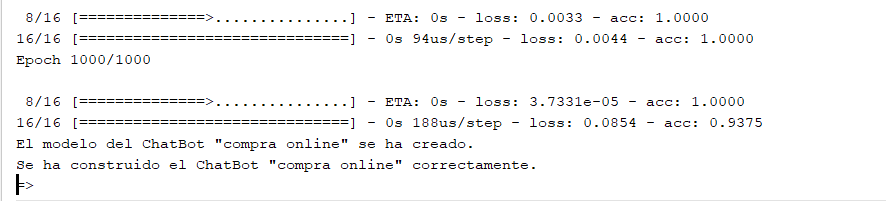


Ilustración : Ejemplo de ejecución - Generar el chatbot (2)

A su vez, va generando los ficheros necesarios tales como el fichero JSON, los ficheros Python de las acciones y el del programa principal.

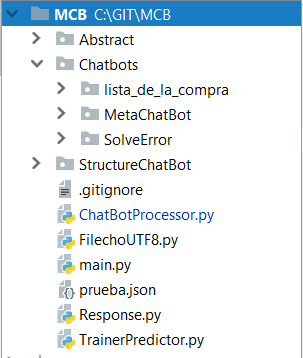


Ilustración : Ejemplo de ejecución - Resultado de ficheros(1)

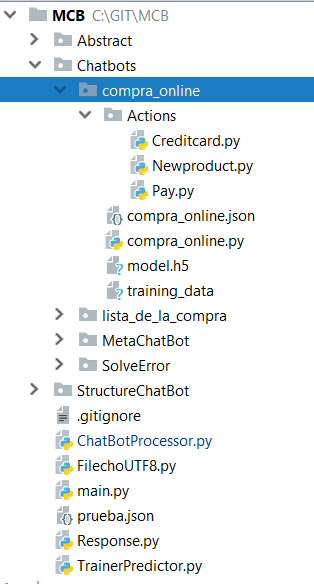


Ilustración : Ejemplo de ejecución - Resultado de ficheros(2)

* Ejecutar “compra online” desde el Meta Chatbot

En este ejemplo después de ejecutar el ChatBot “compra online” se ejecutarán sus acciones. Ya que, en los ficheros de las acciones no está implementado el código que debe ejecutar cada uno de ellos, no mostrará una respuesta.

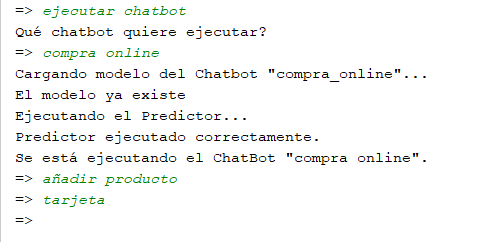


Ilustración : Ejemplo de ejecución - Ejecución de chatbot creado

* Terminar las ejecuciones

En siguiente imagen se muestra la finalización de la ejecución tanto del ChatBot “compra online” como el del Meta ChatBot. Se muestra un mensaje de aviso antes de salir de la ejecución de cada ChatBot.

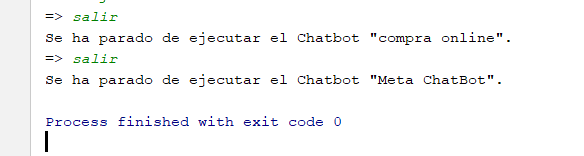


Ilustración : Ejemplo de ejecución - Terminar ejecución

# CAPÍTULO 7: CONCLUSIONES

Durante el desarrollo del sistema, aunque se haya conseguido los objetivos, también se han cometido pequeños errores tales como el no haber programado de forma óptima la encapsulación causando el rediseño de las estructuras de algunas clases; el no investigar más sobre algunas herramientas para evitar los conflictos de las versiones.

Se ha aprendido a programar con un lenguaje nuevo, Python. Conociendo nuevas formas de programación, entornos, librerías, comunidades de apoyo, etc., enriqueciendo el aprendizaje.

Cada vez que el programa necesitaba de nuevas librerías se ha tenido que buscar referencias/ejemplos de uso de las mismas para poder aprender a usarlas, mediante el desarrollo del sistema se ha podido obtener conocimientos de un lenguaje que, sin duda para muchos tal y como se ha mencionado en anteriores capítulos, es flexible y de fácil entendimiento y con una gran potencia.

Aunque ya existan programas propios de empresas de renombre que permiten hacer lo que se planteó con este proyecto, no se ha podido encontrar uno libre que permita hacer lo mismo y con características como que el propio usuario decida qué chatbot desea crear y con una interacción que se pueda programar y usar en el sistema.

Sin duda, de lo aprendido con este proyecto, lo destacado es el desarrollo del sistema de forma modular para poder realizar cambios/mejoras de forma más fácil, lo que facilitará en un futuro las ampliaciones y los conocimientos referente al área del Maching Learning.

# CAPÍTULO 8: TRABAJO FUTURO

Este sistema se puede mejorar teniendo en cuenta distintos focos de interés, por ejemplo:

1. Se podría programar una aplicación móvil que use este sistema para aumentar la facilidad de uso del sistema.
2. Si se quiere enfocar a una aplicación web, el desarrollo de la parte FrontEnd de la misma ayudaría a que la aplicación sea más fácil de entender.
3. Además, la programación de un nuevo ChatBot de ayuda sería ideal para aquellas personas que no entendiesen mucho del tema, lo que haría llegar este sistema a muchas más personas sin que éstas tuviesen mucho conocimiento del tema.
4. También se podría incluir un ChatBot “escritorio”, que permitiese al usuario poder seleccionar el ChatBot que quiera ejecutar.

Siguiendo el símil del escritorio, el nuevo ChatBot sería el escritorio de un ordenador cualquiera y los demás ChatBots (generales o creados por el Meta ChatBot) sería las aplicaciones que el usuario podrá ejecutar seleccionándolas.

1. Tocando la parte funcional interna, se podría cambiar las acciones por otros ChatBots para conseguir encapsular aún más la funcionalidad. Por lo que se conseguiría simular la contextualización del lenguaje
2. Mejorar las interacciones con el sistema.
3. Permitir al usuario el modo de interacción para acercar el sistema a la usabilidad universal.
4. Permitir al usuario indicar la ruta donde desea guardar la estructura del chatbot que está creando y, además darle la opción de introducir una ruta para que el Meta Chatbot cargue todos los ChatBots que se encuentren en esta ruta.
5. Dar la facilidad al usuario para que pueda introducir el código de las acciones de sus ChatBots.

Como se puede ver, este proyecto se puede seguir ampliando añadiendo mejoras considerables, enfocándolo de muchas formas, sus mejoras pueden variar.

[Ilustración 1: Esquema ChatBot 13](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260000)

[Ilustración 2: Gráfico Búsquedas en todo el Mundo 14](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260001)

[Ilustración 3: Gráfico Búsquedas - Top 5 países 14](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260002)

[Ilustración 4: Encuesta a usuarios sobre los beneficios que desean 16](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260003)

[Ilustración 5: Encuesta a usuarios sobre efectividad de un ChatBot 17](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260004)

[Ilustración 6: Estructura de un chatbot - código 24](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260005)

[Ilustración 7: Métodos que ejecutan el chatbot 25](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260006)

[Ilustración 8: Clases de interacción con el usuario 26](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260007)

[Ilustración 9: Clase Interactor 26](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260008)

[Ilustración 10: Estructura de ChatBot 27](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260009)

[Ilustración 11: Estructura de Meta Chatbot y Solve Error 28](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260010)

[Ilustración 12: Arquitectura de carpetas del sistema 34](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260011)

[Ilustración 13: Diagrama de clases - StructureCahtBot y StructureIntent 36](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260012)

[Ilustración 14: Diagrama de clases – Módulos de interacción 37](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260013)

[Ilustración 15: Diagrama de clases - Clases del sistema 37](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260014)

[Ilustración 16: Diagrama de clases - Acciones del Meta Chatbot 38](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260015)

[Ilustración 17: Diagrama de clases - Acciones del Solve Error 39](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260016)

[Ilustración 18: Estructura de los ficheros JSON 40](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260017)

[Ilustración 19: Diagrama de estado - Funcionamiento básico del sistema 42](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260018)

[Ilustración 20: Ejemplo de ejecución - Crear chatbot 43](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260019)

[Ilustración 21: Ejemplo de ejecución - Crear intención, patrón, respuesta y acción 44](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260020)

[Ilustración 22: Ejemplo de ejecución - Generar el chatbot (1) 44](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260021)

[Ilustración 23: Ejemplo de ejecución - Generar el chatbot (2) 45](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260022)

[Ilustración 24: Ejemplo de ejecución - Resultado de ficheros(1) 45](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260023)

[Ilustración 25: Ejemplo de ejecución - Resultado de ficheros(2) 45](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260024)

[Ilustración 26: Ejemplo de ejecución - Ejecución de chatbot creado 46](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260025)

[Ilustración 27: Ejemplo de ejecución - Terminar ejecución 46](file:///C:\Users\achumpita\Desktop\TFG-MEMORIA-MCB_V2.docx#_Toc524260026)

# Bibliografía

elEconomista. «¿Qué son exactamente los chatbots y para qué sirven?» *elEconomista*, 13 de Abril de 2016: 1.

gk\_. «Contextual Chatbots with Tensorflow.» *chatbotmagazine*, 2017: 1.

HAJ-SALEH, ALBERTO. «Qué son exactamente los 'bots' y cómo funcionan.» *GQ*, 2017.

Wikipedia. «Artificial Linguistic Internet Computer Entity.» 2018.

Wikipedia. «Bot Conversacional.» 1. 2018.

Wikipedia. «Test de Turing.» 1. 2018.

1. Prensa online: www.eleconomista.es [↑](#footnote-ref-1)
2. Enciclopedia online: es.wikipedia.org [↑](#footnote-ref-2)
3. Revista online dedicada a Chatbots: chatbotsmagazine.com [↑](#footnote-ref-3)
4. Seudónimo del autor que publicó el artículo del cual se basa este trabajo. [↑](#footnote-ref-4)
5. Es la secuencia más pequeña de instrucciones programadas que puede ser manejada independientemente por un programador. [↑](#footnote-ref-5)
6. Medio digital especializado en Bots: canalbots.com [↑](#footnote-ref-6)
7. Revista online dedicada a inteligencia artificial, chatbots: planetachatbot.com [↑](#footnote-ref-7)
8. Autor del artículo, científico informático retirado y el inventor del motor de búsqueda web de Lycos. [↑](#footnote-ref-8)
9. Profesor de informática en la Universidad de Yale, investigador de algoritmos distribuidos. [↑](#footnote-ref-9)
10. Multinacional estadounidense de tecnología software. [↑](#footnote-ref-10)
11. Es un proyecto para la extracción de datos de [Wikipedia](https://es.wikipedia.org/wiki/Wikipedia) para proponer una versión [Web semántica](https://es.wikipedia.org/wiki/Web_sem%C3%A1ntica). [↑](#footnote-ref-11)
12. Es una [base de datos léxica](https://es.wikipedia.org/w/index.php?title=Base_de_datos_l%C3%A9xica&action=edit&redlink=1) del [Idioma inglés](https://es.wikipedia.org/wiki/Idioma_ingl%C3%A9s). [↑](#footnote-ref-12)
13. Es una herramienta de [Google Labs](https://es.wikipedia.org/wiki/Google_Labs) que muestra los términos de búsqueda más populares del pasado reciente. [↑](#footnote-ref-13)
14. Es un portal en línea de estadísticas, investigación de mercado e inteligencia empresarial. [↑](#footnote-ref-14)
15. Web donde se enseña programación básica, conceptos, estrategia de enseñanza, dirigida a maestros. [↑](#footnote-ref-15)
16. Blog, donde ofrece pequeños cursos de programación. [↑](#footnote-ref-16)
17. Es una web donde se publican contenidos variados sobre tecnología, ocio, motor, publicaciones para

    marketing, etc. [↑](#footnote-ref-17)
18. Es una biblioteca de [código abierto](https://es.wikipedia.org/wiki/C%C3%B3digo_abierto) para [aprendizaje automático](https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico) a través de un rango de tareas, y desarrollado por [Google](https://es.wikipedia.org/wiki/Google), capas de entrenar [redes neuronales](https://es.wikipedia.org/wiki/Red_neuronal_artificial). [↑](#footnote-ref-18)
19. Keras es una biblioteca de red neuronal de código abierto escrita en Python. Es capaz de ejecutarse sobre TensorFlow, Microsoft Cognitive Toolkit o Theano. [↑](#footnote-ref-19)
20. El módulo SimpleXMLRPCServer proporciona un marco de servidor básico para servidores XML-RPC escritos en Python. Los servidores pueden ser independientes, usar SimpleXMLRPCServer o incrustados en un entorno CGI. [↑](#footnote-ref-20)
21. Página web oficial del entorno de desarrollo integrado Pycharm. [↑](#footnote-ref-21)
22. Profesional informático y escritor de artículos de la revista online Quora. [↑](#footnote-ref-22)
23. Colaboradora de la web. [↑](#footnote-ref-23)
24. Plataforma para la innovación abierta, la comunidad de desarrolladores orientada al open banking, open platform o las Fintech APIs. [↑](#footnote-ref-24)