

**UNIVERSIDAD AUTÓNOMA DE SINALOA
POSGRADO EN CIENCIAS DE LA INFORMACIÓN**

**Facultad de Informática Culiacán
Facultad de Ciencias de la Tierra y el Espacio**



**UNA PROPUESTA PARA LA
SEMI-AUTOMATIZACIÓN EN LA GENERACIÓN
DE AGENTES CONVERSACIONALES PARA
SERVICIOS DE MENSAJERÍA INSTANTÁNEA**

TESIS

**QUE PARA OBTENER EL TÍTULO DE
MAESTRÍA EN CIENCIAS DE LA INFORMACIÓN**

PRESENTA

ÁNGEL ANTONIO MARTÍNEZ GÁRATE

DIRECTORES:

**DR. JOSÉ ALFONSO AGUILAR CALDERÓN
DRA. CAROLINA TRIPP BARBA**



Dirección General de Bibliotecas



U n i v e r s i d a d A u t ó n o m a d e S i n a l o a

Restricción de uso

UAS-Dirección General de Bibliotecas

Repositorio Institucional Buelna

Restricciones de uso

Todo el material contenido en la presente tesis está protegido por la Ley Federal de Derechos de Autor (LFDA) de los Estados Unidos Mexicanos (México).

Queda prohibido la reproducción parcial o total de esta tesis. El uso de imágenes, tablas, gráficas, texto y demás material que sea objeto de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente correctamente mencionando al o los autores del presente estudio empírico. Cualquier uso distinto, como el lucro, reproducción, edición o modificación sin autorización expresa de quienes gozan de la propiedad intelectual, será perseguido y sancionado por el Instituto Nacional de Derechos de Autor.

Esta obra está bajo una Licencia Creative Commons Atribución-No Comercial Compartir Igual, 4.0 Internacional.





UNIVERSIDAD AUTÓNOMA DE SINALOA
DIRECCIÓN GENERAL DE BIBLIOTECAS
REPOSITORIO INSTITUCIONAL BUELNA

CARTA CESIÓN DE DERECHOS

En la Ciudad de Culiacan Sinaloa el día 29 del mes Mayo del
año 2023, el (la) que suscribe:
Angel Antonio Martinez Garate alumno (a) del

Programa de: Posgrado en ciencias de la informacion

con número de cuenta: 15934462, de la Unidad Académica:
FACULTAD DE INFORMÁTICA CULIACÁN, manifiesta que es autor (a)

intelectual del presente trabajo de Tesis bajo la dirección de:
Dr. Jose Alfonso Aguilar Calderon y de acuerdo al artículo 27

de la Ley Federal de Derechos de Autor, cede los derechos del trabajo intitulado:
" UNA PROPUESTA PARA LASEMI-AUTOMATIZACIÓN EN LA GENERACIÓN DE

AGENTES CONVERSACIONALES PARASERVICIOS DE MENSAJERÍA INSTANTÁNEA

a la Universidad Autónoma de Sinaloa para su publicación, difusión, edición, reedición, traducción, compilación, distribución y explotación en medios impresos y digitales, con fines académicos y de investigación, la que será titular del mismo, en forma conjunta o separada con el autor.

Todo el material contenido en la presente tesis está protegido por la Ley Federal de Derechos de Autor (LFDA) de los Estados Unidos Mexicanos (México).

Queda prohibido la reproducción parcial o total de ésta tesis. El uso de imágenes, tablas, gráficas, texto y demás material que sea objeto de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente correctamente mencionando al o los autores del presente estudio empírico. Cualquier uso distinto, como el lucro, reproducción, edición o modificación sin autorización expresa de quienes gozan de la propiedad intelectual, será perseguido y sancionado por el Instituto Nacional de Derechos de Autor.

En apego al Art. 27 de la Ley Federal de Derechos de Autor Cedo el derecho de publicación, difusión, edición, reedición, traducción, compilación, distribución y explotación en medios impresos y digitales, con fines académicos y de investigación a la Universidad Autónoma de Sinaloa.

Angel Martinez
Angel Antonio Martinez Garate

Nombre completo y firma



DECLARACIÓN DE ORIGINALIDAD Y ANTIPLAGIO

A través de la presente Yo: **Angel Antonio Martinez Garate**, con número de cuenta: **15934462** y alumno del Programa de **Maestría en Ciencias de la Información**, con sede en la Facultad de Informática Culiacán y Facultad de Ciencias de la Tierra y el Espacio de la Universidad Autónoma de Sinaloa.

DECLARO QUE:

El trabajo de tesis que presento para su defensa y el cual se titula: **"UNA PROPUESTA PARA LA SEMI-AUTOMATIZACIÓN EN LA GENERACIÓN DE AGENTES CONVERSACIONALES PARA SERVICIOS DE MENSAJERÍA INSTANTÁNEA"**

- 1.- Es el resultado de un trabajo personal y constituye un documento original e inédito.
- 2.- Los documentos, artículos, informes, etc. Utilizados para la realización del trabajo están referenciados de forma clara, tanto en el texto principal como en el apartado de bibliografía de la presente tesis.
- 3.- Que las citas de las obras que aparecen reflejadas de forma literal y entre comillas, presentan clara especificación de la fuente (autor, año) d la que ha sido extraídas.
- 4.- Tengo conocimiento que no puedo asumir como propio el trabajo de tesis de otra persona, la copia de frases, párrafos e ideas de otro trabajo sin citar autoría, o cualquier otra acción de similares características conlleva a la realización fraudulenta del manuscrito de tesis y podrá implicar la suspensión del examen de grado en tanto no se corrija o bien la suspensión definitiva, esto acorde con el nivel en que se pudiese incurrir.
- 5.- Soy consciente de que pretender ser el autor de un trabajo, en parte o en su totalidad, escrito por otra persona constituye un delito con base en los artículos 424 y 427 del código penal federal, título vigesimosexto de los delitos en materia de derechos de autor en la República Mexicana.

Culiacán, Sinaloa a 29 de mayo de 2023

Angel Martinez
Angel Antonio Martinez Gárate
Nombre y Firma del alumno

DEDICATORIA

Dedico esta tesis.

A mis padres:

Celestina Gárate Flores y Rosendo Antonio Martínez Naranjo
que siempre me apoyaron incondicionalmente.

A la memoria de mis Abuelos:

Lilia Flores Hernández y Félix Garate Hernández
quienes en vida siempre me apoyaron e inspiraron.

Agradecimientos

En primer lugar, agradezco a Dios por su guía y bendición en mi vida.

Quiero agradecer a mis padres Celestina Gárate Flores y Rosendo Antonio Martínez Naranjo, por su constante apoyo y amor incondicional. Han sido mi pilar durante toda mi vida y en este proyecto no ha sido la excepción.

Agradezco a mis directores de tesis el Dr. José Alfonso Aguilar Calderón y la Dra. Carolina Tripp Barba por su paciencia, orientación y apoyo incansable durante el proceso de investigación y escritura de esta tesis. Gracias por su dedicación y por la confianza depositada en mí.

Agradezco a la Dra. Xiomara P. Zaldívar Colado y al Dr. Aníbal Zaldívar Colado por sus consejos y apoyo en mi proceso de estudios de maestría.

Quiero expresar mi más sincero agradecimiento a mis compañeros que han sido una fuente de motivación y de inspiración para mí y por compartir este proceso conmigo.

Agradezco a los docentes del posgrado por compartir sus conocimientos y contribuir en mi formación académica.

Agradezco a la Universidad Autónoma de Sinaloa y al Posgrado en Ciencias de la Información por brindarme las herramientas y las oportunidades necesarias para llevar a cabo esta investigación.

Por último, pero no menos importante quiero expresar mi agradecimiento al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo otorgado que fue de vital importancia para poder continuar con mi formación académica.

Resumen

El desarrollo de *chatbots* ha presentado desafíos significativos debido al alto costo y tiempo requerido para su implementación. Aunque existen varios entornos de trabajo disponibles, su adaptación a entornos organizacionales se ve limitada por la necesidad de conocimientos técnicos en programación. Esto se debe a que la creación de un agente conversacional involucra una amplia gama de tecnologías y conocimientos, incluyendo el procesamiento del lenguaje natural, tecnologías de *frontend*, *backend* y bases de datos.

Actualmente, existen diversas tecnologías y marcos de trabajo para facilitar la definición y desarrollo de *chatbots*. Lamentablemente, no son lo suficientemente robustas para brindar soporte a todos los requerimientos necesarios para desarrollar un agente conversacional, por ejemplo, las diferentes plataformas de implementación como la plataforma de mensajería instantánea WhatsApp. Esto representa una limitante para la aplicación de estos marcos de trabajo en el ámbito profesional. Además, la experiencia de usuario juega un papel crucial en la aceptación y adopción de nuevas tecnologías, por lo que es fundamental que los desarrolladores construyan herramientas útiles y fáciles de usar para propiciar su rápida adopción en la industria. Con el objetivo de abordar estos desafíos, en esta investigación se planteó la reducción de la complejidad en la implementación de agentes conversacionales y mejorar la experiencia de usuario del marco de trabajo para desarrollo de *chatbots* Xatkitl.

Esta tesis presenta una propuesta para la generación de *chatbots* en el marco de trabajo Xatkit para la plataforma de mensajería WhatsApp. Se empleó el paradigma del desarrollo dirigido por modelos para simplificar el proceso de desarrollo, así como el modelo de lenguaje autorregresivo GPT-3 para mejorar la interacción con el usuario. Los resultados del caso de estudio presentado validan la hipótesis de investigación y el aporte al campo de desarrollo de agentes conversacionales.

Índice general

Índice de figuras	VI
Índice de tablas	VII
Índice de fragmentos de códigos	VII
1. Capítulo 1. Introducción	1
1.1. Motivación	1
1.2. Problemática	2
1.3. Objetivo de la tesis	5
1.4. Objetivo general	6
1.5. Objetivos específicos	6
1.6. Estructura del documento	7
2. Capítulo 2. Marco Teórico	8
2.1. Fundamentos de la Ingeniería Dirigida por Modelos	8
2.1.1. Modelo	8
2.1.2. Metamodelo	10
2.1.3. Domain-Specific Language	11
2.1.4. Eclipse Modeling Framework	13
2.1.5. Model-Driven Engineering	14
2.1.6. Model-Driven Development	15
2.1.7. Model-Driven Architecture	16
2.2. Fundamentos de los Agentes Conversacionales	18
2.2.1. Definición de Agente Conversacional	18
2.2.2. <i>Intent</i> / Intención	20
2.2.3. <i>Entity</i> / Entidad	20
2.3. Procesamiento de Lenguaje Natural	20
2.3.1. Comprensión de Lenguaje Natural	22
2.3.2. Generación de Lenguaje Natural	24
2.3.3. <i>Transformer</i>	24
2.4. Twilio	25
2.5. Xatkit Chat Platform	26
2.5.1. Análisis del lenguaje de dominio específico de Xatkit Chat Platform	27
2.5.2. Mecanismos de extensión	29
2.6. Modelo de lenguaje autorregresivo Generative Pre-trained Transformer 3 .	30

2.6.1.	Funcionamiento	30
2.6.2.	Mecanismos de implementación	33
2.7.	Comentarios Finales	33
3.	Capítulo 3. Estado del Arte	35
3.1.	Mapeo Sistemático de la Literatura	35
3.1.1.	Preguntas de Investigación	36
3.1.2.	Búsqueda de Estudios Primarios	37
3.1.3.	Criterios de Inclusión y Exclusión	39
3.1.4.	Esquema de Clasificación	40
3.2.	Resultados del Mapeo Sistemático de la Literatura	42
3.3.	Discusión	56
3.4.	Comentarios Finales	58
4.	Capítulo 4. Una Propuesta Dirigida por Modelos para la Generación de Agentes Conversacionales	60
4.1.	Introducción	60
4.2.	Semi-automatización de agentes conversacionales mediante el desarrollo dirigido por modelos	61
4.2.1.	Definición de un DSL mediante la implementación de un meta-modelo	62
4.2.2.	Transformaciones modelo a texto para la generación de código fuente	64
4.2.3.	Implementación de GPT-3 como alternativa para resolver consultas de usuario	66
4.3.	Caso de Estudio	75
4.3.1.	Contextualización del caso de estudio	75
4.3.2.	Participantes	76
4.3.3.	Diseño del Chatbot	77
4.3.4.	Desarrollo del Chatbot	83
4.3.5.	Implementación del Chatbot	92
4.3.6.	Evaluación	93
4.3.7.	Resultados	94
4.4.	Comentarios finales	102
5.	Capítulo 5. Conclusiones	104
5.1.	Conclusiones	104
5.2.	Trabajo futuro	107
A.	Plantilla Acceleo M2T para generación de agentes conversacionales	109
B.	Encuesta para caso de estudio	116
	Glosario	120
	Bibliografía	122

Índice de figuras

2.1. Ejemplo de modelo: mapa del estado de Sinaloa.	9
2.2. Ejemplo diagrama de clases UML.	10
2.3. Metamodelo de un formulario.	11
2.4. Metamodelo de un sitio web.	12
2.5. Modelo de un sitio web.	13
2.6. Fragmento de sitio creado a partir de un modelo.	13
2.7. Proceso MDA.	17
2.8. Ejemplo <i>one-hot encoding</i>	22
2.9. Comunicación humano-maquina utilizando el procesamiento de lenguaje natural.	23
2.10. Ejemplo de diagrama de estados.	28
2.11. Metamodelo del <i>platform package</i>	30
3.1. Porcentaje de artículos primarios por categoría de clasificación.	43
3.2. Clasificación de propuesta/herramienta.	45
3.3. Tendencia en las publicaciones.	54
4.1. Metamodelo XatbotMM.	63
4.2. Modelo de <i>Chatbot</i> usando XatbotMM.	64
4.3. Ejemplo de recomendación utilizando la propuesta descrita.	71
4.4. Fragmento del modelo del PCIBot.	86
4.5. Captura de la interacción de PCIBot con un usuario.	94
4.6. Satisfacción general con el uso del <i>Chatbot</i>	95
4.7. Dificultad de uso del <i>chatbot</i>	95
4.8. Aceptación de plataforma.	97
4.9. Preferencias de plataforma.	98
4.10. Otras opciones de plataformas.	98
4.11. Percepción de las recomendaciones.	99
4.12. Aceptación de la función de recomendación.	102

Índice de tablas

3.1.	Palabras clave para cadenas de búsqueda.	37
3.2.	Cadenas de búsqueda para estudios primarios.	38
3.3.	Filtrado de artículos aplicando criterios de inclusión y exclusión.	40
3.4.	Clasificación por tipo de publicación científica.	40
3.5.	Tipo de artículo científico de investigación.	41
3.6.	Clasificación por tipo de contribución.	42
3.7.	Clasificación por tipo de investigación publicada en los estudios primarios.	44
3.8.	Citas tipo de propuesta/herramienta.	52
3.9.	Clasificación por tipo de publicación científica.	55
4.1.	Lista de opciones de <i>prompt</i>	84
4.2.	Ejemplo de sugerencias.	85
4.3.	Ejemplos de recomendaciones en conversaciones con usuarios.	100
4.4.	Ejemplos de sugerencias incorrectas en conversaciones con usuarios.	101

Índice de fragmentos de códigos

4.1. Fragmento de la plantilla de la implementación de la transformación modelo a texto mediante el <i>plugin</i> de Eclipse Acceleo.	65
4.2. Inclusión de la lista de opciones disponibles en el <i>prompt</i> de GPT-3. . . .	68
4.3. Inclusión de las instrucciones en el <i>prompt</i> de GPT-3.	69
4.4. Inclusión de ejemplos en el <i>prompt</i> de GPT-3.	69
4.5. Inclusión del mensaje actual del usuario en el <i>prompt</i> de GPT-3.	70
4.6. Petición al API de GPT-3 para generación de texto.	72
4.7. Definición de plataforma TwilioPlatform.	73
4.8. Uso de la plataforma TwilioPlatform en un Chatbot.	74
4.9. Credenciales de Twilio.	74
4.10. Código para enviar un mensaje de texto mediante el twilioPlatform desde un estado.	74
4.11. Instalación de Xatkit.	86
4.12. Definición de frases de entrenamiento para el intent saludo.	87
4.13. Definición de entidades.	88
4.14. Extracción de información clave con el uso de entidades	88
4.15. Instancia de plataforma de Twilio.	88
4.16. Declaración de estados.	89
4.17. Transiciones entre estados.	89
4.18. Instancia del servicio de OpenAI.	89
4.19. Extracción del mensaje del usuario.	90
4.20. Generación de sugerencia con el servicio de OpenAI.	90
4.21. Configuraciones del Chatbot.	91
A.1. Implementación de la transformación modelo a texto mediante el <i>plugin</i> de Eclipse Acceleo.	109

Capítulo 1

Introducción

1.1. Motivación

Hoy en día el software se ha vuelto indispensable en la vida diaria del ser humano. Esto se debe a que es una herramienta básica en la mayoría de los giros profesionales tales como la comunicación y la educación. El software se encuentra presente desde el dispositivo móvil que se utiliza para la comunicación entre personas hasta en la comodidad de nuestro hogar para activar el funcionamiento de algún dispositivo.

Un mercado en el que se resalta el impacto que ha tenido la evolución del software es en el área de las ventas. La forma en que las empresas pueden llegar a los clientes se ha ampliado y agilizado debido a la rapidez de la tecnología, con lo cual, es indispensable contar con herramientas que sirvan de impulso y canal directo hacia los consumidores finales. Esto, debido a que ahora más que nunca, los clientes demandan tener acceso a la información de productos y servicios de forma eficiente.

Las corporaciones están transformando los canales con los que se comunican con sus clientes. Esto se debe principalmente a que el usuario demanda una mayor personalización en el servicio y soluciones inmediatas a sus necesidades. Para esto se han utilizado las redes sociales como mecanismo de comunicación directa con el cliente-consumidor, de tal forma que es posible que la empresa impacte con sus productos o servicios a un mayor número de clientes potenciales. Lamentablemente, una deficiencia en este sentido es la necesidad de contar con el recurso humano disponible para atenderlas incluso en horas

fuera de oficina que es cuando el cliente potencial ha llegado a solicitar información sobre un producto o servicio y al no recibir respuesta inmediata su interés disminuye.

Recientemente, se ha utilizado la Inteligencia Artificial (IA) para conocer, predecir e influir en las preferencias del consumidor con el propósito de incrementar las ventas o la demanda de servicios. Una de las áreas de implementación de la IA con mayor crecimiento es la enfocada en mejorar la atención al cliente y disminuir los recursos humanos que se requieren para la realización de una tarea. Para esto, se ha propuesto el uso de los agentes conversacionales o *chatbots*, los cuales son programas informáticos (software) que, mediante una interfaz conversacional y la utilización de lenguaje natural, permiten la interacción con los usuarios [1]. Son empleados en distintas áreas, desde la educación, la automatización de tareas, el servicio al cliente o incluso con fines recreativos, probando que tienen un gran potencial y son útiles para realizar una amplia cantidad de tareas.

Un área de aplicación que ha incrementado recientemente el uso de los *chatbots* es en la educación, particularmente, en las instituciones educativas. Las investigaciones señalan que adoptarlos en la educación brinda beneficios para sus usuarios y tienen el potencial de desempeñar distintos roles en las entidades educativas con resultados satisfactorios, tal es el caso de [1] y [2], en donde se utilizan para proveer información institucional en su plataforma web y como herramienta didáctica para los estudiantes. En ambos estudios los resultados muestran un alto nivel de satisfacción por parte de los usuarios.

1.2. Problemática

El desarrollo de software en la actualidad implica la interacción con personas sin los conocimientos técnicos de programación o externos a la codificación, esto según el equipo multidisciplinario que esté constituido en el proyecto. Por lo que se espera que la necesidad de nuevas tecnologías o la evolución de las existentes continúe en aumento. Conforme el software se vuelve más y más complejo surge la necesidad de tratarlo a diferentes niveles de abstracción. Esto dependiendo del perfil de los grupos de interés involucrados en esta tarea. Hoy en día el mercado laboral experimenta escasez de habilidades de programación en comparación con los profesionales disponibles [3]. Lo anterior, implica que el desarrollo de *chatbots*, sea cual sea su área de aplicación (educación, ventas, etc.), se

convierta en una actividad cada vez más compleja. La problemática radica en la cantidad de usuarios para los cuales deberá estar disponible, así como el perfil muy particular de cada uno de ellos. Con esto, la complejidad, disponibilidad y mantenibilidad del software será cada vez más difícil de lograr.

En [4], los autores señalan que la implementación de una aplicación de *chatbot* generalmente requiere una comprensión profunda de las plataformas objetivo, especialmente las conexiones de *back-end* (capa de procesamiento de datos), lo que aumenta los costos de desarrollo y mantenimiento (otro problema sumado a la diversidad de usuarios). Lo anterior es debido a la variedad de tecnologías que suelen necesitar en su desarrollo, por ejemplo, proveedores de procesamiento de lenguaje natural, aprendizaje automático, tecnologías de desarrollo de *front-end* (tecnologías que interactúan directamente con el usuario), sistemas administradores de bases de datos, etc. [5].

En la actualidad existen multitud de entornos de trabajo para desarrollar *bots* (abreviatura de *chatbot*) [6], sin embargo, se encuentran algunas limitaciones para adaptarlos a entornos organizacionales dado que se necesita realizar tareas de programación que requieren conocimientos técnicos. Los desarrolladores, adicionalmente, requieren gestionar la coordinación de los servicios cognitivos al momento de construir la interfaz del *chatbot* además de su integración con servicios externos. Esto sin considerar la extensibilidad, la escalabilidad y el mantenimiento. Por ende, tener control sobre el código fuente se vuelve fundamental si se piensa en la construcción de una herramienta que va a evolucionar con el tiempo y que requiere integrarse con el sistema de información de la empresa.

Una propuesta que ha surgido para reducir el esfuerzo y costo de producción en el desarrollo de software es la Ingeniería Dirigida por Modelos [2, 4, 6, 7, 8] (MDE, *Model-Driven Engineering*, por sus siglas en inglés). Este paradigma para la construcción de software tiene como objetivo semi-automatizar la generación de código fuente a partir de modelos que representen la funcionalidad a implementar. Comercialmente, MDE es el fundamento teórico de lo que hoy se conoce como aplicaciones *no code* y *low code* como por ejemplo Webratio y Microsoft Power Apps por mencionar algunas. Mediante la aplicación de una propuesta basada en MDE es posible crear herramientas software que permitan agilizar el desarrollo de *chatbots*. Lamentablemente, el tiempo de desarrollo no es el único problema referente a la implementación de los agentes conversacionales, sino

que también es necesario evaluar su desempeño en ambientes reales, por tal motivo otra área de estudio que ha tenido un creciente interés en lo referente a este tema se centra en analizar las técnicas más eficaces para evaluarlo. Para ello se considera un área para evaluación de la satisfacción del usuario conocida como experiencia de usuario (UX, User eXperience, por sus siglas en inglés). UX es la percepción que tiene una persona resultante de interactuar con un producto, sistema o servicio [9]. Actualmente, está establecido que la UX es de suma importancia para la aceptación y adopción de la población general a nuevas tecnologías por lo que es crucial que los desarrolladores construyan herramientas que sean útiles y fáciles de usar [10]. A pesar de esto, uno de los eventos que provocan más impresiones negativas en los agentes conversacionales es el problema de la interpretación, es decir, la dificultad que tienen los clientes para hacer que los *bots* los entiendan [9]. Esto resulta en una negativa o desagradable experiencia al utilizar agentes conversacionales.

Aun cuando existe evidencia en la literatura de la implementación de *chatbots* en la industria, por ejemplo, en las instituciones educativas [1, 2, 11, 12, 13], resalta de forma evidente la gran cantidad de tecnologías necesarias para crear los *bots*, por lo que como se indica anteriormente, esto demanda tiempo y costos considerablemente altos al requerir de experiencia en las tecnologías a utilizar para su construcción. Recientemente, se han desarrollado distintas herramientas que aplican MDE para facilitar el desarrollo de los *chatbots* entre las cuales se encuentra Xatkit [4]. Este marco de trabajo de código abierto permite el desarrollo e implementación de *chatbots* para diferentes entornos web y móvil. Xatkit utiliza MDE, mediante la definición de lenguajes específicos de dominio (DSL, *Domain-Specific Language*, por sus siglas en inglés), definición de modelos independientes de plataforma e interpretación en tiempo de ejecución, permite semi-automatizar el desarrollo de agentes conversacionales. Una de las ventajas que ofrece Xatkit es la inclusión de un DSL que brinda la posibilidad de extender su funcionalidad, de tal forma que es posible definir los requerimientos de nuevas plataformas para su implementación. Esto lo convierte en una herramienta adecuada para semi-automatizar el desarrollo de los agentes conversacionales con el potencial de utilizarlos en distintas áreas. Lamentablemente, Xatkit carece de funcionalidad para uno de los entornos para agentes conversacionales más utilizados en la actualidad, WhatsApp (<https://www.whatsapp.com/>). La aplicación de *chat* para teléfonos inteligentes, según el portal *statista*, tuvo dos mil millones

de usuarios activos en marzo del año 2020 [14]. Debido al amplio uso de este tipo de software, se ha vuelto común que las empresas ofrezcan servicio de atención y ventas al cliente de manera automatizada o semi-automatizada por medio de un agente conversacional [15]. A pesar de esto, existe una escasez de herramientas que faciliten la creación de agentes conversacionales para esta plataforma de mensajería instantánea.

1.3. Objetivo de la tesis

En este trabajo de investigación se planteó resolver los problemas mencionados anteriormente en la implementación de agentes conversacionales. Por lo cual, se definió y desarrollo un mecanismo de extensión para la plataforma de código libre basada en MDE llamada Xatkit [4]. La extensión permite el desarrollo semi-automático de *chatbots* para WhatsApp reduciendo el esfuerzo y costo de desarrollo en comparación con las técnicas de desarrollo tradicionales gracias a la generación de código a través de técnicas de MDE. Asimismo, debido a que ofrecer un flujo de conversación adecuado es importante para que el usuario obtenga la información que esta buscando cuando interactúa con el agente y a su vez mejore la UX, se implementó el modelo de lenguaje GPT-3 (*Generative Pre-trained Transformer 3*, por sus siglas en inglés) para proveer recomendaciones al usuario del agente conversacional. Esto permite dotar de capacidad de adaptación al *chatbot* creado con Xatkit, con lo cual, es posible comprender mejor a los usuarios al analizar sus sentimientos en las entradas de texto [16] [17].

Para validar la investigación desarrollada en esta tesis, se elaboró un caso de estudio que consistió en definir, desarrollar e implementar un agente conversacional en la plataforma WhatsApp. El *bot* es utilizado en el Posgrado en Ciencias de la Información (<https://pci.uas.edu.mx/pci/>) de la Universidad Autónoma de Sinaloa para brindar información referente a los planes de estudio, proceso de inscripción y titulación de los programas de maestría y doctorado.

1.4. Objetivo general

Reducir el nivel de complejidad en la implementación de agentes conversacionales utilizando el framework Xatkit, así como mejorar la experiencia de usuario para facilitar el proceso de información al usuario en aplicaciones de mensajería instantánea.

1.5. Objetivos específicos

- Llevar a cabo un mapeo sistemático de la literatura sobre tecnologías que automatizan el proceso de desarrollo de agentes conversacionales para conocer el estado actual de esta tecnología.
- Comprender el funcionamiento de los *frameworks* para desarrollo de agentes conversacionales que existen, seleccionar uno y comprenderlo.
- Analizar y comprender el funcionamiento del *framework* Xatkit.
- Estudiar cómo está estructurado un DSL para definir uno para su aplicación en la generación de *chatbot* para la plataforma de mensajería instantánea WhatsApp.
- Analizar y comprender el DSL de Xatkit y sus mecanismos de extensión.
- Definir una extensión del DSL de Xatkit para incluir la comunicación con WhatsApp.
- Definir una estructura general del prompt de GPT-3 para proveer recomendaciones al usuario.
- Extender la funcionalidad del *frameworks* Xatkit utilizando el lenguaje específico de dominio que este incluye para automatizar el desarrollo de *chatbots* en la plataforma WhatsApp y la implementación de las recomendaciones con GPT-3.
- Realizar una búsqueda de portales web de educación nacionales e internacionales que implementen *chatbots* para su análisis.
- Diseñar e implementar un *chatbot* utilizando la mejora planteada para el *frameworks* Xatkit.

- Analizar el desempeño del *chatbot* desarrollado.

1.6. Estructura del documento

A continuación, el **Capítulo 2** muestra la investigación realizada sobre el estado del arte en el cual se expone la situación actual en el tema de la automatización del desarrollo de agentes conversacionales.

El **Capítulo 3** abarca el marco teórico donde se reúnen los fundamentos teóricos necesarios en las disciplinas manejadas, esta sección contiene conceptos importantes del desarrollo de agentes conversacionales y las definiciones centrales de la Ingeniería Dirigida por Modelos.

El **Capítulo 4** comprende la contribución a la problemática planteada en la sección 1.2 de la tesis donde se presenta la propuesta para generación de agentes conversacionales con el uso del modelo de lenguaje autorregresivo GPT-3 y la automatización de implementación de *chatbots* en la plataforma de WhatsApp mediante un proceso de desarrollo MDE. Además, se muestra la implementación de la propuesta en un caso de estudio particular en el cual se evalúa un agente conversacional desarrollado utilizando la propuesta.

Finalmente, el **Capítulo 5** presenta las conclusiones y trabajo futuro de este trabajo de investigación.

Capítulo 2

Marco Teórico

En este apartado del capítulo se describirán conceptos que son útiles para la comprensión del desarrollo de agentes conversacionales, debido a que son el enfoque principal de este trabajo de investigación es importante conocer los elementos y estructuras que los conforman.

2.1. Fundamentos de la Ingeniería Dirigida por Modelos

La Ingeniería Dirigida por Modelos (*Model-Driven Engineering*, por sus siglas en inglés) es un método utilizado para la semi-automatización del desarrollo de software. La idea central de esta tesis radica en su implementación para *chatbots* por lo que comprender los conceptos básicos es necesario. En este apartado del capítulo se describirán los conceptos fundamentales de esta disciplina.

2.1.1. Modelo

Los modelos han sido utilizados desde la antigüedad, desde las primeras pinturas prehistóricas el hombre los empleaba para comunicar ideas y situaciones relevantes. Según [18], es una representación en un cierto medio de un objeto concepto o idea, el modelo cumple la función de ejemplificar o describir las características de un objeto, sistema o idea. En [3], se señala que los modelos implementan al menos dos roles mediante la aplicación de abstracción: la reducción de características que implica que solo se reflejan las

propiedades de interés del objeto y el mapeo de características que indica que se abstraen y generaliza a partir de un individuo original. En la Figura 2.1, se muestra un mapa del estado de Sinaloa, como un ejemplo fácil de entender de un modelo, en él se representan las características de interés para mostrar la información que necesitamos por medio de la abstracción. En este caso, se está usando un color diferente para resaltar las divisiones entre los municipios de Sinaloa, así se expresa la información de interés a otra persona que pueda observar la imagen.



Figura 2.1: Ejemplo de modelo: mapa del estado de Sinaloa.

Como es lógico, los modelos también pueden utilizarse en la creación de software, razón por la cual este es uno de los conceptos más importantes de los fundamentos de la Ingeniería Dirigida por Modelos pues este es el enfoque principal de MDD. En la Figura 2.2, se muestra un modelo de ejemplo (un diagrama de clases) creado con UML (*Unified Modeling Language*), el cual es un lenguaje de modelado gráfico que se utiliza para representar sistemas, en este se especifica toda la funcionalidad de los procesos. Es utilizado comúnmente como una herramienta de planeación y documentación en los

proyectos de desarrollo de software, si se utiliza una metodología MDD, los modelos serán utilizados directamente como artefacto principal en la generación de software. Hoy en día existen herramientas que permiten la transformación de modelos UML en código fuente de cualquier lenguaje de programación.

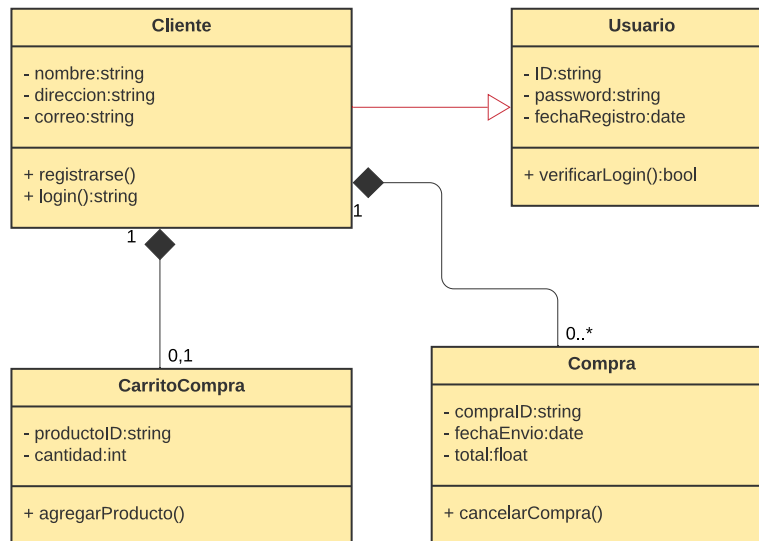


Figura 2.2: Ejemplo diagrama de clases UML.

2.1.2. Metamodelo

Un metamodelo, es la representación de un modelo (un modelo de otro modelo). El *Object Management Group* (OMG) [19], lo define como un tipo especial de modelo que especifica la sintaxis abstracta de un lenguaje de modelado. En este nivel de abstracción se especifican los conceptos, reglas y todos los componentes que serán utilizados para definir otros modelos. En el contexto del desarrollo de software, se pueden ver como todos los elementos que el desarrollador podrá utilizar en un modelo. Cuando se crea un metamodelo, se están definiendo los principios y estructuras que podrán ser utilizados en los modelos que se creen posteriormente.

En la Figura 2.3, se puede apreciar un metamodelo de un formulario web. En el cual se especifican todos los elementos que pueden conformarlo y como están relacionados, por ejemplo, el formulario puede contener de 0 a n entradas (*inputs*). El desarrollador utilizará este metamodelo para crear modelos de formularios, los cuales se convertirán en instancias de otros específicos. Posteriormente, una herramienta transformará estos en

un código que pueda interpretar la computadora, por ejemplo, se pueden transformar a Lenguaje de Marcado de Hipertexto (*HyperText Markup Language*, HTML) utilizando una plantilla.

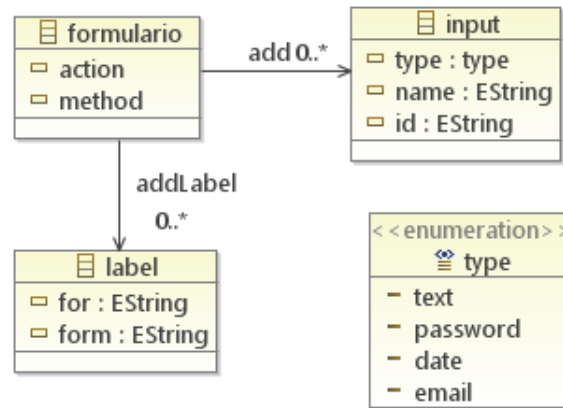


Figura 2.3: Metamodelo de un formulario.

En las Figuras 2.4, 2.5 y 2.6, se puede observar este proceso, primero se creó un metamodelo donde se describen todos los elementos que conforman un sitio web y sus relaciones. Posteriormente, se crea el modelo donde se utilizan los elementos definidos en el metamodelo para crear la web que se desea (este representa la instancia de un modelo de una página web). Finalmente, la herramienta utiliza una plantilla con el código de transformación que fue definida previamente para realizar la transformación del modelo en código HTML y así obtener un sitio web funcional. Esto se realizó utilizando el *plugin* de acceleo el cual fue diseñado para llevar a cabo transformaciones modelo a texto (*Model to Text*, M2T).

2.1.3. Domain-Specific Language

Un *Domain-Specific Language* (DSL) es un lenguaje de programación pensado para utilizarse en un ámbito en particular. A diferencia de los lenguajes de propósito general, (como C o Python), los lenguajes específicos de dominio se diseñan para resolver un problema específico o para utilizarse en situaciones determinadas. En [3], se definen como lenguajes que están diseñados específicamente para un determinado dominio, contexto o empresa para facilitar la tarea de las personas que necesitan describir cosas en ese domi-

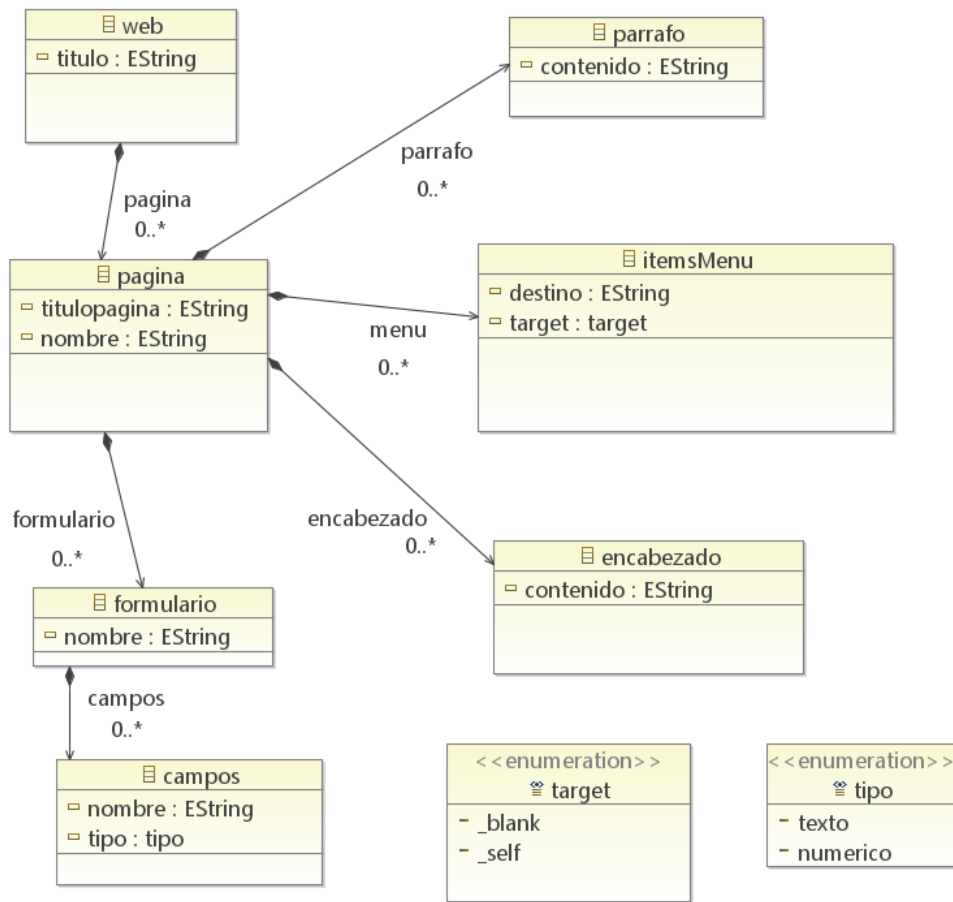


Figura 2.4: Metamodelo de un sitio web.

nio. De acuerdo con [20], un DSL proporciona expresiones predefinidas de un nivel de abstracción alto para representar conceptos que pertenecen al dominio de aplicación. Esto habilita a los expertos de dominio para entender, validar, modificar e incluso desarrollar utilizando el lenguaje sin requerir que tengan conocimientos de programación.

Existen ejemplos de DSL, como HTML, el cual se usa para maquetación web, o SQL que se utiliza para administrar bases de datos, estos dos ejemplos fueron planteados teniendo en cuenta un objetivo muy específico y solo son útiles dentro del entorno para el cual fueron diseñados. Esto tiene sus ventajas, pues se está optimizando el desarrollo a una clase específica de problemas y el lenguaje es comprensible para los expertos del dominio.

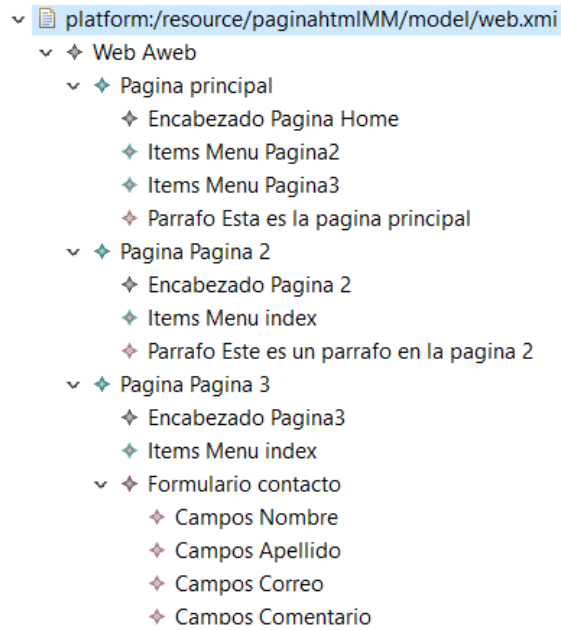


Figura 2.5: Modelo de un sitio web.

[index](#)

Pagina3

contacto

Nombre

Apellido

Correo

Comentario

Figura 2.6: Fragmento de sitio creado a partir de un modelo.

2.1.4. Eclipse Modeling Framework

El *Eclipse Modeling Framework* (EMF) es una herramienta que permite definir modelos basados en el metamodelo Ecore. El cual es la implementación del *metaobject facility* para poder definir DSL a través de metamodelos, los archivos Ecore son metamodelos que

se almacenan en archivos XML (*eXtensible Markup Language*) con la extensión .ecore. EMF es parte del proyecto de modelado Eclipse (*Eclipse Modeling Project*, EMP, por sus siglas en inglés). El cual contiene una gran cantidad de herramientas adecuadas para producir código y esto es realizado mediante los modelos Ecore y sus instancias [21]. EMF es de utilidad para crear programas utilizando modelos. Por otra parte, el *Graphical Modeling Framework* (GMF) puede generar editores gráficos para las instancias de programas definidos mediante los modelos Ecore. La Figura 2.4 muestra un modelo creado con Eclipse EMF.

2.1.5. Model-Driven Engineering

La Ingeniería Dirigida por Modelos (*Model-Driven Engineering*, MDE) es un paradigma de software en el que todas sus etapas del desarrollo y ciclo de vida son guiadas por modelos siendo esto el artefacto más importante. Esta metodología propone que el enfoque esta en los modelos, por el contrario de los paradigmas más tradicionales, en donde el enfoque suele ser el código, esto con el objetivo de reducir la complejidad del proceso de desarrollo de software al elevar el nivel de abstracción [22]. A través de los años, los investigadores de software siempre han buscado un nivel de abstracción mayor y así no tener que enfocarse en desarrollar software programando directamente el hardware y esto fue lo que llevó a la creación de los lenguajes de programación que conocemos hoy en día. Aún con los avances que se lograban, todavía era necesario concentrarse en el aspecto computacional antes que en el dominio del problema que se deseaba solucionar, MDE llegó como una propuesta que tiene el objetivo de solucionar esto [23].

MDE va más allá de la generación automática de código a partir de los modelos, es en este aspecto donde se puede observar su relación y diferencia con el Desarrollo Dirigido por Modelos. Los autores en [3], indican que MDE es un conjunto más amplio que abarca a MDD, pues al incluir a la ingeniería se señala que a diferencia de esta última, se incluyen no solo las actividades de desarrollo, sino el proceso completo de Ingeniería de Software.

En el aspecto empresarial, existen ventajas en aplicar MDE, en [24] se señalan: 1) elevar el nivel de abstracción, 2) integrar todas las vistas del sistema en un mismo entorno y 3) la incorporación de herramientas que gestionen la información del entorno de desarro-

llo. En [25], se considera que la ventaja principal de utilizar esta metodología, contrario a lo que puede pensarse comúnmente, no es la generación de código sino en el soporte que aporta para documentar una buena arquitectura de software.

2.1.6. Model-Driven Development

El Desarrollo Dirigido por Modelos (*Model-Driven Development*, MDD) es un paradigma de desarrollo de software en el que se utilizan los modelos como el artefacto principal del proceso de desarrollo. Consiste principalmente en representar el sistema por medio de los modelos (modelado) y aprovechar las ventajas que esto ofrece. Por el contrario de los paradigmas de programación tradicionales, usando estas técnicas se pueden crear sistemas y programas principalmente modelando y no codificando. Para lograr esto se utilizan herramientas que permiten transformar los modelos a código que pueda interpretar la computadora automáticamente [26].

Según [27], el objetivo principal del Desarrollo Dirigido por Modelos es minimizar el costo y esfuerzo de la producción de software. Para lograrlo, busca mejorar la calidad de las aplicaciones, independiente de las plataformas en donde se ejecuta (refiriéndose a esto al nivel de abstracción al que se llega en MDD en el que se separa la lógica del problema de las características tecnológicas). En [28], se plantea que la motivación detrás de utilizar MDD para mejorar la productividad es aumentar las ganancias que obtienen las empresas desarrolladoras de software tras cada proyecto y esto se logra a través de dos principales beneficios que se obtienen al utilizar este paradigma, en primer lugar aumenta el rendimiento de producción a corto plazo de los programadores al elevar el valor que tiene cada artefacto de software creado. En segundo lugar, también se mejora la productividad a largo plazo, pues se incrementa la vida útil de estos artefactos primarios al reducir la velocidad en la que se vuelven obsoletos. Esto se debe a que se están utilizando los modelos para representar el problema del mundo real y pueden ser re-utilizados independientemente de su soporte tecnológico. Con esto se amplía así las posibilidades de cada pieza de software debido a que se desliga a la solución de su plataforma de desarrollo y/o despliegue volviendo posible migrar o exportar la herramienta a otra plataforma con mayor facilidad si se desea.

Es importante mencionar que es necesario considerar que se requiere que los desarrolladores adopten esta metodología. Esto conlleva un proceso de aprendizaje para lograr modelar los sistemas, además es imprescindible utilizar herramientas que permitan modelar y realizar las conversiones de modelo a código, dos aspectos que son considerados una desventaja de este paradigma a nivel empresarial. Hoy en día existe una variedad de herramientas para MDD y como se ha mencionado anteriormente, las ventajas que ofrece el uso de este paradigma lo convierten en una alternativa útil que justifica el tiempo que puede requerir la capacitación del personal.

2.1.7. Model-Driven Architecture

La arquitectura dirigida por modelos (*Model-Driven Architecture*, MDA) es un procedimiento específico para implementar MDE, propuesto por el *Object Management Group* (OMG). Es un marco de trabajo que proporciona las pautas que deben seguir los desarrolladores de software durante todo el proceso del desarrollo de software a partir de modelos.

De acuerdo con [29], MDA es un cambio fundamental de un diseño orientado por objetos hacia uno dirigido por modelos, el cual fue propuesto por el OMG. Además, señala que la idea central de esta metodología consiste en separar el modelo independiente de la plataforma, el cual no está relacionado con la tecnología de implementación y solo contiene la especificación de la lógica del negocio. Posteriormente, se definen las reglas de conversión de acuerdo a las diferentes tecnologías de implementación con el objetivo de poder convertir el modelo independiente de plataforma a un modelo específico, para finalmente convertir dicho modelo en código. Esta es una de las ideas en las que se fundamenta MDA, separar el desarrollo en 3 modelos: el Modelo Independiente de Computación (*Computation Independent Model*, CIM), el Modelo Independiente de la Plataforma (*Platform Independent Model*, PIM) y los Modelos Específicos de la Plataforma (*Platform Specific Model*, PSM). Estos modelos conforman todo el proceso de desarrollo (MDD) antes de la generación de código, la cual se hace a partir del PSM. En [30], se indica que esto conlleva ventajas en la separación de responsabilidades, productividad, portabilidad, el mantenimiento y la documentación. En la Figura 2.7, se puede

observar el proceso de desarrollo descrito, en la izquierda se muestra la etapa correspondiente a ingeniería de software y en la derecha se muestra los productos obtenidos de cada etapa.

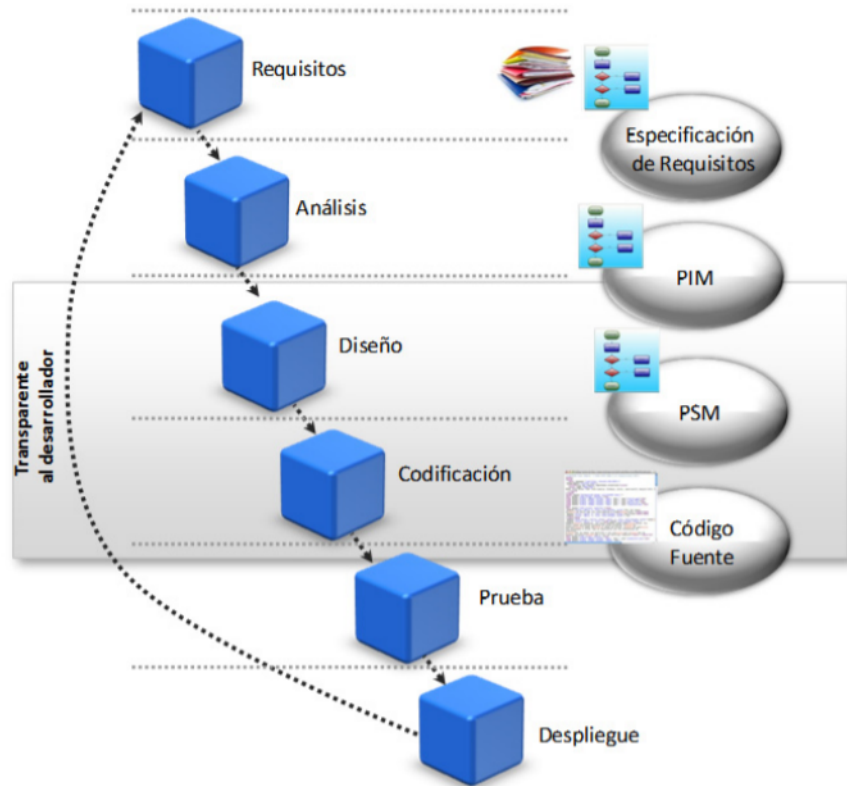


Figura 2.7: Proceso MDA.

Modelo independiente de la computación

El *Computation Independent Model* (CIM) es el modelo donde se especifican los elementos para resolver un problema sin incluir ningún detalle del sistema de computo. En [3] se define como el nivel de modelado más abstracto, que representa el contexto, los requisitos y el propósito de la solución sin ningún vínculo con las implicaciones computacionales. Al utilizar el MDDD es necesario especificar el CIM donde se modela la lógica de negocio de la solución que se desea obtener sin preocuparse por la participación tecnológica.

Modelo independiente de la plataforma

El *Platform Independent Model* (PIM) es el modelo creado a partir del CIM donde se incluye la parte computacional y algorítmica, pero se excluyen los detalles de la plataforma objetivo. El OMG [19] lo define como un modelo de un subsistema que no contiene información específica de la plataforma que se utiliza para realizarlo. La calidad que tiene el modelo es independiente de las características de una plataforma de cualquier tipo en particular. Es posible migrar o adaptar una misma solución a diferentes plataformas a partir de este modelo, debido a que se pueden especificar los detalles de la plataforma objetivo sin modificar el PIM.

Modelo específico de la plataforma

El *Platform Specific Model* (PSM) es el último nivel de abstracción de la arquitectura MDA, donde finalmente se especifican los detalles particulares de la implementación de la plataforma objetivo. En [18], es definido como la vista del sistema desde un punto de vista específico de la plataforma. En esta etapa, los desarrolladores deben definir todas las características y elementos necesarios para poder generar código que sea capaz de ejecutarse.

2.2. Fundamentos de los Agentes Conversacionales

Desde hace años se ha considerado la idea de comunicarse con una computadora, incluso desde el año 1950 Alan Turing ya planteaba varios conceptos similares a partir de abordar la pregunta “¿pueden las maquinas pensar?”. Incluso, proponía la prueba de Turing para evaluar si una máquina tenía la capacidad de mostrar un comportamiento inteligente, similar o indistinguible al de un ser humano [31]. En este apartado se introducen los conceptos fundamentales para comprender el desarrollo de agentes conversacionales.

2.2.1. Definición de Agente Conversacional

Un agente conversacional, también llamado *chatbot*, es un programa que simula una conversación, pueden comunicarse con el usuario a través de distintos medios como la

voz o el texto. En [32] son descritos como un sistema que puede interactuar con usuarios humanos con lenguaje natural. Como señala [33], es un software que imita las conversaciones humanas utilizando inteligencia artificial y está diseñado para distintos fines (asistente virtual, entretenimiento, automatización de tareas, proveedor de información, etc.).

Para el año 1966, fue presentado el que es considerado el primer agente conversacional llamado ELIZA [34] que permitía la comunicación por medio del lenguaje natural a través de analizar las entradas descomponiendo las frases en base a ciertas reglas, obteniendo así las intenciones a través de palabras clave y generando respuestas a partir de esto. A través de los años surgido una gran cantidad de tecnologías y técnicas para diseñar, desarrollar e implementar agentes conversacionales con mayor eficiencia, tal es el caso de AIML (*Artificial Intelligent Markup Language*), el cual es un lenguaje de marcado diseñado exclusivamente para crear agentes conversacionales [35]. Esta tecnología se utilizó para crear la interfaz *chatbot* llamada ALICE (*Artificial Linguistic Internet Computer Entity*) [36], desde entonces su uso se ha extendido para crear una amplia variedad de este tipo de agentes.

Los *chatbots* suelen ser clasificados en base a sus capacidades, en [37] se mencionan dos tipos de agentes conversacionales los cuales son catalogados según su nivel de complejidad en *stateless* o *state-full*. En la primer categoría se encuentran aquellas interfaces de diálogo que interactúan con el usuario como si fuera la primera vez que son utilizados sin importar que se trate de una interacción posterior a un número de sesiones ya completadas. En la segunda categoría se encuentran aquellos con la distinción de *state-full* pueden recordar interacciones anteriores con un usuario específico lo cual mejora la personalización y conlleva una evolución en la calidad de la conversación permitiendo crear nuevas respuestas en base al contexto. También existen otras clasificaciones, como distinguirlos por su dominio de conocimiento si, el agente es capaz de acceder a datos de carácter general y fue entrenado para hablar sobre cualquier tema se trata de un *chatbot* de dominio abierto, por el contrario si fue entrenado para tener una base de conocimiento específica de un tema se considera de dominio cerrado o dominio específico [38].

2.2.2. *Intent* / Intención

En el contexto de un *chatbot* se le llama *intent* a la intención o la información que el usuario desea transmitir en una expresión, un *intent* es un conjunto de expresiones que tienen el mismo objetivo. En [39], se señala que todo *chatbot* requiere un componente de reconocimiento de intención para comprender la meta u objetivo del usuario. El *bot* debe clasificar la expresión del usuario final en una de las intenciones predefinidas. Reconocer el *intent* dentro del *input* es indispensable, pues de esta forma se reconoce la motivación del usuario al introducir una frase o comando, un ejemplo de esto, si el usuario introduce el texto “necesito la dirección de los hospitales en Sinaloa” el *intent* debería ser detectado como “dirección de hospitales” con lo cual el *bot* entendería que el usuario esta haciendo una consulta en la que quiere obtener la dirección de los hospitales, con lo cual podrá proceder a realizar la acción correcta en dicha situación.

2.2.3. *Entity* / Entidad

Una entidad, es un elemento de información contenido en el intercambio de frases entre *chatbot* y usuario. Anteriormente, se observó que en el ejemplo de “necesito la dirección de los hospitales en Sinaloa”, el *intent* debería ser detectado como “dirección de hospitales”, sin embargo, el *bot* también debe ser capaz de reconocer las entidades de interés, en este caso “hospital” y “Sinaloa”, con esto se obtiene la información complementaria necesaria para poder llevar a cabo la consulta solicitada.

2.3. Procesamiento de Lenguaje Natural

El procesamiento del lenguaje natural (*Natural Language Processing*, NLP) es un campo de las ciencias de la computación que se ocupa de investigar la comunicación entre máquinas y personas mediante el uso de lenguas naturales como por ejemplo el español, el inglés o el chino. En [40], se define como una tecnología que permite a las computadoras comprender los lenguajes humanos. Para concretar este objetivo se necesita llevar a cabo un proceso en el que las lenguas naturales que usan las personas sean transformadas a un sistema de codificación que pueda ser interpretado por una máquina, posteriormente

esta información es tratada por el computador y finalmente se concreta una acción la cual puede ser emitir una respuesta en el mismo lenguaje natural. Esto tiene muchas aplicaciones, como por ejemplo, el reconocimiento de voz, minería de datos basada en texto y generación de texto o voz, actualmente se ha vuelto un tema de interés muy importante [41].

El procesamiento de lenguaje natural es un tema que se estudia desde hace años, los primeros proyectos donde se aplicó la lingüística en la computación se remontan a finales de la década de 1940 en donde se trabajó en traducción automática. Los primeros sistemas de traducción eran simples y se basaban en remplazar las palabras a un equivalente en otro idioma lo cual provocaba resultados negativos, con el tiempo estos sistemas se han mejorado tomando en cuenta nuevas gramáticas formales. Hoy en día, se hace énfasis en los aspectos semánticos de la entrada de texto (*input*), la cual se quiere traducir en lugar de solo enfocarse en la sintaxis, teniendo esto en cuenta, en la actualidad se utilizan técnicas de base de conocimiento y técnicas de inteligencia artificial. NLP va más allá de la traducción automática pues necesita comprender completamente la entrada y utilizar conocimiento del mundo real para entender el objetivo del usuario [42].

Para poder procesar el lenguaje natural utilizando una computadora es necesario convertir el texto, voz o cualquier tipo de información que sea transmitida a la máquina a una codificación numérica que esta pueda tratar. En la actualidad, se utilizan diferentes tipos de codificación, por ejemplo, la asignación de valores numéricos a cada palabra, la separación de frases en subpalabras a las cuales se les da un valor, o incluso codificar cada carácter por separado. Cada técnica posee ventajas y desventajas. En la Figura 2.8 se ejemplifica una de estas técnicas que es utilizada actualmente la cual es llamada *one-hot encoding*. Usando esta representación, se codifica cada palabra en un vector numérico, este tendrá tantas posiciones como palabras existan en nuestro vocabulario donde cada posición de este vector representa a cada una de las palabras y se asignará un 1 en la posición de la palabra que se esté representando mientras que todos los otros valores tomarán el valor de 0. Esta es una técnica popular en la actualidad, su utilidad radica en el hecho de inicializar todas las palabras del vocabulario con la misma distancia entre sí, debido a que por cada palabra, se crea una nueva dimensión, esto no ocurriría si se utilizara un proceso de conversión de codificación numérica para cada palabra. Esto es conveniente si se pre-

tende emplear un procedimiento de *machine learning*, ya que usando una representación vectorial se previene de proveer con información falsa a los modelos de aprendizaje. Una vez se codifica el lenguaje natural, existen una gran cantidad de técnicas que son utilizadas con el objetivo de comprender estos datos, debido a que para lograr una real comprensión del lenguaje natural es necesario que el sistema sea capaz de realmente entender la intención y sentimientos de los *inputs* del usuario. Pasada esta etapa, si se requiere de una salida, el sistema debe generar lenguaje natural con la finalidad de comunicar la información al usuario. Estas dos etapas o ramas del NLP son llamadas comprensión de lenguaje natural (*Natural Language Understanding*, NLU) y generación de lenguaje natural (*Natural Language Generation*, NLG) las cuales son discutidas en las secciones siguientes. La relación entre las tres disciplinas se puede observar en la Figura 2.9, donde NLU y NLG son ramas de NLP y juntas suelen llevar un proceso en el que el usuario introduce un *input* y este pasa por la etapa de codificación, procesamiento, entendimiento y finalmente la generación de una respuesta en lenguaje natural lo que producirá una salida.

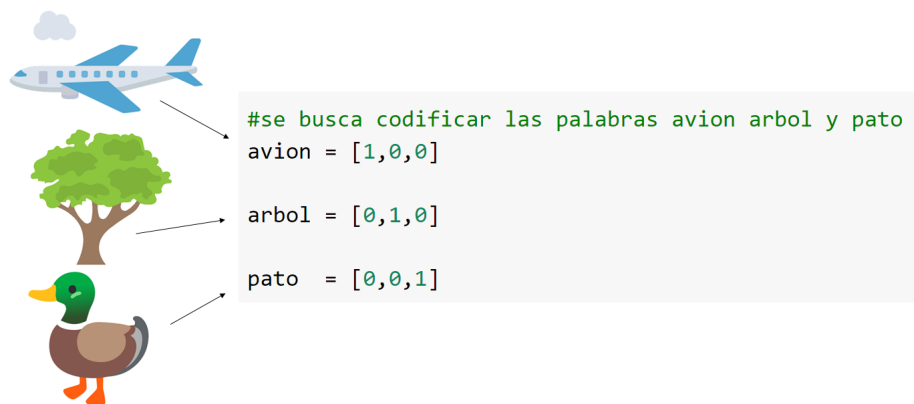


Figura 2.8: Ejemplo *one-hot encoding*.

2.3.1. Comprensión de Lenguaje Natural

El lenguaje natural es la forma de comunicación que usamos los humanos habitualmente, procesar este lenguaje para que se convierta en información útil con la finalidad de que un software la utilice es indispensable en el desarrollo de *chatbots*. En [43], se describe a la comprensión de lenguaje natural (*Natural Language Processing*, NLP) como un campo del procesamiento del lenguaje natural (*Natural Language Understanding*,

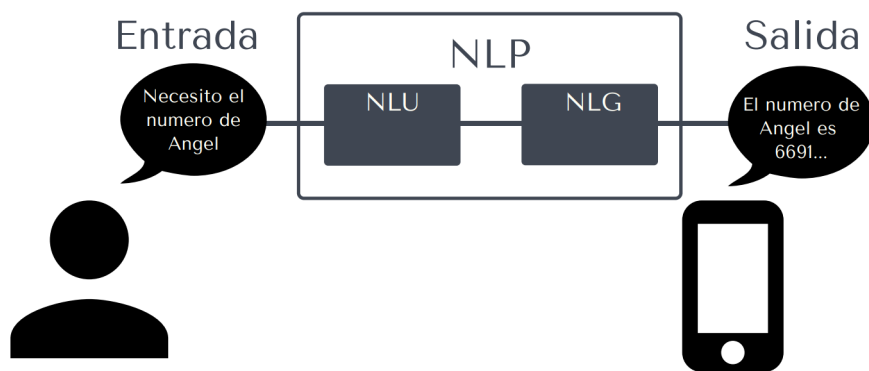


Figura 2.9: Comunicación humano-maquina utilizando el procesamiento de lenguaje natural.

NLU) que se ocupa de la conversión del lenguaje natural en una representación semántica que una computadora puede interpretar. Continuando con el ejemplo de *one-hot encoding* (Figura 2.8) esta técnica es útil para codificar las palabras e inicializarlas con la misma distancia entre ellas, después de realizar la codificación ahora se necesita procesar estos vectores de forma que se obtenga otra representación que incluya la información semántica de cada palabra, este proceso se conoce como *word embedding*, el cual corresponde a una técnica de procesamiento de lenguaje natural que consiste, básicamente, en asignar un vector a cada palabra sin embargo este vector guarda información semántica, lo que permite que pueda ser relacionado con otros vectores según distintos contextos gramaticales. Usando esta técnica los vectores utilizados, por ejemplo los de *one hot encoding* se compactarán a un número de características deseadas; actualmente existen diferentes herramientas para realizar este proceso con gran aceptación por los usuarios como por ejemplo *Word2Vec* y *Glove* [44]. Una vez que se convierten estos datos no estructurados en un conjunto de datos estructurados, NLU se encarga de analizar esta información para permitir extraer las intenciones, sentimientos entidades u objetivos provenientes de la entrada.

2.3.2. Generación de Lenguaje Natural

Para poder llevar a cabo una conversación, el agente debe ser capaz no solo de entender al usuario, también debe estar preparado para expresarse con un método que le resulte comprensible al ser humano de forma nativa. Los autores en [45], definen la generación de lenguaje natural (*Natural Language Generation*, NLG) como la tarea de procesamiento del para generar más lenguaje natural a partir de un sistema de representación de máquina con una base de conocimientos, utilizando una forma lógica. Se podría decir que un sistema NLG es como un traductor que convierte datos en una representación en lenguaje natural. El objetivo es capturar el significado del texto y luego crear un procedimiento de transmisión de información que una persona pueda entender y usar para interactuar con un sistema. Además, el sistema NLG debería poder aclarar el significado del texto utilizando un lenguaje natural. Este NLP es una parte importante del sistema NLG.

Los sistemas NLG se utilizan para diferentes tareas, como creación y moderación de contenido, traducción y resumen de información, comunicación de datos, etc. En la actualidad, son diversos métodos los que se utilizan con el fin de generar lenguaje natural como el uso de *templates* (o plantillas) para adaptar los datos y comunicarlos al usuario con palabras amigables. El uso de NLG a partir de modelos que son capaces de aprender de un conjunto de datos son más usados cada día lo que permite que el lenguaje se adapte y aprenda a partir de lenguaje natural real. Uno de los sistemas de generación de lenguaje natural que ha surgido en la actualidad es *Generative Pre-trained Transformers*, GPT-3, el cual es un modelo pre entrenado que utiliza *Deep Learning* para producir texto de la forma en la que la haría un humano. Este sistema ha demostrado la utilidad que pueden tener estas herramientas de NLG al emplear aprendizaje profundo en su campo.

2.3.3. Transformer

Se le llama *transformer* a un tipo de arquitectura de redes neuronales. Actualmente, es ampliamente utilizada en modelos de lenguaje; se caracteriza principalmente por utilizar una codificación posicional, en la cual se le asigna un número a cada componente en una cadena de caracteres representando su posición dentro de la oración, a diferencia de las alternativas que eran utilizada anteriormente con redes neuronales recurrentes en la cual el

texto es analizado secuencialmente [46]. Esta característica permite paralelizar de forma eficiente el entrenamiento, lo que posibilita que sean utilizadas en grandes modelos con conjuntos de entrenamiento que se encuentran en el orden de los terabytes. Esta arquitectura introdujo el concepto de *textitself-attention*, que permitió que los modelos crearan una representación interna del lenguaje después de analizar una gran cantidad de datos facultando a estas herramientas desempeñar una gran cantidad de tareas [46]. En la actualidad, existen herramientas que utilizan esta arquitectura que han conseguido un gran éxito al lograr desempeñar diferentes tareas de procesamiento de lenguaje. Sus modelos han sido entrenados con millones de parámetros tales como *Bidirectional Encoder Representations from Transformers* (BERT) [47] o *Generative Pre-trained Transformer 3* (GPT-3) [48] e incluso existen alternativas de código abierto, por ejemplo, GPT-NeoX-20B [49].

2.4. Twilio

Twilio es un servicio de comunicación en la nube que permite enviar y recibir mensajes de texto, llamadas de voz y video, y correos electrónicos a través de una API fácil de usar [50], [51]. Un aspecto importante de Twilio es su capacidad para interactuar con la popular aplicación de mensajería, WhatsApp.

La plataforma de Twilio es fácil de integrar en los sistemas existentes y ofrece una variedad de opciones de personalización para que las empresas puedan adaptar la experiencia del cliente a sus necesidades. Además, Twilio ofrece una amplia gama de herramientas de análisis y seguimiento, lo que permite a las empresas rastrear y medir el impacto de sus interacciones con los clientes.

La plataforma cuenta con una infraestructura de alta calidad y una amplia variedad de herramientas y características. Además, Twilio cuenta con un soporte técnico y una comunidad de desarrolladores activa que puede ayudar a los negocios y empresas a sacar provecho de la plataforma.

2.5. Xatkit Chat Platform

Actualmente, han emergido distintas plataformas para el desarrollo de agentes conversacionales, como *dialogflow* [52], la plataforma de Google que ofrece un proveedor de lenguaje natural y herramientas para diseñar interfaz de usuario de conversación.

Xatkit es un *framework* de código abierto que utiliza el lenguaje Java y técnicas del MDD como los lenguajes específicos de dominio, definiciones independientes de la plataforma e interpretación en tiempo de ejecución. Su objetivo es disminuir los problemas relacionados a la complejidad en el desarrollo de agentes conversacionales en el entorno empresarial. Los proyectos creados en esta plataforma, se benefician si el equipo de desarrollo se encuentra con problemas como lo son la necesidad de una comprensión profunda de las plataformas objetivo y otras tecnologías o conocimientos requeridos, o si existe una dificultad para adaptarse a las necesidades de las compañías en alguna etapa del desarrollo (por ejemplo debido a la gran dificultad de cambiar de proveedor de NLP), etc. Esta herramienta esta diseñada para minimizar estos inconvenientes.

Xatkit provee un conjunto de lenguajes específicos de dominio (Sección 2.1.3 de la tesis) para definir agentes conversacionales de forma independiente de la plataforma. Además, permite la implementación automática de la aplicación de *chatbots* y administra la lógica de conversación definida sobre las plataformas seleccionadas, permitiendo cambiarlas en cualquier momento sin necesidad de alterar el código correspondiente a la lógica de la conversación. Esta herramienta tiene una arquitectura modular para facilitar la evolución de los componentes por separado.

En comparación con otras herramientas de desarrollo de agentes conversacionales como por ejemplo Dialogflow, Xatkit ofrece una mayor flexibilidad en el diseño y la implementación de los agentes conversacionales. En lugar de estar limitado a las herramientas proporcionadas por una plataforma específica, esta herramienta permite adaptarse mejor a las necesidades específicas de un proyecto de desarrollo. Además, Xatkit es un *framework* de código abierto, lo que significa que el usuario tiene acceso completo al código fuente y puede personalizar y modificar la plataforma según sus necesidades. Adicionalmente, Xatkit permite la implementación automática de la aplicación de *chatbot* y administra la lógica de conversación definida sobre las plataformas seleccionadas, lo que permite

cambiarlas en cualquier momento sin necesidad de alterar el código correspondiente a la lógica de la conversación. Esta característica es especialmente útil si se requiere cambiar de proveedor de NLP, ya que no se tendría que reescribir todo el código de la lógica de la conversación para adaptarse a la nueva plataforma, lo que significa que el usuario no está limitado a un solo proveedor de NLP (como sería el caso de simplemente utilizar DialogFlow).

Xatkit utiliza técnicas del Model-Driven Development (MDD) que permiten definir los agentes de forma independiente de la plataforma y con lenguajes específicos de dominio, lo que facilita su adaptación a las necesidades de las compañías en cualquier etapa del desarrollo. Por otro lado, Xatkit tiene una arquitectura modular que permite la evolución de los componentes por separado, lo que facilita la incorporación de nuevas funcionalidades y la actualización de las existentes. Esto puede ser especialmente beneficioso para proyectos que tienen como objetivo desarrollar agentes conversacionales más complejos y sofisticados.

En resumen, Xatkit se selecciona sobre herramientas como Dialogflow para este proyecto de investigación debido a su enfoque en la personalización y la flexibilidad en el desarrollo de agentes conversacionales, su arquitectura modular y la facilidad para cambiar de plataforma sin alterar la lógica de la conversación.

2.5.1. Análisis del lenguaje de dominio específico de Xatkit Chat Platform

Actualmente, Xatkit utiliza un DSL interno, lo cual significa que está contenido dentro de un lenguaje huésped de propósito general. Esto conlleva ciertos beneficios para los desarrolladores, puesto que no es necesario aprender un nuevo lenguaje para utilizar este *framework* debido a que se utiliza Java, el cual es ampliamente utilizado por una gran cantidad de programadores. Además, esto permite la posibilidad de utilizar las herramientas disponibles para Java mientras se crean, depuran o prueban *chatbots*, incluso se habilita la oportunidad, en caso de ser necesario, el uso de librerías ya existentes para este lenguaje.

Es de suma importancia conocer el funcionamiento del DSL de Xatkit para desarrollar los objetivos de esta investigación, es necesario comprender sus mecanismos de extensión

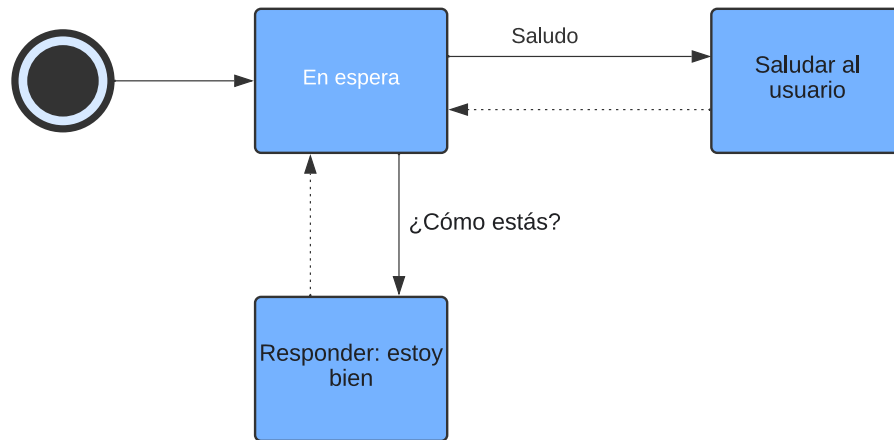


Figura 2.10: Ejemplo de diagrama de estados.

para que pueda ser implementado en nuevas plataformas. Esto permitirá reducir aún más la complejidad en el desarrollo de *chatbots* en la actualidad y añadir nuevas funciones que permitan mejorar la experiencia de los usuarios que utilicen los agentes conversacionales desarrollados. Por este motivo a continuación se describen sus características necesarias para definir *chatbots*.

El DSL de Xatkit está basado en la semántica de una máquina de estados para desarrollar agentes conversacionales, existen tres componentes básicos para crear un *chatbot* estos son: los *intents*, las plataformas y la definición de flujo de conversación y reacciones de *bot*. En primer lugar, es necesario diseñar la máquina de estados, a continuación se utilizará un ejemplo sencillo de un *bot* que responde a los saludos, en la Figura 2.10 se puede observar el diagrama del agente conversacional, en el se puede apreciar como el *chatbot* se traslada inmediatamente del estado inicial al estado de espera a alguna interacción que el usuario tenga con este, después existen dos caminos diferentes de conversación.

En el ejemplo se puede observar como se definen los estados que corresponden a los *intents*, a los cuales podemos añadir un conjunto de frases de entrenamiento para cada una, así como las transiciones entre estos, además se puede apreciar como se está definiendo la acción a realizar por el *chatbot* al entrar a “Saludo” y “¿Cómo estás?”, en este caso es enviar un mensaje a través de la plataforma React. Por último, se define un estado *default* en caso de no realizar la transición a ningún otro caso definido.

Además de definir los estados es posible añadir un conjunto de frases de entrenamiento para cada uno de los estados que corresponden a los *intents*, estos sirven para dotar de más

ejemplos al proveedor de procesamiento de lenguaje natural para reconocer los *intents* en más casos. Por último, es necesario que se definan las plataformas objetivo con las que se desea que el *chatbot* interactúe. Una vez se han definido todos los aspectos que conformaran al agente se puede crear el modelo del *bot* para ejecutarlo.

Es necesario comprender el funcionamiento principal del DSL de Xatkit para poder crear una extensión de este, ahora que se han expuesto los conceptos básicos de este lenguaje es preciso conocer las herramientas que este ofrece para definir nuevas plataformas, en el siguiente apartado se discuten los mecanismos de extensión del *framework* los cuales serán usados para definir nuevas mejoras que servirán para desarrollar las metas de esta investigación.

2.5.2. Mecanismos de extensión

Xatkit incluye un DSL en su *Platform package* para definir las capacidades de nuevas plataformas, se pueden definir las acciones que se pueden ejecutar y los eventos que esta puede emitir, dependiendo de los *ProviderDefinitions* que ofrece. Estas son las capacidades que se utilizan explícitamente en el modelo de ejecución para interactuar con ella, podemos ver el ejemplo de la Figura 2.11.

En primera instancia, una plataforma debe ser definida con un atributo de nombre y una ruta usada por el Xatkit Runtime (el motor de ejecución basado en eventos que despliega y administra la ejecución del *chatbot*) para enlazar el modelo con la implementación de la plataforma. Además, a cada ruta le corresponde un conjunto de acciones que deben ser definidas. El metamodelo nos permite crear plataformas abstractas lo cual significa que no se incluye una implementación concreta para sus acciones debido a que se utiliza para representar una familia de plataformas similares. Esto permite definir *bots* de forma genérica. Las implementaciones de cada plataforma se conforman de proyectos Java independientes las cuales emplean la interfaz de la plataforma Xatkit esta es utilizada internamente por el componente Xatkit Runtime para cargarla dinámicamente, crear instancias de acción y ejecutarlas cuando se cumple una regla de ejecución. Cada una se compone de una clase Java principal que contiene los datos de acceso a sus acciones permitidas y un conjunto de clases que representan cada una de las acciones que pueden ser

utilizadas en esta. Las comunicaciones con las diferentes plataformas normalmente son manejadas a través de peticiones REST (*REpresentational State Transfer*).

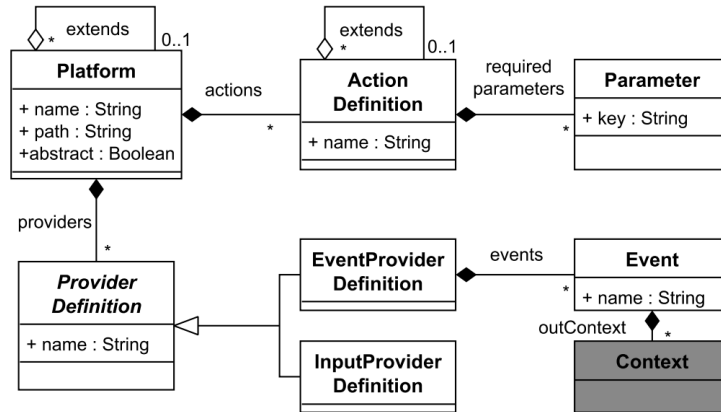


Figura 2.11: Metamodelo del *platform package*.

2.6. Modelo de lenguaje autorregresivo Generative Pre-trained Transformer 3

GPT-3 (*Generative Pre-trained Transformers*) es un modelo de lenguaje autorregresivo que utiliza el aprendizaje profundo para producir textos similares a los humanos. Es capaz de generar lenguaje natural sofisticado y completar tareas como clasificación, respuesta a preguntas y generar resúmenes [53], [48]. Fue creada por la compañía de investigación de inteligencia artificial OpenAI (<https://openai.com/>).

En este trabajo de investigación se experimentará el uso de este modelo para generar recomendaciones a los usuarios en base a sus preguntas al agente conversacional asistente de modo que se podrá explorar la utilidad de esta herramienta en este ámbito, para lograr este objetivo es necesario comprender el funcionamiento externo y los mecanismos de implementación de esta plataforma los cuales serán analizados en esta sección.

2.6.1. Funcionamiento

El uso principal de GPT-3 es la comprensión y generación de lenguaje natural. Por medio del API de OpenAI el usuario introduce un texto denominado (*prompt*) y como re-

sultado se producirá una respuesta que intente coincidir con las instrucciones o el contexto dado. Esta dinámica de entrada y salida, así como el uso de un conjunto de parámetros disponibles que modifican el comportamiento del modelo, es la manera por medio de la cual se “programa” esta herramienta. Debido a su importancia, este apartado se dedica a analizar el comportamiento y utilidad de los modelos y principales parámetros disponibles.

GPT-3 se divide en un conjunto de modelos de NLU (*Natural Language Understanding*) y NLG (*Natural Language Processing*) ordenados en función de su potencia y capacidades:

- text-davinci-001
- text-curie-001
- text-babbage-001
- text-ada-001

Siendo el primero, “Davinci”, el modelo más potente y Ada el más rápido difiriendo además en el costo por *token*, por lo que es decisión del desarrollador la selección del modelo. Los modelos inferiores pueden ser capaces de desempeñar algunas tareas con la misma competencia que los modelos más potentes, pero con un menor costo, económico y de computación. Esta tesis, se centrará en utilizar los modelos Davinci y Curie debido que son los de mayor potencia, sin embargo, se puede experimentar en diferentes casos con modelos mas baratos la posibilidad de obtener un buen rendimiento.

Los modelos de GPT-3 procesan el texto por medio de una conversión de texto a *tokens* en donde una palabra puede ser dividida en sub-palabras, por lo que las palabras mas grandes generalmente son divididas en una cantidad mayor de *tokens* que las más pequeñas. La documentación de OpenAI señala que 1 *token* tiene aproximadamente 4 caracteres o 0.75 palabras para texto en inglés. Además, se debe tomar en cuenta que la cantidad de *tokens* contenidos en la entrada de texto (la cual es llamada *prompt*), más aquellos generados a partir de esta, no deben de superar la longitud máxima de contexto del modelo (la cual para la mayoría de los casos es de 2048 *tokens*, o alrededor de 1500

palabras). Es importante tomar esto en consideración para establecer una longitud máxima en el *prompt* y en el texto generado el cual se puede limitar por medio de un parámetro *Response length*. Otra alternativa útil para controlar la longitud de la respuesta es utilizar *Stop Sequences*, este concepto permite definir un conjunto de secuencias que forzarán la detención de texto una vez se genere alguna, lo que permite controlar la salida de texto.

La generación de texto de GPT-3 puede variar en cada interacción aún si se proporciona el mismo *prompt*, esto se debe a que la API es estocástica por defecto. Esto se puede controlar por medio de un parámetro de temperatura *Temperature*, el cual controla la capacidad de exploración y aleatoriedad del texto generado, mientras el parámetro de temperatura se aproxima a 0, el modelo se volverá determinista y repetitivo, por lo que es de interés hacer un buen manejo de este parámetro en dependencia si se quiere obtener respuestas mas variadas y creativas u obtener resultados similares en caso de emplear entradas idénticas.

Es importante diseñar una estructura de *prompt* adecuada para obtener respuestas satisfactorias debido a que esta entrada es la información principal a partir de la cual el modelo genera el texto. Existen tres pautas básicas para crear *prompts* que produzcan texto más relevante, estas son:

- Exponer claramente el comportamiento deseado, esto se realiza mediante instrucciones explicitas, ejemplos o combinaciones de ambos
- Proporcionar datos de calidad, se debe proveer suficientes ejemplos que sean adecuados, además si se desea que las respuestas sean formales y con una buena gramática se debe procurar que los ejemplos proporcionados también tengan estas características.
- Proveer una configuración adecuada, los parámetros como la temperatura y longitud de respuesta, influyen directamente en la respuesta del modelo.

A continuación, se exponen los mecanismos de implementación que provee OpenAI para interactuar con los modelos de GPT-3.

2.6.2. Mecanismos de implementación

OpenAI ofrece dos formas de interactuar con GPT-3, la primera es el *playground*, el cual es una interfaz diseñada para usar fácilmente los modelos; este permite modificar fácilmente todos sus parámetros. Su función principal es poder experimentar configuraciones rápidamente con la herramienta. La segunda alternativa, es el uso de un API, la cual se puede utilizar para comunicar directamente cualquier aplicación web con la plataforma, este será el medio por el cual se implementará la comunicación del *chatbot* con GPT-3. El API de OpenAI puede ser consumido por medio de URL, para lo cual es necesario proveer dos parámetros para comunicarse con el servicio, una clave de organización y la clave de API. Ambos son obtenidos a través del portal de OpenAI

2.7. Comentarios Finales

Para cumplir con los objetivos de esta investigación fue fundamental comprender a profundidad la tecnología y conceptos presentados en este capítulo. En primer lugar, se exploraron los conceptos clave del desarrollo de agentes conversacionales, los cuales incluyen el agente conversacional en sí mismo, el procesamiento de lenguaje natural, la comprensión de lenguaje natural, la intención y la entidad y la generación de lenguaje natural. Estos conceptos son esenciales para comprender cómo funcionan los agentes conversacionales y cómo se pueden construir y mejorar. Posteriormente, se abordaron los fundamentos de la ingeniería dirigida por modelos, que es una técnica de desarrollo de software que se enfoca en la creación y uso de modelos para construir sistemas de software. Dentro de este marco, se discutieron conceptos como Model-Driven Development, Model-Driven Engineering, modelos y metamodelos, el Eclipse Modeling Framework, Model-Driven Architecture, Domain-Specific Language y los modelos independientes de la computación y de la plataforma. Todos estos conceptos son relevantes para entender cómo se puede utilizar la ingeniería dirigida por modelos para el desarrollo de agentes conversacionales. Después, se analizó la Xatkit Chat Platform, que es una plataforma de código abierto para la creación de agentes conversacionales. Se examinó su lenguaje de dominio específico y sus mecanismos de extensión, lo que permitió entender cómo

se puede utilizar esta plataforma para desarrollar agentes conversacionales personalizados además de comenzar por comprender su alcance, ventajas y limitaciones. Por último, se profundizó en la comprensión del modelo de lenguaje autorregresivo Generative Pre-trained Transformer 3 (GPT-3), que es uno de los modelos de procesamiento de lenguaje natural más avanzados y potentes disponibles en la actualidad. Se explicó su funcionamiento y los mecanismos de implementación, lo que permitió entender cómo se puede utilizar este modelo para mejorar la capacidad de los agentes conversacionales de comprender y generar lenguaje natural.

Finalmente, este capítulo proporcionó una base teórica sólida para la comprensión del desarrollo de agentes conversacionales y las técnicas de ingeniería dirigida por modelos que se pueden utilizar para construirlos. También se presentaron tecnologías avanzadas como GPT-3, que pueden ser utilizadas para mejorar la capacidad de los agentes conversacionales. Estos conceptos y tecnologías son fundamentales para el desarrollo exitoso de este trabajo de investigación.

Capítulo 3

Estado del Arte

Este capítulo tiene como objetivo proporcionar el estado del arte del ámbito de las tecnologías que utilizan técnicas de Ingeniería Dirigida por Modelos para automatizar el proceso de desarrollo de agentes conversacionales o *chatbots* con el fin de aportar una mejor percepción de la investigación actual de esta área. Para lograr esta meta se condujo un mapeo sistemático de la literatura.

Este capítulo está organizado de la siguiente forma, la sección 3.1 titulada *Mapeo sistemático de la literatura*, introduce el protocolo de investigación y conducción llevado a cabo para obtener el estado del arte. En la sección 3.2, *Resultados del mapeo* se presentan los datos obtenidos del mapeo y el análisis producido de los estudios primarios. La discusión derivada de la investigación se presenta en la sección 3.3. Finalmente, en la sección 3.4, *Comentarios finales*, se presentan las consideraciones finales de este capítulo.

3.1. Mapeo Sistemático de la Literatura

Un mapeo sistemático de la literatura (SMS) es un estudio en el que se sintetiza la información útil de un tema dotándola de una estructura por medio de la clasificación de los reportes de investigación y resultados ofreciendo un resumen de las publicaciones disponibles [54]. Básicamente, es el proceso de identificar, categorizar y analizar la literatura existente que es relevante para un determinado tema de investigación.

En esta tesis se llevó a cabo un mapeo sistemático de la literatura para identificar la

tecnología y temática actual respecto a la automatización en la implementación de agentes conversacionales (*chatbots*). Además, se utilizó para conocer ventajas y desventajas que se obtienen en cada tecnología de implementación en comparación con los métodos de desarrollo tradicionales. Lo anterior, con el propósito de reconocer las áreas de oportunidad en este campo.

El procedimiento que fue implementado para conducir este SMS consta de 5 etapas: (1) definición de las preguntas de investigación, (2) búsqueda de estudios primarios, (3) selección de documentos aplicando criterios de inclusión y exclusión (4) definición del esquema de clasificación y (5) análisis de clasificación y mapeo sistemático. Cada una de ellas se detalla a continuación.

3.1.1. Preguntas de Investigación

Teniendo en cuenta los objetivos de esta tesis, se definieron un conjunto de preguntas de investigación las cuales servirán para extraer los datos de interés de las publicaciones al igual que clasificar y sintetizar la información disponible sobre las tecnologías que utilizan técnicas de Ingeniería Dirigida por Modelos para automatizar el proceso de desarrollo de agentes conversacionales en la literatura. Estas preguntas de investigación son descritas a continuación.

- Pregunta 1. ¿Qué temas de investigación en Ingeniería Dirigida por Modelos se están abordando actualmente en el dominio de los agentes conversacionales?
- Pregunta 2. ¿Cuáles son las tecnologías que implementan técnicas de Ingeniería Dirigida por Modelos en el desarrollo de agentes conversacionales?
- Pregunta 3. ¿Cuáles son los desafíos que existen al usar tecnologías que aplican alguna técnica de Ingeniería Dirigida por Modelos en la industria?
- Pregunta 4. ¿Cuáles son las bases de datos con mayor número de publicaciones en este tema?
- Pregunta 5. ¿Cuál ha sido la tendencia en los últimos cinco años en las publicaciones relacionadas con el Desarrollo de *chatbot* a través de MDE?

- Pregunta 6. ¿Cuáles son los países más activos en este tema?
- Pregunta 7. ¿Cuál es el tipo de publicación de las investigaciones encontradas?

A continuación, se muestra el proceso de búsqueda de los estudios primarios que posteriormente serán utilizados para contestar las preguntas de investigación.

3.1.2. Búsqueda de Estudios Primarios

Con el objetivo de contestar las preguntas de investigación se procedió con la búsqueda, para esto se elaboró una selección de palabras clave. A partir de estas se identificaron sinónimos y nombres alternativos para poder construir las cadenas de búsqueda adecuadas para utilizar en las fuentes de información, las palabras clave obtenidas fueron *model driven engineering*, *model driven development*, *MDE*, *MDD*, *chatbot*, *conversational agents*, *conversational system*, *automation*, *chatbot design*, *chatbot deployment*, *modeling*, *DSL*, *code generator*, *code generation*, *framework*, *tool*, *enviroment* resumidas en la Tabla 3.1, con las cuales se definieron las cadenas de búsqueda mostradas en la Tabla 3.2.

Tabla 3.1: Palabras clave para cadenas de búsqueda.

Concepto	Sinónimos / Términos relacionados
“Model driven engineering”	“Model driven”, MDE, “model driven development”, MDD, “Model Based”
Chatbot	“conversational agents”, “conversational system”, “bot”
Automation	Modeling, “chatbot design”, “code generator”, DSL, “chatbot deployment”, framework, tool, enviroment

Las bases de datos bibliográficas *IEEEExplorer*, *ACM Digital Library*, *Science Direct*, *Springer Link*, *Web of Science* y *Google Scholar* fueron seleccionadas para este mapeo sistemático debido a un análisis preliminar de las fuentes de estudios primarios accesibles en los que se obtuvieron resultados relacionados con el tema de interés. Los trabajos identificados como “literatura gris”, tales como tesis y estudios obtenidos de *OpenGray* y

Tabla 3.2: Cadenas de búsqueda para estudios primarios.

Base de datos	Cadena de búsqueda
IEEEExplorer	(“Full Text .AND. Metadata”:“Model driven engineering” OR “Full Text .AND. Metadata”: MDE OR “Full Text .AND. Metadata”:“model driven development” OR “Full Text .AND. Metadata”:MDD) AND (“Full Text .AND. Metadata”:Chatbot OR “Full Text .AND. Metadata”:“conversational agents” OR “Full Text .AND. Metadata”:“conversational system”) AND (“Full Text .AND. Metadata”: Automation OR “Full Text .AND. Metadata”:modeling OR “Full Text .AND. Metadata”:“chatbot design” OR “Full Text .AND. Metadata”:“code generator” OR “Full Text .AND. Metadata”: DSL OR “Full Text .AND. Metadata”:“chatbot deployment” OR “Full Text .AND. Metadata”:framework OR “Full Text .AND. Metadata”:tool OR “Full Text .AND. Metadata”:enviroment)
ACM Digital Library	(acmdlTitle:“Model-Driven Engineering” OR acmdlTitle: “MDE” OR acmdlTitle:“model driven development” OR acmdlTitle:“MDD.”OR acmdlTitle: “Model based”) AND (acmdlTitle:Chatbot OR acmdlTitle:“conversational agents” OR acmdlTitle:“conversational system”) AND (acmdlTitle: Automation OR acmdlTitle:modeling OR acmdlTitle:“chatbot design” OR acmdlTitle:“code generator” OR acmdlTitle:DSL OR acmdlTitle:“chatbot deployment” OR acmdlTitle:framework OR acmdlTitle:tool or acmdlTitle:enviroment)
Science Direct	(“Model driven engineering” OR MDE OR MDD) AND (Chatbot OR “conversational agents” OR “conversational system”) AND (Automation OR modeling OR framework)
Springer Link	(“Model driven engineering” OR MDE OR “model driven development” OR MDD) AND (Chatbot OR “conversational agents” OR “conversational system”) AND (Automation OR modeling OR “chatbot design” OR “code generator” OR DSL OR “chatbot deployment” OR framework OR tool or enviroment)
Google scholar	(“Model driven engineering” OR “Model driven” OR MDE OR “model driven development” OR MDD OR “Model Based”) AND (Chatbot OR “conversational agents” OR “conversational system”) AND (Automation OR modeling OR “chatbot design”

Teseo también fueron considerados como fuente de búsqueda. Posteriormente, se llevó a cabo la búsqueda de estudios primarios en las fuentes de datos seleccionadas. Se obtuvieron 429 artículos científicos, a los cuales se aplicaron los criterios de inclusión y exclusión (sección 3.1.3) para poder decidir su incorporación a este análisis.

3.1.3. Criterios de Inclusión y Exclusión

Se continuó con la definición de los criterios de inclusión, los cuales quedaron de la siguiente forma:

Criterios de inclusión: se incluyen estudios disponibles en español e inglés, pueden ser artículos de revistas científicas, conferencias y tesis de posgrado cuyo tema incluya alguna tecnología que hace uso de técnicas de Ingeniería Dirigida por Modelos o discuta su utilización en la semi-automatización del desarrollo de agentes conversacionales independientemente de su dominio. Además, el alcance no se limitará a ninguna ubicación de publicación específica.

Criterios de exclusión: se excluyen las publicaciones en un idioma distinto al inglés, en caso de existir dos artículos con contenido similar en el que uno sea un artículo de revista y otro artículo de conferencia se excluirá la versión de conferencia. Si un estudio idéntico se publicó en distintas fuentes por los mismos autores solo se incluirá el más reciente. Si un estudio primario se encuentra en un *keynote*, un tutorial, sección de póster o sesión de panel de discusión, no se tomará en cuenta para este mapeo sistemático de la literatura.

A continuación, se procedió a definir el proceso de selección, el cual constó de 4 etapas en las cuales se aplicaron los criterios de inclusión y exclusión para determinar la relevancia del estudio primario con el tema de interés con el fin de filtrar los resultados. Inicialmente se analizó el título y resumen del documento. Posteriormente se realizó un segundo filtro en el cual se incluyó al análisis la sección de conclusión del artículo, en el tercer filtro se descartaron las publicaciones con contenido idéntico para finalizar con una última etapa de selección en la que se analizó el documento completo.

Al finalizar el proceso de selección se obtuvo un total de 20 estudios primarios a partir de los 429 resultados iniciales (ver Tabla 3.3). Los estudios primarios se utilizaron para

contestar a las preguntas de investigación y realizar la clasificación de este mapeo.

Tabla 3.3: Filtrado de artículos aplicando criterios de inclusión y exclusión.

Base de datos	Total de resultados	Resultados finales
IEEEExplorer	35	4
ACM Digital Library	20	0
Science Direct	19	0
Springer Link	98	6
Google scholar	257	10
Total	429	20

3.1.4. Esquema de Clasificación

Se realizaron cuatro esquemas de clasificación para los estudios primarios. Esto permitió clarificar el tipo de estudios primarios encontrados en la búsqueda bibliográfica realizada en la sección 3.1.2.

El primer esquema se implementó para clasificar los estudios primarios por tipo de publicación científica, es decir, si es una publicación de revista, conferencia, revisión de literatura o tesis (ver Tabla 3.4).

Tabla 3.4: Clasificación por tipo de publicación científica.

Tipo de publicación científica	Descripción
Revista Científica	Trabajo de investigación que tiene la finalidad de exponer los resultados de una investigación científica, es publicado en revistas especializadas.
Conferencia	Artículo científico que tiene el objetivo de comunicar los resultados de una investigación realizada inicialmente para un congreso especializado.
Revisión	Investigación realizada a partir de analizar estudios primarios con un enfoque estadístico.
Tesis	Investigación realizada con el fin de obtener un grado académico.

El segundo criterio de clasificación, se diseñó con base en el trabajo presentado en [55] (ver Tabla 3.5), en el se evalúa el tipo y diseño del artículo científico. De tal forma

que con esto se obtiene información importante sobre cómo se están abordando las investigaciones en el tema de interés. Es importante mencionar al igual que lo indica el autor, no siempre es posible enmarcar en una sola clase una publicación pues en ocasiones las investigaciones tienen más de un objetivo que los lleva a entrar en más de una categoría.

Tabla 3.5: Tipo de artículo científico de investigación.

Tipo de artículo científico	Descripción
Investigación de evaluación	Una investigación sobre un problema técnica o solución en la práctica y muestra sus beneficios y desventajas la novedad de la investigación reside obligatoriamente en la técnica o solución sino en el criterio con la que el fenómeno está siendo evaluado.
Propuesta de solución	Una propuesta de solución para un problema la cual debe ser novedosa o incluir una mejora o extensión a una ya existente. Normalmente muestra los beneficios de la propuesta mediante algún recurso como por medio de ejemplos o argumentación.
Investigación de validación	Esta investigación muestra las características de una propuesta o solución que utilizando alguna metodología sin embargo esta todavía no ha sido implementada en la práctica.
Artículo filosófico	Artículos que plantean una nueva forma de abordar los temas como por ejemplo un nuevo marco conceptual.
Artículo de opinión	Estos artículos contienen la opinión del autor acerca de una solución y problema, sus beneficios, desventajas y su propuesta al respecto.
Artículo de experiencia personal	Artículos que contienen la opinión del autor basándose en su experiencia en trabajos previos, comúnmente se utilizan para compartir la experiencia de algún profesional de la industria o investigador que tiene conocimientos previos de un problema o herramientas en la práctica.

En el tercer esquema (ver Tabla 3.6), se utilizó para clasificar el tipo de contribución a nivel práctico. Esto con la finalidad de determinar en qué forma están surgiendo las contribuciones y herramientas a los usuarios de la comunidad en el desarrollo de *chatbots*. Este

esquema resulta particularmente importante debido a que las técnicas de automatización en el desarrollo comúnmente están destinadas a utilizarse en el ámbito empresarial.

Tabla 3.6: Clasificación por tipo de contribución.

Tipo de contribución	Descripción
Librería	Conjunto de funciones y/o procedimientos utilizados comúnmente para utilizarse dentro de un dominio específico.
SDK (Software Development Kit)	Conjunto de herramientas para el desarrollo de software orientados a una plataforma específica.
API. (Application Programming Interface)	Conjunto de funciones y procedimientos creadas con el objetivo de interactuar con una plataforma externa.
Framework	Entorno de desarrollo que cuenta con un conjunto de herramientas útiles en la creación de software que ahorran tiempo y esfuerzo para el desarrollador.
Arquitectura	Guía o forma de como se debe desarrollar software en base a ciertos lineamientos con el fin de tener una estructura general para futuros proyectos.

Al finalizar la definición y aplicación de los pasos anteriores se ejecutó la extracción de información de los 20 estudios primarios los cuales fueron clasificados utilizando los esquemas presentados anteriormente. Toda la información obtenida de este proceso fue empleada para responder a las preguntas de investigación y cumplir el objetivo de este estudio en el cual se generó nuevo conocimiento acerca de las tendencias en las publicaciones de las tecnologías que utilizan técnicas de Ingeniería Dirigida por Modelos para automatizar el proceso de desarrollo de agentes conversacionales. Esto localizar las oportunidades que existen en la investigación de este ámbito con mayor facilidad, así como ofrecer una perspectiva más amplia de los antecedentes en este tema.

3.2. Resultados del Mapeo Sistemático de la Literatura

En esta sección se presentan los resultados obtenidos de analizar la información extraída de los estudios primarios que contenían datos relevantes para contestar las preguntas

de investigación planteadas en esta tesis.

Pregunta 1. ¿Qué temas de investigación en Ingeniería Dirigida por Modelos se están abordando actualmente en el dominio de los agentes conversacionales?

La información que proveen los estudios seleccionados indica que la investigación actual en este tema se centra en la automatización del desarrollo. Para lograr este objetivo, una gran parte de los investigadores se han concentrado en crear sus propios metamodelos y lenguajes específicos de dominio. Esto para crear entornos de desarrollo robustos, que permitan construir agentes conversacionales reduciendo costo y esfuerzo.

Como se puede observar en la gráfica presentada en la Fig. 3.1, la mayoría de los artículos (76 %) son propuestas de solución que se han centrado en crear *frameworks* completos para que el usuario solo se centre en usarlo para crear sus *chatbots*. El 14 % son investigaciones de validación, es decir, propuestas que utilizan alguna metodología pero que no han sido implementadas fuera de la academia, particularmente se refiere a la automatización del proceso de migración de plataformas, lo cual se ha comprobado es posible mediante el uso de modelos independientes de plataforma (Xatkit, Conga). El 10 % restante del total de estudios primarios clasificados, son propuestas para la comunicación de componentes y herramientas externas utilizadas para automatizar el desarrollo o añadir nuevas funcionalidades (como por ejemplo el uso de APIs). En la Tabla (3.7) se detallan los estudios primarios clasificados por tipo de investigación publicada.

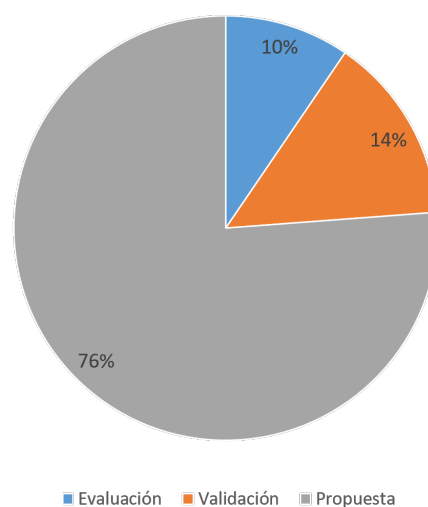


Figura 3.1: Porcentaje de artículos primarios por categoría de clasificación.

Tabla 3.7: Clasificación por tipo de investigación publicada en los estudios primarios.

Clasificación por tipo de investigación	Estudios primarios
Investigación de evaluación	[7], [56]
Propuesta de solución	[57], [4], [6], [8], [58], [59], [60], [61], [62], [63], [64], [17], [16], [65], [66] [67]
Investigación de validación	[68], [61], [69]

Pregunta 2. ¿Cuáles son las tecnologías que implementan técnicas de Ingeniería Dirigida por Modelos en el desarrollo de agentes conversacionales?

En los estudios seleccionados, se encontró una variedad de tecnologías que implementan Ingeniería Dirigida por Modelos en el desarrollo de *chatbots*. Como se había mencionado con anterioridad, se puede observar en la Figura 3.2 que la mayoría de los trabajos (55 %) se centraron en desarrollar un *framework* de desarrollo completo, con el cual le ofrecen al desarrollador todas las herramientas que necesita para implementar un agente conversacional. El 27 % de los estudios primarios proponen arquitecturas. El 14 % se enfocan en la comunicación a través del desarrollo de APIs y solo el 4 % en el desarrollo de librerías reutilizables.

Las propuestas tecnológicas que fueron encontradas en los estudios que se muestran a continuación:

- *Creating and Migrating Chatbots with CONGA* [57]. En este artículo se presenta un IDE (*Integrated Development Environment*) llamado CONGA que permite la creación y migración de *chatbots*. Entre las ventajas que brinda se encuentran: un lenguaje específico de dominio para modelar *chatbots* independientemente de una herramienta de desarrollo y un módulo de recomendación de herramientas que permite orientar fácilmente al desarrollador según los requerimientos del *chatbot*. Además, ofrece generadores de código y un módulo que permite extraer los modelos de agentes conversacionales creados anteriormente. Este es uno de los ejemplos de propuestas tecnológicas mas completas que se encontraron en la literatura, sin

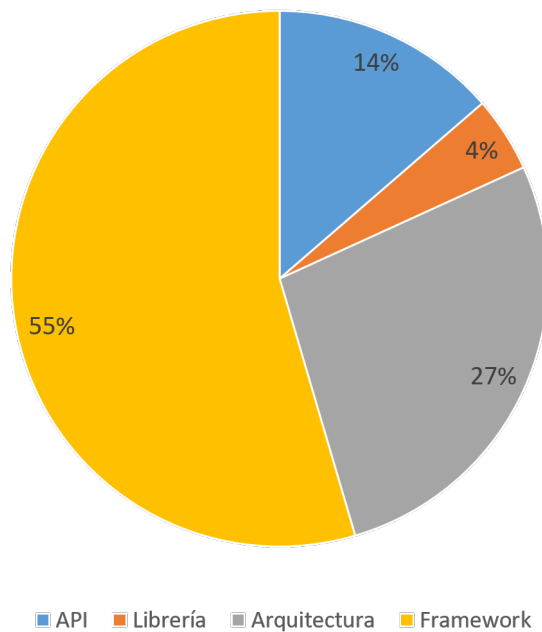


Figura 3.2: Clasificación de propuesta/herramienta.

embargo, actualmente el proceso de ingeniería inversa para obtener los modelos a partir de un *chatbots* solo está disponible para proyectos de DialogFlow ¹. Incluso, existen ocasiones en las que es necesario la intervención manual, como el uso de *emojis*, característica que actualmente no soporta CONGA. En el uso de librerías de Google, los autores señalaron que hay situaciones en las que no es posible realizar la migración de manera completamente automática debido al uso de tecnologías nativas de cada plataforma objetivo.

- *Xatkit: A Multimodal Low-Code Chatbot Development Framework* [4]. Una de las propuestas de mayor presencia en los estudios primarios es Xatkit (anteriormente conocido como *jarvis Multi-platform Chatbot Modeling and Deployment with the Jarvis Framework* [8]). Es un *framework* de código abierto para desarrollo de *chatbots* aplicando *Model-Driven Development*. Uno de los puntos a favor que sitúan esta propuesta a un nivel superior del resto es que brinda al desarrollador una estructura definida y una metodología para cada proyecto de desarrollo. También, implementa sus propios lenguajes específicos de dominio los cuales son indepen-

¹<https://cloud.google.com/dialogflow>

dientes entre sí, lo que permite rehusar fácilmente todos los modelos de cualquier etapa de desarrollo en diferentes proyectos. Además, contiene un módulo llamado *Platform Package*, el cual permite definir nuevas plataformas para la implementación de agentes conversacionales, por lo que se puede ampliar el alcance de todos los *chatbots* que sean creados o los ya existentes. Adicionalmente, incluye un *Xatkit Development Toolkit* para que más desarrolladores puedan ampliar su proyecto.

- *MDD based case tool for Automatic Generation of ChatBot* [6]. Los autores presentan una herramienta para generar agentes conversacionales en un entorno MDD. Esto se lleva a cabo mediante el IDE Eclipse y el Project Eclipse Modeling Project². A través de la definición de un meta-metamodelo ECORE y una serie de reglas de transformación M2T *Model to Text* la propuesta es capaz de generar el código fuente para crear un *chatbot* para Facebook^{footnote}<https://www.facebook.com>, así como módulos de configuración para integrar el *bot* a servicios REST (*REpresentational State Transfer*). Permitir la interacción con servicios REST proporciona la posibilidad de comunicarse con una gran cantidad de herramientas. Lo que representa una gran ventaja, sin embargo, no hace uso de algún servicio proveedor de lenguaje natural, por lo que el NLP (*Natural Language Processing*) de los asistentes generados está muy limitado.
- *A Framework to Create Conversational Agents for the Development of Video Games by End-Users* [58]. En este estudio primario, los autores presentan una herramienta para crear *chatbots*. Para esto, utilizan MDD con el objeto de crear modelos a través del diseño de un DSL. En el caso de estudio, se muestra el desarrollo de una herramienta para crear máquinas de estados en Eclipse, el *framework* en sí es una para casos específicos, no para crear todo tipo de *chatbots* para distintas plataformas.
- *Model-based Chatbot Generation Approach to Converse with Open Data Sources* [59]. Esta publicación propone una herramienta para desarrollar *chatbots* capaces de interactuar con OpenAPI (especificación estándar que se usa para describir las API REST) por medio de Xatkit y un *plugin* de Eclipse. Esta propuesta permite la

²<https://www.eclipse.org/modeling/>

importación directa de la especificación de un OpenApi a un modelo UML (Unified Modeling Language) que es utilizado para generar la comunicación entre el *chatbot* con una API. La idea presentada en este trabajo mejora la automatización en el proceso de desarrollo de *chatbots* que pueden ser utilizados para consultar diferentes OpenApis. Como limitante se encuentra el hecho de que para importar el API, este debe ser del tipo Socrata, CKAN, OData u OpenAPI solamente. Esto se logra mediante configuración manual por parte del desarrollador, lo cual limita ese proceso de automatización en la generación de *chatbots*.

- *Diplomat: A conversational agent framework for goal-oriented group discussion* [60]. Un *framework* para desarrollar *chatbots* moderadores de grupos de discusión. La herramienta, consiste en un conjunto de clases de Python que son usadas para definir las características del agente, dichas características son clasificadas en 6 tipos. El archivo de transcripción, contiene el historial del *chat* del grupo, incluyendo autores de los mensajes. Las intervenciones que puede realizar el *chatbot* consisten en enviar mensajes. Las interacciones pasivas que se activan con el paso del tiempo, interacciones reactivas las cuales se activan dependiendo de ciertas acciones de los usuarios, el tipo de características del agente, que son sus reglas de conversación y relacionan el archivo de transcripción con las intervenciones disponibles y, por último, el tipo de característica de configuración que modifican el comportamiento de la aplicación del agente. Es una herramienta que facilita el desarrollo de agentes conversacionales en grupos de discusión, un ámbito específico de aplicación. No obstante, debido a que la implementación del *framework* es un conjunto de clases de Python y no se creó un DSL, se requiere conocimientos técnicos de programación para ser utilizado. Además, se necesita volver a cargar el archivo de transcripción completo cada vez que existe una modificación en este. En la publicación no se muestra un ejemplo de un *chatbot* desarrollado con esta herramienta, solo se utilizó el enfoque de *wizard of oz* (metodología de diseño utilizada en el desarrollo rápido de productos para mejorar la experiencia del usuario) para validar su desarrollo.
- *Towards a model-driven approach for multiexperience AI-based user interfaces* [61]. Esta investigación propone un DSL (el cual es una generalización del DSL

de Xatkit) para el desarrollo de todo tipo de interfaces de usuario para *chatbots*. La propuesta presentada en esta investigación facilita la definición de flujos y comportamientos más complejos en la interacción con el usuario. La sintaxis abstracta del DSL está compuesto por el *intent package*, el *behavioral package* y el *runtime package*. El primero describe los conceptos que pueden usarse para modelar la definición de *intents*, en este se describen los tipos de eventos que pueden ocurrir en el sistema y como son provocados. También se definen aquí los *intents*, así como entidades para el caso de los eventos provocados por un usuario. El segundo, sirve para representar una máquina de estados para modelar el comportamiento del *bot* ante los distintos eventos. Por último, en el *runtime package* se definen las clases que se utilizarán en tiempo de ejecución del *bot* desplegado. Una vez es definido el agente conversacional, el trabajo propone una extensión del lenguaje de modelado de flujos de interacción (*Interaction Flow Modeling Language*, IFML, por sus siglas en inglés) para describir las relaciones entre el agente y los demás componentes de software. Lo anterior se utiliza para especificar el posicionamiento y visibilidad del *chatbot* en la interfaz gráfica. Particularmente, permite especificar la información que puede ser compartida entre el agente y el resto de la interfaz, la forma en la que pueden interactuar los componentes con el *bot* o este con los componentes y la ejecución de acciones y los componentes cuando se active un evento. Esta propuesta muestra un acercamiento en el que se puede facilitar el desarrollo de interfaces conversacionales que están integradas con distintos componentes a la vez, es un primer paso para atacar el problema. Lamentablemente, aún existen una cantidad de características de los *bots* que pueden ser añadidas al DSL, por ejemplo el control de acceso a través de roles en los que un usuario puede o no consultar cierta información, esto de acuerdo con sus permisos de acceso.

- *Towards Automating the Synthesis of Chatbots for Conversational Model Query* [62]. Plantea un esquema de desarrollo de *chatbots* para hacer consultas utilizando lenguaje natural, el esquema propone enlazar un *intent* a cada tipo de consulta, para esto se creó una plataforma EMF (*Eclipse Modeling Framework*) para permitir que el *framework* Xatkit consulte modelos EMF dependiendo de los *intents* detectados

por el *bot*. El diseñador del *chatbot* proveerá un metamodelo del dominio que define la estructura de los modelos que serán consultados y lo complementará con sinónimos para cada característica y clase para el entrenamiento del modelo de NLP, a partir de esta información se genera el agente que es capaz de realizar estas consultas. Este esquema puede ser utilizado para crear *chatbots* que hagan consultas a partir de una serie de configuraciones y modelos, en si. También, los autores de esta propuesta muestran una extensión del *framework* Xatkit.

- *Extensible Chatbot Architecture Using Metamodels of Natural Language Understanding* [63]. En este estudio primario se propone una arquitectura para crear *chatbots* mediante un metamodelo en el cual se definen los conceptos de los servicios de NLU (*Natural-Language Understanding*). La propuesta presentada, permite crear metamodelos para nuevos servicios (actualmente se desarrollaron 2, Dialogflow y Rasa) así como la definición de un conjunto de reglas para relacionarlos. El uso de esta arquitectura, permite al desarrollador la posibilidad de intercambiar fácilmente el proveedor de lenguaje natural del *chatbot* siempre y cuando existan los metamodelos correspondientes. Además, su metamodelo permite la definición de roles y agrupaciones de las entidades.
- *Towards Conversational Syntax for Domain-Specific Languages using Chatbots* [64]. Presenta un *framework* para añadir soporte de lenguaje natural a través de *chatbots* a la sintaxis de los lenguajes específicos de dominio. En él se propone un proceso automatizado para modelar los *chatbots* en el que genera una configuración general. Además, declara cómo hacer referencia a objetos y características de las clases instanciables, así como el nivel de inconsistencia tolerable permitido durante el proceso de modelado. Posteriormente, el diseñador ajusta esta configuración manualmente añadiendo sinónimos para las clases o características o para declarar que clases no son instanciables. Después, el *framework* sintetiza una descripción del *chatbot* y finalmente, implementa el *chatbot* en una plataforma.
- *Collaborative Creation and Training of Social Bots in Learning Communities* [17]. desarrollaron un *framework* para crear *chatbots* en el que se definen propiedades

como el ambiente con el que debe interactuar el agente, las acciones del *bot* y los usuarios con los que puede relacionarse. Estos agentes, son capaces de implementar funciones de clasificación y generación de texto utilizando técnicas de aprendizaje profundo, lo cual le da su acercamiento a los *bots* generativos pero al mismo tiempo permite el uso de declaraciones *If-Then* si se busca un enfoque “*retrieval chatbot*”. Una vez inicializadas estas características, se puede inducir el *bot* a la fase de entrenamiento donde se le provee de datos de entrenamiento para dotarlo de una base de conocimiento. Es uno de los pocos estudios que permite generar *chatbots* de enfoque generativo (*generative chatbot*), es decir, bot de dominio abierto que genera combinaciones originales de lenguaje en lugar de seleccionar respuestas predefinidas.

- *Architecture of a Framework for Generic Assisting Conversational Agents* [69]. En este estudio primario se explica la utilidad del uso de agentes conversacionales para asistir a usuarios inexpertos en el manejo de componentes de software. Los autores afirman que es necesario tener una opción para desarrollar estos asistentes con un tiempo de esfuerzo y costo igual o menor al desarrollo de los propios componentes. Es por eso que proponen este *framework* el cual se basa en la secuencia convencional de sistemas de ayuda: 1) formalización de la solicitud del usuario, 2) resolución de la solicitud y 3) presentación de la asistencia, en donde siguieron distintos enfoques como la generalización de los agentes definidos a partir del *framework*. La idea es que desarrolladores que saben poco de NLP puedan desarrollar estos asistentes y aprovechar las técnicas de MDA (Model Driven Architecture) para generar nuevos componentes a través de modelos. El *framework*, fue probado con distintos componentes de diferente dominio, sin embargo, todos son componentes sencillos por lo que no existe una validación en un caso más complejo. Además, no se hace uso de ningún servicio proveedor de procesamiento de lenguaje natural que utilice aprendizaje profundo.
- *A Modular Data-Driven Architecture for Empathetic Conversational Agents* [16]. Es una arquitectura para desarrollar un agente conversacional, el cual puede ser modificado para utilizarse en diferentes escenarios. Se le integra un controlador empá-

tico. Trabaja con el historial de diálogo para percibir las emociones y sentimientos del usuario y crear contenido acorde a sus necesidades. La arquitectura, está basada en los agentes conversacionales generativos dirigidos por datos y está dividida en cinco tipos de módulos. El primer tipo procesa la entrada convirtiendo el sonido en texto para después usar un componente de NLU para extraer el significado del texto. El segundo, se encarga de controlar la estructura del dialogo y se genera una respuesta que es convertida de texto a dialogo hablado. En el tercer tipo de módulos se encuentran aquellos encargados de extraer las características de alto nivel de la entrada hablada y tratar de encontrar las emociones y sentimientos del usuario. El quinto tipo tiene como objetivo moldear la salida según el contexto global de la conversación. En él, se encuentran los módulos de control de empatía, encargados de modelar el sistema de empatía. Siguiendo esta arquitectura es posible crear agentes conversacionales de dominio abierto utilizando algoritmos de aprendizaje reforzado para generar agentes conversacionales más empáticos.

- *Dynamic Natural Language User Interfaces Using Microservice* [65]. Este trabajo presenta un acercamiento automatizado para construir agentes conversacionales para reemplazar o complementar a las interfaces gráficas. Esto se hace con base a un conjunto de microservicios definidos mediante la especificación de OpenAPI y un conjunto de requerimientos escritos en lenguaje natural. Las operaciones se pueden definir a través del análisis de los metadatos de la especificación de los microservicios. De la misma forma se pueden inferir las operaciones, parámetros, sus propiedades y las salidas de cada operación. Gracias a esto se puede obtener un modelo representativo de los microservicios. Por otra parte se extraen los *intents* de cada requerimiento aplicando detección de frases clave y se enlaza cada uno con el modelo de los microservicios. Adicionalmente, se generan múltiples frases para invocar cada *intent* a partir de los requerimientos y los *intent* detectados. Por último, las frases son utilizadas para entrenar una red neuronal que enlazará los enunciados del usuario a los *intents*. Este acercamiento automatiza en gran medida la creación interfaces conversacionales para sistemas existentes o nuevos sistemas siempre que exista la especificación de OpenAPI de los microservicios que se utilizarán, solo es

necesario proveer estos y las declaraciones de los requerimientos para ahorrar una parte del trabajo.

- *Model-Driven Chats: Enabling Chatbot Development for Non-technical Domain Experts Through Chat Flow Visualization and Auto-generation* [66]. En este trabajo, los autores presentan una simbología gráfica estandarizada del flujo de diálogo utilizando los conceptos de MDE para crear una forma más sencilla de representar los flujos de dialogo utilizando formatos basados en XML (eXtensible Markup Language), con lo que se espera mejorará la trazabilidad, testeo, mantenimiento y documentación. Para lograr esto se crearon los metamodelos correspondientes para la representación de los flujos de diálogo de *Watson Assistant* (WA) y se definieron las reglas de transformación, desde la representación de dialogo de WA a la propuesta y viceversa.
- *Huey: Intelligent Agents for Natural Human to Machine Communication* [67]. En este trabajo se desarrolló una arquitectura la cual incluye dos nuevos lenguajes de programación que permite crear *voicebots* (software que combina el procesamiento de lenguaje natural y la inteligencia artificial para entablar conversaciones) y *chatbots* a partir de gramáticas. Funciona independiente de cualquier sistema operativo en particular, la nube, entorno o hardware propietario. Estos lenguajes fueron creados utilizando los conceptos de los DSLs.

Tabla 3.8: Citas tipo de propuesta/herramienta.

Tipo de herramienta	Estudios primarios
API (interfaz de programación de aplicaciones)	[7], [64], [65]
Librería	[60]
<i>Framework</i>	[68], [57], [4], [6], [8], [58], [59], [61], [64], [17], [69], [66]
Arquitectura	[59], [62], [63], [16], [65], [67]

Es importante señalar que, aunque en el sistema de clasificación, la categoría de arquitectura y *framework* son independientes, la mayoría de estos entornos ofrecen un patrón de diseño concreto para desarrollar agentes conversacionales. Pero, como se puede observar en la clasificación, también se encontraron trabajos de investigación que desarrollaron una arquitectura propia.

Pregunta 3. ¿Cuáles son los desafíos que existen al usar tecnologías que aplican alguna técnica de la Ingeniería Dirigida por Modelos en la industria?

Al analizar la literatura disponible se puede observar que muchos investigadores se han centrado en desarrollar un *framework* de desarrollo con el fin de tener un entorno completo en el que se puedan crear *chatbots* ahorrando tiempo y esfuerzo en su construcción. Dentro de este campo, uno de los retos al que se han enfrentado las investigaciones son la gran cantidad de tecnologías que son necesarias en el desarrollo de un agente conversacional y de las plataformas objetivo que existen para dicho fin. Esto es complicado de resolver, porque podría provocar problemas de compatibilidad y escalabilidad. Por tal motivo, se considera que el principal reto que enfrenta el desarrollo automatizado de *bots* es la capacidad de proveer la compatibilidad entre plataformas y distintos proveedores de herramientas de desarrollo. En este sentido, han surgido propuestas que hacen énfasis en la creación de metamodelos (DSLs) que tomen en cuenta la posibilidad de crear nuevas plataformas objetivo y tecnologías de desarrollo considerando para esto herramientas de procesamiento de lenguaje natural. Por otra parte, se encuentran pocos artículos que presenten herramientas para crear agentes conversacionales generativos los cuales tengan una base de conocimientos general evitando que sean para un dominio en específico. Algunos investigadores también señalan que es complicado conseguir los datos para poder entrenar sus modelos de procesamiento de lenguaje natural, apuntando a que muchos negocios y empresas suelen requerir esperar para obtener años de datos provenientes de las interacciones con los usuarios para tener un conjunto de datos confiables.

Pregunta 4. ¿Cuáles son las bases de datos con mayor número de publicaciones en este tema?

Como puede observarse en la Tabla 3.3 la mayoría de estudios primarios fueron encontrados mediante *Google Scholar*, sin embargo muchos de estos provenían originalmente de *Springer Link* y *IEEE Explorer*. Las bases de datos con más publicaciones en este tema

son éstos dos con once y siete publicaciones respectivamente.

Pregunta 5. ¿Cuál ha sido la tendencia en los últimos cinco años en las publicaciones relacionadas con el Desarrollo de *chatbot* a través de MDE?

Como se puede observar en la Figura 3.3, la mayoría de las publicaciones son menores de diez años. También, se puede apreciar un creciente interés en la investigación de este tema, la mayoría de ellas fueron publicadas en el año 2021. Cabe destacar que la publicación más antigua incluida en este mapeo es de año 2016, lo cual es un indicador del interés que existe en la comunidad científica.

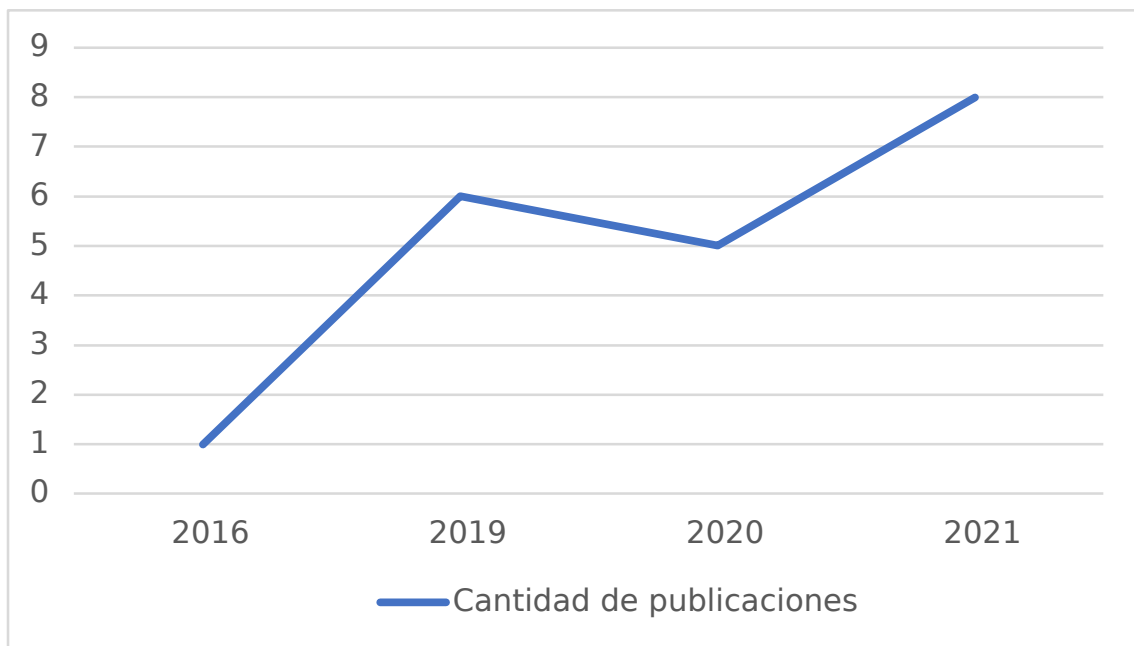


Figura 3.3: Tendencia en las publicaciones.

Pregunta 6. ¿Cuáles son los países más activos en este tema?

Acorde con los datos extraídos de este mapeo sistemático, España se posiciona como el país más activo en este tema. También se encontraron publicaciones provenientes de Estados Unidos, Alemania, Italia, Francia, Colombia, Canadá y Serbia.

Pregunta 7. ¿Cuál es el tipo de publicación de las investigaciones encontradas?

Se puede observar en la Tabla 3.4 que la mayoría de las investigaciones de este tema surgen de trabajos para conferencias, encontrando un total de diez publicaciones en los resultados finales, sin embargo existen una variedad de artículos de revista (siete publicaciones). En cuanto a trabajos de revisión si bien en los filtrados preliminares se encontra-

ron trabajos que examinaban otros trabajos de investigación o tecnologías de desarrollo de *chatbots* no se encontró uno especializado en las tecnologías que utilizaban técnicas de MDE para el desarrollo de los agentes conversacionales, se encontró e incluyo un solo trabajo de tesis relacionado con el tema proveniente de la universidad de Harvard.

Tabla 3.9: Clasificación por tipo de publicación científica.

Herramienta	Descripción
Journal	[4], [8], [61], [56], [63], [64], [17]
Conference	[68], [57], [6], [58], [59], [62], [69], [16], [65], [66]
Tesis	[67]

El interés de la comunicad científica por hacer investigación en este tema se han incrementado y mantenido los últimos años. Los investigadores, se han enfocado en disminuir el tiempo de desarrollo de agentes conversacionales a través de proponer herramientas que mejoren la automatización de desarrollo utilizando técnicas de MDE. El objetivo de utilizar Ingeniería Dirigida por Modelos parte de la construcción de metamodelos y DSLs que aceleren y faciliten la creación de *chatbots*, así como mejoren el proceso de migración entre tecnologías y plataformas. También, se ha propuesto arquitecturas que permiten utilizar la información ya disponible para la construcción de asistentes conversacionales con el uso de OpenAPI.

Las propuestas respecto a este tema han mejorado el desarrollo de *chatbots* a nivel empresarial, sin embargo aún existen retos y áreas de oportunidad debido a que cada una de las propuestas analizadas poseen sus ventajas, así como también tienen limitaciones. Entre las principales se encuentran i) la creencia de ciertas características en las definiciones de sus metamodelos, por ejemplo, de roles de usuario, lo que limita las capacidades del *bot*, ii) no utilizar proveedores de lenguaje natural para mejorar la experiencia de usuario, iii) carecer de un DSL que permita facilitar el desarrollo de agentes conversacionales para usuarios no expertos, y iv) No brindar desarrollo multiplataforma.

3.3. Discusión

Mediante mapeo sistemático realizado para profundizar en el estado del arte del tema de estudio de esta tesis, se obtuvo información acerca de las publicaciones que muestran el uso de las técnicas de la Ingeniería Dirigida por Modelos para automatizar el proceso de desarrollo de agentes conversacionales. En esta sección, se discutieron las percepciones sobre las tendencias en la investigación en este tema y las principales ventajas de las propuestas actuales, así como las áreas que se pueden abordar en futuras investigaciones.

Se puede destacar que desde hace tiempo ya existía una extensa variedad de tecnologías, herramientas y plataformas para desarrollar *chatbots* en el mercado. En [56], se analizan las principales herramientas de desarrollo, los autores presentan una comparación de las principales características de cada tecnología y se indican los factores esenciales a tomar en cuenta para seleccionar una. Estos factores se dividen en dos categorías, los técnicos y los administrativos. Sin embargo, es importante mencionar que esta diversidad de herramientas puede crear dificultades a la hora de elegir la adecuada para el proyecto a desarrollar. Esto se debe a que cada proyecto de desarrollo tiene características muy particulares, por ejemplo, si se quiere incluir una nueva característica posterior en el *bot* que no estaba planeada inicialmente, es posible que sea necesario migrar a otra plataforma de desarrollo. El estudio concluye que las herramientas existentes cubren una amplia variedad de posibilidades y escenarios en la implementación de *chatbots*. Incluso, haciendo uso de estas herramientas siguen existiendo retos en todo el proceso de desarrollo, por ejemplo aún existe un margen de mejora en las metodologías debido a que no ofrecen patrones de diseño ni métricas de calidad. Incluso, no ofrecen un método adecuado para las pruebas de software. Además, la mayoría de herramientas están enfocados en agentes conversacionales de dominio cerrado, debido a que utilizan frases de entrenamiento para especificar los *intents*. Por último, el autor menciona que cada tecnología utiliza su propio formato para definir a los *chatbots*, por este motivo en el W3C (World Wide Web Consortium) se está definiendo una estandarización para su construcción.

La mayoría de los trabajos encontrados proponen resolver las consultas de los usuarios a través de la detección de *intents* predefinidos en los mensajes de los usuarios. También, se identificaron propuestas que van más allá de esos *intents*, encontrando información va-

liosa para el usuario en las páginas de preguntas frecuentes de las empresas, manuales de usuario, OpenAPIs o documentos ya disponibles. Estos documentos pueden utilizarse para responder las preguntas que no fueron encontradas en los *intents* definidos, debido a que no siempre es posible contemplar todos los casos y preguntas del usuario esta es una alternativa muy útil con la que el *chatbot* puede ampliar su base de conocimiento. La siguiente alternativa para conseguir la información para el usuario debe provenir de una biblioteca de datos más amplia, para lo cual se podría usar un modelo de lenguaje que posea conocimiento general, como los Transformers GPT-3, GPT-J, GPT-NEO, Megatron-Turing, etc. Los cuales, pueden ser usados como ultima alternativa para contestar las preguntas de los usuarios, sin embargo ninguna de las propuestas encontradas toma en cuenta estas herramientas.

En los últimos años, se ha mostrado un creciente interés en la creación de herramientas que mejoren el desarrollo de agentes conversacionales. Algunos autores hicieron uso de las tecnologías que ya existían para crear *frameworks* y arquitecturas capaces de desarrollar agentes conversacionales utilizando una herramienta integral. Parte de la investigación realizada hasta ahora ha buscado la reutilización de componentes y la escalabilidad. Se ha buscado que se puedan integrar herramientas a través del uso de DSLs. Otros autores, definieron sus propias arquitecturas en las cuales utilizan su propia metodología para dotar de NLP a los agentes conversacionales. Esto ocasiona que no exista una estandarización en la construcción de este tipo de software. Algunas propuestas, incluso permiten crear *chatbot* generativos debido a que estos tienen el potencial de tener una base de conocimientos general y conversar acerca de cualquier tema.

Las nuevas propuestas que han surgido para desarrollar agentes conversacionales no están exentas de presentar sus propios retos, pues en las mismas se puede observar que todas tienen sus propias limitaciones. En primer lugar, si la herramienta propuesta utiliza un servicio de procesamiento de lenguaje natural externo es necesario integrarlas por medio de la definición de metamodelos que permitan la posibilidad de definir nuevas tecnologías. Esto es importante, pues se está hablando de un medio cambiante y de confiar en una sola herramienta, esta podría modificarse o incluso desaparecer. Además, las distintas herramientas ofrecen diferentes prestaciones y utilidades, por lo que sería peligroso depender de solo una. Así mismo, la compatibilidad debe de ser un factor clave y permitir

la migración. Otra oportunidad que se detectó, y que es poco explorada en las propuestas tecnológicas encontradas en los estudios primarios analizados en el mapeo sistemático, fue la posibilidad de dotar a los *chatbots* con la capacidad de ofrecer recomendaciones o sugerencias para guiar al usuario en el hilo de la conversación para aumentar la facilidad de uso de los agentes. Por otra parte, es poca la atención que se le dedica en los trabajos encontrados a la integración con plataformas de mensajería que son ampliamente utilizadas en el mercado de los *chatbots* como WhatsApp, Telegram, etc. que son usadas tanto por grandes empresas y pequeños negocios para comunicarse con sus clientes. Debido a esto, puede ser de interés explorar la automatización en la integración y comunicación con estas plataformas utilizando las herramientas existentes.

Finalmente, como se puede observar en la Figura 3.1, la mayoría de la investigación se a centrado en crear nuevas propuestas, pero han comenzado a surgir evaluaciones e investigaciones de validación que utilizan propuestas previas. Una recomendación, es promover la investigación de evaluación y validación y centrarse en mejorar o ampliar las propuestas ya existentes. Esto permitiría obtener herramientas robustas y con más posibilidades para desarrollar agentes conversacionales estandarizados.

3.4. Comentarios Finales

Este capítulo presentó los resultados obtenidos a través de realizar un estudio denominado mapeo sistemático de la literatura. Esto permitió obtener información acerca del estado del arte en el ámbito de la automatización del proceso de desarrollo de agentes conversacionales por medio de las técnicas de Ingeniería Dirigida por Modelos. Se consideraron un total de 429 publicaciones de diferentes fuentes científicas como *IEEE Explorer*, *ACM Digital Library*, *Science Direct*, *Springer Link*, *Web of Science* y *Google Scholar*. Posteriormente, se seleccionaron y analizaron 22 estudios primarios mediante una serie de criterios de inclusión y exclusión previamente definidos. Después, se contestaron las preguntas de investigación. Finalmente, se obtuvo el estado actual de la investigación en este tema.

El mapeo sistemático de la literatura mostró que la investigación respecto a este tema ha tenido un incremento en los últimos años, comprobando que es un tema de interés para

la comunidad científica actualmente. Además, este estudio muestra que la mayor parte de la investigación se ha centrado en desarrollar nuevas propuestas tecnológicas con el objetivo de obtener herramientas completas que faciliten y aceleren el desarrollo de agentes conversacionales. Con esto, se pretende como beneficio, disminuir así el costo y tiempo a los desarrolladores. Si bien existen artículos que parten de otras propuestas para extender funcionamiento se requieren más estudios que validen y evalúen los distintos acercamientos. Otra de las áreas de oportunidad detectadas se centra en la búsqueda de herramientas que permitan obtener datos para entrenar y definir *chatbots* con mayor facilidad. Es verdad que existen propuestas que presentan arquitecturas que se aprovechan de APIs REST para definir agentes conversacionales con cierto grado de automatización en el proceso, pero estas posibilidades no son muy explotadas en la literatura científica analizada.

Capítulo 4

Una Propuesta Dirigida por Modelos para la Generación de Agentes Conversacionales

Se inicia este capítulo de la tesis recordando que el objetivo principal de esta investigación consistió en reducir el tiempo de desarrollo de *chatbots* para aplicaciones de mensajería instantánea en el *framework* Xatkit Chat Platform a través de una propuesta bajo el paradigma de desarrollo de software MDD. Además, se implementó el modelo de lenguaje autorregresivo GPT-3 para mejorar la experiencia del usuario al utilizar un *bot*, de tal forma el agente puede generar respuestas (recomendaciones) automáticas cuando no cuente la información solicitada en su base de conocimiento.

4.1. Introducción

De acuerdo con los estudios analizados en el Capítulo 3 de esta tesis, existen herramientas que brindan un entorno para el desarrollo semi-automático de agentes conversacionales, cada una de ellas ofrece un ambiente de desarrollo muy particular. Para esta tesis se seleccionó el *framework* Xatkit Chat Platform. Esta plataforma utiliza el paradigma de desarrollo de software MDD para la generación de agentes a partir de modelos y transformaciones de modelo a texto; además de ser un entorno de código abierto, in-

cluye un *Platform Package* con el cual es posible definir agentes conversacionales para distintas plataformas, lo que permite que se pueda ampliar la plataforma de implementación de los *chatbots* creados. Debido a que utiliza el paradigma MDD, dispone de un DSL implementado mediante el lenguaje de programación Java, con el cual permite su extensión y personalización sin dejar de lado la mantenibilidad. Asimismo, una más de sus fortalezas es que incluye soporte para integrar proveedores de procesamiento de lenguaje natural. Lamentablemente, actualmente Xatkit no cuenta con soporte para la totalidad de las plataformas de mensajería instantánea que existen en el mercado, particularmente para WhatsApp. Esto representa un problema, pues limita su campo de aplicación, debido a que es el servicio de mensajería instantánea con más usuarios hoy en día en el mundo. En enero de 2023, dos mil millones de usuarios accedían al chat de WhatsApp mensualmente [70]. El uso de la aplicación es particularmente fuerte en mercados fuera de los Estados Unidos.

La solución que se presenta en este capítulo de la tesis mejora la problemática mencionada anteriormente (consultar sección 1.2 de la tesis para ampliar la información) mediante la extensión de la funcionalidad del *framework* Xatkit y su integración a WhatsApp mediante Twilio a través de un proceso dirigido por modelos. Esto permite que los desarrolladores de agentes conversacionales sean capaces de integrar los *chatbots* que creen al servicio de mensajería instantánea más utilizado hoy en día.

4.2. Semi-automatización de agentes conversacionales mediante el desarrollo dirigido por modelos

En esta sección se describe el proceso definido para la semi-automatización de agentes conversacionales mediante el desarrollo dirigido por modelos. MDD fue seleccionado como base para esta propuesta debido a las ventajas que brinda, principalmente, la reducción de tiempo de desarrollo (consultar sección 2.1.6 de la tesis para mayor información). La propuesta en esta tesis está integrada de la siguiente manera: i) un metamodelo para definir un modelo que represente el *bot* que se desarrolló y ii) la definición de una plantilla para implementar transformaciones M2T a través del *plugin* Acceleo Model to Text.

La plantilla se utiliza para convertir el modelo generado (definición conceptual del agente conversacional) a partir del metamodelo a código fuente Java. El código obtenido es utilizado para la integración del agente con la plataforma Xatkit. A continuación se describen cada uno de los componentes mencionados.

4.2.1. Definición de un DSL mediante la implementación de un metamodelo

Un DSL es un lenguaje diseñado para ser utilizado en dominio en específico, permite plasmar los conceptos claves en él, así como generar modelos y código fuente para un área en concreto, para una explicación más detallada se puede consultar la sección 2.1.3 de la tesis. En la propuesta que se presenta en esta investigación, se definió un DSL y se implementó en un metamodelo denominado XatbotMM, a través del marco de trabajo Eclipse a través del *plugin* Eclipse Modeling Framework, ambos pertenecientes al proyecto Eclipse Modeling Project [71]. En él, se especificaron metaclases que representan cada uno de los elementos que pueden conformar un *Chatbot* del *framework* Xatkit Chat Platform, así como las relaciones entre ellos. Esto permitirá la creación de *bots* a partir de instancias de las metaclases del metamodelo creado, es decir, a partir de un modelo de un agente conversacional.

En la Figura 4.1, se muestra el diseño del metamodelo llamado XatbotMM, implementado en Eclipse a través del Eclipse Modeling Project. Se definieron nueve metaclases para representar cada uno de los conceptos del dominio agentes conversacionales en Xatkit Chat Platform, estos son: la metaclase *Platform*, la cual es la plataforma que utilizará el *bot* para comunicarse con los usuarios, cabe destacar que XatbotMM permite la definición de modelos de agentes conversacionales para ser integrados en las plataformas WhatsApp y React. La metaclase *Provider*, es la que permite seleccionar el servicio proveedor de lenguaje natural, brinda soporte para los proveedores de procesamiento de lenguaje natural DialogFlow y NLP.JS. La metaclase *Intents*, es utilizada para definir la capacidad del agente conversacional a través de sus atributos nombre y sentencias de entrenamiento. Finalmente, las metaclases *State*, *Transition*, *TestMessage*, *MultimediaMessage* y *ConditionalTransition* que conforman el comportamiento del *bot*. Por lo tanto, es posible construir

diversos *Chatbots*, solo se requiere que el usuario modele la lógica del agente conversacional mediante la definición de los múltiples *intents*, frases de entrenamiento, estados, transiciones y respuestas en el modelo.

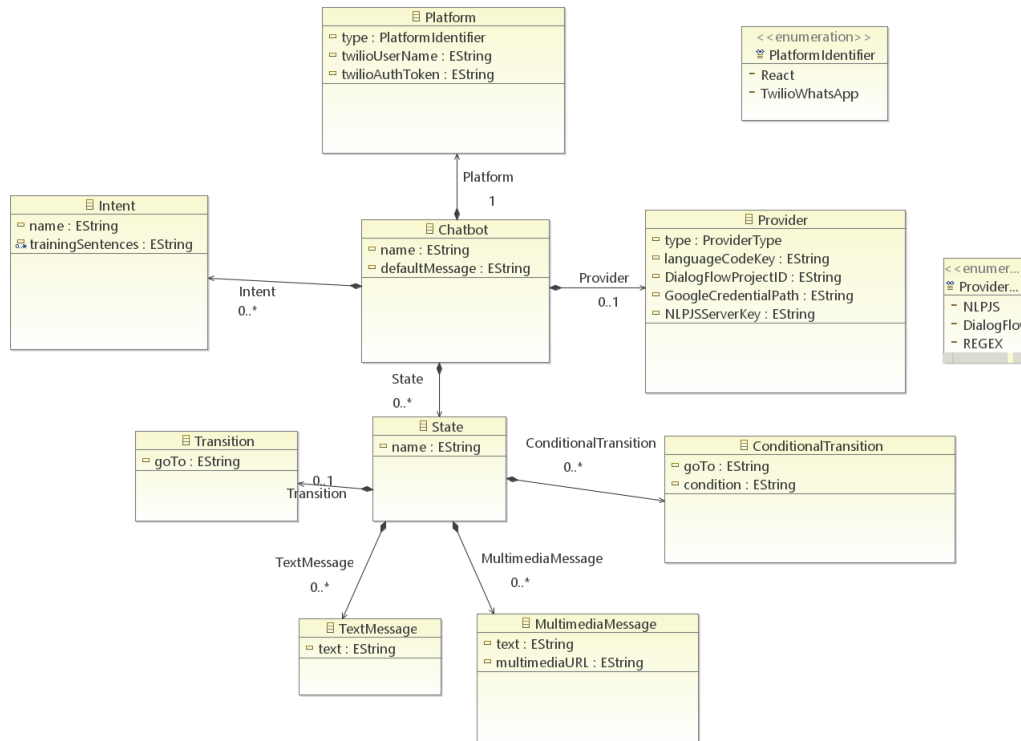


Figura 4.1: Metamodelo XatbotMM.

En la Figura 4.2, se observa el modelo de un agente conversacional como ejemplo, construido con editor de modelos proporcionado por el marco de trabajo Eclipse, a través del *plugin* Eclipse Modeling Framework. El *bot*, es capaz de saludar y presentarse cuando interactúe con el usuario. El agente se conforma de cuatro estados, estos son: *init*, *estadoEspera*, *estadoPresentacion* y *estadoSaludo*. Los estados se utilizan para inicializar el *Chatbot* en el estado *init*, el cual pasa el control al *estadoEspera* en donde se verificará si el usuario introduce un texto que corresponda a un saludo o a una consulta sobre información del agente. Posteriormente, el agente, en caso de detectar uno de las dos intenciones previamente definidas, cederá el control a uno de los dos estados que corresponda con la intención detectada, la cual puede ser *estadoSaludo* o *estadoPresentacion*. Después, el *bot* enviará un mensaje de texto apropiado a la petición del usuario y con esto devolverá el control al *estadoEspera* para completar el ciclo. Asimismo, en el *bot* se define un *provider*

para especificar el proveedor de procesamiento de lenguaje natural que se utilizará, así como un elemento *platform*, que especifica que el agente será desplegado en la plataforma de React en este ejemplo en particular. Finalmente, la transformación M2T, definida en *plugin* Acceleo Model to Text, se encargará de generar el código necesario para ejecutar el agente conversacional.

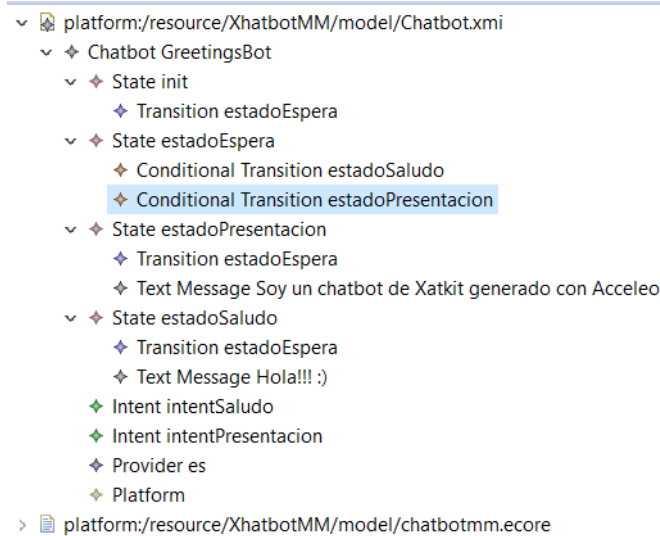


Figura 4.2: Modelo de *Chatbot* usando XatbotMM.

4.2.2. Transformaciones modelo a texto para la generación de código fuente

Las transformaciones modelo a texto permiten convertir un modelo en código fuente en el contexto del MDD. Son un componente esencial para la automatización y semi-automatización en el desarrollo de software. En la propuesta que se presenta en ésta tesis, se diseñó y programó una plantilla para la generación del código fuente de los *chatbots* que sean definidos a partir de modelos instanciados a partir del metamodelo XatkitMM.

La implementación de la transformación M2T desarrollada se llevó a cabo mediante el *plugin* del *framework* Eclipse denominado Acceleo Model to Text. Éste se encarga de extraer de un modelo la información necesaria para poder crear y ejecutar un agente conversacional de Xatkit. La información permite generar archivos de código fuente en el lenguaje de programación Java. La transformación M2T funciona con cualquier modelo creado a partir del metamodelo XatkitMM.

Acceleo permite definir funciones, emplear condiciones y ciclos para poder personalizar la generación de código fuente como sea requerido. Lo anterior, posibilita que el usuario solo necesite definir la lógica de su agente conversacional declarando los *intents*, frases de entrenamiento, estados, transiciones y respuestas en el modelo. Una vez realizado esto, Acceleo se encargará de convertirlo en código Java, así como de importar las librerías necesarias para el uso de la plataforma y proveedor de procesamiento de lenguaje natural, es decir, se ocupará de realizar la configuración adecuada para el funcionamiento del *bot*. En el Fragmento de código 4.1, se puede observar un parte de la plantilla creada, la plantilla completa se encuentra en el Anexo A.

```

1 [comment encoding = UTF-8 /]
2 [module generate( 'http://xhatbotMMU' )]
3
4 [template public generateElement(aChatbot : Chatbot)]
5 [comment @main/]
6 [file (aChatbot.name.concat( '.java' ), false , 'UTF-8')]
7 package com.xatkit.example;
8
9 import com.xatkit.core.XatkitBot;
10
11 [generatePlatformImports(aChatbot)/]
12 import lombok.val;
13 import org.apache.commons.configuration2.BaseConfiguration;
14 import org.apache.commons.configuration2.Configuration;
15
16 import static com.xatkit.dsl.DSL.eventIs;
17 import static com.xatkit.dsl.DSL.fallbackState;
18 import static com.xatkit.dsl.DSL.intent;
19 import static com.xatkit.dsl.DSL.intentIs;
20 import static com.xatkit.dsl.DSL.model;
21 import static com.xatkit.dsl.DSL.state;
22 import com.xatkit.core.recognition.
    IntentRecognitionProviderFactory;
23 [generateIntentProviderImports(aChatbot)/]
24
25 public class [aChatbot.name/] {
26
27     /*
28      * Your bot is a plain Java application: you need to
29      * define a main method to make the created jar executable.
30      */
31     public static void main(String [][' ']/[][' ']/ args) {

```



```

31
32 [ generateIntentDefinitions ( aChatbot ) /]
33 [ generatePlatformDefinitions ( aChatbot ) /]
34 [ generateStatesDefinitions ( aChatbot ) /]
35 [ generateStatesBehaviour ( aChatbot ) /]
36 [ generateDefaultFallback ( aChatbot ) /]
37 [ generateBotModel ( aChatbot ) /]
38
39         Configuration botConfiguration = new
BaseConfiguration () ;
40
41 [ generateIntentProviderConfigurations ( aChatbot ) /]
42 [ generatePlatformConfigurations ( aChatbot ) /]
43
44         XatkitBot xatkitBot = new XatkitBot ( botModel ,
botConfiguration ) ;
45         xatkitBot . run () ;
46         /*
47         * The bot is now started , you can check http ://
localhost:5000/admin to test it .
48         * The logs of the bot are stored in the logs
folder at the root of this project .
49         */
50     }
51 }
52 [ / file ]

```

Fragmento de código 4.1: Fragmento de la plantilla de la implementación de la transformación modelo a texto mediante el *plugin* de Eclipse Acceleo.

4.2.3. Implementación de GPT-3 como alternativa para resolver consultas de usuario

Las preguntas que un agente conversacional de dominio cerrado es capaz de contestar son limitadas, esto debido a que la mayoría de los diseñadores se apoyan en definir *intents* en los cuales clasificar las preguntas del usuario utilizando un conjunto de frases de entrenamiento para cada uno de ellos. Como se puede observar en los trabajos encontrados en el estado del arte (consultar Capítulo 3 de la tesis), esto es comprensible debido a que la intención principal de la construcción de un *chatbot* es contestar las preguntas comunes de las cuales ya se conoce la respuesta, de forma que se pueda desarrollar una herramienta que asista a los usuarios de forma automática. Lamentablemente, aunque la

definición de *intents* y frases de entrenamiento suelen funcionar la mayoría de las veces, esta formula tiene sus limitaciones, por ejemplo, cuando el usuario realice una petición racional al agente conversacional en la que el desarrollador no consideró o formuló la pregunta correspondiente. Esto conlleva a que el servicio NLP no sea capaz de detectar y responder correctamente con los ejemplos de entrenamiento. Por este motivo, se han investigado otras alternativas para dotar al *chatbot* de la capacidad de resolver las consultas de los usuarios cuando no sea capaz de contestar con el procedimiento normal.

En la actualidad, una de las alternativas utilizadas para responder las preguntas de los usuarios no consideradas en el entrenamiento, de acuerdo con los artículos científicos encontrados en la Sección 3.3 de la tesis, consiste en consultar en los documentos que posee el propietario del *bot*. Éstos pueden ser archivos publicados anteriormente, manuales de usuario, *blogs* de preguntas frecuentes, etc. Esta estrategia se aplica debido a que puede ser fuente de información útil al usuario, por lo que se utiliza alguna técnica para buscar en esas fuentes la respuesta de una solicitud que no pudo ser atendida por no tener un *intent* definido o al no ser capaz de detectar la intención del usuario en su petición [72] [73] [74]. Conforme aumenta el tamaño de la base de conocimiento del *bot*, se vuelve capaz de contestar más preguntas, a su vez, se convierte en un agente de dominio abierto, es decir, que puede conversar acerca de cualquier tema. Por este motivo, cuando las fuentes de información que se mencionaron no son suficientes para contestar la pregunta del usuario, aún queda la opción de recurrir a una fuente de conocimientos general. En este caso, es cuando los modelos conocidos como *Transformers* pueden ser de utilidad, pues son capaces de producir texto que simule el comportamiento humano y dispone de un conocimiento general puede ser utilizado para contestar las preguntas de los usuarios de un *chatbot*.

Para paliar la limitante mencionada anteriormente, en ésta tesis se implementó el modelo de lenguaje natural autorregresivo GPT-3 [75], un modelo que cuenta con aproximadamente 175 mil millones de parámetros de entrenamiento y un conocimiento amplio de temas generales. Por tal motivo, se consideró como alternativa para contestar las preguntas que pueda realizar un usuario al agente conversacional cuando el servicio de procesamiento de lenguaje natural no fue capaz de detectar la intención y responder. GPT-3 es uno de los modelos de lenguaje que utilizan aprendizaje profundo más potentes actualmente.

Generación de sugerencias con GPT-3

En esta sección se muestra el proceso propuesto para utilizar de manera práctica el modelo GPT-3 como alternativa para resolver consultas de un usuario de *chatbot*. El método planteado es para utilizar el modelo cuando el agente reciba un mensaje que el servicio de NLP no sea capaz de clasificar en uno de los *intents* definidos. Al suceder este escenario, el *bot* contestará con el mensaje por *default*, simplemente se le indica al usuario que la consulta no puede ser contestada, pero que es posible ofrecerle información al usuario que tenga relación con el mensaje. De esta forma, se instruye al usuario sobre cómo preguntar algo que pueda interesarle con el objetivo de atraer su atención y retomar el hilo de la conversación que pudo perderse cuando no fue posible atender su petición.

El primer paso del procedimiento definido en esta tesis, consistió en diseñar una entrada adecuada para el modelo, considerando las recomendaciones de OpenAI para diseñar un buen *prompt*. Posteriormente, se realizaron una serie de pruebas para distintos escenarios de preguntas, se concluyó que el modelo debería contener cuatro elementos para ser capaz de generar sugerencias útiles para el usuario cuando el *bot* lo requiera. Los elementos definidos son los siguientes:

Proporcionar información sobre el contexto en el que se está empleando el *chatbot*. Esta información puede ser extraída directamente de las frases de entrenamiento del *chatbot* y puede ser implementada en forma de una lista de opciones.

```
1 String prompt="Lista de opciones:\n"+
2     "Tienes una guía del examen? \n"+
3     "¿Dónde están ubicados? \n"+
4     "Datos de Contacto\n"+
5     "Información sobre los docentes\n"+
6     "Cuáles son las líneas de investigación\n"+
7     "Información sobre movilidad\n"
```

Fragmento de código 4.2: Inclusión de la lista de opciones disponibles en el *prompt* de GPT-3.

Instrucciones precisas sobre como el modelo debe actuar. Aquí se especifica con palabras breves y claras la tarea que el modelo debe realizar. En el Fragmento de código 4.3, se detalla como se le indica explícitamente que genere una sugerencia a partir del mensaje del usuario. La sugerencia debe ser seleccionada de la lista de opciones anteriormente mencionada. Las instrucciones dependerán de las necesidades o preferencias del

diseñador del *chatbot*, por ejemplo, se puede especificar que genere las sugerencias en un idioma específico o utilice el mismo idioma del mensaje del usuario.

```
1 prompt = prompt + "Instrucciones:\n"+
2 "pregunta al usuario la opción que puede interesarle
3 de la Lista de opciones según el mensaje del
4 usuario en la conversación de la misma forma
5 que en el Ejemplo, solo puedes contestar
6 con ejemplos de la Lista de opciones y debes
7 realizar una sugerencia que tenga relación
8 con el mensaje del usuario, la sugerencia
9 debe ser siempre en español, si no puedes
10 ofrecer una recomendación acorde o útil de
11 la Lista de opciones responde con un
12 'lo siento no tengo una sugerencia"
13 "para mostrar '\n"
```

Fragmento de código 4.3: Inclusión de las instrucciones en el *prompt* de GPT-3.

Una lista de ejemplos. Esta lista muestra como se espera que se generen las sugerencias. Los ejemplos deben seguir las instrucciones del elemento anterior, por lo general aumentar la cantidad de ejemplos provoca una mayor calidad de las sugerencias, pero se debe considerar que incrementar la cantidad de caracteres del *prompt* implica un mayor costo y tiempo de respuesta.

```
1 "prompt = prompt + " "Ejemplo:\n"+
2 "usuario: ¿da clases el profe Juan? \n"+
3 "sugerencia: si te interesa puedes intentar escribir:
4 Información sobre los docentes\n"+
5 "usuario: ¿cuándo empieza? \n"+
6 "sugerencia: si te interesa puedes intentar escribir:
7Cuál es la fecha de inicio de actividades\n"+
8 "usuario: pero no tengo dinero\n"+
9 "sugerencia: si te interesa puedes intentar escribir:
10 Información sobre becas\n"+
```

Fragmento de código 4.4: Inclusión de ejemplos en el *prompt* de GPT-3.

El mensaje del usuario. Finalmente, se debe incluir, al menos, el último mensaje que escribió el usuario. La razón para hacerlo de esta forma se debe a que es la pieza clave de información a partir de la cual el modelo generará una sugerencia.

En la mayoría de los escenarios entre el usuario y el agente conversacional, los tres primeros elementos serán estáticos, es decir, previamente definidos, mientras que el cuarto cambiará dinámicamente según la conversación. La forma menos complicada de hacerlo

es concatenar el último mensaje al final y añadir una frase de inicio para que el modelo genere el texto a partir de lo previamente mostrado, ver Fragmento de código 4.5.

```
1 String promptTextoUsuario = prompt +  
2     mensajeUsuario +  
3     "\nsugerencia:";
```

Fragmento de código 4.5: Inclusión del mensaje actual del usuario en el *prompt* de GPT-3.

Los tres elementos mencionados anteriormente fueron diseñados para un *prompt* que será utilizado en alguno de los modelos base de GPT-3. En caso de tener planeado utilizar *fine tuning*, los elementos número 1 y 2 deberán ser remplazados o eliminados y utilizar una gran cantidad de instancias del elemento 3 en su lugar. Además, se recomienda filtrar el texto generado del modelo para estandarizar el formato de la respuesta, con esto se pretende impedir o minimizar respuestas inadecuadas, por ejemplo, comprobando la longitud del texto proporcionado por el modelo o comprobando si contiene palabras inapropiadas. También, se puede hacer uso del parámetro de temperatura y las secuencias de parada según le convenga al diseñador del *chatbot*. Siguiendo este procedimiento se puede construir el sistema de recomendaciones rápidamente sin importar el área de aplicación del agente conversacional. Pero, es importante mencionar que esta propuesta fue diseñada especialmente para *chatbots* utilizados en servicio al cliente, por lo que podrían ser necesarias adecuaciones dependiendo de los objetivos del diseñador. En la Figura 4.3 se puede observar un ejemplo en el que se aplica esta propuesta, en este se utiliza el *playground* de OpenAI para generar una sugerencia, el texto formado por GTP-3 es mostrado en color verde.

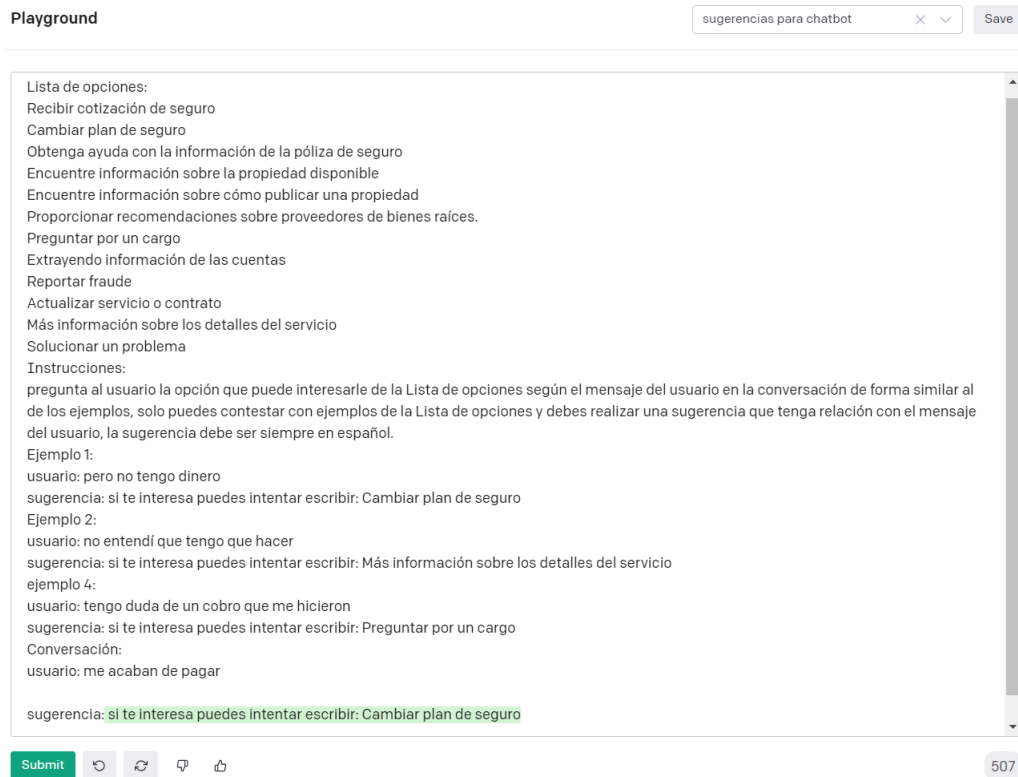


Figura 4.3: Ejemplo de recomendación utilizando la propuesta descrita.

Integración de GPT-3 con Xatkit Chat Platform

El procedimiento definido en esta tesis para generar sugerencias con GPT-3 fue implementado en un proyecto de Xatkit. La dinámica de generación de sugerencias puede iniciarse en el momento que un usuario envía un mensaje que no pudo ser clasificado en ningún *intent* por el servicio de procesamiento de lenguaje natural. Al presentarse esta situación, el agente conversacional consultaría el *fallbackState*, en donde se realiza el procedimiento. Los primeros 3 elementos del *prompt* son definidos previamente y el mensaje del usuario es concatenado cuando es necesario (se puede acceder fácilmente al último mensaje enviado por el usuario desde “`context.getIntent().getMatchedInput()`”) y una vez preparado se puede realizar la generación de texto.

Con el fin de comunicarse con el servicio de OpenAi desde un proyecto de Xatkit se empleó la librería TheoKanning `openai-java` [76]. La finalidad de la librería es utilizar el API de OpenAI desde un proyecto Java. Una vez importada, se pueden realizar peticiones

al servicio para que un modelo genere texto a través de un *prompt*. Un ejemplo de esta implementación se muestra en el Fragmento de código 4.6.

```
1 \begin{lstlisting}[language=Java, frame=none]
2 OpenAiService service = new OpenAiService(your_token)
3 CompletionRequest completionRequest = CompletionRequest
4     .builder()
5     .prompt("hola mi nombre es")
6     .echo(true)
7     .build();
8 service.createCompletion("ada", completionRequest)
9     .getChoices()
10    .forEach(System.out::println);
```

Fragmento de código 4.6: Petición al API de GPT-3 para generación de texto.

Interacción con WhatsApp a través de Twilio API

La forma de establecer comunicación entre software con el servicio de WhatsApp *Business* es a través de una API propietario. Para ser utilizada en un proyecto, es necesario solicitar la autorización de la empresa WhatsApp, así como llevar a cabo un proceso de identificación, que finaliza con la aprobación o negación del uso de la API. Esto representa un obstáculo para los desarrolladores que necesiten comunicarse con esta plataforma. Por tal motivo, la misma empresa recomienda utilizar alguno de sus proveedores para soluciones empresariales. Por esta razón se seleccionó Twilio (consultar el Capítulo 2 de la tesis), el cual brinda una API REST para establecer comunicación entre aplicaciones software y WhatsApp. Este servicio, puede ser utilizado cuando se registra un número de teléfono para que los usuarios puedan comunicarse con el *bot*.

Definición de la plataforma TwilioPlatform en Xatkit para la integración con la plataforma de mensajería WhatsApp

Para definir la plataforma de Xatkit se creó una clase llamada TwilioPlatform. La clase, entre sus funciones, permite inicializar la plataforma, almacenar las credenciales y llamar a las acciones que puede ejecutar el *bot*. Para comenzar, se inicializa el cliente de Twilio, en conjunto con el twilioEventProvider, el cual se encarga de recibir la llamada del *webhook* y convertirla en un evento. Una vez realizado esto, el servidor de Twilio

es notificando, señalando la llegada de un nuevo mensaje. La clase `twilioEventProvider` hereda de la clase abstracta `WebhookEventProvider` que recibe la llamada del *webhook* y maneja las peticiones realizadas por el servidor Xatkit, en el cual se crea una instancia de `TwilioRestHandler`. En esta última clase, se encuentran los métodos que convierten el cuerpo de la petición de formato *application/x-www-form-urlencoded* a un formato JSON. Además, se extrae el contenido del mensaje para procesarlo como un *intent*, de esta forma, cuando un usuario envíe un mensaje este viajará vía *webhook* al *bot*, el cual reconocerá el *intent* proveniente del texto. Asimismo, se extraerán los números de teléfono involucrados. Por último, se creará una sesión basándose en el número de teléfono del remitente del mensaje para que el *chatbot* almacene el estado de la conversación de cada usuario. Cada vez que un mensaje es enviado al número de teléfono con la sesión activa en la cuenta de Twilio, el *bot* que emplee la plataforma de Twilio realizará este proceso internamente y enviará un evento para que el *bot* pueda contestar con una de las acciones definidas. A continuación, en el Fragmento de código 4.7, se muestra una sección de las instrucciones en esta clase, el código completo se puede encontrar en [77].

```
1 public class TwilioPlatform extends RuntimePlatform {
2     public TwilioEventProvider getTwilioEventProvider() {
3         return new TwilioEventProvider(this);
4     }
5     @Override
6     public void start(XatkitBot xatkitBot,
7         Configuration configuration) {
8         super.start(xatkitBot, configuration);
9         String username =
10             configuration.getString(TWILIO_ACCOUNT_SID);
11         if (nonNull(username)) {
12             String authToken = configuration.
13                 getString(TWILIO_AUTH_TOKEN);
14             checkArgument(nonNull(authToken) &&
15                 !authToken.isEmpty(),
16                 "Cannot construct a client from the " +
17                 "provided credentials");
18             checkTwilioClient(username, authToken);
19         } else {
20             Log.warn("No credentials
21                 set in the configuration");
22         }
23     }
```



```

24     private void checkTwilioClient(String username ,
25         String password) throws AuthenticationException {
26         Twilio.init(username , password);
27     }
28     public void reply(@NonNull StateContext context ,
29         @NonNull String message) {
30         Reply action =
31             new Reply(this , context , message);
32         RuntimeActionResult result =
33             action.call();
34     }
35 }

```

Fragmento de código 4.7: Definición de plataforma TwilioPlatform.

Una vez que la plataforma está definida en Xatkit, el siguiente paso consiste en que el desarrollador cree una nueva instancia de la clase TwilioPlatform y de la clase TwilioEventProvider, según se ilustra en el Fragmento de código 4.8, así como agregar sus credenciales a la configuración, como se muestra en el Fragmento de código 4.9. Posteriormente, indicar que se desea utilizar la plataforma de Twilio para que el *chatbot* sea capaz de recibir procesar y enviar mensajes de WhatsApp. Las acciones definidas para la plataforma son capaces de enviar mensajes cuyo contenido sea texto o multimedia y se puede hacer uso de estas acciones como se muestra en el Fragmento de código 4.10.

```

1 TwilioPlatform twilioPlatform = new TwilioPlatform();
2 TwilioEventProvider twilioEventProvider = new
3     TwilioEventProvider(twilioPlatform);

```

Fragmento de código 4.8: Uso de la plataforma TwilioPlatform en un Chatbot.

```

1 botConfiguration.addProperty("xatkit.twilio.username",
2     userName);
3 botConfiguration.addProperty("xatkit.twilio.auth.token",
4     authToken);

```

Fragmento de código 4.9: Credenciales de Twilio.

```

1 estadoSaludo
2     .body(context->twilioPlatform.reply(context,"Hola"))
3     .next()
4     .moveTo(estadoEspera);

```

Fragmento de código 4.10: Código para enviar un mensaje de texto mediante el twilioPlatform desde un estado.

4.3. Caso de Estudio

En este apartado se describe el diseño e implementación de un *chatbot* como prueba de concepto de la propuesta dirigida por modelos para la generación de agentes conversacionales presentada en esta tesis. La funcionalidad implementada en la plataforma Xatkit Chat Platform permite desarrollar *bots* para la aplicación de mensajería instantánea WhatsApp y React. Además, a través del modelo de lenguaje autorregresivo GPT-3, es posible la generación de sugerencias a preguntas del usuario no consideradas en la base de conocimiento del agente. Para ejemplificar la propuesta, se decidió desarrollar un *bot* asistente que auxilie al usuario que solicita información sobre el Posgrado en Ciencias de la Información (PCI) de la Universidad Autónoma de Sinaloa.

4.3.1. Contextualización del caso de estudio

Actualmente, la Universidad Autónoma de Sinaloa, oferta Maestría y Doctorado dentro del Posgrado en Ciencias de la Información (PCI). El estudiante puede seleccionar entre dos áreas del conocimiento, la primera de ellas es “Computación y Sistemas” y la segunda es “Geomática y Geodesia”. El programa se encuentra inscrito en el Sistema Nacional de Posgrados (SNP) del Consejo Nacional de Ciencia y Tecnología (CONACYT).

La administración del PCI lleva a cabo un proceso de admisión para nuevos alumnos una vez por año, previo a esta etapa se incrementa el interés de la institución en publicitar su oferta educativa. A pesar de esto, a razón del interés de los posibles alumnos, independientemente de las fechas de convocatoria oficial de ingreso al PCI, se necesita una fuente de información en cualquier época del año que sea accesible para los interesados. Hoy en día, el posgrado cuenta con una página de Facebook y un sitio web para consulta de información [?]. Lamentablemente, para el interesado suele ser insuficiente el mecanismo brindado, o no puede encontrar la información precisa que necesita. En este sentido, un asistente conversacional puede solucionar esta problemática de disponibilidad de la información, permitiendo que este accesible todos los días del año sin importar la hora. Debido a lo mencionado anteriormente, en este caso de estudio se detalla el diseño, desarrollo, implementación y evaluación de un agente conversacional que auxilie al usuario con la información acerca del posgrado. A través de este caso de estudio se pretende:

- Implementar en la plataforma de WhatsApp un *chatbot* en un entorno real. Esto mediante el *framework* Xatkit y la propuesta MDD para la generación de agentes conversacionales presentada en esta tesis.
- Evaluar la satisfacción de los usuarios y sus preferencias al interactuar con el agente conversacional del PCI para mensajería instantánea por la plataforma WhatsApp a través de una encuesta de satisfacción.
- Evaluar la satisfacción del usuario acerca de las sugerencias recibidas por parte del *chatbot* en el caso de no contar con el conocimiento para contestar la pregunta realizada. Esto a través del método propuesto en esta tesis para generar sugerencias al usuario y lograr retomar el flujo de conversación mediante el modelo de lenguaje natural autorregresivo GPT-3.

4.3.2. Participantes

Con el propósito de llevar a cabo la validación de la propuesta presentada en esta tesis, en el caso de estudio se contempló a 100 usuarios (interesados en ingresar al PCI) para que interactuaran con el agente conversacional. El perfil de usuario para este ejercicio corresponde a estudiantes de licenciatura y posgrado, cuya área de estudio se relaciona con el perfil de ingreso al programa del PCI. Ésto se definió así debido a que los posibles futuros estudiantes serán el consumidor directo del *chatbot* del posgrado.

Para recolectar información de los participantes se proporcionó una encuesta a los usuarios. Adicionalmente, se registraron todas las interacciones (mensajes enviados y recibidos) entre el usuario y el agente conversacional durante todo el periodo de tiempo que permanezca en línea interactuando con el usuario. Por último, con el objetivo de no influir en el comportamiento del participante durante el uso del *bot*, no se le proporcionó información sobre el PCI, solo el número de teléfono por el cual es posible comunicarse con el *chatbot* a través del servicio de mensajería instantánea WhatsApp. Además del enlace a la encuesta, sin incluir ningún detalle del funcionamiento del agente conversacional.

4.3.3. Diseño del Chatbot

En este apartado se muestra el diseño de los componentes que conforman el agente conversacional que se implementó en este caso de estudio. Para su diseño, se crearon cuatro categorías, esto para llevar a cabo un enfoque estratégico planificando el diseño estableciendo una jerarquía de componentes. Las categorías, fueron definidas para este proyecto en particular, su objetivo es validar las tecnologías y métodos propuestos en la tesis, las categorías son las siguientes:

- Elementos conceptuales.
- Elementos funcionales.
- Elementos tecnológicos.
- Definición de *prompt*.

Elementos conceptuales

Los elementos conceptuales corresponden a la definición de las características del agente conversacional, tales como su objetivo de implementación, el perfil de usuarios con los que interactuará, de que forma llevará a cabo su comunicación con el usuario (texto e imágenes) y el deseo o intención del usuario al interactuar con el *bot*. Con base en ello, se desarrollan los elementos en las categorías posteriores, las cuales sirven para definir la esencia del *chatbot*.

- Objetivo

El agente conversacional, tiene como objetivo brindar información de forma rápida acerca del Posgrado en Ciencias de la Información a cualquier usuario, en especial a aquel que sea un alumno potencial del programa. Concretamente, es responder de forma confiable y rápida a las consultas realizadas por el usuario.

- Usuarios

Los principales usuarios del *chatbot* son las personas que hayan terminado o se encuentren próximos a finalizar sus estudios de licenciatura y estén interesados en

ingresar a un programa de maestría o doctorado en el Posgrado en Ciencias de la Información. El rango de edad en promedio es para personas adultas mayores de 21 años. Por las características del usuario, la mayoría debería estar familiarizado con el uso de software, sin embargo, el agente tiene la capacidad para ser de utilidad para cualquier persona que tenga la facultad de leer y escribir en español indistintamente de su experiencia con el uso de la tecnología.

- Identidad del *chatbot*

El *bot* es un asistente cordial y contesta formalmente a la mayoría de las preguntas, no tiene un avatar y su imagen es el logo del PCI. El agente, siempre contesta a cualquier consulta del usuario, en caso de no poder resolver esta debe disculparse. Además, en caso de ser posible, ofrecer una respuesta alternativa útil.

- Comunicación

El agente tendrá la capacidad de expresarse a través de texto e imágenes, así como *emojis*, las cuales son una imagen o icono digital que se usa en las comunicaciones electrónicas para representar una emoción, un objeto, una idea, etc. [78]. El idioma que utilizará será solamente el español, no se contempla recibir consultas en otro idioma diferente al español al momento de definir los *intents*, frases de entrenamiento y respuestas. El usuario podrá comunicarse con el agente solamente a través de texto.

Elementos funcionales

Los elementos funcionales son las actividades que debe ser capaz de realizar el *chatbot* en cada escenario para cumplir con su objetivo principal. Los escenarios que debe ser capaz de manejar cuando interactúa con un usuario vienen definidos en sus *intents*, los cuales son un conjunto de expresiones que reflejan un propósito o un deseo del usuario, el cual se desea transmitir al agente conversacional. Para este caso de estudio, el *bot* no tiene la necesidad de desempeñar ninguna tarea automáticamente más allá de resolver las consultas que le sean solicitadas.

Actualmente, la mayoría de interesados en ingresar al PCI obtiene información a tra-

vés del sitio web oficial. El SNP, demanda que los posgrados ofrezcan un sitio web con una serie de requerimientos para ser considerados en su padrón de calidad, el PCI lo cumple, es decir, contiene datos confiables y actualizados del programa [79]. Por tal motivo, la mayor parte de los *intents* que se definieron para el agente conversacional de este caso de estudio fueron diseñados para hacer uso de esa información y, por medio de ésta, contestar correctamente las posibles consultas que puedan realizar los candidatos en ingresar al posgrado.

Los *intents* definidos en este caso de estudio están divididos en 2 categorías: los propios del *bot* y los relacionados a ofrecer información acerca del posgrado. Se diseñaron un total de 36 *intents*, las acciones que desencadenan se dividen en respuestas de texto y respuestas multimedia. Además, diversas intenciones tienen más de una respuesta posible, la cual se selecciona dependiendo de ciertas entidades que se extraen de la consulta del usuario. Las respuestas a las consultas son extraídas, como se mencionó anteriormente, del sitio web oficial del PCI y de los representantes (personal administrativo) de la institución. A continuación, se listan 30 de los 36 *intents* referentes al posgrado de este caso de estudio. Posteriormente, se muestran los 6 restantes referentes a la funcionalidad del *chatbot*:

Intents definidos con la funcionalidad necesaria para dar información acerca del posgrado.

1. Información acerca del posgrado.
2. Para qué sirve el posgrado.
3. Perfil de egreso.
4. Duración del posgrado.
5. Requisitos.
6. Proceso de preinscripción.
7. Recepción de documentos.
8. Proceso de inscripción.

9. Costos y procesos de becas.
10. Existencia de beca.
11. Resultados de admisión.
12. Información de entrevista.
13. Modalidad del posgrado.
14. Modalidad de exámenes.
15. Fecha de preinscripciones.
16. Fecha recepción de documentos.
17. Fecha curso propedéutico y examen de conocimientos.
18. Fecha de examen psicométrico.
19. Fecha de examen CENEVAL.
20. Inicio de actividades.
21. Frecuencia de convocatoria.
22. Información de movilidad.
23. Líneas de investigación.
24. Plan de estudios.
25. Información de docentes.
26. Información de exámenes.
27. Datos de contacto.
28. Dirección.
29. Guías examen.

30. Información de titulación.

Intents creados para dotar de funcionalidad al agente conversacional.

1. Saludo.
2. Información del *bot*.
3. Información general.
4. Ofrecer ayuda.
5. Respuesta Negativa.
6. Información sobre el desarrollo del *bot*.

Elementos tecnológicos

En esta sección se detalla el ecosistema tecnológico utilizado para definir, desarrollar e implementar el agente conversacional para este caso de estudio. Se detalla el sistema operativo, plataformas, lenguajes de programación, *frameworks*, etc.

- Sistema operativo: Ubuntu 20.04 (LTS) x64.
- Lenguaje de programación: Java.
- *Framework*: Xatkit.
- Servicio de procesamiento de lenguaje natural: Dialogflow / NLP.JS.
- *Middleware*: Twilio API.
- Servicio de mensajería instantánea: WhatsApp.
- Alternativa de consultas de usuario: GPT-3 modelo text-curie-001, TheoKanning/openai-java.
- Equipo de cómputo: 4 GB Memoria RAM, 2 AMD vCPUs 1996MHz compartido, 80 GB de almacenamiento NVMe SSD.

Para la implementación se optó por el sistema operativo Ubuntu 20.04 (LTS) x64 con la finalidad de igualar el caso de estudio a un ambiente de ejecución real en un proceso de desarrollo habitual. Ubuntu suele utilizarse como servidor web para este tipo de proyectos. También, se utilizó el *framework* Xatkit y la implementación de la plataforma xatkit-whatsapp-via-twilio-platform, propuesta en esta tesis, para desarrollar el agente conversacional. Así mismo, se utilizó el lenguaje de programación Java, en su versión 11; el motivo principal de dicha elección radica en que esa versión es la requerida por TheoKanning/openai-java para comunicarse con OpenAi API y así utilizar GPT-3.

Una de las ventajas que ofrece Xatkit es la facilidad de integrar y cambiar el proveedor de procesamiento de lenguaje natural en cualquier etapa de desarrollo, esto es debido a las técnicas MDD que utiliza. Por tal motivo, no es necesario designar uno específico en tiempo de diseño. Esto tiene como ventaja que es posible experimentar una vez se encuentre desarrollado el agente, lo que proporcionará la posibilidad de elegir la opción que se ajuste a las necesidades del desarrollador o por otra parte la que demuestre un mejor rendimiento para el caso de uso particular. Por lo anterior, en este apartado del caso de estudio, queda indefinido en esta etapa del desarrollo el proveedor de lenguaje natural. Posteriormente, se seleccionará alguno de los dos proveedores disponibles en Xatkit: Dialogflow y NLP.JS. No se consideraron el analizador de expresiones regulares, debido ser solo para testeo y, Xatkit NLU *Engine* porque a que aún se encuentra en fase experimental.

Definición del *prompt*

En el contexto de la inteligencia artificial y los agentes conversacionales, un *prompt* se refiere a una frase o pregunta que se le proporciona al sistema para que genere una respuesta relevante. El *prompt* desarrollado para este caso de estudio se presenta en la Tabla 4.1, se diseño a través de las pautas descritas en la Sección 4.2.3 de esta tesis.

En un agente conversacional, el *prompt* debe contener cuatro elementos, estos son: una lista de opciones, instrucciones detalladas, una lista de ejemplos y la consulta del usuario. En este caso de estudio la lista de opciones se creó a partir de las frases de entrenamiento del *chatbot* y solo se utilizó una frase de entrenamiento por cada *intent*. Esto debido a

que incluir más sería redundante y aumentaría la cantidad de *tokens* (forma de división de texto utilizada por GPT-3, la cantidad varía según la palabra encontrada) utilizados debido al incremento de texto, lo que ocasionaría que se incremente el costo y tiempo de procesamiento. Las instrucciones y ejemplos proporcionados en el proceso de diseño del *prompt* para este caso de estudio son suficientes para garantizar que el formato deseado del texto generado sea claro y preciso. Por lo tanto, al diseñar un *prompt* utilizando las pautas descritas en la sección de propuesta de generación de sugerencias con GPT-3 se recomienda utilizar, para una mejor comprensión del *chatbot*, sólo una opción para cada *intent*. Además, se especificó un texto que describía las pautas que GPT-3 debía tomar en cuenta para generar el texto y la lista de ejemplos se conforma de un conjunto de consultas hipotéticas que un usuario puede realizar con una respuesta que sea adecuada para cada consulta.

El *prompt* resultado se muestra en la Tabla 4.1. A cada cadena de texto que corresponde con una pregunta en el *prompt* se le añadirá el mensaje del usuario cuando sea necesario y la cadena “sugerencia:”. Para continuar con el formato deseado, se experimentó con el uso de ligeras modificaciones como eliminar los signos de puntuación y las instrucciones. Esta modificación, solo provocó ligeros cambios en la generación de texto, principalmente logró reducir la cantidad de *tokens*. En total, se obtuvo un estimado de 1,376 *tokens*. Este hecho ocasionó que se optará por utilizar el motor de GPT-3 llamado Curie al ser más rápido y de menor costo por consumo en comparación al motor más potente llamado Davinci. Al no representar una mejora significativa, se decidió mantenerlo sin modificaciones significativas para el experimento. El ejemplo se detalla en la Tabla 4.2.

4.3.4. Desarrollo del Chatbot

En esta sección se describe el proceso de desarrollo de los componentes del agente conversacional propuesto para este caso de estudio. El código fuente completo del proyecto del *chatbot* llamado PCIBot puede encontrarse en [80]. Es importante mencionar que se ocultaron algunos elementos tales como las credenciales de verificación de OpenAi, DialogFlow y Twilio por cuestiones de seguridad.

Tabla 4.1: Lista de opciones de *prompt*.

Sugerencia de cadena de texto en el <i>prompt</i>
¿Tienes una guía del examen?
¿Dónde están ubicados?
Datos de Contacto
Información sobre los docentes
Cuál es el plan de estudios del programa de maestría
Cuáles son las líneas de investigación
Información sobre movilidad
¿Quién te programó?
Información frecuencia de la convocatoria
Cuál es la fecha de inicio de actividades
Cuál es la fecha del Curso propedéutico y examen de conocimientos
Información sobre fecha de recepción de documentos
Información sobre fecha de preinscripciones
Información sobre la modalidad de los exámenes
Información sobre la modalidad del posgrado
Información sobre entrevista
¿Cuándo obtengo los resultados de admisión?
Información sobre becas
¿Cuál es el proceso de titulación?
¿Qué tengo que hacer para entrar al posgrado?
Información sobre proceso de preinscripción.
¿Cuáles son los requisitos para ingresar a maestría?
Información sobre duración del programa de posgrado
Información acerca del posgrado.
Información sobre costos
¿Cómo te hicieron?
Cuál es la fecha del curso propedéutico y examen de conocimientos

El primer paso de la implementación de este caso de estudio consistió en utilizar el metamodelo de XatbotMM (Fig. 4.1) para definir el modelo del agente conversacional creado para el PCI. Esto se llevó a cabo a través del Eclipse Modeling Framework utilizando el lenguaje Ecore. En este modelo se definieron los estados, *intents*, transiciones y acciones que utilizará el *bot*. Además, se especifica, entre las propiedades del modelo, la plataforma y el proveedor de procesamiento de lenguaje natural a utilizar, el cual puede ser TwilioPlatform o DialogFlow, para este *bot* se seleccionó TwilioPlatform. El modelo resultante se observa en la Figura 4.4.

Una vez creado el modelo del agente PCIBot, se procedió con la ejecución de la plan-

Tabla 4.2: Ejemplo de sugerencias.

Instrucciones
Pregunta al usuario la opción que puede interesarle de la Lista de opciones según su mensaje en la conversación, de la misma forma que en el Ejemplo . Solo puedes contestar con ejemplos de la Lista de opciones y debes realizar una sugerencia que tenga relación con el mensaje del usuario. Si no puedes ofrecer una recomendación acorde o útil de la Lista de opciones responde con un “ <i>Lo siento no tengo una sugerencia para mostrar</i> ”
Ejemplos:
<i>usuario:</i> ¿da clases el profe Carlos? <i>sugerencia:</i> si te interesa puedes intentar escribir: <i>Información sobre los docentes</i>
<i>usuario:</i> hora <i>sugerencia:</i> si te interesa puedes intentar escribir: <i>Cuál es la fecha de inicio de actividades</i>
<i>usuario:</i> pero no tengo dinero <i>sugerencia:</i> si te interesa puedes intentar escribir: <i>Información sobre becas</i>
<i>usuario:</i> no entendí que tengo que hacer <i>sugerencia:</i> si te interesa puedes intentar escribir: <i>¿Que tengo que hacer para entrar al posgrado?</i>
<i>usuario:</i> ¿me queda muy lejos? <i>sugerencia:</i> si te interesa puedes intentar escribir: <i>¿Dónde están ubicados?</i>
<i>usuario:</i> es difícil hablar japones? <i>sugerencia:</i> <i>lo siento no tengo una sugerencia para mostrar</i>
<i>usuario:</i> esta muy caro? <i>sugerencia:</i> si te interesa puedes intentar escribir: <i>Información sobre costos</i>
<i>usuario:</i> is expensive? <i>sugerencia:</i> si te interesa puedes intentar escribir: <i>Información sobre costos</i>
<i>usuario:</i> Información sobre perros <i>sugerencia:</i> <i>Lo siento no tengo una sugerencia para mostrar</i>
<i>usuario:</i> Is it too far from my house? <i>sugerencia:</i> si te interesa puedes intentar escribir: <i>¿Dónde están ubicados?</i>
<i>usuario:</i> quiero sacar un kardex <i>sugerencia:</i> si te interesa puedes intentar escribir: <i>Datos de Contacto</i>
<i>usuario:</i> What are the requirements to enter a master’s degree? <i>sugerencia:</i> si te interesa puedes intentar escribir: <i>¿Cuáles son los requisitos para ingresar a maestría?</i>
<i>usuario:</i> te quiero <i>sugerencia:</i> <i>Lo siento no tengo una sugerencia para mostrar</i>
<i>usuario:</i> estoy aburrido <i>sugerencia:</i> si te interesa puedes intentar escribir: <i>Información sobre movilidad</i>
<i>usuario:</i> I don’t know what to do <i>sugerencia:</i> si te interesa puedes intentar escribir: <i>Información general</i>
<i>usuario:</i> ¿que pasa si no puedo ir a hacer el examen? <i>sugerencia:</i> si te interesa puedes intentar escribir: <i>Información sobre la modalidad de los exámenes</i>
<i>usuario:</i> quiero escuchar música <i>sugerencia:</i> <i>Lo siento no tengo una sugerencia para mostrar</i>

tilla de Acceleo M2T para generar la mayor parte del código fuente para el *framework* Xatkit Chat Platform. Mediante la plantilla para transformaciones M2T se logró semi-automatizar la construcción del agente conversacional, evitando programarlo desde cero, lo que permitió un ahorro de tiempo en su desarrollo. Cabe destacar que el Chatbot generado a partir del modelo es implementable en un entorno real, pero, debido a que se decidió incluir una serie de características y funcionalidades extra, como el uso de entidades para respuestas personalizadas y la implementación del servicio de GPT-3 para la generación de sugerencias, fue necesario intervención manual mínima en el código fuente.

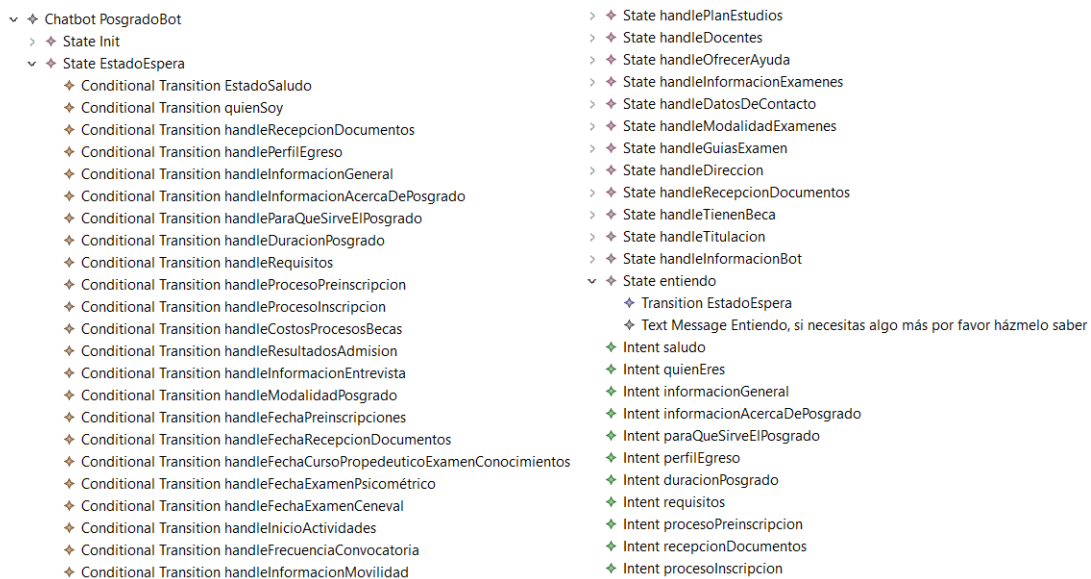


Figura 4.4: Fragmento del modelo del PCIBot.

El siguiente paso en el proceso de desarrollo del agente conversacional para este caso de estudio consistió en la instalación del *framework* Xatkit Chat Platform. Para hacerlo, se consultó su documentación oficial [81], en donde se encontró, como requerimiento previo, instalar Java Development Kit (JDK), Maven (*framework* de gestión de proyectos de software, es un modelo estándar de gestión y descripción de proyectos) y Git (software de código abierto para el control de versiones de código fuente). En el Fragmento de código 4.11, se detallan los comandos necesarios para descargar del repositorio la plataforma Xatkit e instalarlo vía línea de comandos.

```

1 git clone
2 https://github.com/xatkit-bot-platform/xatkit.git
3 cd xatkit

```

```

4 git submodule update --init --recursive
5
6 mvn clean install -DskipTests

```

Fragmento de código 4.11: Instalación de Xatkit.

Después de tener configurado el ambiente de desarrollo se procedió a crear un proyecto Maven. Para esto, se tomó como base el proyecto de ejemplo de Xatkit Chat Platform. Posteriormente, se procedió a implementar todos los *intents* de la Tabla 4.1 por medio del DSL de Xatkit. Para hacerlo, se creó una variable para cada uno, utilizando la función *intent()* y definiendo un conjunto de oraciones de entrenamiento. Cada una de ellas es una forma en la cual el usuario podría referirse a una intención (pregunta) específica y el *bot* será capaz de clasificar el texto en alguno de los *intents* disponibles. En el Fragmento de código 4.13 se muestra la definición de frases de entrenamiento del agente conversacional creado en este caso de estudio.

Con lo que respecta a la integración de la capacidad de respuesta a mensajes del usuario que no se encontraban en los *intents* del agente conversacional, se implementó aprendizaje profundo, a través de la puesta en funcionamiento del modelo de lenguaje autorregresivo GPT-3. La librería *theokanning.openai-gpt3-java* API GPT-3 de OpenAI permite realizar la integración y generar texto automáticamente para responder al usuario del *bot*. Después, se importaron las librerías que permiten comunicarse con los proveedores de procesamiento de lenguaje natural Dialogflow y NLP.js. Finalmente, se integró la capacidad de ejecución del agente en WhatsApp utilizando Twilio.

```

1 \begin{lstlisting}[language=Java,frame=none]
2 val saludo = intent("saludo")
3     .trainingSentence("Hola")
4     .trainingSentence("Buen día")
5     .trainingSentence("buenas tardes")
6     .trainingSentence("que onda")
7     .trainingSentence("que tal");

```

Fragmento de código 4.12: Definición de frases de entrenamiento para el intent saludo.

También, se crearon *entities* (Fragmento de código 4.14), con el objetivo de incrementar la capacidad de respuesta del agente conversacional y dotarlo de la capacidad de conocer si el mensaje del usuario estaba solicitando información sobre un programa de estudios en particular. Con esto, se logró personalizar la respuesta del *bot* del PCI y no

agobiar al usuario con información que no solicitó. Con las *entities*, es posible extraer información del mensaje del usuario.

```
1 val ePrograma = mapping("PPROGRAMA")
2     .entry()
3     .value("maestria").synonym("master")
4     .entry()
5     .value("doctorado")
6     .synonym("DOC");
```

Fragmento de código 4.13: Definición de entidades.

```
1 val perfilEgreso = intent("perfilEgreso")
2     .trainingSentence("Cual es el perfil de egreso "+
3         "de un alumno de maestría")
4     )
5     .trainingSentence("que obtengo al "+
6         "estudiar la maestría")
7     )
8     .trainingSentence("De que me sirve "+
9         "estudiar el maestría")
10    )
11    .trainingSentence("como sale un" +
12        "egresado de maestría")
13    )
14    .parameter("PPROGRAMA")
15        .fromFragment("maestria")
16        .entity(ePrograma);
```

Fragmento de código 4.14: Extracción de información clave con el uso de entidades .

Posteriormente, se instanció la plataforma de Twilio y el TwilioEventProvider para que el agente conversacional pueda recibir entradas (mensajes) del usuario y ejecutar las acciones de enviar un mensaje de texto o multimedia como respuesta, como se observa en el Fragmento de código 4.15. Cabe destacar que el código fuente para la definición de *intents*, estados, transiciones y acciones fue generado automáticamente a partir del modelo mostrado en la sección 4.2 de la tesis.

```
1 TwilioPlatform twilioPlatform = new TwilioPlatform();
2 TwilioEventProvider twilioEventProvider
3     = new TwilioEventProvider(twilioPlatform);
```

Fragmento de código 4.15: Instancia de plataforma de Twilio.

Continuando con la implementación del agente conversacional llamado PCIBot, se procedió a definir los estados en los que el *bot* puede encontrarse, así como las transiciones

para moverse de uno a otro. Esto se debe a que los *chatbots* en Xatkit Chat Platform funcionan como máquinas de estados, con lo cual pueden ser eventos de todo tipo, por mencionar un ejemplo, se encuentra el caso en que se produce la detección de un *intent* proveniente del mensaje del usuario. La declaración de estados en PCIBot y la transición entre ellos se ejemplifica en los Fragmentos de código 4.16 y 4.17.

```
1 val init = state("Init");
2 val estadoEspera = state("EstadoEspera");
3 val estadoSaludo = state("EstadoSaludo");
```

Fragmento de código 4.16: Declaración de estados.

```
1 init
2     .next()
3     .moveTo(estadoEspera);
4
5 estadoEspera
6     .next()
7     .when(intentIs(saludo)).moveTo(estadoSaludo)
```

Fragmento de código 4.17: Transiciones entre estados.

Cada uno de los pasos explicados anteriormente permitió establecer las respuestas para cada consulta definida que se le podría hacer al PCIBOt. Se programaron un total de 42 estados con sus acciones y transiciones.

En el caso particular, cuando el proveedor de procesamiento de lenguaje natural del agente conversacional no tenga una respuesta registrada en sus *intents*, existe un estado especial, llamado *defaultFallback*. Su finalidad es permitir al desarrollador programar una respuesta predefinida para indicarle al usuario que la solicitud no se pudo resolver, pero que brindará una recomendación relacionada con está utilizando el método propuesto en la Sección 4.2.3. Para este caso de estudio, la implementación de esta funcionalidad se llevó a cabo de la siguiente forma: se definió una variable de tipo *String*, o cadena de texto, llamada *promptVar*, para inicializar el servicio de OpenAi. Una vez hecho esto se procede a definir un arreglo de opciones de respuesta, como se observa en el Fragmento de código 4.18.

```
1 OpenAiService service =
2     new OpenAiService(credencialOpenAi);
3 ArrayList<String> storyArray = new ArrayList<String>();
4 String promptVar="Lista de opciones:\n"+
```



```
5 "Tienes una guía del examen?\n" + ...
```

Fragmento de código 4.18: Instancia del servicio de OpenAI.

A continuación, el *defaultFallback* recuperará el último mensaje del usuario utilizando la función *getMatchedInput()*. El mensaje, será concatenado al final de *promptVar* y se concatenará la cadena “sugerencia:” para continuar con el formato deseado descrito en las instrucciones y ejemplos del *prompt*. La codificación de este paso se detalla en el Fragmento de código 4.19.

```
1 String textoUsuario = context
2     . getIntent ()
3     . getMatchedInput ();
4 String promptMasTextoUsuario = promptVar
5     + textoUsuario
6     + "\nsugerencia: ";
7 storyArray . clear ();
```

Fragmento de código 4.19: Extracción del mensaje del usuario.

El siguiente paso, consistió en utilizar el servicio y generar texto que servirá para ser enviado al usuario como respuesta, éste se registrará en un archivo. Antes de hacerlo, se debe de realizar un filtrado del texto debido a que existen ocasiones en las que el modelo no será capaz de generar una recomendación adecuada. Para este caso de estudio que implementa el PCIBot, se procedió a realizar el filtrado de cadenas de texto que no superan los 25 caracteres, consultar el Fragmento de código 4.20.

```
1 CompletionRequest completionRequest
2     = CompletionRequest . builder ()
3         . promptVar ( promptMasTextoUsuario )
4         . echo ( false )
5         . maxTokens ( 96 )
6         . temperature ( 0.1 )
7         . stop ( stopList )
8         . build ();
9 service . createCompletion ( "text-curie-001" ,
10 completionRequest ) . getChoices () . forEach ( line ->
11 { storyArray . add ( line . getText () ); System . out . println ( line );
12 } );
13
14 if ( storyArray . get ( 0 ) . length () > 25 ) {
15     twilioPlatform . reply ( context , storyArray . get ( 0 ) );
16     sugerenciaGuardar = "\n" + textoUsuario +
17         "\n" + storyArray . get ( 0 );
```

```

18     guardarFichero ( sugerenciaGuardar );
19 } else {
20     sugerenciaGuardar = "\n"+textoUsuario+"\n"+ "-----";
21     guardarFichero ( sugerenciaGuardar );
22 }

```

Fragmento de código 4.20: Generación de sugerencia con el servicio de OpenAI.

Finalmente, una vez implementadas las capacidades del agente conversacional, el comportamiento del agente quedó completamente definido. Por lo que se procedió a a realizar los últimos ajustes de configuración: indicarle al *bot* las plataformas que va a utilizar, el servicio de procesamiento de lenguaje natural que se va a encargar de detectar los *intents* del usuario y las credenciales de los servicios a emplear. En el Fragmento de código 4.21 se muestra la codificación de la configuración final del agente.

```

1 val botModel = model()
2     .usePlatform ( twilioPlatform )
3     .listenTo ( twilioEventProvider )
4     .initState ( init )
5     .defaultFallbackState ( defaultFallback );
6
7 Configuration botConfiguration = new BaseConfiguration ();
8
9 botConfiguration . addProperty ( " xatkit . twilio . username " ,
10     userName );
11 botConfiguration . addProperty ( " xatkit . twilio . auth . token " ,
12     authToken );
13
14 botConfiguration . addProperty (
15     IntentRecognitionProviderFactory . INTENT_PROVIDER_KEY ,
16     DialogFlowConfiguration
17         . DIALOGFLOW_INTENT_PROVIDER );
18
19 botConfiguration . addProperty (
20     DialogFlowConfiguration . PROJECT_ID_KEY ,
21     " pruebaxatkit - sipv " );
22
23 botConfiguration . addProperty (
24     DialogFlowConfiguration . GOOGLE_CREDENTIALS_PATH_KEY ,
25     " xDialog . json " );
26
27 botConfiguration . addProperty (
28     DialogFlowConfiguration . LANGUAGE_CODE_KEY ,
29     " es " );
30 botConfiguration . addProperty (

```

```

31     DialogFlowConfiguration.CLEAN_AGENT_ON_STARTUP_KEY,
32     true);
33
34 botConfiguration.addProperty(
35     IntentRecognitionProviderFactory.INTENT_PROVIDER_KEY,
36     NlpjsConfiguration.NLPJS_INTENT_PROVIDER);
37
38 botConfiguration.addProperty(
39     NlpjsConfiguration.AGENT_ID_KEY,
40     "default");
41
42 botConfiguration.addProperty(
43     NlpjsConfiguration.LANGUAGE_CODE_KEY,
44     "es");
45 botConfiguration.addProperty(NlpjsConfiguration
46     .NLPJS_SERVER_KEY,
47     "http://localhost:8080");
48
49 XatkitBot xatkitBot = new XatkitBot(botModel,
50     botConfiguration);
51 xatkitBot.run();

```

Fragmento de código 4.21: Configuraciones del Chatbot.

El proceso de desarrollo del agente conversacional se simplificó gracias al uso de la propuesta presentada en esta tesis. El *framework* Xatkit Chat Platform extendió su funcionalidad gracias al metamodelo XatbotMM, la plantilla para transformaciones M2T en Acceleo y la implementación de la plataforma de Twilio. Con esta tecnología, es posible que Xatkit Chat Platform pueda permitir al equipo de desarrollo concentrarse en la fase de diseño conceptual de su agente conversacional. Esto es posible debido a que la implementación de los principios de MDD permiten agilizar la codificación, es decir, reducir el tiempo de desarrollo.

4.3.5. Implementación del Chatbot

Una vez realizado el proceso de codificación del agente conversacional se procedió con su implementación. En primer lugar, se instaló Xatkit Chat Platform en un sistema operativo Linux Ubuntu. Posteriormente, como se tenía considerado en la fase de diseño, se utilizó el servicio de hospedaje de DigitalOcean para crear una cuenta de WhatsApp Business. Después, se asignó un número de teléfono al servicio de Twilio para poder

comunicar al usuario con el *Chatbot*.

4.3.6. Evaluación

En este apartado se detalla la evaluación que se realizó al agente conversacional PCIBot. Ésta, consistió principalmente en analizar la calidad de las respuestas del agente conversacional y las recomendaciones de GPT-3. Se buscó obtener la percepción del usuario al momento de interactuar con el *bot*. Adicionalmente, se estudió la forma como los usuarios interactuaron con él cuando no fue capaz de resolver su consulta.

Para lograr los objetivos mencionados anteriormente, se inició la evaluación centrándose en la recolección de datos de los usuarios por medio de una encuesta. El agente conversacional PCIBot, fue creado para brindar información del Posgrado en Ciencias de la Información de la Universidad Autónoma de Sinaloa. Se implementó en la plataforma de mensajería instantánea más utilizada a nivel mundial, WhatsApp. Las preguntas de la encuesta se encuentran disponibles para su consulta en el Anexo B.

Para iniciar el caso de estudio, se compartió el número de teléfono de WhatsApp del PCIBot con 100 usuarios pertenecientes/egresados de cuatro universidades, la Universidad Autónoma de Sinaloa (UAS), la Universidad Politécnica de Sinaloa (UPSIN), la Universidad Autónoma de Occidente (UAdeO) y el Instituto Tecnológico de Mazatlán (ITMAZ). Estas instituciones tienen en común el área de estudio de la informática y sistemas computacionales en nivel licenciatura. En la Figura 4.5, se muestra el funcionamiento del *chatbot* interactuando con un usuario por medio de lenguaje natural en la aplicación de WhatsApp instalada en su teléfono. El PCIBot, es capaz de responder a la consulta mediante texto e imágenes. Posteriormente, se procedió a compartir una encuesta de satisfacción con los usuarios del agente conversacional. La encuesta se compartió a través de la herramienta Google Forms [82], una vez aplicada, por medio del análisis de las preguntas referentes a las conversaciones entre usuario y *bot*, se evaluaron los objetivos de este caso de estudio.

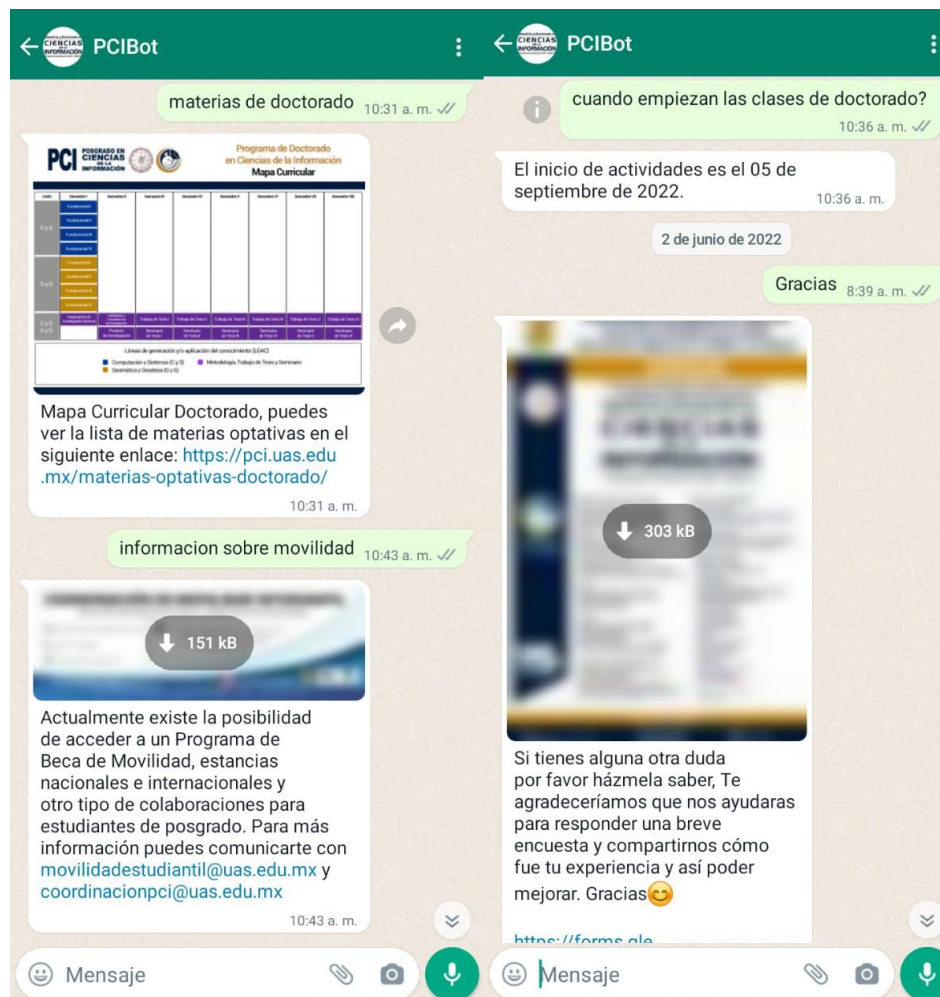


Figura 4.5: Captura de la interacción de PCIBot con un usuario.

4.3.7. Resultados

En esta sección se exponen los resultados de la evaluación realizada en este caso de estudio. Los datos obtenidos de la encuesta pueden consultarse en [83] y la encuesta en el Anexo B. A continuación se presenta cada una de las preguntas realizadas a los usuarios del PCIBot con una gráfica y análisis del resultado obtenido.

1. ¿Cuál es tu satisfacción general con el uso del Chatbot para obtener información sobre el Posgrado en Ciencias de la Información?

Como se puede observar en la Figura 4.6, si bien los resultados muestran una buena aceptación en general de los usuarios del PCIBot (66 %), existieron casos en los que el participante no estuvo del todo conforme con su experiencia. Realizando un

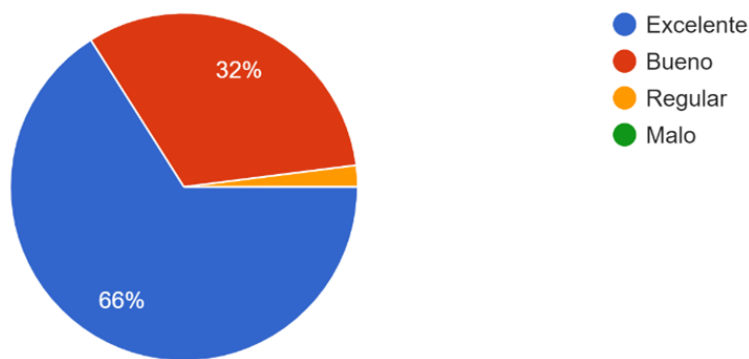


Figura 4.6: Satisfacción general con el uso del *Chatbot*.

análisis del texto registrado en la conversación (interacción PCIBot-usuario), se llegó a la conclusión que los usuarios que tuvieron un nivel de satisfacción regular es porque estaban buscando una función del agente no estaba disponible en el momento de utilizarlo. Es así, como en ocasiones, esto provoca que hagan intentos de solicitudes que el *bot* no pudo resolver. Se resalta el hecho de que el proceso de desarrollo de un *chatbot*, al igual que en el área del software, requiere de una etapa de mantenimiento en el que puede ser actualizado para adaptarse a las necesidades de los clientes.

2. ¿Qué tan difícil te resultó hacer consultas en el *chatbot*?

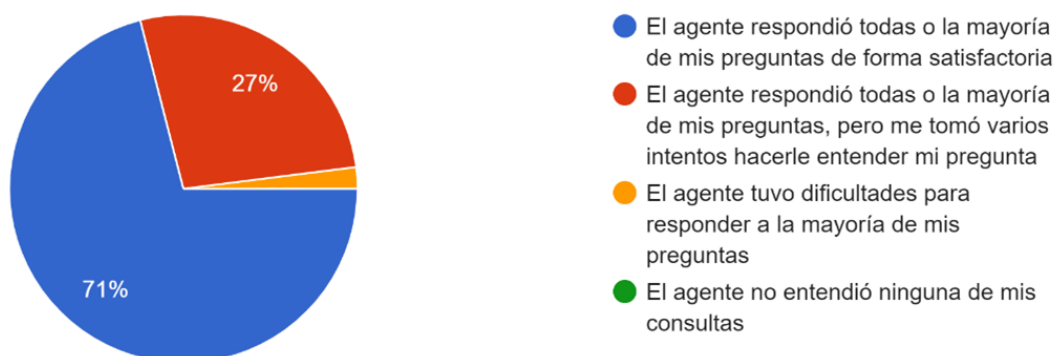


Figura 4.7: Dificultad de uso del *chatbot*.

Los resultados de esta pregunta como se aprecia en la Figura 4.7, son acordes a su

predecesora debido a que la mayoría de los usuarios valoran su nivel de satisfacción general respecto al agente conversacional con la facilidad y utilidad de uso. En este caso, la mayoría de los usuarios, es decir el 71 %, resolvieron sus solicitudes fácilmente.

Después de analizar los diálogos de la interacción PCIBot-Usuario, se observó que, en diversas ocasiones, los participantes pudieron encontrar una forma de resolver su consulta a través de las recomendaciones que ofreció el *bot*. A pesar de ello, existieron casos en los que el agente tuvo dificultades para responder las preguntas de los usuarios porque existieron conversaciones en las que el agente conversacional no fue capaz de clasificar adecuadamente la intención del usuario.

Esta pregunta resultó ser muy importante para considerar como trabajo futuro, mejorar esa interacción del PCIBot, de tal forma que se logre disminuir este tipo de eventos por medio de mejores secuencias de entrenamiento para el agente.

3. ¿Consideras cómodo y/o útil poder utilizar un *Chatbot* desde la plataforma de WhatsApp para obtener información sobre el Posgrado en Ciencias de la Información? ¿Por qué?

Los resultados de esta pregunta son concluyentes y muestran una clara aceptación de los participantes a este tipo de herramientas en la plataforma de WhatsApp (ver Fig. 4.8) con el 99 % de aceptación por parte de los encuestados. En los comentarios, los usuarios resaltaron lo práctico y fácil que resulta tener disponible esta solución las 24 horas del día. Incluso, sobresale en las respuestas el hecho de poder utilizar el PCIBot directamente desde su celular en una aplicación que utilizan prácticamente en el día a día. Además, los usuarios encuestados expresaron que el uso de un agente conversacional para obtener información del Posgrado en Ciencias de la Información es de gran utilidad y que su implementación va de acuerdo con el área estudio del posgrado. Incluso, que les permite obtener información de utilidad rápida y fácilmente en un horario fuera del horario de oficina, pues algunos interesados, solo tienen tiempo de buscar información por la noche y no durante su horario de trabajo.

El resultado de esta pregunta, sustenta la propuesta presentada en esta tesis y muestra la utilidad de crear herramientas que faciliten el desarrollo de *Chatbots* en las plataformas de mensajería más populares.

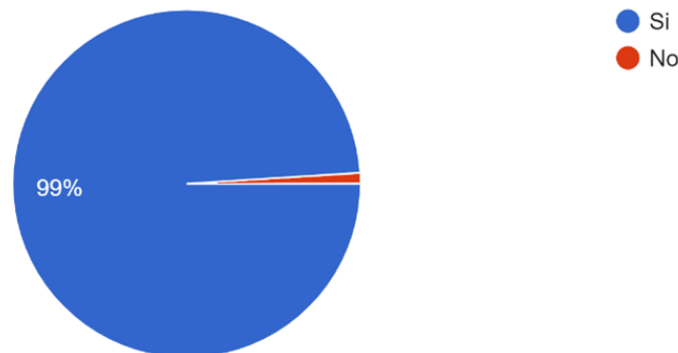


Figura 4.8: Aceptación de plataforma.

4. ¿Consideras a WhatsApp tu plataforma de mensajería instantánea preferida para interactuar con este tipo de herramientas de servicio al cliente?

La gran mayoría de los participantes, es decir, el 96 % de los encuestados, prefieren utilizar WhatsApp como plataforma de mensajería instantánea para interactuar con este tipo de agentes conversacionales, dejando atrás plataformas como Facebook (ver Fig. 4.9). Lo que demuestra la utilidad de una herramienta que facilite la generación e integración de agentes conversacionales con WhatsApp, como la propuesta desarrollada en este trabajo de investigación.

5. Si seleccionaste No ¿cuál es tu preferida?

La mayoría de los usuarios encuestados seleccionaron a la plataforma de mensajería instantánea WhatsApp como su medio preferido para interactuar con agentes conversacionales. Sin embargo, existen otras plataformas que son del agrado de los usuarios, en este caso de estudio, como se observa en la Figura 4.10, se detectaron un total de cuatro usuarios que seleccionaron Telegram y Facebook Messenger como sus plataformas preferidas. Este resultado es interesante, porque se pueden considerar ambas, como trabajo futuro, para ser implementadas al momento de de-

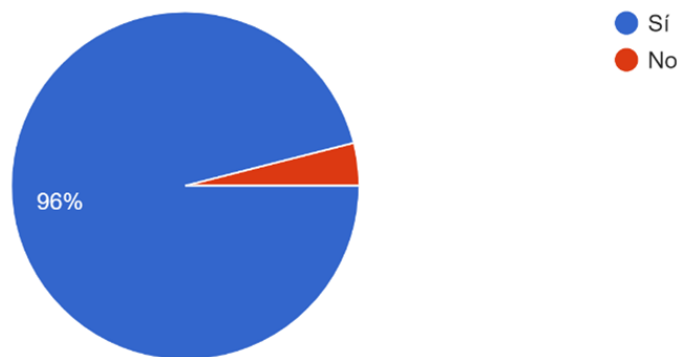


Figura 4.9: Preferencias de plataforma.

sarrollar un agente conversacional de servicio al cliente. Otras aplicaciones mencionadas, con menor porcentaje, son WeChat y Line, las cuales gozan de una gran popularidad en el mercado asiático [84].

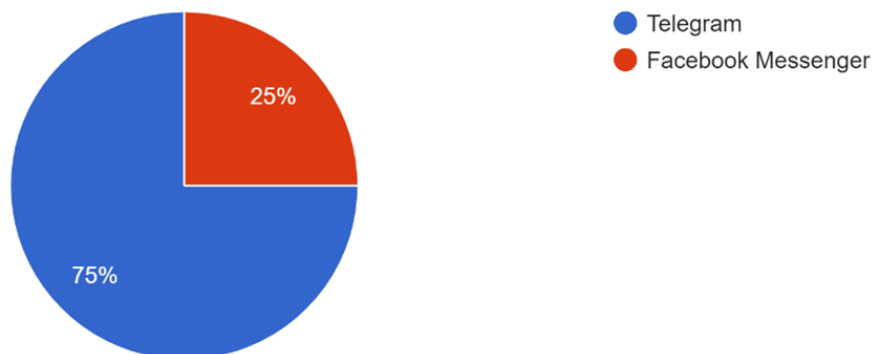


Figura 4.10: Otras opciones de plataformas.

6. Cuando el Chatbot del posgrado no entiende una pregunta regularmente ofrece una recomendación en el siguiente formato: (Perdón, no lo entiendo o no estoy capacitado para responder esa pregunta... si te interesa puedes intentar escribir:...)
Selecciona una de las siguientes afirmaciones:

En la Figura 4.11 se muestran los resultados de la encuesta a los participantes de Esta pregunta, tiene el objetivo de evaluar eficacia de las recomendaciones del modelo de lenguaje de procesamiento de lenguaje natural autorregresivo GPT-3, im-



Figura 4.11: Percepción de las recomendaciones.

plementado en la propuesta de este trabajo. Aquí, se demuestra que la mayoría de los usuarios estuvieron conformes con las sugerencias, sin embargo, al contrastar con los datos recolectados se detectó que la mayoría de los usuarios no hicieron uso de esta función. Esto significa que no se requirió el servicio de procesamiento de lenguaje natural DialogFlow y que el *bot* fue capaz de clasificar todas sus consultas. En las conversaciones de la interacción PCIBot con el usuario, se pueden observar diálogos en los que los usuarios aprovecharon esta cualidad para formular de nuevo su pregunta o para obtener información extra.

La percepción del usuario acerca de las recomendaciones recibidas por parte del PCIBot se encuentran en la Figura 4.11. El 55 % contestó que el agente le proporcionó recomendaciones de utilidad. El 36 % señaló que no recibió ninguna recomendación por parte del *bot* porque recibió la respuesta correcta a su pregunta. Solo el 8 % de los encuestados afirmaron que no recibieron una recomendación de utilidad por parte del agente.

Si bien es cierto que las recomendaciones ofrecidas auxiliaron al usuario cuando el agente conversacional no fue capaz de contestar una pregunta, se observó que los usuarios utilizaron esa capacidad del agente para volver a formular su pregunta, con lo cual, se logró que el *chatbot* la entendiera u ofreciera una alternativa relacionada con su petición.

A continuación, Tabla 4.3, se muestran extractos de las conversaciones de los usua-

Tabla 4.3: Ejemplos de recomendaciones en conversaciones con usuarios.

Usuario	Bot
quiénes son los directores de tesis del area de sistemas y computacion?	Si te interesa puedes intentar escribir: Información sobre los docentes
En que fechas se entrega la documentación?	Si te interesa puedes intentar escribir: Información sobre fecha de Recepción de documentos
queda muy lejos de mi casa?	si te interesa puedes intentar escribir: ¿Dónde están ubicados?
En que puedes ayudarme?	Si quieres puedes intentar escribir: Información sobre la modalidad de los exámenes
Dónde?	Si quieres puedes intentar escribir: Datos de Contacto
Horario	Si te interesa puedes intentar escribir: Cuál es la fecha de inicio de actividades
En qué me puedes ayudar?	Si quieres puedes intentar escribir: ¿Qué tengo que hacer para entrar al posgrado?
La universidad ofrece servicios extras?	si te interesa puedes intentar escribir: Información sobre movilidad

rios en los que se muestran ejemplos de las recomendaciones emitidas por el agente PCIBot cuando no entendió alguna consulta.

A pesar de que existieron casos de éxito al utilizar esta función del agente conversacional, tanto en la encuesta como en el análisis de los datos, se detectaron situaciones en las cuales las sugerencias no resultaron satisfactorias para los usuarios. En algunos casos, señalaron que, cuando se escribe una consulta muy alejada del entorno de la escuela y el posgrado, las sugerencias pueden tornarse confusas o erróneas. En este contexto, la mayoría de las ocasiones en las que el usuario intentaba conversar de un tema ajeno al PCI, el sistema de recomendaciones GPT-3 contestaba acertadamente con un *“Lo siento no tengo una sugerencia para mostrar”*, pero se presentaron incidentes en los que las sugerencias resultaron irrelevantes en el contexto manejado. Algunos de estos casos de sugerencias erróneas extraídas de los datos de las conversaciones entre el PCIBot y el usuario son mostradas en la Tabla 4.4.

Tabla 4.4: Ejemplos de sugerencias incorrectas en conversaciones con usuarios.

Usuario	Bot
Sabes tocar el piano?	si te interesa puedes intentar escribir: Información sobre movilidad
oye conozco a otro <i>bot</i> de mensajería interesante, Cleverbot, lo conoces?	si te interesa puedes intentar escribir: Ejemplo de conversación con Cleverbot
c++	Si te interesa puedes intentar escribir: Información sobre C++
league of legends	Si te gusta el juego de League of Legends, puedes intentar preguntar sobre él en la lista de preguntas
when are the inscriptions?	If you are interested, you can try to write: Information about the teachers
Calidad?	Si quieres puedes intentar escribir: ¿Cuál es la calidad del programa de maestría?

7. ¿Consideras que la capacidad de ofrecer recomendaciones que ayuden a retomar la conversación cuando el agente no es capaz de contestar una pregunta una cualidad útil de un *Chatbot*? ¿Por qué?

EL 95 % los usuarios consideran útil la cualidad de ofrecer recomendaciones por parte del PCIBot (ver Fig. 4.12). Entre los comentarios los usuarios encuestados, se encontró que algunos de ellos mostraron la inconformidad que les provoca cuando un agente conversacional solo responde que no es capaz de solucionar su petición sin ofrecer alguna alternativa. También, entre la información obtenida, destaca el hecho de que los usuarios comentaron tres propiedades que les parecen útiles por parte del *bot*, la primera de ella es la función de sugerencias, se valoró que el agente pudiera asistir en la resolución de dudas o conocer otro tipo de información que les podría interesar. La segunda es la de aportar conocimiento sobre como volver a formular la pregunta. Finalmente, la tercera propiedad a destacar del agente es la de apoyar a retomar la conversación cuando se brinda una sugerencia, al contrario de otros asistentes que simplemente cortan la conversación y no ofrecen esta función.

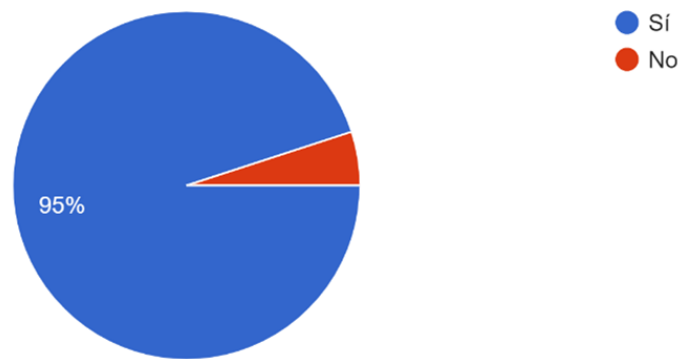


Figura 4.12: Aceptación de la función de recomendación.

4.4. Comentarios finales

Los resultados de la evaluación demuestran una aprobación por parte de los participantes hacia la herramienta desarrollada, a razón de que el 66 % de los encuestados calificaron su interacción con el bot como excelente y un 32 % como buena. Además, se observó una preferencia por parte de los usuarios posicionando la plataforma WhatsApp como la favorita para comunicarse con los *chatbots* de servicio al cliente. Pero, como se pudo observar en los resultados del mapeo sistemático de la literatura, realizado en el Capítulo 3 de la tesis, las herramientas actuales para facilitar el desarrollo de este tipo de *software* no contemplan la integración con esta aplicación de mensajería instantánea. Por lo que el desarrollo de la propuesta presentada en esta tesis para facilitar el desarrollo de agentes conversacionales para el servicio de WhatsApp se considera una contribución importante en el área de agentes conversacionales.

Los datos extraídos de la encuesta realizada en este capítulo, sugieren el uso de, al menos, una alternativa para atender las consultas de los usuarios cuando el agente no sea capaz de resolver una consulta a través del mecanismo principal de detección de *intents* y respuestas predefinidas. También, es importante señalar que el 95 % de los participantes consideraron la capacidad de ofrecer recomendaciones que ayuden a retomar la conversación cuando el agente no es capaz de contestar una pregunta una cualidad útil en un *chatbot*. Lo que es más, los usuarios consideraron muy útil el uso del modelo de lenguaje autorregresivo GPT-3 como alternativa a las consultas realizadas al agente con-

versacional. Esta afirmación se sustenta en que la mayoría de los integrantes de la encuesta calificaron las sugerencias del agente como útiles. Es por este motivo que el uso de estos modelos de lenguaje puede resultar positivo en el contexto de los *bots*, lo que expone la necesidad de estudiar esta alternativa más a profundidad.

Capítulo 5

Conclusiones

A continuación se explican los resultados que se obtuvieron durante la investigación descrita en esta tesis. Posteriormente, se listan y comentan una serie de conclusiones que se presentan dentro del ámbito del trabajo realizado.

5.1. Conclusiones

En los últimos años, los agentes conversacionales se han vuelto relevantes debido a la forma en la que los usuarios prefieren interactuar, por ejemplo, mensajes de voz o texto, en lugar de llamadas telefónicas. Además, el creciente uso de los dispositivos inteligentes activados por voz en los últimos años, como Alexa de Amazon [85], Google Assistant [86] y Siri de Apple [87] [88], para automatizar procesos, ha ocasionado que los agentes conversacionales sean objeto de demanda entre los usuarios de servicios, quienes esperan que una empresa esté disponible 24 horas del día, los 7 días de la semana. Esto se debe a que permiten reducir costos a las compañías cuando se automatizan las interacciones que demandan información de productos o servicios, como pueden ser las llamadas o consultas por información básica como lo es el precio o disponibilidad.

Si bien es cierto que la demanda de *chatbots* está aumentando en la actualidad, se incrementa también la necesidad de conocimientos técnicos en una gran cantidad de herramientas para su desarrollo. Diversos autores han presentado propuestas que permiten facilitar la definición e implementación de agentes conversacionales proporcionando téc-

nicas, librerías y *frameworks* especializados, como se detalla en el Capítulo 3. Sin embargo, los trabajos no consideran plataformas tecnológicas para envío de mensajes ampliamente usadas a nivel mundial, como por ejemplo WhatsApp y Telegram. Este hecho limita el campo de aplicación de los agentes conversacionales. Incluso, es posible que su programación se complique debido a que provoca la necesidad de agregar una capa intermediaria extra a su codificación al no contar con soporte para alguna plataforma de mensajería. Además, destaca el hecho de que el uso de los modelos de lenguajes autorregresivos no ha sido investigado en su totalidad [89]. Esta tecnología, que está cobrando una gran importancia en el ámbito del procesamiento de lenguaje natural a razón de sus capacidades de clasificación y desempeño en diversas tareas, se consideró en esta tesis.

A continuación se describen las conclusiones del presente trabajo y algunos de los posibles trabajos futuros que pueden continuar desarrollándose como resultado de la investigación realizada.

Este trabajo de Tesis presentó una propuesta para la generación de agentes conversacionales para la plataforma de mensajería instantánea WhatsApp a través del *framework* Xatkit Chat Platform. Los componentes principales son cuatro:

- Vinculación de WhatsApp Xatkit Chat Platform mediante Twilio. Se implementó un intermediario con Twilio Platform, una plataforma de comunicación en la nube que ofrece herramientas para enviar y recibir mensajes de texto, voz y video a través de diversas aplicaciones. El trabajo presentado en esta tesis, facilita el desarrollo de agentes conversacionales personalizados que pueden interactuar con los usuarios a través de la plataforma de mensajería WhatsApp.
- La definición de un DSL mediante un metamodelo llamado XatkitMM. Fue construido con el lenguaje Ecore. En él se definieron los principales componentes que conforman a un *chatbot* en *framework* Xatkit Chat Platform. Por consiguiente, es posible crear modelos de agentes conversacionales con las propiedades (atributos) requeridos por Xatkit Chat Platform y con ello el programador no necesita escribir código fuente, solo modelar el agente con sus propiedades.
- Transformación M2T. Un componente destacable en la tesis es la inclusión de las técnicas MDD para transformaciones M2T. Dentro de este marco, es posible la ge-

neración semi-automatizada de agentes conversacionales a partir de modelos. Para tal efecto, se diseñó e implementó una transformación modelo a texto en el *plugin* Acceleo Model to Text en el *framework* Eclipse, por ende, es posible generar el código fuente del *bot* a partir de un modelo que contiene sus propiedades y, gracias a esto, no es necesario escribir código fuente.

- Implementación de GPT-3. Se adaptó en Xatkit Chat Platform el modelo de lenguaje autorregresivo GPT-3 para la generación de sugerencias al usuario. Esto permite resolver consultas no consideradas en los *intents* del agente. Se consideró su aplicación porque emplea aprendizaje profundo para producir textos que simulan la redacción humana, por lo que cuenta con 175,000 millones de parámetros de aprendizaje automatizado.

Como prueba de concepto, se llevó a cabo un caso de estudio en el que se desarrolló un *chatbot* para brindar información al usuario interesado en ingresar en el Posgrado en Ciencias de la Información (PCI) de la Universidad Autónoma de Sinaloa. Esto tuvo como objetivo evaluar su funcionamiento y aceptación, lo cual se realizó mediante una encuesta en línea y el posterior análisis de los resultados.

Los resultados del caso de estudio evidenciaron la satisfacción de los participantes en lo referente al uso de un agente conversacional en WhatsApp para obtener información del PCI. También, la implementación del modelo de lenguaje autorregresivo GPT-3 resultó de gran aceptación, es por esta razón que se considera como un aporte significativo al área de definición y desarrollo de agentes conversacionales vinculado al concepto de sugerencias al usuario cuando el *chatbot* no es capaz de resolver una consulta. Dotar de esta capacidad al agente permite mejorar la experiencia de interacción *bot*-usuario.

Esto permite validar la hipótesis de esta investigación sobre como el desarrollo dirigido por modelos aplicado en la generación de agentes conversacionales para la plataforma WhatsApp agiliza su integración y mejora el proceso de información al usuario.

Para concluir se presenta un listado de los aportes que se obtuvieron en la investigación realizada en esta tesis:

- Investigación sobre el estado del arte del ámbito de los *frameworks* para automatizar el proceso de desarrollo de agentes conversacionales. Se obtuvieron áreas de

oportunidad y futuras líneas de investigación en este tema.

- Se obtuvo un proyecto de código abierto, desarrollado para agilizar el desarrollo de *chatbots* para la plataforma de mensajería instantánea WhatsApp en el *framework* Xatkit Chat Platform.
- Una propuesta dirigida por modelos para asistir a los desarrolladores en la creación de agentes conversacionales complejos para WhatsApp, sin requerir de conocimiento técnico en las tecnologías necesarias para la creación de un *chatbot*. Lo cual, puede permitir una mayor cantidad de personas accedan a estas herramientas.
- Se desarrolló una propuesta para el uso del modelo de lenguaje autorregresivo GPT-3 para la generación de sugerencias al usuario como alternativa a los servicios de procesamiento de lenguaje natural. Esto permitió aumentar la satisfacción general del usuario del agente.
- Se llevó a cabo el análisis de los datos de un caso de estudio en donde se observó la necesidad de implementar técnicas para la resolución de consultas en los agentes conversacionales para intentar proveer siempre al cliente de una alternativa de respuesta que resulte satisfactoria.

5.2. Trabajo futuro

La implementación de soporte para WhatsApp via Twilio en Xatkit Chat Platform presentada en la tesis representa una alternativa para desarrollar *chatbots* mediante un proceso de desarrollo semi-automatizado. A pesar de esto, durante el desarrollo de esta tesis han surgido algunas líneas futuras de investigación. Algunas de ellas, están directamente relacionadas con este trabajo y son el resultado de interrogantes que han emergido durante la realización de la investigación.

Una línea clave que queda como trabajo futuro es la creación de una plataforma que automatice la implementación de agentes conversacionales interactuando directamente con la API de WhatsApp *Business* sin necesidad de un intermediario. Esto puede ser de

gran utilidad en casos en los que se necesite el pago de un servicio extra cuando se desee generar un agente conversacional, por ejemplo, para Telegram.

Por lo que respecta al estado actual de la automatización de la generación de agentes conversacionales a partir de modelos sin necesidad de escribir código fuente, resalta el hecho de la falta de *frameworks* robustos que permitan llevar a cabo un proceso dirigido por modelos que guía al desarrollador desde la fase de requerimientos hasta la implementación del código fuente generado.

Especial atención merece la línea de investigación referente al uso de modelos de lenguaje autorregresivos como alternativa en la resolución de sugerencias para el usuario del *bot*. El dominio de aplicación, el entrenamiento y el idioma, necesitan ser objeto de mayor investigación. La diferencia entre los agentes que surgieron en la década de los sesenta y aquellos que cobraron fuerza recientemente, consiste precisamente en las opciones de lenguajes específicos para un dominio predefinido que permita esa inferencia de respuestas al usuario cuando no cuente el agente con el dato exacto. Lo que es más, existen otros modelos de código abierto que deberían considerarse, por ejemplo GPT-Neo [90], que han llegado a demostrar resultados similares a GPT-3 con un menor número de parámetros. También existen otras alternativas que se especializan en algún idioma en concreto, tal es el caso de BERTIN-GPT-J-6B [91], para el idioma Español.

Se identificó como futura línea de investigación incluir técnicas de ingeniería y diseño de *prompt*, así como los llamados *modifiers* (palabras clave que cambian el comportamiento de los modelos generativos). Durante la investigación realizada en la tesis, se ha observado que su uso puede generar a mejores resultados en algunos casos, como en los sistemas generativos de texto a imágenes [92].

Finalmente, una opción a futuro es el estudio del uso de *fine tuning*, la cual es una función que permite entrenar un modelo pre-entrenado con grandes volúmenes de ejemplos con el fin de mejorar su desempeño en una tarea concreta. Se puede utilizar esta técnica para incrementar los resultados satisfactorios y disminuir los *tokens* utilizados en el *prompt* disminuyendo los costos de uso.

Apéndice A

Plantilla Acceleo M2T para generación de agentes conversacionales

```
1 [comment encoding = UTF-8 /]
2 [module generate( 'http://xhatbotMMU' )]
3
4 [template public generateElement(aChatbot : Chatbot)]
5 [comment @main/]
6 [file (aChatbot.name.concat( '.java' ), false , 'UTF-8')]
7 package com.xatkit.example;
8
9 import com.xatkit.core.XatkitBot;
10
11 [generatePlatformImports(aChatbot)/]
12 import lombok.val;
13 import org.apache.commons.configuration2.BaseConfiguration;
14 import org.apache.commons.configuration2.Configuration;
15
16 import static com.xatkit.dsl.DSL.eventIs;
17 import static com.xatkit.dsl.DSL.fallbackState;
18 import static com.xatkit.dsl.DSL.intent;
19 import static com.xatkit.dsl.DSL.intentIs;
20 import static com.xatkit.dsl.DSL.model;
21 import static com.xatkit.dsl.DSL.state;
22 import com.xatkit.core.recognition.
    IntentRecognitionProviderFactory;
23 [generateIntentProviderImports(aChatbot)/]
24
25 public class [aChatbot.name/] {
26
```

```

27     /*
28     * Your bot is a plain Java application: you need to
    define a main method to make the created jar executable.
29     */
30     public static void main(String [][' ']/[' ']/ args) {
31
32     [ generateIntentDefinitions(aChatbot)/]
33     [ generatePlatformDefinitions(aChatbot)/]
34     [ generateStatesDefinitions(aChatbot)/]
35     [ generateStatesBehaviour(aChatbot)/]
36     [ generateDefaultFallback(aChatbot)/]
37     [ generateBotModel(aChatbot)/]
38
39         Configuration botConfiguration = new
    BaseConfiguration();
40
41     [ generateIntentProviderConfigurations(aChatbot)/]
42     [ generatePlatformConfigurations(aChatbot)/]
43
44         XatkitBot xatkitBot = new XatkitBot(botModel,
    botConfiguration);
45         xatkitBot.run();
46         /*
47         * The bot is now started, you can check http://localhost:5000/admin to test it.
48         * The logs of the bot are stored in the logs
    folder at the root of this project.
49         */
50     }
51 }
52 [/ file]
53 [/ template]
54
55 [template public generateIntentDefinitions(chatbot: Chatbot
    )]
56 [for(intent: Intent | chatbot.Intent) separator('\n')]
57     val [intent.name/] = intent("[intent.name/]")
58     [for(tSentence: String | intent.trainingSentences)
    separator('\n') after(';')]
59         .trainingSentence("[tSentence/]")[/ for]
60 [/ for]
61
62
63 [/ template]
64
65 [template public generatePlatformDefinitions(chatbot:

```

```

Chatbot)]
66 [if (chatbot.Platform.type.toString() = 'React')]
67     ReactPlatform reactPlatform = new ReactPlatform();
68     ReactEventProvider reactEventProvider =
        reactPlatform.getReactEventProvider();
69     ReactIntentProvider reactIntentProvider =
        reactPlatform.getReactIntentProvider();
70 [/if]
71
72 [if (chatbot.Platform.type.toString() = 'TwilioWhatsApp')]
73     TwilioPlatform twilioPlatform = new TwilioPlatform();
74     TwilioEventProvider twilioEventProvider = new
        TwilioEventProvider(twilioPlatform);
75 [/if]
76 [/template]
77
78 [template public generateStatesDefinitions(chatbot: Chatbot
    )]
79     [for (state: State | chatbot.State) separator('')]
80         val [state.name/] = state("[state.name/]");
81     [/for]
82 [/template]
83
84 [template public generateStatesBehaviour(chatbot: Chatbot)]
85     [for (state: State | chatbot.State) separator('\n')]
86         [state.name/]
87         [if (state.name.toString() = 'init')]
88             [if (chatbot.Platform.type.toString() = '
                TwilioWhatsApp')]
89                 [if (not state.Transition.oclIsUndefined())]
90                     .next()
91                     .moveTo([state.Transition.goTo/]);
92                 [/if]
93             [/if]
94
95             [if (chatbot.Platform.type.toString() = 'React')]
96                 [if (not state.Transition.oclIsUndefined())]
97                     .next()
98                     .when(eventIs(ReactEventProvider.ClientReady)).
moveTo([state.Transition.goTo/]);
99                 [/if]
100             [/if]
101         [/if]
102         [if (not (state.name.toString() = 'init'))]
103             .body(context -> {

```

```

104     [for(txtMessage:TextMessage | state.TextMessage)
separator('')]
105     [if (chatbot.Platform.type.toString() = 'React')]
106         reactPlatform.reply(context, "[txtMessage.text /]" )
;
107     [/if]
108     [if (chatbot.Platform.type.toString() = '
TwilioWhatsApp')]
109         twilioPlatform.reply(context, "[txtMessage.text /]"
);
110     [/if]
111     [/for]
112     [for(multimediaMessage:MultimediaMessage | state.
MultimediaMessage) separator('')]
113         twilioPlatform.replyMedia(context, "[
multimediaMessage.text /]", "[multimediaMessage.
multimediaURL /]");
114     [/for]
115     })
116     .next()
117     [for(conditionalTransition:ConditionalTransition |
state.ConditionalTransition) separator('\n') after(';')]
118         .when(intentIs([conditionalTransition.condition]))
.move([conditionalTransition.goTo]) [/for]
119         [if (not state.Transition.oclIsUndefined())]
120             .move([state.Transition.goTo]);
121         [/if]
122     [/if]
123 [/for]
124 [/template]
125
126 [template public generateIntentProviderConfigurations(
chatbot: Chatbot)]
127 [if (chatbot.Provider.type.toString() = 'NLPJS')]
128 botConfiguration.addProperty(
IntentRecognitionProviderFactory.INTENT_PROVIDER_KEY,
NlpjsConfiguration.NLPJS_INTENT_PROVIDER);
129 botConfiguration.addProperty(NlpjsConfiguration.
AGENT_ID_KEY, "default");
130 botConfiguration.addProperty(NlpjsConfiguration.
LANGUAGE_CODE_KEY, "[chatbot.Provider.languageCodeKey]"
);
131 botConfiguration.addProperty(NlpjsConfiguration.
NLPJS_SERVER_KEY, "[chatbot.Provider.NLPJSServerKey /]");
132 [/if]
133 [if (chatbot.Provider.type.toString() = 'DialogFlow')]

```

```

134 botConfiguration.addProperty(
    IntentRecognitionProviderFactory.INTENT_PROVIDER_KEY,
    DialogFlowConfiguration.DIALOGFLOW_INTENT_PROVIDER);
135 botConfiguration.addProperty(DialogFlowConfiguration.
    PROJECT_ID_KEY, "[chatbot.Provider.DialogFlowProjectID/]
    ");
136 botConfiguration.addProperty(DialogFlowConfiguration.
    GOOGLE_CREDENTIALS_PATH_KEY, "[chatbot.Provider.
    GoogleCredentialPath/]");
137 botConfiguration.addProperty(DialogFlowConfiguration.
    LANGUAGE_CODE_KEY, "[chatbot.Provider.languageCodeKey/]
    ");
138 botConfiguration.addProperty(DialogFlowConfiguration.
    CLEAN_AGENT_ON_STARTUP_KEY, true);
139 [/if]
140 [/template]
141
142 [template public generatePlatformConfigurations(chatbot:
    Chatbot)]
143 [if (chatbot.Platform.type.toString() = 'TwilioWhatsApp')]
144     botConfiguration.addProperty("xatkit.twilio.username",
        "[chatbot.Platform.twilioUserName/]");
145     botConfiguration.addProperty("xatkit.twilio.auth.
        token", "[chatbot.Platform.twilioAuthToken/]");
146 [/if]
147 [/template]
148
149 [template public generateBotModel(chatbot: Chatbot)]
150 [if (chatbot.Platform.type.toString() = 'TwilioWhatsApp')]
151     val botModel = model()
152         .usePlatform(twilioPlatform)
153         .listenTo(twilioEventProvider)
154         .initState(init)
155         .defaultFallbackState(defaultFallback);
156 [/if]
157 [if (chatbot.Platform.type.toString() = 'React')]
158     val botModel = model()
159         .usePlatform(reactPlatform)
160         .listenTo(reactEventProvider)
161         .listenTo(reactIntentProvider)
162         .initState(init)
163         .defaultFallbackState(defaultFallback);
164 [/if]
165 [/template]

```



```

166
167 [template public generateDefaultFallback(chatbot: Chatbot)]
168     val defaultFallback = fallbackState()
169     [if (chatbot.Platform.type.toString() = 'TwilioWhatsApp')
170         .body(context -> twilioPlatform.reply(context, "[
171             chatbot.defaultMessage/]"));
172     [/if]
173     [if (chatbot.Platform.type.toString() = 'React')]
174         .body(context -> reactPlatform.reply(context, "[
175             chatbot.defaultMessage/]"));
176     [/if]
177 [/template]
178
179 [template public generatePlatformImports(chatbot: Chatbot)]
180     [if (chatbot.Platform.type.toString() = 'TwilioWhatsApp')
181         ]
182     import com.xatkit.plugins.twilio.platform.TwilioPlatform;
183     import com.xatkit.plugins.twilio.platform.io.
184         TwilioEventProvider;
185     [/if]
186     [if (chatbot.Platform.type.toString() = 'React')]
187     import com.xatkit.plugins.react.platform.ReactPlatform;
188     import com.xatkit.plugins.react.platform.io.
189         ReactEventProvider;
190     import com.xatkit.plugins.react.platform.io.
191         ReactIntentProvider;
192     [/if]
193 [/template]
194
195 [template public generateIntentProviderImports(chatbot:
196     Chatbot)]
197     [if (chatbot.Provider.type.toString() = 'NLPJS')]
198     import com.xatkit.core.recognition.nlpjs.
199         NlpjsIntentRecognitionProvider;
200     import com.xatkit.core.recognition.nlpjs.NlpjsConfiguration
201         ;
202     [/if]
203     [if (chatbot.Provider.type.toString() = 'DialogFlow')]
204     import com.xatkit.core.recognition.dialogflow.
205         DialogFlowIntentRecognitionProvider;
206     import com.xatkit.core.recognition.dialogflow.
207         DialogFlowConfiguration;
208     [/if]

```

¹⁹⁸ [/ template]

Fragmento de código A.1: Implementación de la transformación modelo a texto mediante el *plugin* de Eclipse Acceleo.

Apéndice B

Encuesta para caso de estudio

1. ¿Cuál es tu satisfacción general con el uso del Chatbot para obtener información sobre el Posgrado en Ciencias de la Información?

Respuestas:

- Excelente.
- Bueno.
- Regular.
- Malo.

Descripción:

Esta pregunta está enfocada en evaluar si la experiencia general del usuario resulto ser positiva o negativa.

2. ¿Qué tan difícil te resultó hacer consultas en el Chatbot?

Respuestas:

- El agente respondió todas o la mayoría de mis preguntas de forma satisfactoria
- El agente respondió todas o la mayoría de mis preguntas, pero me tomó varios intentos hacerle entender mi pregunta
- El agente tuvo dificultades para responder a la mayoría de mis preguntas
- El agente no entendió ninguna de mis consultas

Descripción:

Con esta pregunta se pretende analizar el nivel de dificultad que puede tener el usuario para comunicarse con el agente conversacional.

3. ¿Consideras cómodo y/o útil poder utilizar un *Chatbot* desde la plataforma de WhatsApp para obtener información sobre el Posgrado en Ciencias de la Información? ¿Por qué?

Respuestas:

- Si.
- No.
- respuesta abierta

Descripción:

Preguntas enfocadas en la aceptación de los usuarios en el uso de la herramienta por el medio mencionado, lo cual asistirá en el análisis de la utilidad de mejorar la automatización de generación de agentes conversacionales en la plataforma de mensajería instantánea WhatsApp.

4. ¿Consideras a WhatsApp tu plataforma de mensajería instantánea preferida para interactuar con este tipo de herramientas de servicio al cliente?

Respuestas:

- Si.
- No.

Descripción:

Pregunta enfocada en las preferencias de los usuarios, tiene el objetivo de ser auxiliar en el análisis de la popularidad y aceptación de la plataforma WhatsApp por parte de los consumidores.

5. Si seleccionaste No ¿cuál es tu preferida?

Respuesta abierta.

Descripción:

Pregunta relacionada con la anterior con el fin de conocer otras opciones que pueden ser solicitadas por los usuarios.

6. Cuando el Chatbot del posgrado no entiende una pregunta regularmente ofrece una recomendación en el siguiente formato: (Perdón, no lo entiendo o no estoy capacitado para responder esa pregunta... si te interesa puedes intentar escribir:...)

Selecciona una de las siguientes afirmaciones:

Respuestas:

- El agente me ofreció recomendaciones las cuales me fueron de utilidad.
- El agente no me ofreció ninguna recomendación.
- El agente me ofreció recomendaciones, pero no eran de utilidad.
- El agente me ofreció recomendaciones, pero no tenían sentido (eran irrelevantes en el contexto de la escuela y el posgrado).

Descripción:

Con esta pregunta se pretende analizar la interacción del usuario con el agente cuando este no es capaz de resolver una consulta.

7. ¿Consideras que la capacidad de ofrecer recomendaciones que ayuden a retomar la conversación cuando el agente no es capaz de contestar una pregunta una cualidad útil de un Chatbot? ¿Por qué?

Respuestas:

- Si.
- No.
- Respuesta abierta.

Descripción:

Estas preguntas tienen como objetivo analizar la aceptación de los usuarios con la solución propuesta para abordar las consultas no manejadas.

Glosario

AIML Artificial Intelligent Markup Language. 19

ALICE Artificial Linguistic Internet Computer Entity. 19

BERT Bidirectional Encoder Representations from Transformers. 25

CIM Computation Independent Model. 16

DSL Domain-Specific Language. 4

EMF Eclipse Modeling Framework. 13, 48

EMP Eclipse Modeling Project. 14

GMF Graphical Modeling Framework. 14

GPT-3 Generative Pre-trained Transformer 3. 5

HTML HyperText Markup Language. 11

IA Inteligencia Artificial. 2

IDE Integrated Development Environment. 44

IFML Interaction Flow Modeling Language. 48

M2T Model to Text. 11

MDA Model Driven Architecture. 50

MDD Model-Driven Development. 15

MDE Model-Driven Engineering. 3

NLG Natural Language Generation. 22

NLP Natural Language Processing. 20

NLU Natural Language Understanding. 49

OMG Object Management Group. 10

PIM Platform Independent Model. 16

PSM Platform Specific Model. 16

REST Representational State Transfer. 46

UML Unified Modeling Language. 9

UX User eXperience. 4

WA Watson Asistant. 52

XML eXtensible Markup Language. 14

Bibliografía

- [1] Cordero J., Toledo A., Guaman F., and Barba-Guaman L., “Use of chatbots for user service in higher education institutions,” in *Iberian Conference on Information Systems and Technologies (CISTI)*, 2020, pp. 24–27.
- [2] Ondas S., Pleva M., and Hladek D., “How chatbots can be involved in the education process,” in *17th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 2019, pp. 575–580.
- [3] Brambilla M., Cabot J., and Wimmer M., *Model-Driven Software Engineering in Practice*. Springer, 2012.
- [4] Daniel G., Cabot J., Deruelle L., and Derras M., “Xatkit: A multimodal low-code chatbot development framework,” *IEEE Access*, vol. 8, pp. 15 332–15 346, 2020.
- [5] Abdellatif A., Costa D., Badran K., Abdalkareem R., and Shihab E., *Challenges in Chatbot Development: A Study of Stack Overflow Posts*. New York, NY, USA: Association for Computing Machinery, 2020, p. 174–185. [Online]. Available: <https://doi.org/10.1145/3379597.3387472>
- [6] Parada W. V., Giraldo F. A., and Londoño R M. I., “Mdd based case tool for automatic generation of chatbot,” in *XLV Latin American Computing Conference (CLEI)*, 2019, pp. 1–7.
- [7] Pérez-Soler S., Guerra E., and de Lara J., “Model-driven chatbot development,” in *Conceptual Modeling*, Dobbie G., Frank U., Kappel G., Liddle S. W., and Mayr H. C., Eds. Cham: Springer International Publishing, 2020, pp. 207–222.
- [8] Daniel G., Cabot J., Deruelle L., and Derras M., “Multi-platform chatbot modeling and deployment with the jarvis framework,” in *Advanced Information Systems Engineering*, Giorgini P. and Weber B., Eds. Cham: Springer International Publishing, 2019, pp. 177–193.
- [9] Følstad A. and Brandtzaeg P. B., “Users’ experiences with chatbots: findings from a questionnaire study,” *Quality and User Experience*, vol. 5, p. 3, 2020. [Online]. Available: <https://doi.org/10.1007/s41233-020-00033-2>
- [10] Hornbæk K. and Hertzum M., “Technology acceptance and user experience: A review of the experiential component in hci,” *ACM Transactions on Computer-Human Interaction*, vol. 24, pp. 1–30, 2017.

- [11] Sandu N. and Gide E., “Adoption of AI-chatbots to enhance student learning experience in higher education in india,” in *18th International Conference on Information Technology Based Higher Education and Training (ITHET)*. IEEE, 2019, pp. 1–5.
- [12] Carlander-Reuterfelt D., Carrera A., Iglesias C. A., Araque O., Sanchez Rada J. F. S., and Munoz S., “JAICOB: A Data Science Chatbot,” *IEEE Access*, vol. 8, pp. 180 672–180 680, 2020.
- [13] Heryandi A., “Developing chatbot for academic record monitoring in higher education institution,” *IOP Conference Series: Materials Science and Engineering*, vol. 879, no. 1, pp. 1–9, 2020. [Online]. Available: <https://dx.doi.org/10.1088/1757-899X/879/1/012049>
- [14] Statista, “Aplicaciones de mensajería más populares según el número de usuarios mensuales activos a nivel mundial a enero de 2023.” [Online]. Available: <https://es.statista.com/estadisticas/599043/aplicaciones-de-mensajeria-mas-populares-a-nivel-mundial-de/>
- [15] Vorontsov M. and Radmir S. I., “Automation of message sending processes using specialized software,” in *IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*, 2021, pp. 746–748.
- [16] Scotti V., Tedesco R., and Sbattella L., “A modular data-driven architecture for empathetic conversational agents,” in *IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2021, pp. 365–368.
- [17] Neumann A. T., de Lange P., and Klamma R., “Collaborative creation and training of social bots in learning communities,” in *IEEE 5th International Conference on Collaboration and Internet Computing (CIC)*, 2019, pp. 11–19.
- [18] Fernández P. a., “Un Análisis Crítico de la Aproximación Model-Driven Architecture,” p. 121, 2009.
- [19] “Terms and Acronyms | Object Management Group.” [Online]. Available: https://www.omg.org/gettingstarted/terms_and_acronyms.htm
- [20] Sosa J. D., Díaz O., and Trujillo S., “Defining DSL expressions collaboratively in multidisciplinary embedded engineering,” in *Proceedings 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, 2011, pp. 217–220.
- [21] Jörges S. and Steffen B., “Exploiting ecore’s reflexivity for bootstrapping domain-specific code-generators,” in *35th Annual IEEE Software Engineering Workshop*, 2012, pp. 72–81.
- [22] Bucchiarone A., Cicchetti A., and Marconi A., “Gdf: A gamification design framework powered by model-driven engineering,” in *ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2019, pp. 753–758.

- [23] Schmidt D., “Guest editor’s introduction: Model-driven engineering,” *Computer*, vol. 39, no. 2, pp. 25–31, 2006.
- [24] Cuevas C., Barros L., Martínez P. L., and Drake J. M., “Beneficios que aportala metodología MDE a los entornos de desarrollo de sistemas de tiempo real,” *RIAI - Revista Iberoamericana de Automatica e Informatica Industrial*, vol. 10, no. 2, pp. 216–227, 2013.
- [25] Whittle J., Hutchinson J., and Rouncefield M., “The state of practice in model-driven engineering,” *IEEE Software*, vol. 31, no. 3, pp. 79–85, 2014.
- [26] Selic B., “The pragmatics of model-driven development,” *IEEE Software*, vol. 20, no. 5, pp. 19–25, 2003.
- [27] Pascuas Rengifo Y. S., Mendoza Suarez J. A., and Córdoba Correa E. D., “Desarrollo Dirigido por Modelos (MDD) en el Contexto Educativo,” *Scientia et technica*, vol. 20, no. 2, p. 172, 2015.
- [28] Atkinson C. and Kuhne T., “Model-driven development: a metamodeling foundation,” *IEEE Software*, vol. 20, no. 5, pp. 36–41, 2003.
- [29] Liu Y. and Wang Y., “A study of metamodeling based on MDA,” in *3rd International Conference on Computer Research and Development (ICCRD)*, vol. 2, 2011, pp. 171–173.
- [30] Meaurio V. S. and Schmieder E., “La Arquitectura de Software en el Proceso de Desarrollo: Integrando MDA al Ciclo de Vida en Espiral,” *Revista Latinoamericana de Ingenieria de Software*, vol. 1, no. 4, p. 142, 2014.
- [31] Turing A. M., “I.—Computing Machinery and Intelligence,” *Mind*, vol. LIX, no. 236, pp. 433–460, 1950. [Online]. Available: <https://doi.org/10.1093/mind/LIX.236.433>
- [32] Rarhi K., Bhattacharya A., Mishra A., and Mandal K., “Automated Medical Chatbot,” *SSRN Electronic Journal*, pp. 1–2, 2018.
- [33] Ranoliya B. R., Raghuwanshi N., and Singh S., “Chatbot for university related FAQs,” in *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017, pp. 1525–1530.
- [34] Weizenbaum J., “Eliza—a computer program for the study of natural language communication between man and machine,” *Commun. ACM*, vol. 9, no. 1, p. 36–45, 1966. [Online]. Available: <https://doi.org/10.1145/365153.365168>
- [35] Science N., Bishwabidyalay G., and Mamun S. A., “Review of Integrated Applications with AIML Based Chatbot Review of Integrated Applications with AIML Based Chatbot,” in *1st International Conference on Computer and Information Engineering*, 2015, pp. 87–90.

- [36] Shawar B. A. and Atwell E., “ALICE chatbot: Trials and outputs,” *Computacion y Sistemas*, vol. 19, no. 4, pp. 625–632, 2015.
- [37] Prathyusha V., Sri G. L., Meenakshi G., and Chakravarti Y. K., “Information acquisition chatbot system using luis,” in *6th International Conference on Inventive Computation Technologies (ICICT)*, 2021, pp. 1025–1029.
- [38] Adamopoulou E. and Moussiades L., *An Overview of Chatbot Technology*. Springer International Publishing, 2020, vol. 584 IFIP. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-49186-4_31
- [39] Qasse I., Mishra S., and Hamdaq M., “iContractBot: A Chatbot for Smart Contracts’ Specification and Code Generation,” 2021. [Online]. Available: <http://arxiv.org/abs/2103.09314>
- [40] Wang D., Su J., and Yu H., “Feature extraction and analysis of natural language processing for deep learning english language,” *IEEE Access*, vol. 8, pp. 46 335–46 345, 2020.
- [41] Jaf S. and Calder C., “Deep learning for natural language parsing,” *IEEE Access*, vol. 7, pp. 131 363–131 373, 2019.
- [42] Kumar E., *Natural language processing*. IK International Pvt Ltd, 2011.
- [43] Nigam A., Sahare P., and Pandya K., “Intent Detection and Slots Prompt in a Closed-Domain Chatbot,” in *Proceedings - 13th IEEE International Conference on Semantic Computing (ICSC)*, 2019, pp. 340–343.
- [44] Rafat A. A. A., Salehin M., Khan F. R., Hossain S. A., and Abujar S., “Vector representation of bengali word using various word embedding model,” in *8th International Conference System Modeling and Advancement in Research Trends (SMART)*, 2019, pp. 27–30.
- [45] Virkar M., Honmane V., and Rao S. U., “Humanizing the chatbot with semantics based natural language generation,” in *IEEE International Conference on Intelligent Computing and Control Systems (ICCS)*, 2019, pp. 891–894.
- [46] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser L., and Polosukhin I., “Attention is all you need,” 2017. [Online]. Available: <https://arxiv.org/abs/1706.03762>
- [47] Devlin J., Chang M.-W., Lee K., and Toutanova K., “Bert: Pre-training of deep bidirectional transformers for language understanding,” 2018. [Online]. Available: <https://arxiv.org/abs/1810.04805>
- [48] Floridi L. and Chiriatti M., “GPT-3: Its Nature, Scope, Limits, and Consequences,” vol. 30, pp. 681–694, 2020. [Online]. Available: <https://doi.org/10.1007/s11023-020-09548-1>

- [49] Black S., Biderman S., Hallahan E., Anthony Q., Gao L., Golding L., He H., Leahy C., McDonell K., Phang J., Pieler M., Prashanth U. S., Purohit S., Reynolds L., Tow J., Wang B., and Weinbach S., “Gpt-neox-20b: An open-source autoregressive language model,” 2022. [Online]. Available: <https://arxiv.org/abs/2204.06745>
- [50] Venkatesan S., Jawahar A., Varsha S., and Roshne N., “Design and implementation of an automated security system using twilio messaging service,” in *2017 International Conference on Smart Cities, Automation Intelligent Computing Systems (ICON-SONICS)*, 2017, pp. 59–63.
- [51] Daniel G., Cabot J., Deruelle L., and Derras M., “Twilio Integration with Dialogflow for Effective Communication,” *Journal of Web Development and Web Designing*, 2019.
- [52] “Documentación de Dialogflow | Google Cloud.” [Online]. Available: <https://cloud.google.com/dialogflow/docs>
- [53] McGuffie K. and Newhouse A., “The radicalization risks of gpt-3 and advanced neural language models,” 2020.
- [54] Petersen K., Feldt R., Mujtaba S., and Mattsson M., “Systematic mapping studies in software engineering,” *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, vol. 17, 2008.
- [55] Wieringa R., Maiden N., Mead N., and Rolland C., “Requirements engineering paper classification and evaluation criteria: A proposal and a discussion,” *Requir. Eng.*, vol. 11, pp. 102–107, 2006.
- [56] Pérez-Soler S., Juárez-Puerta S., Guerra E., and de Lara J., “Choosing a chatbot development tool,” *IEEE Software*, vol. 38, no. 4, pp. 94–103, 2021.
- [57] Pérez-Soler S., Guerra E., and de Lara J., “Creating and migrating chatbots with conga,” in *IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2021, pp. 37–40.
- [58] Baena-Perez R., Ruiz-Rube I., Dodero J. M., and Bolivar M. A., “A framework to create conversational agents for the development of video games by end-users,” in *Optimization and Learning*, Dorronsoro B., Ruiz P., de la Torre J. C., Urda D., and Talbi E.-G., Eds. Cham: Springer International Publishing, 2020, pp. 216–226.
- [59] Ed-douibi H., Cánovas Izquierdo J. L., Daniel G., and Cabot J., “A model-based chatbot generation approach to converse with open data sources,” in *Web Engineering*, Brambilla M., Chbeir R., Frasincar F., and Manolescu I., Eds. Cham: Springer International Publishing, 2021, pp. 440–455.
- [60] Hogan K., Baer A., and Purtilo J., “Diplomat: A conversational agent framework for goal-oriented group discussion,” in *Contemporary Issues in Group Decision and Negotiation*, Morais D. C., Fang L., and Horita M., Eds. Cham: Springer International Publishing, 2021, pp. 143–154.

- [61] Planas E., Daniel G., Brambilla M., and Cabot J., “Towards a model-driven approach for multiexperience ai-based user interfaces,” *Software and Systems Modeling*, vol. 20, 2021.
- [62] Pérez-Soler S., Daniel G., Cabot J., Guerra E., and de Lara J., “Towards automating the synthesis of chatbots for conversational model query,” in *Enterprise, Business-Process and Information Systems Modeling*, Nurcan S., Reinhartz-Berger I., Soffer P., and Zdravkovic J., Eds. Cham: Springer International Publishing, 2020, pp. 257–265.
- [63] Matic R., Kabiljo M., Zivkovic M., and Cabarkapa M., “Extensible chatbot architecture using metamodels of natural language understanding,” *Electronics*, vol. 10, no. 18, 2021. [Online]. Available: <https://www.mdpi.com/2079-9292/10/18/2300>
- [64] Pérez-Soler S., González-Jiménez M., Guerra E., and Lara J., “Towards conversational syntax for domain-specific languages using chatbots.” *The Journal of Object Technology*, vol. 18, p. 5:1, 01 2019.
- [65] Mahmood R., Joshi A., Lele A., and Pennington J., “Dynamic natural language user interfaces using microservices,” in *CEUR Workshop Proceedings*, vol. 2848, 2020.
- [66] Khalil A., Leiva F. H., Shonibare A., Arsenault E. M., Turner L., khalifa S., Tam-Seto L., Linden B., Wood V., Stuart H., Nolan J., and McDowell C., “Model-driven chats: Enabling chatbot development for non-technical domain experts through chat flow visualization and auto-generation,” in *Advances in Information and Communication*, Arai K., Ed. Cham: Springer International Publishing, 2021, pp. 1036–1050.
- [67] Levinson H., “Huey: Intelligent Agents for Natural Human to Machine Communication,” 2021. [Online]. Available: <https://dash.harvard.edu/handle/1/37367716>
- [68] Pérez-Soler S., Guerra E., and de Lara J., “Flexible modelling using conversational agents,” in *ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2019, pp. 478–482.
- [69] Sansonnet J.-P., Leray D., and Martin J.-C., “Architecture of a framework for generic assisting conversational agents,” in *Intelligent Virtual Agents*, Gratch J., Young M., Aylett R., Ballin D., and Olivier P., Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 145–156.
- [70] Statista, “WhatsApp: number of users 2013-2020,” 2022. [Online]. Available: <https://www.statista.com/statistics/260819/number-of-monthly-active-whatsapp-users/>
- [71] Gronback R. C., *Eclipse modeling project: a domain-specific language (DSL) toolkit*. Pearson Education, 2009.
- [72] “How to build a chatbot that reads all your data to find the right answer - Xatkit.” [Online]. Available: <https://xatkit.com/chatbot-open-question-qa-haystack/>

- [73] Agrawal P., Menon T., Kam A., Naim M., Chouragade C., Singh G., Kulkarni R., Suri A., Katakam S., Pratik V., Bansal P., Kaur S., Duggal A., Chalabi A., Choudhari P., Satti S. R., Nayak N., and Rajput N., *QnAMaker: Data to Bot in 2 Minutes*. New York, NY, USA: Association for Computing Machinery, 2020, p. 131–134. [Online]. Available: <https://doi.org/10.1145/3366424.3383525>
- [74] Syed Z. H., Trabelsi A., Helbert E., Bailleau V., and Muths C., “Question answering chatbot for troubleshooting queries based on transfer learning,” *Procedia Computer Science*, vol. 192, pp. 941–950, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050921015854>
- [75] Dale R., “Gpt-3: What’s it good for?” *Natural Language Engineering*, vol. 27, no. 1, pp. 113–118, 2021.
- [76] “TheoKanning/openai-java: OpenAI GPT-3 Api Client in Java.” [Online]. Available: <https://github.com/TheoKanning/openai-java>
- [77] “xatkit-bot-platform/xatkit-whatsapp-via-twilio-platform: Receive and send messages from WhatsApp using the Twilio API.” [Online]. Available: <https://github.com/xatkit-bot-platform/xatkit-whatsapp-via-twilio-platform>
- [78] “Rela academia española: Diccionario de la lengua española, 23.^a ed.” [Online]. Available: <https://dle.rae.es>
- [79] “Portal Web Posgrado en Ciencias de la Información (PCI).” [Online]. Available: <https://pci.uas.edu.mx/>
- [80] “AngelTM/PCIBot.” [Online]. Available: <https://github.com/AngelTM/PCIBot>
- [81] “Xatkit wiki.” [Online]. Available: <https://xatkit.com/>
- [82] “Google forms.” [Online]. Available: <https://forms.google.com.mx>
- [83] “AngelTM/TesisDatosPCIBot.” [Online]. Available: <https://github.com/AngelTM/TesisDatosPCIBot>
- [84] “Asia: most popular Asia-based mobile messenger apps 2020 | Statista.” [Online]. Available: <https://www.statista.com/statistics/250548/most-popular-asian-mobile-messenger-apps/>
- [85] “Amazon alexa official site: What is alexa?” [Online]. Available: <https://developer.amazon.com/es-ES/alexa>
- [86] “Asistente de google: tu google personal.” [Online]. Available: https://assistant.google.com/intl/es_es/
- [87] “Siri - apple (mx).” [Online]. Available: <https://www.apple.com/mx/siri/>
- [88] Lokman A. S. and Aamedeen M. A., “Modern chatbot systems: A technical review,” in *Proceedings of the Future Technologies Conference (FTC)*, Arai K., Bhatia R., and Kapoor S., Eds. Cham: Springer International Publishing, 2019, pp. 1012–1023.

- [89] Min B., Ross H., Sulem E., Veyseh A. P. B., Nguyen T. H., Sainz O., Agirre E., Heinz I., and Roth D., “Recent advances in natural language processing via large pre-trained language models: A survey,” 2021. [Online]. Available: <https://arxiv.org/abs/2111.01243>
- [90] Black S., Leo G., Wang P., Leahy C., and Biderman S., “GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow,” 2021, If you use this software, please cite it using these metadata. [Online]. Available: <https://doi.org/10.5281/zenodo.5297715>
- [91] la Rosa J. D., Ponferrada E. G., Romero M., Villegas P., de Prado Salas P. G., and Grandury M., “BERTIN: Efficient pre-training of a spanish language model using perplexity sampling,” *Procesamiento del Lenguaje Natural*, vol. 68, no. 0, pp. 13–23, 2022. [Online]. Available: <http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/6403>
- [92] Oppenlaender J., “A taxonomy of prompt modifiers for text-to-image generation,” 2022. [Online]. Available: <https://arxiv.org/abs/2204.13988>
- [93] Daniel G., Cabot J., Deruelle L., and Derras M., “Xatkit: a Multimodal Low-Code Chatbot Development Framework,” *IEEE Access*, vol. 8, pp. 15 332–15 346, 2020.
- [94] —, “Multi-platform Chatbot Modeling and Deployment with the Jarvis Framework,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11483 LNCS, pp. 177–193, 2019.
- [95] Aguilar J., Garrigós I., Mazón J.-N., and Trujillo J., “An mda approach for goal-oriented requirement analysis in web engineering,” *Journal of Universal Computer Science*, vol. 16, pp. 2475–2494, 2010.
- [96] “OpenAI API tokenizer.” [Online]. Available: <https://beta.openai.com/tokenizer>
- [97] Barroilhet Díez A., “Criptomonedas, economía y derecho,” *Revista chilena de derecho y tecnología*, vol. 8, no. 1, pp. 29–67, 2019.