TESIS TITULADA:

DISEÑO DE UNA ARQUITECTURA CONVERSACIONAL POR TEXTO QUE CONSUMA SERVICIOS COGNITIVOS DE NLP Y MACHINE LEARNING PARA LA INTERACCIÓN CON CHATBOTS MEDIANTE EL USO DE LENGUAJE NATURAL

PARA OPTAR POR EL TÍTULO DE MAGÍSTER EN:

INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

PRESENTADO POR:

NELSON ORLAN ESCOBAR CEBALLOS
1088318391

SERGIO ALEXANDER FLOREZ GALEANO
1088297456

ASESOR:

MGS. JULIO HERNANDO VARGAS MORENO

LUGAR Y AÑO DE SUSTENTACIÓN:

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERÍAS
MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
PEREIRA, RISARALDA
15 DE JUNIO DE 2021

Pereira 15 de junio de 2021

DEDICATORIA

Dedico este proyecto y todos mis logros pasados, presentes y futuros; a mi padre y a mi madre, ya que gracias a su esfuerzo y dedicación han hecho de mí un hombre de bien y me han preparado para la vida.

Sergio Alexander Florez Galeano

Dedico este proyecto a cada uno de los éxitos y fracasos que me han formado como profesional pasados, presentes y futuros; a mi familia que me ha apoyado incansablemente principalmente mi padre y a madre, a los cuales les debo cada buena decisión que he tomado en mi vida, pero principalmente este proyecto se lo dedico a Laura Ruiz que jamás ha dejado de creer en mí que siempre me ha apoyado y ha dado por mi más que muchas otras personas, espero este proyecto los llene de orgullo y sepan de corazón que el esfuerzo realizado en este fue hecho pensando en ustedes.

Nelson Orlan Escobar Ceballos

AGRADECIMIENTOS

Agradezco principalmente a mi familia por ser mi apoyo en mi desarrollo como profesional, a todos mis maestros que ayudaron en mi formación compartiendo sus conocimientos y a mis compañer@s, amig@s y demás personas que estuvieron presentes y aún continúan aportándome con su compañía, consejos y experiencia.

Sergio Alexander Florez Galeano

Agradezco principalmente a mi padre quien fue la persona que me apoyo desde un inicio para realizar esta maestría. Y quiero agradecer a mi novia Laura Ruiz que también supo apoyarme en los momentos más difíciles en los cuales no quería continuar o no sabía que dirección tomar, fue una parte esencial y una gran ayuda para realizar este proyecto. Finalmente agradecer al profesor Julio que siempre nos apoyó en los cambios y mejoras que teníamos en el proyecto, usando su experiencia y conocimiento.

Nelson Orlan Escobar Ceballos

TABLA DE CONTENIDO

1. CAPÍTULO I: GENERALIDADES **9**

1.1. Título 9

1.2. Definición del problema 10

1.2.1. Descripción del problema 10

1.2.2. Formulación del problema 11

1.3. Objetivo de la investigación 12

1.3.1. Objetivo general 12

1.3.2. Objetivos específicos 12

1.4. Justificación de la investigación 13

2. CAPÍTULO II: MARCO REFERENCIAL **14**

2.1. Marco Conceptual 14

2.2. Marco de antecedentes 19

2.2.1. Historia del NLP 19

2.2.2. Inicios del NLP 19

2.2.3. Historia y evolución de los chatbots 24

2.3. Marco de teórico 27

2.3.1. Tendencias, casos de uso clave y necesidades de los chatbots 27

2.3.2. Expectativas o necesidades de los consumidores 27

2.3.3. Expectativas o necesidades de los desarrolladores 29

2.3.4. Arquitectura de un chatbot 29

2.3.5. Limitaciones y retos en los chatbots 30

2.3.6. Falta de información para entrenamiento 31

2.3.7. Poca capacidad de interpretación conversacional 31

2.3.8. Soporte multilenguaje y/o multiplataforma 32

2.3.9. Regulación en el manejo de la información 32

2.3.10. Introducción a las Arquitecturas Serverless 33

3. CAPÍTULO III: ESTADO DEL ARTE **42**

3.1. Arquitecturas de Software 42

3.1.1. Diseño y elementos de la Arquitectura de software 42

3.1.2. Etapas del desarrollo de una arquitectura de software 45

3.1.3. Architecture Description Language (ADL) 56
3.1.4. Métodos para el análisis y elaboración de arquitecturas 59
3.1.5. Métodos de evaluación de arquitecturas de software 67
3.1.6. Metodología SEI (software Engineering Institute) para la creación de arquitecturas de software 69
3.1.7. Metodología SEI para equipos pequeños de desarrollo 72
3.1.8. Metodología SEI-A, adaptación del SEI 73
3.2. NLP y NLU en el desarrollo de chatbots 76
3.2.1. ¿Qué es el NLP? 76
3.2.2. Objetivos del NLP 77
3.2.3. Natural Language Understanding (NLU) 78
3.2.4. Técnicas usadas en el NLP 78
3.2.5. Arquitecturas basadas en NLP 84
3.2.6. Frameworks y plataformas NLP como servicio para el desarrollo de chatbots 89
4. CAPÍTULO V: DESARROLLO METODOLÓGICO
FASE 1 - ANÁLISIS (QAW-A) 98
4.1. Presentación de “LaraBot” cómo caso del estudio 98
4.2. Objetivos de negocio 99
4.3. Descripción del sistema 101
4.3.1. Presentación de componentes del sistema 101
4.4. Usuarios del sistema 104
4.4.1. Usuario Chat 104
4.4.2. Usuario Web 104
4.4.3. Usuario Desarrollador o Administrador 105
4.5. Requisitos específicos del sistema 105
4.5.1. Requisitos Funcionales 105
4.5.2. Requisitos no Funcionales o Atributos de Calidad 113
4.5.3. Restricciones de diseño 120
4.6. Aplicando QAW-A 122
4.6.1. Involucrados en QAW-A 122
4.6.2. Despliegue del método QAW-A 123
4.6.3. Resultados de QAW-A 144

5. CAPÍTULO VI: DESARROLLO METODOLÓGICO FASE 2 - DISEÑO (ADD-A) 145
5.1. Involucrados en ADD-A 145
5.2. Despliegue del método ADD-A 145
5.2.1. ITERACIÓN #1 146
5.2.2. ITERACIÓN #2: #ChatPlatform 152
5.2.3. ITERACIÓN #3: #WebApp 159
5.2.4. ITERACIÓN #4: #InputGateway 165
5.2.5. ITERACIÓN #5: #IntentDetector 171
5.2.6. ITERACIÓN #6: #BotEngine 177
5.2.7. ITERACIÓN #7: #OutputInterface 182
5.2.8. ITERACIÓN #8 - Final 186
5.3. Resultados de ADD-A 201
6. CAPÍTULO VII: DESARROLLO METODOLÓGICO FASE 3 - EVALUACIÓN (ATAM-A) 202
6.1. Involucrados en ATAM-A 202
6.2. Despliegue del método ATAM-A 202
7. CAPÍTULO VIII: DESARROLLO METODOLÓGICO FASE 4 - DOCUMENTACIÓN (VAB-A) 233
7.1. Involucrados en VaB-A 233
7.2. Despliegue del método VaB-A 233
8. CAPÍTULO IX: IMPLEMENTACIÓN DE PROTOTIPO (VAB-A) 234
8.1. Tecnologías de desarrollo 234
8.2. Arquitectura de chatbot basada en componentes de tecnología específica 235
9. CAPÍTULO X: CONCLUSIONES Y TRABAJO FUTURO 237
9.1. Conclusiones 237
9.2. Trabajos futuros 241
10. CAPÍTULO XI: REFERENCIAS BIBLIOGRÁFICAS 243

ÍNDICE DE FIGURAS

Figura 1: Linea de tiempo del NLP. 19
Figura 2: Ejemplo de la máquina de traducción 19
Figura 3: Traducción Basada en estadística. 22
Figura 4: Visualización de n-gramas en una fase. 22
Figura 5: Tendencia de papers de NLP basados en deep Learning y redes neuronales. 24
Figura 6: Mercado de los chatbots por regiones (Billones de Dólares). 27
Figura 7: Ciclo de expectativas de la inteligencia artificial en 2020 según Gartner 28
Figura 8: Arquitectura de un chatbot conversacional 30
Figura 9: Comparativa de modelos de comparación en la nube 33
Figura 10: Ejemplo de aplicación monolito desplegada en instancia de servidor. 34
Figura 11: Ejemplo de aplicación escalada horizontalmente a 4 réplicas. 35
Figura 12: Plataforma FAAS. Invocando función dentro de contenedor virtual. 36
Figura 13:Plataforma FAAS escalando horizontalmente llamado a la función 2. 36
Figura 14: Ejemplo arquitectura cliente-servidor simple. 44
Figura 15: Procesos de arquitectura y sus interacciones 46
Figura 16: Relación de las diferentes etapas de una arquitectura 47
Figura 17: Relación de stakeholders. 50
Figura 18: Plantilla de la documentación de una vista. 52
Figura 19: Modelo de vistas 4+1. 53
Figura 20: Ejemplo simple de UML 55
Figura 21: ADL en los escenarios de elaboración de arquitectura de software. 56
Figura 22: Evaluación en la arquitectura de software. 57
Figura 23: ABD dentro del ciclo de vida. 60
Figura 24: Descomposición del sistema en elementos de diseño. 61
Figura 25: Pasos para la descomposición de un elemento de diseño. 61
Figura 26: Definición de la vista lógica. 62
Figura 27: Estructura ADM-TOGAF, por ares (2013) Recuperado. 63
Figura 28: Actividades principales de QAW. 65
Figura 29: Los casos de uno integran el trabajo. 66
Figura 30: Fases, iteraciones y disciplinas. 66
Figura 31: ATAM: Entradas y salidas. 69
Figura 32: Enfoque del SEI para la calidad. 69
Figura 33: Representación visual de la metodología del SEI. 70
Figura 34: Representación de la metodología SEI adaptada. 72
Figura 35: Metodología SEI-A. 75
Figura 36: Ejemplo de red semántica. 79
Figura 37: Ejemplo Árbol de sintaxis en Python. 79
Figura 38: Visualización de N-gramas en una frase. 80
Figura 39: Funcionamiento de un algoritmo evolutivo. 81
Figura 40: Representación de un modelo oculto de Márkov. 82
Figura 41: Muestra de un sistema simple vectorial para clasificación 82

Figura 42: Explicación de una red neuronal. 83
Figura 43: Redes neuronales recurrentes en la traducción. 84
Figura 44: Ejemplo de framework para el uso de NLP con retroalimentación de usuarios. 85
Figura 45: Clasificación de sistemas basados en NLP. 86
Figura 46: Sistema de etiquetación de una frase. 87
Figura 47: arquitectura propuesta por spacy. 87
Figura 48: Estructuración de una frase por Intel AI. 88
Figura 49: Estructuración de una frase por Spacy. 88
Figura 50: Arquitectura de ejemplo de NLP en un sistema de chat. 89
Figura 51: Arquitectura de NLP de intel AI lab. 90
Figura 52: Logo de LaraBot. 98
Figura 53: Componentes del sistema 101
Figura 54: Plantilla de la documentación de una vista. 145
Figura 55: Descomposición de componentes del #ChatPlatform. 156
Figura 56: Diagrama de casos de uso del #ChatPlatform. 159
Figura 57: Descomposición de componentes de la #WebApp. 162
Figura 58: Casos de uso de la #WebApp. 165
Figura 59: Descomposición de componentes de la #InputGateway. 169
Figura 60: Descomposición de componentes de la #IntentDetector. 175
Figura 61: Descomposición de componentes de la #BotEngine. 180
Figura 62: Descomposición de componentes de la #OutputInterface. 185
Figura 63: Diagrama de casos de uso del sistema #LaraBot. 187
Figura 64: Diagrama de secuencia para el caso de uso CU-01. 197
Figura 65: Diagrama de secuencia para el caso de uso CU-02. 198
Figura 66: Diagrama de secuencia para el caso de uso CU-03. 198
Figura 67: Diagrama de secuencia para el caso de uso CU-04. 198
Figura 68: Diagrama de secuencia para el caso de uso CU-05. 199
Figura 69: Diagrama de secuencia para el caso de uso CU-06. 199
Figura 70:Diagrama de secuencia para el caso de uso CU-07. 200
Figura 71: Diagrama de secuencia para los casos de uso del CU-08 a CU-13. 200
Figura 72: Diagrama de contexto del sistema #LaraBot. 201
Figura 73: Arquitectura de chatbots prototipo. 236

ÍNDICE DE TABLAS

Tabla 1: Frameworks de Serverless más populares. 39
Tabla 2: Comparativa de plataformas FaaS según el enfoque. 40
Tabla 3: Comparativa de límites de plataformas FaaS. 40
Tabla 4: Comparativa de límites de la capa gratuita de plataformas FaaS. 41
Tabla 5: Comparativa de límites de la capa gratuita de plataformas FaaS. 41
Tabla 6: Plantilla de refinamiento de escenarios propuesta por Len Bass. 129
Tabla 7: Motivaciones arquitectónicas priorizadas. 148
Tabla 8: Elementos resultantes de la descomposición del sistema. 149
Tabla 9: Motivantes arquitectónicas de #ChatPlatform. 153
Tabla 10: Candidatos a motivantes arquitectónicos principales para el #ChatPlatform. 154
Tabla 11: Acciones permitidas por el usuario en #ChatPlatform 157
Tabla 12: Motivantes arquitectónicas de #WebApp. 160
Tabla 13: Candidatos a motivantes arquitectónicos principales para el #WebApp. 161
Tabla 14: Acciones permitidas por el usuario en #WebApp. 164
Tabla 15: Motivaciones arquitectónicas de #InputGateway. 166
Tabla 16: Candidatos a motivantes arquitectónicos principales para el #InputGateway. 167
Tabla 17: Acciones permitidas por el usuario en #InputGateway. 171
Tabla 18: Motivaciones arquitectónicas priorizadas. 172
Tabla 19: motivantes arquitectónicos principales candidatos para el #IntentDetector. 173
Tabla 20: Acciones permitidas en #IntentDetector. 177
Tabla 21: Motivaciones arquitectónicas de #BotEngine. 178
Tabla 22: Candidatos a motivantes arquitectónicos principales para el #BotEngine. 179
Tabla 23: Acciones permitidas por el usuario en #BotEngine. 182
Tabla 24: Motivantes arquitectónicas de #OutputInterface. 183
Tabla 25: Motivantes arquitectónicos principales (ASRs) de #OutputInterface. 184
Tabla 26: Acciones permitidas por el usuario en #OutputInterface. 186
Tabla 27: Matriz de trazabilidad entre los objetivos de negocio y motivantes arquitectónicos. 228
Tabla 28: Matriz de trazabilidad entre atributos de calidad y escenarios de calidad. 229
Tabla 29: Matriz de trazabilidad entre los motivantes arquitectónicos y componentes. 231
Tabla 30: Matriz de trazabilidad entre los casos de uso y los requerimientos funcionales. 232

1. CAPÍTULO I: GENERALIDADES

1.1. Título

Diseño de una arquitectura conversacional por texto que consuma servicios cognitivos de NLP y Machine Learning para la interacción con chatbots mediante el uso de lenguaje natural.

1.2. Definición del problema

1.2.1. Descripción del problema

Los chatbots, también conocidos como agentes conversacionales, son un software que usualmente se desarrolla con una arquitectura simple que busca la identificación de intenciones[1] y entidades[2], las cuales permiten clasificar una respuesta por medio de controladores[3] que siguen un conjunto de patrones. Estas intenciones y entidades limitan el software al uso y/o implementación de estructuras de datos para responder a frases preestablecidas [2, 10].

Debido a la naturaleza de los lenguajes, una misma intencionalidad se puede representar con un número casi infinito de frases sinónimas que tienden a ser repetitivas, redundantes y a cambiar en detalles irrelevantes dada la limitada capacidad de una máquina de entender el idioma. Esto hace que la base de conocimiento predefinida que se puede llegar a crear no sea suficiente y tenga poca capacidad para abarcar todos los casos de clasificación de una misma intención [11], dado que el lenguaje humano es difuso, ambiguo y requiere mucho contexto para ser interpretado.

Recientes investigaciones demuestran que cerca del 75% de las experiencias de usuario usando chatbots es pobre y, a pesar de los avances significativos en el área de la inteligencia artificial para crear agentes conversacionales basados en Machine Learning, resulta complejo relacionar adecuadamente la petición de un usuario con los recursos adecuados [6]. Por lo tanto, en el mercado actual los chatbots basados en reglas o patrones siguen siendo predominantes, pese a la gran cantidad de frameworks y soluciones emergentes. Esto es debido en gran parte a la falta de una arquitectura clara que permita una implementación sostenible basada en atributos de calidad como la extensibilidad, escalabilidad y mantenibilidad [12].

Teniendo en cuenta lo mencionado, se evidencia una gran área en la que se podría mejorar la forma en la que los chatbots son implementados a la fecha. Esto principalmente relacionado a la poca capacidad de entendimiento y la falla constante del agente conversacional para asociar de forma adecuada las frases o peticiones del usuario con su intención correspondiente; sumado a la necesidad de soluciones agnósticas a la plataforma que permitan integrar estos chatbots con servicios cognitivos y múltiples canales de comunicación como Slack, Microsoft Teams y Google Chat.

---

[1] Intención: La intención del usuario extraída del mensaje [1].

[2] Entidad: Estructura de la información que da significado a la intención [1].

[3] Controlador: Comúnmente conocido como “handler” en inglés, se encarga de consumir recursos como intenciones y entidades para generar una respuesta [1].

1.2.2. Formulación del problema

Se conoce como chatbot o agente conversacional al software con el que se puede interactuar a través de una interfaz de texto y/o audio para la ejecución de tareas simples de negocio. Normalmente su interacción se limita a responder y obedecer una serie de órdenes previamente definidas [1].

Una gran expectativa en los chatbots se evidencia en el número considerable de estos en los diferentes mercados, como lo son el soporte y PQR (Peticiones, Quejas y Reclamos); incluso un 40% fueron desarrollados en 2018 [24], pero se prevé que serán descartados en 2021 [22], debido a la incapacidad de estos para relacionar frases con los recursos adecuados.

En la búsqueda de desarrollar chatbots que entreguen mejores resultados, se han implementado técnicas de procesamiento de lenguaje natural (a partir de ahora NLP, por sus siglas en inglés) [9] que han tenido un desempeño positivo al permitir entender un poco mejor la intencionalidad de una frase. El NLP es un campo amplio y se han visto mejoras cuando es usado en conjunto con el machine learning, ya que este último aporta un manejo de contexto y aprendizaje basado en la experiencia previa [13, 16]. Gracias a estos avances, actualmente existen servicios cognitivos, como luis.ai, IBM watson y wit.ai, que ofrecen mecanismos de reconocimiento de intenciones y entidades a través de APIs[4] que permiten a los desarrolladores obtener predicciones con un alto porcentaje de acierto y a un bajo costo.

Para ofrecer mejores experiencias conversacionales, los desarrolladores se enfrentan a arquitecturas cada vez más complejas de mantener, dada la necesidad de implementar sistemas que coordinen integración con servicios cognitivos, como entrada para la interfaz conversacional e integración a múltiples canales de comunicación que deben respetar o permitir manejar de forma sostenible atributos de calidad como la extensibilidad, escalabilidad y mantenibilidad. Serverless se presenta entonces como una solución que permite a los desarrolladores delegar el mantenimiento de la infraestructura a proveedores externos, ya que desde su concepción está diseñado para ser flexible y escalable al permitir al desarrollador enfocarse en la implementación del software de forma sostenible [14].

Dados estos hechos, es conveniente y posible diseñar una arquitectura que especifique cómo implementar chatbots de forma sostenible y, de este modo, satisfacer la extensibilidad, escalabilidad y mantenibilidad; y, a su vez, que permita hacer uso de sistemas cognitivos que apliquen técnicas de Machine Learning y NLP para la detección de intenciones, y su posterior uso en conversaciones más naturales (en consecuencia del poco número de chatbots comerciales que puedan entender las conversaciones y su relación con los recursos).

[4] API: Application Programming Interface.

1.3. Objetivo de la investigación

1.3.1. Objetivo general

Diseñar una arquitectura conversacional por texto, que sea extensible, escalable y mantenible mediante serverless, haciendo uso de servicios cognitivos de Machine Learning y NLP que facilite la implementación de chatbots con la capacidad de mantener conversaciones naturales a través de la generación de respuestas más precisas en el contexto de la intención del mensaje enviado al chatbot.

1.3.2. Objetivos específicos

1. Investigar, analizar y definir metodologías a usar para el diseño de una arquitectura de chatbot basada en serverless integrando servicios cognitivos de Machine Learning (ML) y procesamiento de lenguaje natural (NLP).

2. Realizar un estudio comparativo de las arquitecturas usadas en las diferentes tecnologías y frameworks más populares en el desarrollo de interfaces conversacionales.

3. Establecer el conjunto de características y métricas que mejor se adapten al diseño de un agente conversacional (chatbot), basado en el aprendizaje automático, que entienda el significado de las oraciones para satisfacer atributos de calidad como extensibilidad, escalabilidad y mantenibilidad.

4. Investigar, enumerar y describir los diferentes servicios cognitivos y frameworks de código abierto que permiten de una mejor forma clasificar la intencionalidad de mensajes recibidos usando NLP y Machine Learning.

5. Investigar las mejores prácticas en el desarrollo de arquitecturas basadas en Serverless que permitan satisfacer las necesidades de un agente conversacional.

6. Modelar una arquitectura para agentes conversacionales (chatbots) por texto basada en serverless que hagan uso de NLP y machine learning.

7. Realizar un estudio comparativo sobre las métricas existentes para validar el diseño de una arquitectura.

8. Validar la arquitectura diseñada usando las métricas obtenidas en el objetivo anterior para certificar y/o aprobar la objetividad del sistema.

1.4. Justificación de la investigación

Las interfaces de usuario en el mercado durante muchos años han hecho uso de la visión y el tacto como medios predominantes de interacción. Sin embargo, gracias a los avances en las interfaces y formas de interactuar con el software junto a las mejoras en comprensión del lenguaje natural (NLU, por sus siglas en inglés), han permitido a los usuarios usar el texto y la voz de forma más común como medios para la ejecución de recursos [7, 18].

En el mundo de las interacciones por texto y audio, tenemos las interfaces de usuario conversacional (CUI), conocidas como agente o chatbot, que buscan responder a mensajes hechos por los usuarios, los cuales demandan interfaces con capacidad de contexto y comprensión humana [7], y exigen conversaciones naturales que les permitan tener una experiencia más sofisticada con las empresas y el software con el que interactúan día a día [8].

Hay técnicas como el procesamiento de lenguaje natural (NLP) que buscan darles a las máquinas la capacidad de comprender la emocionalidad e intencionalidad de un mensaje [4] y de esta forma proveer al software la capacidad de entender y mantener una conversación con los usuarios [2]. Sin embargo, debido a que es un área enfocada en usar las relaciones de las palabras para entenderlas, procesarlas y sacar utilidad de ellas, el NLP no es una herramienta que se pueda usar de forma única para resolver otros retos de los chatbots como lo es un manejo de contexto y aprendizaje basado en experiencia previa; esta área puede ser complementada haciendo uso de Machine Learning [17].

Las arquitecturas y tecnologías usadas para desarrollar chatbots están condicionadas a marcos de trabajo complejos para su sostenibilidad y, dada la necesidad de soportar integración con múltiples plataformas y canales de comunicación, implican diversas tecnologías, bases de datos y soporte de alta concurrencia. Serverless se presenta, entonces, como un modelo de computación en la nube ideal para el desarrollo de chatbots, ya que ofrece una solución escalable que simplifica a los desarrolladores el mantenimiento de la infraestructura del sistema y, por consiguiente, permite integrar de forma flexible servicios cognitivos para generar respuestas que se relacionen con lo que un usuario requiere [14].

Este proyecto presenta el estado del arte de las interfaces conversacionales y busca lograr un impacto positivo en el desarrollo de chatbots comerciales a partir de la propuesta de un diseño de arquitectura de software sostenible que satisfaga atributos de calidad clave, como la extensibilidad, escalabilidad y mantenibilidad, siguiendo las mejores prácticas y patrones de diseño que permitan implementar un agente conversacional con capacidades para mantener una conversación basada en contexto e intencionalidad del mensaje recibido.

2. CAPÍTULO II: MARCO REFERENCIAL

2.1. Marco Conceptual

2.1.1. ChatBot

Un chatbot es un programa informático que permite a los humanos interactuar con la tecnología utilizando una variedad de métodos de entrada, como voz, texto, gestos y tacto, para mantener una conversación con una persona al proveer respuestas automáticas a entradas hechas por el usuario. Habitualmente, la conversación se establece mediante texto, aunque también hay modelos que disponen de una interfaz de usuario multimedia [23].

2.1.2. Conceptos relacionados con la arquitectura de software

2.1.2.1. Ambiente (Environment)

El ambiente en un sistema hace referencia principalmente al entorno en el que los componentes del sistema actúan, siendo así un conjunto de variables y aspectos de ejecución que actúan sobre los componentes del sistema [29].

El ambiente determina la totalidad de las características y circunstancias de las influencias sobre el sistema a lo largo de su ciclo de vida, incluyendo sus interacciones con este entorno. El ambiente del sistema puede contener otros sistemas [31].

2.1.2.2. Arquitectura de sistema [29]

La arquitectura de un sistema define la organización de sus conceptos o propiedades fundamentales, incluyendo sus componentes, interfaces, procesos, restricciones, comportamientos, relaciones entre sí y el ambiente; además, de los principios que gobiernan su diseño y evolución. Algunas nociones sobre las funciones de una arquitectura son:

- Proporcionar una vista estructural de alto nivel.
- Definir el estilo o combinación de estilos para una solución.
- Se concentra en requerimientos no funcionales.
- Los requerimientos funcionales se satisfacen mediante modelado y diseño de aplicación.

2.1.2.3. Stakeholder [55]

Individuo, equipo u organización que comparte intereses sobre un sistema, ya sea porque tiene derechos o el mismo posee características que satisfacen un conjunto de necesidades o expectativas. Dentro de los Stakeholders podemos encontrar, usuarios u organizaciones finales, administradores, clientes, desarrolladores, proveedores, socios, inversionistas, entre otros.

2.1.2.4. Arquitectura de software

La arquitectura de software provee el diseño de más alto nivel de la estructura o estructuras del sistema, que comprenden elementos de software, las propiedades externamente visibles de estos elementos y las relaciones entre ellas [31].

2.1.2.5. Estilo arquitectónico

Los estilos arquitectónicos son conceptos abstractos de "alto nivel", que ofrecen soluciones reutilizables a problemas que son recurrentes en el diseño. Definen roles, la topología y el comportamiento de las interacciones que rodea a los componentes [29].

2.1.3. Elementos correspondientes de una arquitectura para chatbot

2.1.3.1. Intención (Intent)

En el área de los chatbots, la intención (intent) se refiere al significado de un mensaje en una conversación. La intención es un factor crítico en la funcionalidad del chatbot porque es la capacidad de este para analizar lo que realmente se quiere decir y los recursos correspondientes para dar una respuesta apropiada, lo cual finalmente determina el éxito de la interacción [23].

Para que un chatbot pueda entregar respuestas más precisas con base a la intención identificada debe:

- Estar bien programado y entrenado con un modelo útil que incluye muchos datos de entrenamiento.
- Aprovechar el aprendizaje automático para avanzar y mejorar constantemente aprendiendo de la experiencia previa.

2.1.3.2. Entidad (Slot/Entity)

En un chatbot, una entidad se refiere al modificador que el cliente usa para describir su problema, este puede ser texto que describe casi cualquier cosa: una hora, lugar, persona, elemento, número, etc. Mediante el procesamiento del lenguaje natural (PNL), los chatbots pueden extraer entidades de las entradas que los usuarios escriben para modificar la intención del usuario, lo cual permite dar recomendaciones y respuestas precisas. Una entidad en un chatbot se usa para agregar valores a la intención de búsqueda [23].

2.1.3.3. Contexto

El contexto es ese conjunto de circunstancias o situaciones durante un proceso de comunicación donde se encuentran el emisor y el receptor y donde se produce el mensaje. El contexto es muy importante en la comunicación, pues las variaciones en un mismo lenguaje y las diferencias culturales hacen que aquello que para unos es correcto, para otros pueda no serlo.

En el caso de un chatbot el contexto es aquel conjunto de interacciones unidas en una estructura de árbol lógica. La posición en ese árbol define cuándo la prioridad y la disponibilidad de la interacción hace que se generen diferentes respuestas de bot [2]. El contexto cambia cada vez que el bot se mueve a lo largo de la historia y abre nuevas interacciones posibles, así como también cierra las anteriores [23].

2.1.4. Controlador (Handler)

Un handler es un componente responsable de relacionar la intención del usuario con los recursos adecuados. Un chatbot puede contener un gran conjunto de controladores, los cuales tienen la responsabilidad de dar respuesta a una única intención, por lo tanto, al recibirse una intención, todos los controladores disponibles deben ser evaluados y sólo uno de ellos podrá retornar una respuesta [24].

Cada controlador evalúa su capacidad de dar respuesta a la intención recibida teniendo en cuenta también un contexto previo, dado que interacciones previas pueden modificar totalmente la intencionalidad del mensaje recibido.

2.1.5. Procesamiento del Lenguaje Natural (NLP)

El procesamiento de lenguaje natural se utiliza para hacer posible que las computadoras lean texto, escuchen la voz hablada, la interpreten, midan el sentimiento y determinen qué partes son importantes, esto se logra mediante el proceso de dividir la entrada del usuario en oraciones y palabras.

También se estandariza el texto a través de una serie de técnicas, por ejemplo, convirtiéndolo todo en minúsculas o corrigiendo errores ortográficos antes de determinar, entender, interpretar y manipular el lenguaje humano [23].

2.1.5.1. Entendimiento del Lenguaje Natural (NLU)

Es un campo de procesamiento de lenguaje natural (NLP) que se ocupa de la conversión del lenguaje en una representación semántica que una computadora puede interpretar [10]. La comprensión del lenguaje natural ayuda al chatbot a comprender lo que dijo el usuario utilizando objetos de lenguaje generales y específicos del dominio, como léxicos, sinónimos y temas. Estos se usan junto con algoritmos o reglas para construir flujos de diálogo que le dicen al chatbot cómo responder [23].

2.1.5.2. Generación de Lenguaje Natural (NLG)

Brindar una experiencia significativa y personalizada más allá de las respuestas pre escritas requieren la generación de lenguaje natural. Esto permite al chatbot interrogar a los repositorios de datos, incluidos los sistemas de back-end integrados y las bases de datos de terceros, y utilizar esa información para crear una respuesta [23].

2.1.6. Machine Learning

El ML es una aplicación de la inteligencia artificial (IA) que proporciona a los sistemas la capacidad de aprender y mejorar automáticamente a partir de la experiencia sin ser programado explícitamente.

El ML se centra en el desarrollo de programas informáticos que pueden acceder a los datos y utilizarlos para aprender por sí mismos.

El proceso de aprendizaje comienza con observaciones o datos, como ejemplos, experiencia directa o instrucción, para buscar patrones en los datos y tomar mejores decisiones en el futuro en función de los ejemplos proporcionados. El objetivo principal es permitir que las computadoras aprendan automáticamente sin intervención o asistencia humana y ajustar las acciones en consecuencia.

2.1.7. Serverless

Serverless es un modelo de ejecución de computación en la nube donde el proveedor de la nube administra dinámicamente la asignación y aprovisionamiento de servidores [20]. Una aplicación sin servidor se ejecuta en contenedores de cómputo sin estado que son activados por eventos efímeros (pueden durar una invocación) y totalmente administrados por el proveedor de la nube [21].

El precio se basa en el número de ejecuciones y duración de estas en lugar de la capacidad de cómputo pre-comprada. La gran diferencia entre un sistema serverless y uno convencional es su capacidad para funcionar durante cortos periodos de tiempo, al solo ser activados por eventos de invocación, además de poder escalar de forma dinámica sin ningún esfuerzo de desarrollo por parte de quien contrata el servicio [19].

2.1.8. Insight [61]

Un insight es un resultado obtenido con base a un análisis que permite llegar a una percepción, entendimiento o conocimiento acerca de un comportamiento estudiado. Los insights se obtienen a partir de información que posee el investigador, pero no están a simple vista y requiere de un tratamiento de datos especial por medio de la transformación de estos en diferentes tipos de visualizaciones a análisis estadísticos, dando como resultado una deducción, la raíz de un problema o una oportunidad.

2.1.9. HTTP (Hypertext Transfer Protocol) [97]

Abreviatura que en español traduce cómo “Protocolo de transferencia de hipertextos”, el cual es un protocolo de comunicación que permite la transferencia de información en la nube muy orientado al concepto de transacciones, siguiendo el esquema petición-respuesta, comúnmente utilizado en el modelo cliente-servidor. HTTP es un protocolo sin estado, dado que no guarda ningún tipo de información sobre peticiones anteriores.

2.1.10. Framework [95]

En desarrollo de software, un framework es un marco de trabajo que ofrece un conjunto de componentes genéricos, librerías y herramientas las cuales han sido construidas, probadas y optimizadas por diversos profesionales en busca de ofrecer una solución que permita crear aplicaciones de forma versátil, robusta y eficiente. El uso de frameworks da la posibilidad al desarrollador enfocarse en la lógica de negocio y la funcionalidad de la aplicación a alto nivel.

2.1.11. API (Application Programming Interface) [96]

El término API en español traduce cómo interfaz de programación de aplicaciones, pero en el mundo de desarrollo de software se referencia por su definición en inglés. Se trata de un conjunto de definiciones y protocolos que ofrecen una interfaz de comunicación entre dos aplicaciones, muy comúnmente referenciado en los modelos cliente-servidor, lo que permite el paso de mensajes para el consumo o envío de recursos.

2.1.12. REST (Representational State Transfer) [96]

REST hace referencia a un conjunto de restricciones y especificaciones orientadas a la creación de un estilo de arquitectura de software, que permita la comunicación entre servicios o aplicaciones de una manera estandarizada. Se aplica principalmente a la definición de APIs, las cuales se suelen referenciar cómo REST APIs, y aquellas que cumplen casi al 100% con la especificación se suelen definir cómo RESTFul APIs.

2.1.13. SDK (Software Development Kit) [95]

Conjunto de herramientas de desarrollo ofrecidas por una plataforma o proveedor específico, para la construcción de aplicaciones de software sobre su infraestructura.

2.1.14. Sistema Agnóstico [67]

La premisa de un sistema agnóstico hace referencia a un producto que independiente de la plataforma se ejecuta igualmente bien y de forma estandarizada, eliminando el alto acoplamiento, lo cual permite desarrollar y desplegar características en una amplia gama de canales de distribución. En el área del desarrollo de software se suele asociar con sistemas que poseen SDKs que permiten implementaciones en diferentes lenguajes de programación y/o pueden ser desplegados en diferentes plataformas o proveedores.

2.2. Marco de antecedentes

2.2.1. Historia del NLP

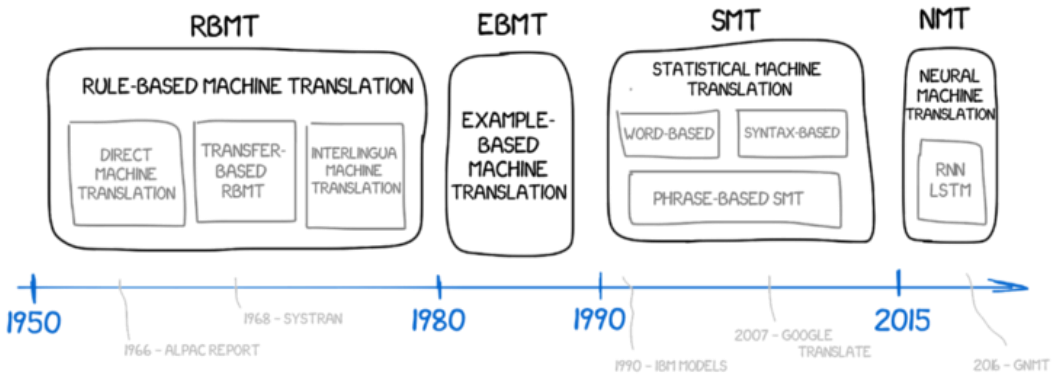

Figura 1: Linea de tiempo del NLP [57].

2.2.2. Inicios del NLP

Algunos mencionan la relación de los orígenes del NLP a la década de 1950 [56], lo cual no es totalmente erróneo, pero si se toma en cuenta que el NLP es un campo muy amplio y entre las técnicas de procesamiento se pueden destacar la interpretación de un idioma a otro, la traducción y el manejo de frases, se debe mencionar como en 1933 el científico soviético Peter Troyanskii presentó "la máquina para la selección e impresión de palabras al traducir de un idioma a otro"[57] a la Academia de Ciencias de la URSS. Este incluía tanto el diccionario bilingüe como un método para tratar los roles gramaticales entre idiomas, basado en esperanto.

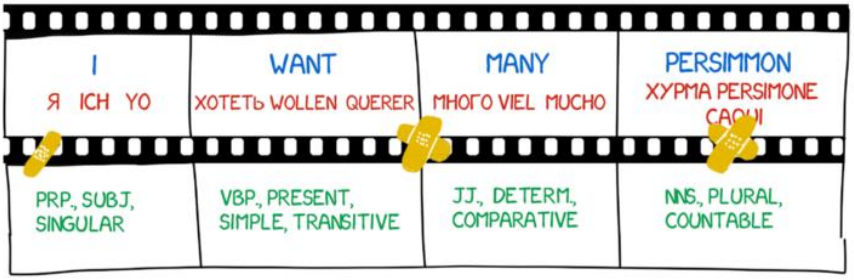

Figura 2: Ejemplo de la máquina de traducción [57]

La idea de esta era que el operador tomará la primera palabra del texto, y así encontrar la tarjeta correspondiente, para posteriormente tomar una foto y escribir sus características morfológicas (sustantivo, plural, genitivo) en la máquina de escribir. Las teclas de la máquina de escribir codificaban una de las funciones.

La cinta y la película de la cámara se utilizaban simultáneamente, realizando un conjunto de fotogramas con palabras y su morfología. Desgraciadamente no fue sino hasta 1956 que se conoció el trabajo del profesor Troyanskii [57].

A Pesar de ser un sistema simple esto daría pie a los primeros pasos del NLP como herramienta para trabajar sobre idiomas de forma automatizada. También cabe mencionar que las primeras patentes para "máquinas traductoras" fueron solicitadas a mediados de la década de 1930, por Georges Artsrouni, siendo su patente simplemente un diccionario bilingüe **automático** que usaba cinta de papel [57].

2.2.2.1. NLP basado en símbolos y reglas de sintaxis (1950s - comienzos de los 1990s)

La década de 1950 fue una época especialmente importante para la ciencia, ya que con esta se marcaba la recuperación de la segunda guerra mundial y el inicio de la guerra fría [57]. Muchos de los avances que se habían logrado en la segunda guerra mundial comenzaron a ser compartidos, se tiene por ejemplo a Alan Turing publicando en 1950 su famoso artículo "Computing Machinery and Intelligence" en el que proponía lo que ahora se llama la **prueba de Turing** como criterio de inteligencia. Esta prueba es una de las más importantes que se conoce para validar la capacidad de algoritmos y entes que hagan uso de inteligencia artificial y su capacidad para entender.

En 1954 comienza con IBM lo que sería unos años importantes para la traducción automatizada de idiomas [57]. Para esta época IBM se interesó especialmente en desarrollar una tecnología que permitiera traducir en tiempo real frases de un idioma a otro, en este caso del ruso al inglés debido al creciente conflicto entre rusia y estados unidos. Es así como en la sede de IBM en Nueva York, comenzó el experimento Georgetown-IBM, con el cual se consiguió que la computadora IBM 701 tradujera automáticamente 60 frases en ruso al inglés por primera vez en la historia.

En 1957, las estructuras sintácticas de Noam Chomsky revolucionaron la lingüística con la "gramática universal", un sistema de estructuras sintácticas basado en reglas [58]. Este sistema se basaba en reglas de producción que mantenían una memoria de trabajo de las afirmaciones de memoria en curso.

La operación básica para un sistema de reglas de producción implica un ciclo de tres pasos ('reconocer', 'resolver conflicto ', y' actuar ') que se repite hasta que no se apliquen más reglas a la memoria de trabajo. Cada regla era independiente de las demás, permitiendo que las reglas se agregaran y se pudieran eliminar fácilmente. Sin embargo, existen problemas de escalabilidad cuando los sistemas de reglas de producción aumentan de tamaño; se requiere una cantidad significativa de mantenimiento para mantener un sistema con miles de reglas.

A pesar de que lo conseguido con la computadora IBM 701 y las estructuras semánticas de Chomsky fueron hitos importantes, el progreso real alcanzado en años posteriores fue mucho más lento, especialmente después del informe ALPAC en 1966, en el que se destacó que diez años de investigación no cumplieron con las expectativas, generando que la financiación para la traducción automática y por ende también el NLP se redujera drásticamente. Esto como se esperaba desembocó en poco interés en el campo del NLP conduciendo a pocas investigaciones

adicionales en traducción automática hasta finales de la década de 1980, cuando se desarrollan los primeros sistemas estadísticos de traducción automática [56].

En 1969 Roger Schank introdujo la teoría de la dependencia conceptual para la comprensión del lenguaje natural. Este modelo, parcialmente influenciado por el trabajo de Sydney Lamb, fue ampliamente utilizado por los estudiantes de Schank en la Universidad de Yale, como Robert Wilensky, Wendy Lehnert y Janet Kolodner [57].

En 1970, William A. Woods introdujo la red de transición aumentada (ATN) para representar la entrada del lenguaje natural. En lugar de reglas de estructura de frases, los ATN utilizaron un conjunto equivalente de autómatas de estado finito que se llamaban de forma recursiva. Los ATN y su formato más general llamado "ATN generalizado" continuaron utilizándose durante varios años [57]. Durante los años 70, muchos programadores comenzaron a escribir "ontologías conceptuales", que estructuraban la información del mundo real en datos comprensibles por computadora.

Como se puede notar desde la década de 1950, la investigación del NLP se centró mucho en tareas como traducción automática, recuperación de información, resumen de texto, respuesta a preguntas, extracción de información y modelado de temas [58]; esto condujo a que muchos de los sistemas de NLP hasta la década de 1980 estuviesen basados en conjuntos de reglas escritas de forma manual, en consecuencia a la naturaleza de las investigaciones que enfocaban más su visión en problemas de sintaxis en lugar de soluciones alternativas como el machine learning.

Sin embargo, con el comienzo de los 80s hubo un cambio en el enfoque del NLP con la introducción de algoritmos de machine learning para el procesamiento del lenguaje [56]. Algunos de los primeros algoritmos de machine learning utilizados, como los árboles de decisión, producían sistemas de reglas estrictas “if-then” similares a las reglas escritas a mano existentes. Esto último facilitaba más la mantenibilidad e implementación de estos sistemas, cambiando la forma de trabajar con el NLP con estudios posteriores enfocados más en la estadística.

2.2.2.2. NLP basado en estadística (1990s - 2010s)

Con los avances conseguidos en las etapas tempranas del desarrollo de redes que permitían evaluar mejor las frases, un ejemplo son las redes bayesianas (Pearl, 1985) (también conocidas como redes de creencias) que proporcionan un medio para expresar distribuciones de probabilidad conjuntas sobre muchas hipótesis interrelacionadas. Hizo que la investigación se centrará cada vez más en modelos estadísticos, que toman decisiones probabilísticas suaves basadas en asignar ponderaciones de valor real a las características que componen los datos de entrada.

A comienzos de la década de 1990 se empezaron a notar grandes avances en el campo de la traducción de texto (no olvidemos que gran parte de lo que se buscaba con el NLP en sus orígenes era conseguir sistemas automatizados de traducción) gracias en gran medida a los métodos estadísticos usados en el NLP. Un ejemplo de esto se mostró por primera vez con el

trabajo de IBM Research, con un sistema de traducción automática que no sabía nada sobre reglas y lingüística en su conjunto. Este analizaba textos similares en dos idiomas y a través de ponderaciones trataba de comprender los patrones.

El modo en el que se operaba era el siguiente, se tomaba una oración idéntica en dos idiomas dividida en palabras, que luego se emparejaban. Esta operación se repetía unos 500 millones de veces para contar, por ejemplo, cuántas veces la palabra "Das Haus" se traducía como "House" frente a "building", frente a "construcción", y así sucesivamente. Estos sistemas tomaban ventaja de textos multilingües existentes que habían sido elaborados por el Parlamento de Canadá y la Unión Europea como resultado de leyes que exigían la traducción de todos los procedimientos gubernamentales a todos los idiomas oficiales de los sistemas de gobierno correspondientes.

La primicia de esta solución era que, si la forma en la que los intérpretes traducen las frases era la correcta, las máquinas también debían hacerlo igual, parametrizando estadísticamente como se hacían las traducciones.

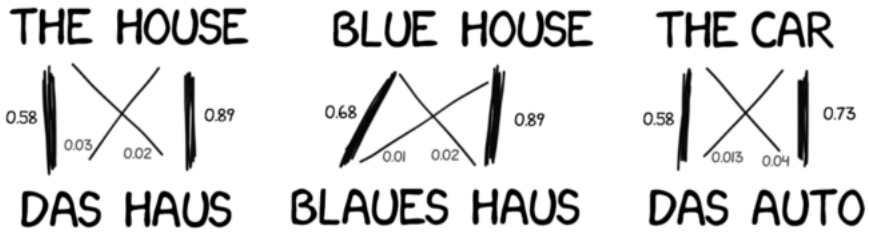

Figura 3: Traducción Basada en estadística [57].

Sin embargo, la mayoría de los demás sistemas dependían de un corpora desarrollada específicamente para las tareas implementadas por estos sistemas, lo que fue (y a menudo sigue siendo) una limitación importante en el éxito de estos sistemas. Como resultado, una gran cantidad de investigación se enfocó a métodos para aprender de manera más efectiva a partir de cantidades limitadas de datos.

Ya con los avances detallados por IBM usando estadística para la clasificación de palabras, se buscó mejorar aún más los resultados y es así como en década del 2000, se empezó a utilizar los n-gramas, que eran una secuencia contigua de n palabras en una fila. Así, la máquina ya no solo aprendía a entender palabras individuales sino también combinaciones constantes de palabras, lo que mejoró notablemente la precisión.

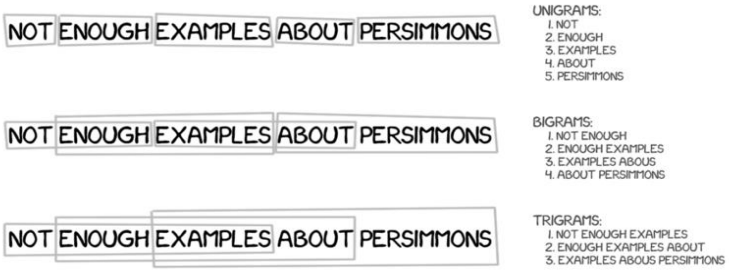

Figura 4: Visualización de n-gramas en una fase [57].

De igual forma con el crecimiento de la web, desde mediados de la década de 1990 se dispone de cantidades cada vez mayores de datos lingüísticos sin procesar (sin anotar). Lo que permitió que la investigación se centrará cada vez más en algoritmos de aprendizaje no supervisados, semi-supervisados y redes semánticas [58]. Dichos algoritmos pueden aprender de los datos que no se han anotado manualmente con las respuestas deseadas o utilizando una combinación de datos anotados y no anotados. Generalmente, esta tarea es mucho más difícil que el aprendizaje supervisado y, por lo general, produce resultados menos precisos para una determinada cantidad de datos de entrada.

Los modelos de lenguaje en los que ahora se basan muchos sistemas de reconocimiento de voz son ejemplos de tales modelos estadísticos. Estos modelos son generalmente más robustos cuando se les da una entrada desconocida, especialmente una entrada que contiene errores (como es muy común para los datos del mundo real), y producen resultados más confiables cuando se integran en un sistema más grande que comprende múltiples subtareas.

Algunos trabajos destacables en el NLP del comienzo de los 2000s son la lógica difusa, la cual tiene una relación directa con la programación neurolingüística PNL (Carvalho, Batista y Coheur, 2012) para tareas como el análisis de sentimientos (Subasic y Huettner, 2001), el resumen lingüístico (Kacprzyk y Zadrozny, 2010), la representación del conocimiento (Lai, Wu, Lin y Huang, 2011) y la inferencia del significado de las palabras (Kazemzadeh, Lee y Narayanan, 2013).

Las redes neuronales artificiales pueden ayudar a completar tareas de PNL como resolución de ambigüedad (Chan y Franklin, 1998; Costa, Frasconi, Lombardo y Soda, 2005), inferencia gramatical (Lawrence, Giles y Fong, 2000). La computación evolutiva se puede explotar para tareas como la evolución gramatical (O'Neill & Ryan, 2001), el descubrimiento de conocimientos (Atkinson-Abutridy, Mellish y Aitken, 2003), la categorización de textos (Araujo, 2004) y el aprendizaje de reglas (Ghandar, Michalewicz, Schmidt, To y Zurbruegg, 2009) [58].

2.2.2.3. NLP basado en sistemas neuronales (2010s - presente)

En años recientes diferentes investigaciones se han desarrollado con el objetivo de mejorar más las capacidades del NLP, ya que había una preocupación respecto a los algoritmos que se estaban usando, principalmente porque al estar basados en palabras están limitados solo a la información que pueden "ver" y, por lo tanto, tarde o temprano alcanza una saturación [58].

Es así como se cambia nuevamente el enfoque gracias a los avances realizados en el desarrollo de modelos basados en redes neuronales a comienzos del 2000permitiendo que las investigaciones recientes se hayan centrado cada vez más en los algoritmos de aprendizaje no supervisados y semi-supervisados [59]. En la figura 5 se denota el aumento en el número de investigaciones recientes de NLP centras cada vez más en el uso de nuevos métodos de aprendizaje profundo.

Las redes neuronales basadas en representaciones de vectores densos han estado produciendo resultados superiores en varias tareas de NLP. Esta tendencia es provocada por el éxito del uso de representación de palabras (Luong, Socher y Manning, 2013) y reconocimiento de emociones (Cambria, Gastaldo, Bisio y Zunino, 2014) y los métodos de deep learning. Dichos algoritmos y métodos pueden aprender de los datos que no se han anotado manualmente con las respuestas deseadas, o utilizando una combinación de datos clasificados y no clasificados. Actualmente hay una enorme cantidad de datos no clasificados disponibles (que incluyen, entre otras cosas, todo el contenido de la World Wide Web), que a menudo pueden compensar los resultados obtenidos.

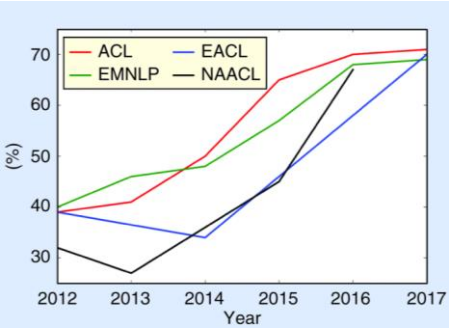

Figura 5: Tendencia de papers de NLP basados en deep Learning y redes neuronales [59].

Con los avances logrados en esta década, varios modelos de deep learning se han convertido en los nuevos métodos de vanguardia para los problemas de NLP. En general, el aprendizaje puede ser supervisado (cada elemento de los datos de entrenamiento está etiquetado con la respuesta correcta) o no supervisado, cuando no lo está, y el proceso de aprendizaje trata de reconocer patrones automáticamente (como en el análisis de clúster y factores) [58].

El aprendizaje supervisado es la práctica más popular en la investigación reciente de aprendizaje profundo para el NLP. En muchos escenarios del mundo real, sin embargo, tenemos datos sin etiquetar que requieren enfoques avanzados sin supervisión o poca supervisión. Esto es uno de los problemas actuales que se busca resolver en el NLP y el aprendizaje automático ya que el modelo puede aprender haciendo uso de información con ruido a través de los datos de entrenamiento en lugar de solo sus características esenciales deseadas [59].

2.2.3. Historia y evolución de los chatbots

Aunque los chatbots son una tecnología que ha tomado mucho auge en años recientes, y con expectativas elevadas sobre esta, su historia se remonta a muchos años atrás, específicamente a 1964 con el científico informático del MIT Joseph Weizenbaum y su desarrollo ELIZA. Para 1966 el desarrollo de ELIZA la llevo se considerada la primera máquina capaz de hablar utilizando los primeros avances en el procesamiento del lenguaje natural (NLP) [4].

Eliza podía engañar a muchas personas para que creyeran que estaban hablando con un humano, esto lo conseguía tomando el rol de una psicoterapeuta, simplemente sustituyendo sus propias

palabras en guiones y retroalimentando a los usuarios para mantener la conversación [5]. Estos avances condujeron a la creación de un nuevo agente bautizado PARRY en 1972, con algunas aproximaciones hacia un modelo basado en paranoia [23]. Estos pasos iniciales en el desarrollo de chatbots, generaron una buena expectativa, pero llegaron a un estancamiento debido al poco avance en el NLP (ref. informe ALPAC en 1966 genero desmotivación en las investigaciones relacionadas al NLP y desinterés), y falta de soluciones alternativas para conseguir mejores resultados. Es por ello por lo que durante 20 años los avances en este campo fueron pocos, presentando soluciones como lo fueron Racter (1984), Jabberwacky (1988), Loebner Prize, Loebner Prize (1990), Dr. Sbaitso (1991) [23].

En 1995 sin embargo hubo una nueva aproximación un poco más interesante en creación de chatbots con ALICE (Artificial Linguistic Internet Computer Entity), este trajo grandes avances ya que se basó en el Lenguaje de marcado de inteligencia artificial (AIML), que es muy similar a la estructura de las soluciones modernas de hoy en día [5], tuvo una buena aceptación ganando en tres ocasiones el Loebner, galardón que evalúa cual es el software más inteligente de los que se han presentado; esta se basaba en la implementación de detección de patrones heurísticos para relacionar entrada con reglas preestablecidas, es importante aclarar que tomó cerca de 5 años en ser liberada para su uso, siendo el periodo del 2000 al 2004 cuando pudieron verse sus resultados.

Los avances logrados durante esta época demostraron la necesidad creciente de un desarrollo más orientado hacia el uso de inteligencia artificial, debido al poco avance en el desarrollo de bots que pudieran mantener un contexto de la conversación. El NLP había dado grandes resultados, pero por si solo no permitía desarrollar una charla fluida, es por ello que gran parte del éxito conseguido por ALICE, se debiera a que al inspirarse en ELIZA, tomo su experiencia y la llevó más allá en una etapa temprana del manejo de técnicas de inteligencia artificial [5].

Continuando con el desarrollo e implementación de la inteligencia artificial y el machine learning en este campo se llega a Watson, un software desarrollado por IBM, inicialmente con el objetivo de participar en un concurso de televisión llamado “leopardy”. Lo fundamental de Watson ha sido que no se detuvo en ese punto, IBM vio una gran oportunidad debido a los avances que se habían conseguido en el área del machine learning para obtener buenos resultados con Watson, es así como desde entonces, Watson se ha convertido en referencia para la implementación de NLP y machine learning para revelar información de grandes cantidades de datos, ampliando sus funcionalidades al convertirse en un asistente de inteligencia artificial para automatización [6].

Con Watson se refleja lo que sería en la década del 2010 un fuerte interés en la creación de asistentes personales y el desarrollo de inteligencia artificial, que permiten interactuar con una máquina de forma más natural; es aquí donde surgen las soluciones más actuales como el asistente de Google, Cortana, Siri e incluso los chatbots creados por Facebook [6]. Lo anterior

como se ve en la curva de Gartner (Figura 7) dio como resultado una acelerada creación de chatbots, con expectativas bastante infladas, pero que como se ha dicho previamente no han dado los resultados esperados al no tener claro cómo conseguir una implementación adecuada de un sistema basado en modelos de machine learning con NLP, situación actual que comparten a la fecha la mayoría de los chatbots.

2.3. Marco de teórico

2.3.1. Tendencias, casos de uso clave y necesidades de los chatbots

La industria de los chatbots es un mercado prometedor, el cual ha ido creciendo a un ritmo acelerado y según algunas predicciones de "Markets and Markets"[5] se espera pasar de un tamaño de mercado de cerca de 5 billones de dólares en 2020 a 15 billones en 2025.

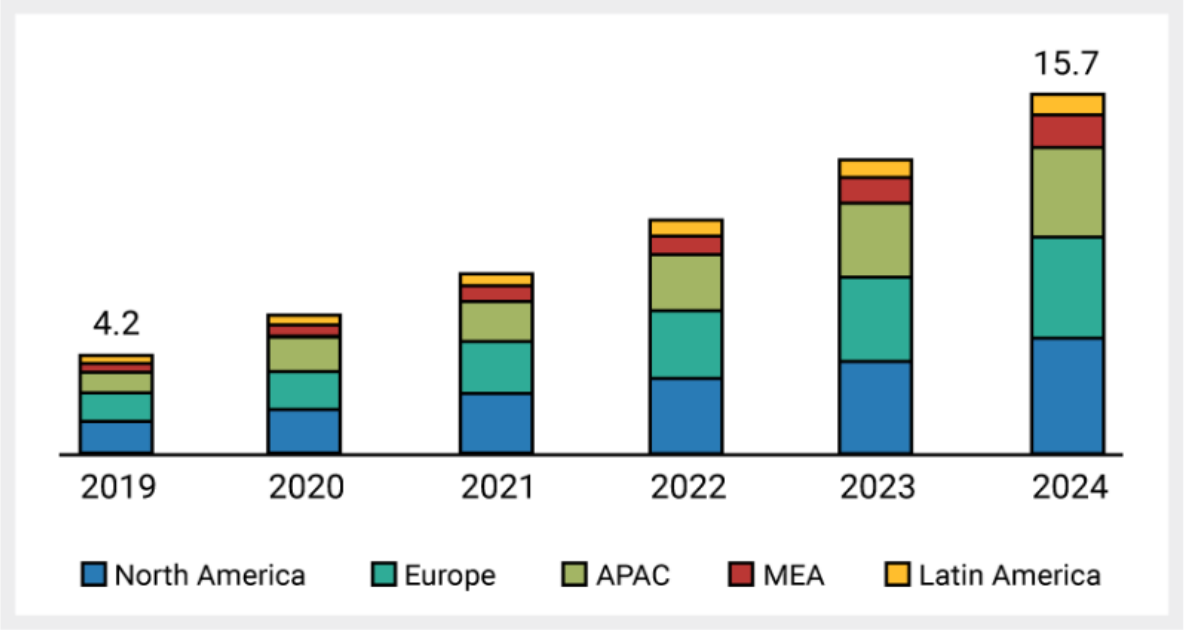

Figura 6: Mercado de los chatbots por regiones (Billones de dólares) [23].

2.3.2. Expectativas o necesidades de los consumidores

A continuación, se comparten algunos de los insights más representativos sobre el mercado de los chatbots compartidos por empresas destacadas en la industria tecnológica [23]:

- "El 63% de los clientes piensan en la interacción con chatbots cuando quieren escribir un mensaje a una compañía o marca" - Mindshare.
- "80% de las empresas planean o ya están utilizando chatbots en 2020" – Oracle.
- "50% de las grandes compañías invertirán más en creación de chatbots en 2021 que en desarrollo de aplicaciones móviles tradicionales" - Gartner.
- "Los chatbots serán responsables de ahorrar costos a las empresas de cerca de $8 billones de dólares anuales en 2022" - Juniper Research.
- "Los clientes y las empresas ahorran 2.5 billones de horas gracias a los chatbots en 2023" - Juniper Research.
- "El tamaño global del mercado en 2025 se espera que alcance los $15 billones de dólares" - Markets and Markets.

[5] Markets and Markets: Empresa dedicada a desarrollar estudios de mercado y consultorías [60].

- “En 2025 las empresas de servicio al cliente que incorporen inteligencia artificial en sus plataformas elevarán la eficiencia operacional en un 25%” – Gartner.

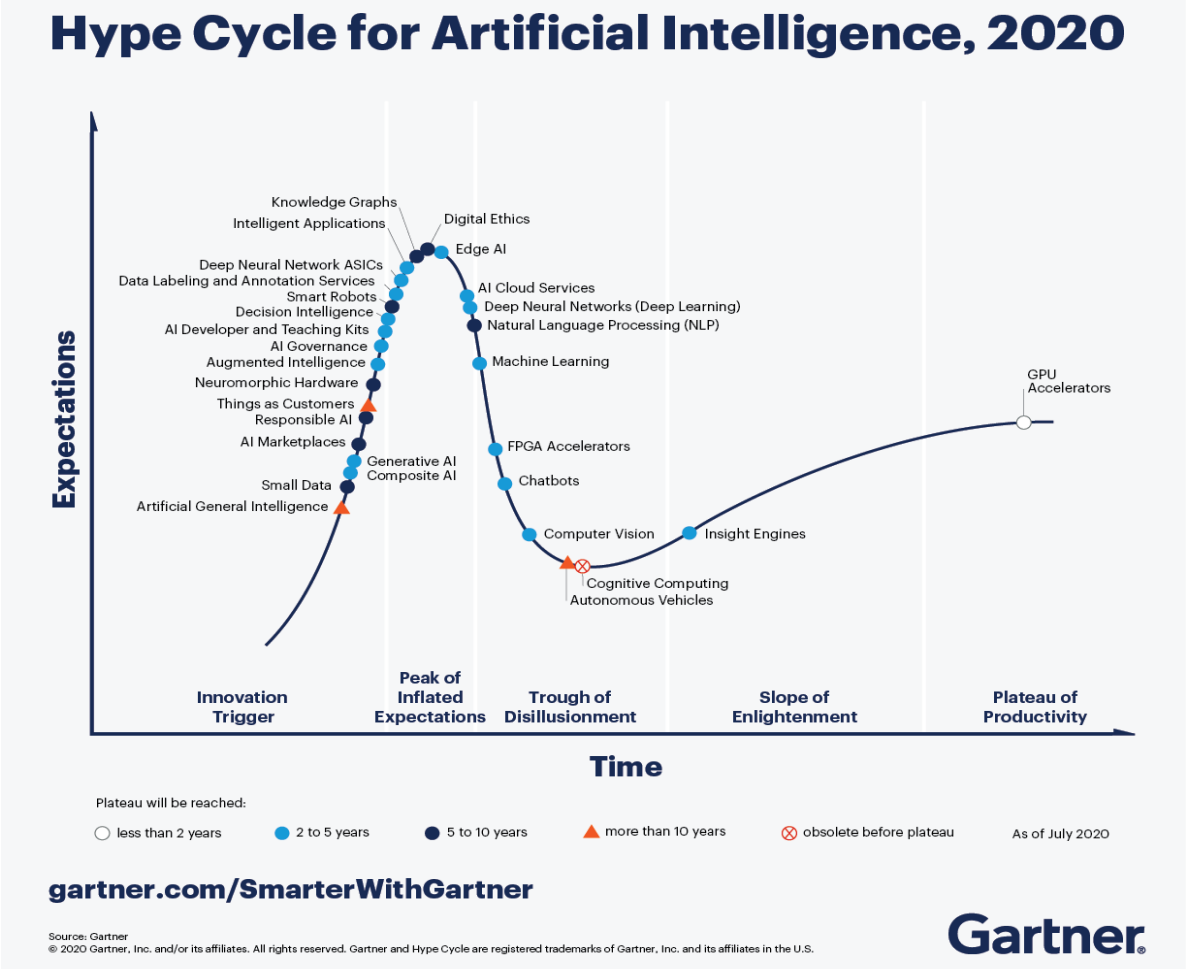

Figura 7: Ciclo de expectativas de la inteligencia artificial en 2020 según Gartner [25]

Por otro lado, Gartner ha compartido en su último reporte insights interesantes donde se puede observar que los chatbots han caído de estar en el pico de las expectativas en 2019 a empezar a caer al valle de la desilusión en 2020, y se prevé que el 40% de los chatbots desarrollados en 2018 serán descartados o ya han sido abandonados en 2020. Esto lejos de ser algo negativo, se debe a la incapacidad de muchos chatbots a cumplir con una serie de expectativas clave por parte de los usuarios cómo lo son el entendimiento del lenguaje natural y la capacidad de asociar intenciones con los recursos adecuados.

Teniendo en cuenta que sobre los chatbots se tienen grandes expectativas, a continuación, se presentan algunos de los casos de uso más demandados por los usuarios según algunos estudios de mercado [62]:

- Poder enviar peticiones, quejas o reclamos.
- Obtener ayuda o resolver preguntas frecuentes.
- Poder tener una respuesta rápida en caso de emergencia.

- Se espera un trato humano en la conversación por parte de los chatbots. Se espera un saludo y despedida cordial.
- Poder pagar una factura.
- Hacer compras online.
- Obtener recomendaciones o ideas para comprar algo.
- Estar disponibles 24x7.
- Atención al cliente vía redes sociales.
- Los chatbots deben poder ser multiplataforma.
- A nivel interno las empresas esperan automatizar procesos de recursos humanos cómo onboardings o seguimiento a empleados.
- El chatbot debe poder ofrecer soporte en múltiples lenguajes.
- Poder establecer recordatorios o alarmas de forma sencilla.
- Poder obtener información precisa acerca de las capacidades del bot incorporando comandos de ayuda.
- Permitir crear atajos o recordar acciones para ayudar a automatizar tareas.

2.3.3. Expectativas o necesidades de los desarrolladores

Dada las altas expectativas anteriormente mencionadas sobre los chatbots, los desarrolladores afrontan nuevos retos y necesidades respecto a las herramientas, soluciones y servicios disponibles a su alcance, entre estos se destacan [2]:

- Tener soluciones multiplataforma que permitan al chatbot estar disponible en diferentes canales de comunicación cómo Slack, Microsoft Teams, Google Chat y entre otros. Hoy en día existen muchos frameworks que suelen tener un acoplamiento alto a plataformas específicas, y los que tienen un desarrollo más agnóstico suelen ser poco flexibles.
- Desarrollar soporte a múltiples idiomas de manera ágil mediante el uso de servicios de traducción.
- Arquitecturas escalables que permitan soportar el alta y a la vez variable demanda de usuarios. Dada esta necesidad serverless se ha convertido en una solución óptima.
- Poder consumir servicios cognitivos que permitan la identificación de la intención del usuario de manera sencilla.
- Dotar de flexibilidad al chatbot en términos de acceso a recursos y capacidad de aprender de la experiencia previa.

2.3.4. Arquitectura de un chatbot

Los chatbots siguen un flujo conversacional bastante simple donde un usuario envía un mensaje y espera una respuesta. La arquitectura para generar esta respuesta se observa con detalle en la figura 8, donde tenemos un **clasificador de intenciones** y **extractor de entidades** los cuales implementan técnicas de NLP, NLU y Machine Learning para obtener **intenciones** y **entidades** a partir del mensaje recibido.

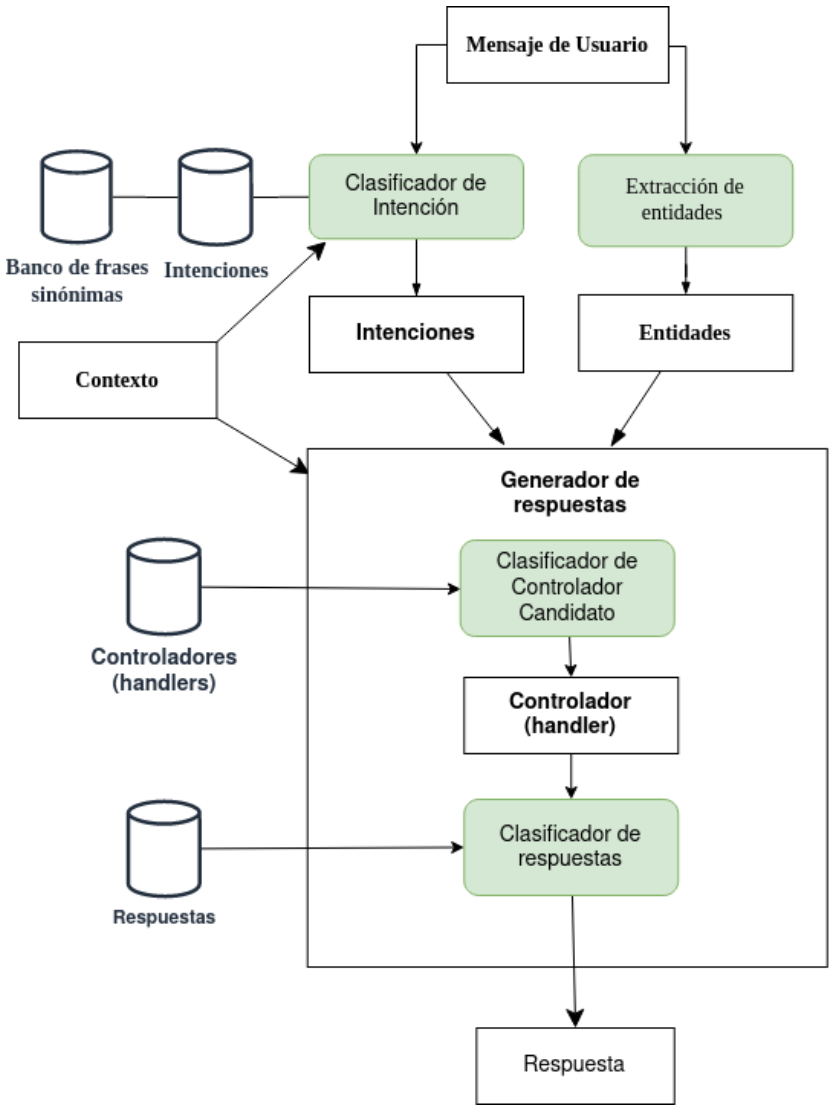

Figura 8: Arquitectura de un chatbot conversacional [2].

La **intención** y las **entidades** sirven de entradas a un motor de generación de respuestas el cual posee una lista de **controladores** responsables de cada tipo de posible respuesta a retornar al usuario, el cual también recibe como entrada un **contexto** en memoria de conversaciones pasadas, ya que la intención y la respuesta esperada pueden cambiar completamente con base a mensajes anteriores.

2.3.5. Limitaciones y retos en los chatbots

Los chatbots poseen un valor importante en el negocio de cualquier compañía que quiera aumentar su fidelización de clientes, mejorar su marca e incluso dar capacidad a los usuarios de realizar acciones sobre su negocio [23]. Sin embargo, actualmente se habla de descartar cerca del 40% de los chatbots desarrollados en los últimos años [22], esta situación se encuentra principalmente relacionada a la naturaleza de las tecnologías usadas en chatbots y su poca capacidad de dar soluciones reales en una conversación humana.

Los chatbots sujetos a este tipo de tecnologías no tiene capacidad de responder preguntas para hacer aclaraciones ni mucho menos redirigir al usuario al recurso correcto que este requiere. Actualmente se pueden mencionar cuatro áreas principales sobre retos y limitantes de los chatbots:

2.3.6. Falta de información para entrenamiento

Es importante aclarar que un sistema de inteligencia artificial requiere grandes cantidades de datos para dar resultados aceptables. Esto para grandes compañías como Facebook, IBM, Google o Microsoft por nombrar algunas, no representa un gran reto debido a un común denominador, todas estas grandes compañías ya tienen en sus sistemas un gigantesco flujo de información, que se nutren día a día.

Puede que obtener data de entrenamiento parezca sencillo, pero resulta ser una gran limitante cuando no se tienen los recursos y sistemas de compañías con tanto flujo de información, generando un gran muro para el desarrollo de cualquier tecnología basada en IA [11].

Analizando el caso concreto de los sistemas de lenguaje, tenemos que lo que es algo natural en humanos: las relaciones entre palabras, frases, oraciones, sinónimos, entidades léxicas, conceptos, etc., deben ser "aprendidas" por una máquina [2]. Esto implica que el uso del aprendizaje automático para comprender a los humanos requiere que los humanos recopilen, seleccionen y depuren cada pieza de datos de entrenamiento de una cantidad asombrosa de información [13]. Estas condiciones hacen que la construcción de aplicaciones conversacionales especialmente los basados en IA sean bastante costosos y de mucho consumo de tiempo.

2.3.7. Poca capacidad de interpretación conversacional

Los sistemas que hacen uso del NLP son generalmente sistemas que permiten entender la intencionalidad de los textos y hacer modificaciones sobre estos, sin embargo, cuando esto se pone en práctica a una conversación real muchas variables afectan el desarrollo de esta, se puede señalar en el historial de una conversación que el conocimiento previo y en especial el contexto como ejemplos importantes que deben ser manejados [5].

Los sistemas que hacen uso del NLP no tienen en cuenta muchas de estas variables, ya que se basan en la premisa de procesar un texto casi estático, esto en particular para los chatbots es un problema ya que en un sistema de conversación basado en la lingüística, los usuarios esperan que las preguntas con el mismo significado reciban la misma respuesta, lo que hemos llamado previamente frases sinónimas, lo cual en la mayoría de casos por la poca capacidad de interpretación del chatbot no se consigue [12].

Estas limitantes en chatbots que no usan NLP y/o IA dan como resultado el no reconocer correctamente preguntas similares formuladas de diferentes maneras, incluso dentro de la misma conversación, perdiendo el interés del usuario y por tanto el valor que el chatbot da a la compañía [8].

2.3.8. Soporte multilenguaje y/o multiplataforma

Los chatbots están fuertemente atados al idioma al que este es implementado debido a su propia naturaleza idiomática ya que diferentes idiomas requieren diferentes desarrollos. De igual forma los chatbots requieren una interfaz de entrada de texto, además que también se requieren implementaciones únicas por cada plataforma soportada [15].

Esta puede ser una limitante fuerte para compañías que requieren un desarrollo más global, ya que la mayoría de las tecnologías de desarrollo de chatbot requieren un gran esfuerzo y, a menudo, reconstrucciones completas para cada nuevo idioma y canal que necesitan ser soportados, lo que lleva a múltiples soluciones dispares, todas torpemente coexistentes [23].

2.3.9. Regulación en el manejo de la información

La información es el activo principal de cualquier organización que haga uso de la IA como parte de su catálogo de productos, los chatbots basados en IA requieren de grandes cantidades de información para dar desempeños adecuados, esto sin embargo también va de la mano a la forma en la que la información es recolectada para su posterior uso.

Para las organizaciones, el desafío no es solo almacenar los datos, sino también recuperar la información para exportarla o eliminarla de forma segura y auditable. Muchos de los sistemas de almacenamiento han funcionado sin muchos inconvenientes al recolectar información de los usuarios, esto sin embargo se convierte en un reto debido a las regulaciones en el manejo de la información del usuario [12]. Es probable que la regulación aumente en muchos países en el futuro.

Además, muchas tecnologías de chatbot restringen el acceso a los datos de conversación generados, lo que significa que las empresas pierden uno de los beneficios clave de implementar un chatbot. Sin estos datos, las empresas están efectivamente ciegas a sus clientes.

2.3.10. Introducción a las Arquitecturas Serverless [50]

Serverless es un término que se traduce cómo "sin servidor", lo cual suele ser un concepto confuso dado que en la práctica es necesario que haya entornos donde se ejecutan las aplicaciones. Sin embargo, gracias a Serverless, el desarrollador puede contratar servicios de infraestructura donde no debe encargarse del mantenimiento de estos entornos o incluso sus requerimientos.

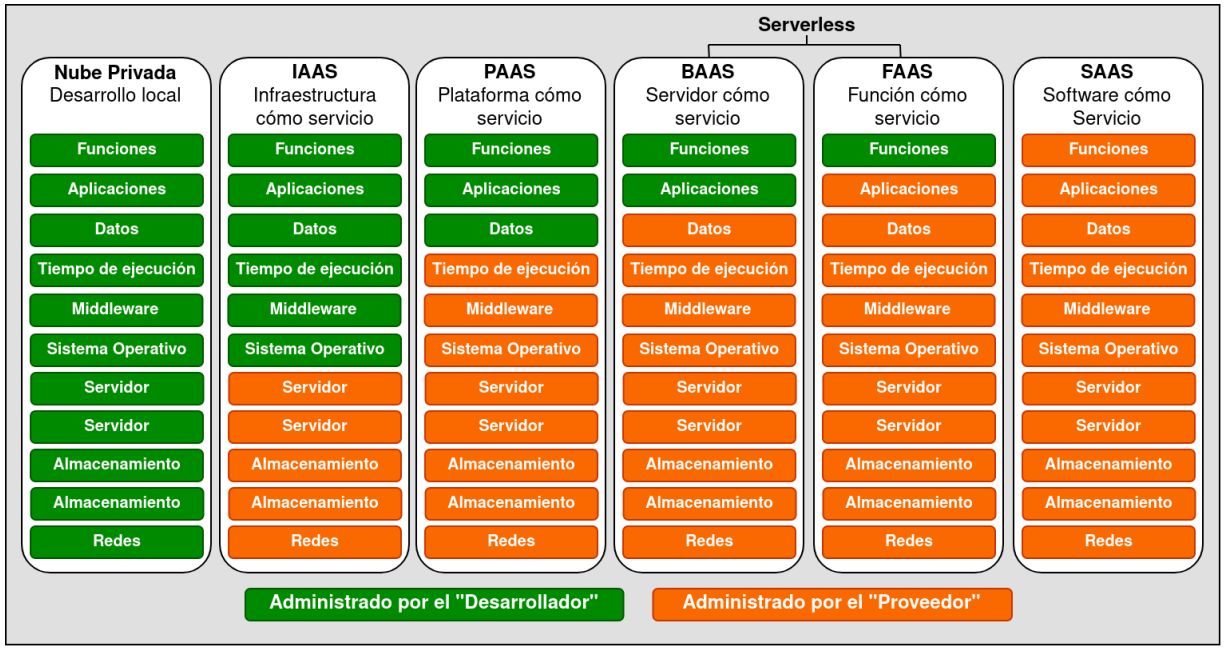

Figura 9: Comparativa de modelos de comparación en la nube

Más en detalle, Serverless cómo concepto se refiere al uso de dos modelos de computación en la nube a la vez, los cuales son "Backend as a Service"[6] (BAAS) y "Function as a Service"[7] (FAAS). El BAAS existe hace más de 15 años desde la aparición de Amazon S3, el cual es un servicio que ofrece almacenamiento en la nube. Cómo concepto el BAAS se refiere a servicios de componentes genéricos que se pueden contratar por los desarrolladores para ser conectados a sus aplicaciones de forma transparente mediante APIs, de manera que puedan ser utilizados para construir otros servicios más específicos, dada la necesidad de no invertir esfuerzos en componentes que ya existen, y tampoco tener que mantenerlos.

Algunos ejemplos de servicios BAAS son:

- Servicios de almacenamiento: Amazon S3, Azure Blob Storage, Google Cloud Storage.
- Servicios de autenticación: Auth0, Amazon Cognito, Firebase Authentication.
- Servicios de bases de datos: Amazon RDS, Amazon DynamoDB.
- Servicios de colas: Amazon SQS.

[6] Backend as a Service (BAAS): Se traduce cómo Servidor cómo Servicio, pero se seguirá referenciando en inglés.
[7] Function as a Service (FAAS): Se traduce cómo Función cómo Servicio, pero se seguirá referenciando en inglés.

- Servicios de respaldo: AWS Backup.
- Servicios de búsqueda en tiempo real: Algolia, ElasticSearch, Searchify.
- Servicios de notificación: Amazon SNS, Firebase Cloud Messaging.

Por otro lado tenemos FAAS, el cual es el responsable de volver popular el término Serverless, y nace con la aparición de AWS Lambda en el 2014 y experimentó una adopción significativa a mediados del 2016; este modelo ofrece una nueva forma de diseñar y ejecutar aplicaciones, donde el desarrollador solo se encarga de implementar código de funciones específicas y desplegarlas, su administración de recursos, ejecución y mantenimiento se delega totalmente al proveedor del servicio FAAS, por lo cual es ideal para desarrollo de servicios de scripting que ejecuten tareas pequeñas.

Algunos ejemplos de servicios FAAS son:

- AWS Lambda.
- Google Functions.
- Azure Functions.
- OpenWhisk.

2.3.10.1. Arquitecturas Tradicionales [50]

Comúnmente en una arquitectura tradicional se requiere contratar una instancia de un servidor, la cual se debe aprovisionar con un sistema operativo y herramientas que permitan administrar recursos cómo el uso de memoria, procesamiento y almacenamiento, también se deben implementar mecanismos de seguridad; además de los frameworks, librerías y los requerimientos de la aplicación que se desee desplegar. Todo esto conlleva a tener un mantenimiento y ciclos de actualización periódica, labor que comúnmente se le conoce cómo DevOps, término que más allá de referirse a un rol o área de desarrollo, se utiliza para describir un conjunto de estrategias y buenas prácticas ligadas a la cultura de una empresa para mantener el ciclo de vida del software [52].

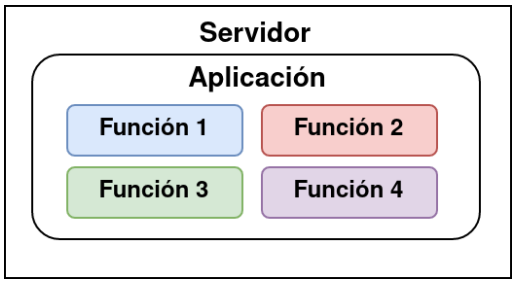

Figura 10: Ejemplo de aplicación monolito desplegada en instancia de servidor.

El servidor posee aplicaciones que están conformadas por un conjunto de funciones que resuelven problemas específicos, donde el ciclo de vida de cada función está atado al de la aplicación, por lo tanto, si se tuviera una aplicación conformada por 4 funciones cómo en la figura anterior, un cambio sobre la función 2, implicaría desplegar la aplicación entera.

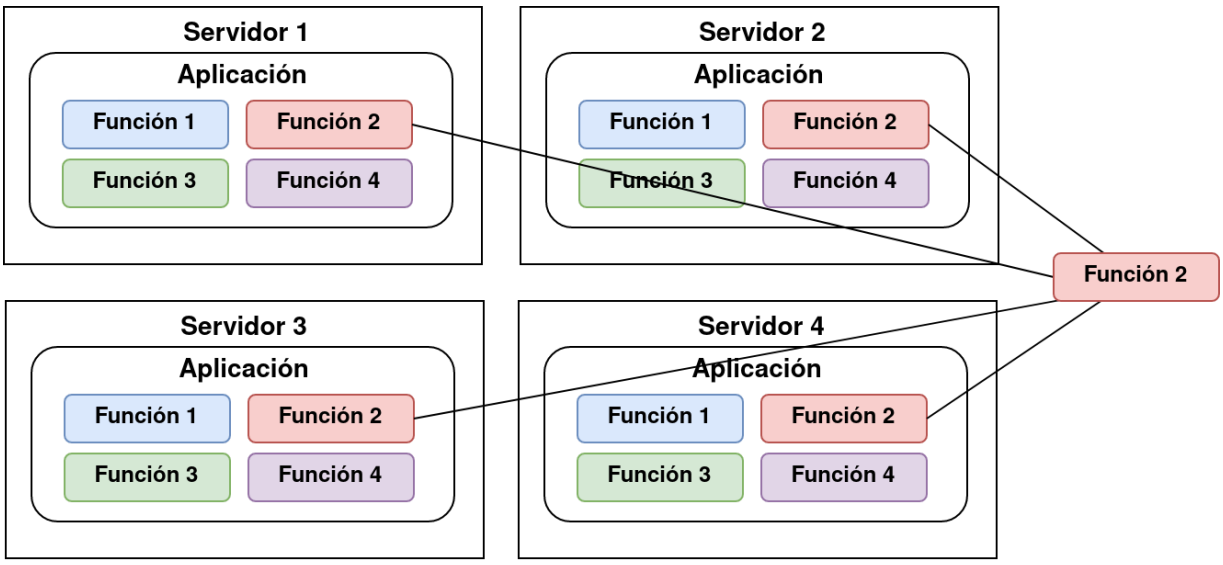

Figura 11: Ejemplo de aplicación escalada horizontalmente a 4 réplicas.

Posteriormente las aplicaciones requieren escalar tanto vertical cómo horizontalmente, y es en esto último donde el ciclo de actualización de la aplicación y estrategias DevOps se vuelven más complejas de gestionar. Debido a que, si se tienen múltiples instancias de la aplicación, un cambio sobre la función 2 implica actualizar todas las réplicas de la aplicación en paralelo. Sumado a las políticas de balanceo de carga y enrutamiento que hay que implementar para mantener balanceado del uso de recursos y el flujo de peticiones de manera coordinada sobre los múltiples nodos del sistema.

2.3.10.2. Arquitecturas Serverless [50]

El ciclo de vida del software y las estrategias DevOps cambian y se simplifican de forma destacable en una arquitectura serverless. Siguiendo el ejemplo de la figura 12, en una plataforma FAAS se despliegan sólo el código de las funciones y no la aplicación entera, por lo cual cada una puede evolucionar por separado de manera independiente, y el desarrollador solo se debe preocupar por escribir el código que satisfaga los objetivos de negocio. Toda la labor de aprovisionamiento y mantenimiento de los servidores es asumida por la plataforma FAAS.

Una característica muy importante de FAAS es que todas estas funciones individuales no se están ejecutando todo el tiempo, sino que se encuentran en reposo y evitando el consumo de recursos hasta que sean invocadas, ya que se pueden parametrizar para ser ejecutadas por algún evento específico, cómo una inserción en una base de datos, la creación de un usuario, una actualización de estado en un registro o una simple petición HTTP.

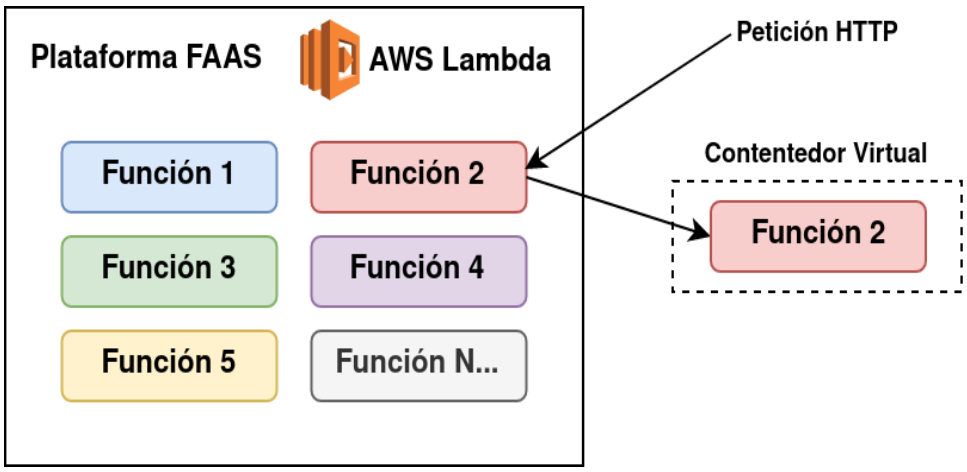

Figura 12: Plataforma FAAS. Invocando función dentro de contenedor virtual.

Una vez la función es invocada, se aprovisiona y ejecuta dentro de un contenedor virtual, el cual es una máquina virtual muy liviana que consume muy pocos recursos y se comporta cómo un proceso más del sistema operativo (implementaciones más modernas hacen uso de contenedores), al cual se le puede parametrizar fácilmente la cantidad de recursos en memoria, CPU, almacenamiento y tiempo de ejecución que puede consumir. Esta característica permite que ejecutar múltiples instancias del contenedor virtual sea muy sencillo y óptimo, por lo cual, en caso de mucha demanda, la escalabilidad vertical y horizontal está garantizada de forma transparente para el desarrollador.

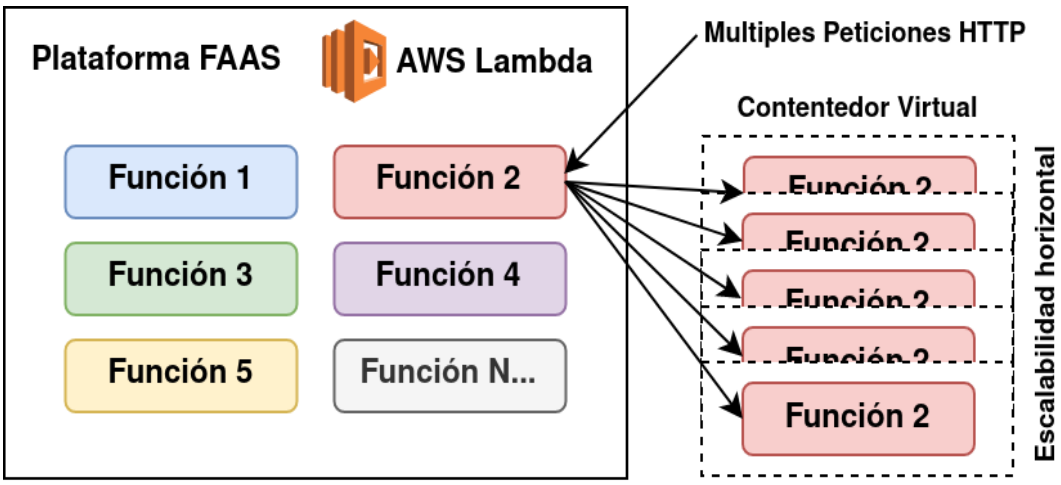

Figura 13:Plataforma FAAS escalando horizontalmente llamado a la función 2.

En la figura 13 se observa cómo llegan múltiples eventos que disparan una misma función, en este caso la plataforma FAAS escala horizontalmente creando múltiples réplicas del contenedor virtual con copias de la función 2, lo cual permitirá procesar todos los eventos en paralelo. Cabe aclarar que a medida que un contenedor termina su ejecución permanece encendido durante un lapso de tiempo que le permite ser reutilizado por más eventos y hacer que la siguiente ejecución sea más rápida, esto es especialmente útil ya que la inicialización del contenedor demora un par de segundos mientras transición de reposo a ejecución; y cómo es de esperar, después de un tiempo de inactividad el contenedor procederá a volver a su estado de reposo evitando costes.

2.3.10.3. Ventajas y desventajas de usar Serverless [50]

No existe solución perfecta que sirva para resolver cualquier tipo de problema. Como se observa en la figura 13, existen múltiples modelos de computación en la nube que entregan al desarrollador el control de diferentes capas de la infraestructura, de las cuales en el caso de serverless se puede observar que se delegan muy pocas capas al desarrollador, lo cual trae consigo una serie de ventajas al permitir simplificar mucha labor de aprovisionamiento y mantenimiento, pero también hay desventajas como las que se compartirán más adelante:

Ventajas de serverless

- **No hay que administrar la infraestructura:** No hay que preocuparse por los servidores, ni por el sistema operativo, ni los contenedores, ni las redes, entre otros recursos**.**
- **Está diseñado para escalar automáticamente:** La plataforma FAAS escala bajo demanda sin necesidad de configuraciones extras por parte del desarrollador.
- **Posee una arquitectura orientada a eventos:** Las funciones pueden ser invocadas por cualquier tipo de acción cómo una petición HTTP, un registro en una tabla, una actualización en base de datos, una notificación, entre otros.
- **No hay costos de contratación:** Desplegar las funciones en el proveedor no tiene ningún costo mientras se mantenga en reposo.
- **Se cobra bajo demanda:** Solo se cobra por el uso de recursos cómo consumo de memoria, CPU, almacenamiento y tiempos de ejecución.
- **Límites flexibles y uso de recursos variables:** A cada función se le pueden asignar una cantidad de uso de recursos limitados que pueden escalar de manera flexible con base a las necesidades de la función.

Desventajas de serverless

- **Acoplamiento con proveedores específicos:** Conocido como "Vendors Lock-In" (en inglés), el delegar toda la administración de la infraestructura a un tercero crea una dependencia que obliga al desarrollador a usar servicios de un proveedor específico y puede ser difícil diversificar.
- **Arranque en frío**: Conocido como "Cold Start" (en inglés), es un tiempo de espera que ocurre mientras que el contenedor virtual pasa de estado de reposo a ejecución. No es un tiempo destacable, pero para aplicaciones que requieran de tiempos de respuesta en tiempo real es una gran desventaja.
- **Restricciones del proveedor (en tiempo, tamaño y ejecución):** En una plataforma FAAS los recursos asignados a las funciones son muy limitados, por lo cual se debe buscar resolver solo tareas pequeñas.
- **Es difícil hacer monitoreo:** Las plataformas FAAS no cuentan con un buen sistema de seguimiento y depuración de errores.
- **Es difícil calcular los costos:** En FAAS se cobra bajo demanda, por lo cual no es fácil predecir cuál será el costo de las facturas a final de mes, ya que dependerá de la demanda variable.

2.3.10.4. Cuando usar o no usar Serverless [50]

Teniendo en cuenta las ventajas y desventajas mencionadas en la sección anterior, a continuación, se destacan los principales casos de uso para usar serverless, así como también en qué contextos no debería usarse:

Cuando usar serverless

- Tareas cortas que se ejecutan con una frecuencia fija.
- Automatización de eventos de infraestructura.
- Procesamiento de datos para hacer cálculos matemáticos o generar informes estadísticos.
- Aplicaciones web orientadas a eventos disparados por el usuario.
- Chatbots.
- Para construir Rest APIs.

Cuando no usar serverless

- Cuando se desea depender de un proveedor.
- Cuando las funciones tienen ejecuciones muy largas.
- Para ejecuciones complejas o pesadas en términos de consumo de recursos.

2.3.10.5. Frameworks de desarrollo Serverless [71, 72]

Los frameworks de serverless proveen un conjunto de herramientas que proveen los mecanismos necesarios para permitir al desarrollador abstraerse de las especificidades técnicas de las plataformas FAAS o infraestructuras en la nube, y en lugar de ello permiten un fácil despliegue y aprovisionamiento de las funciones en los diferentes proveedores. Muchos frameworks son desarrollados enfocado a un solo proveedor, pero de manera ideal los desarrolladores demandan herramientas que ofrezcan soluciones agnósticas al proveedor de infraestructura en la nube.

Los frameworks de serverless los podemos clasificar en dos tipos:

- **Frameworks de abstracción:** Abstraen las especificidades técnicas y permiten desplegar funciones en una o más plataformas FAAS.
- **Frameworks de aprovisionamiento:** Son frameworks open source que ofrecen la capacidad de crear una plataforma FAAS en infraestructuras privadas, administradas por el mismo desarrollador.

A continuación, se presenta una tabla comparativa de los frameworks de serverless más populares:

Framework	Lenguajes de Desarrollo	Plataforma de Despliegue (CI/CD)	Ejecución o tipo de invocación
Fission	NodeJs, Python, Ruby, Go, PHP, Bash.	Kubernetes en IaaS.	HTTP, Cron, MQ, CLI
Kubeless	NodeJs, Python, Ruby, PHP, Go, Ballerina.	Kubernetes en IaaS.	HTTP, Cron, MQ, Stream, CLI
IronFunctions	Go, .NET, Javascript, Java, Lambda, Python, Ruby, Rust.	Picasso	HTTP, MQ, Alarm, CLI
Sparta	Go	AWS Lambda	HTTP, MQ, Stream, CLI
Fn Project	Agnóstico	AWS Lambda	HTTP, MQ
Snafu	Python, Java	AWS Lambda, OpenWhisk, Fission, Kubeless.	HTTP, MQ, FS, Cron, CLI
Serverless	Javascript, C#, F#, Scala, Python, Java, Golang, Groovy, Kotlin, PHP & Swift.	AWS Lambda, Azure Functions, Google Functions, OpenWhisk, SpotInst, Kubeless, Fn	HTTP, Cron, MQ, CLI
Chalice	Python	AWS Lambda	HTTP, Cron, MQ, Stream, CLI
ClaudiaJs	NodeJs	AWS Lambda	HTTP, Cron, MQ, CLI
Architect	NodeJs, Python, Ruby	AWS Lambda	HTTP, Cron, MQ, CLI
Openwhisk	NodeJs, Go, Java, Scala, PHP, Python, Ruby, Swift, .Net, Rust.	Kubernetes en IaaS.	HTTP, Cron, MQ, CLI

Tabla 1: Frameworks de Serverless más populares.

2.3.10.6. Plataformas FaaS [73]

Si bien Serverless y FaaS cómo concepto nacieron gracias a AWS Lambda, hoy en día existen muchísimas alternativas. A continuación, se presentan algunas tablas comparativas de las principales características de las plataformas FaaS.

2.3.10.6.1. Tabla comparativa de plataformas FaaS según enfoque

Enfoque: Hazlo tú mismo (Permiten crear un FaaS en IaaS)					
Open-FaaS	Fn Project	Fission	Open-Whisk	Kubeless	Fly.io*

Enfoque: Configurables, administradas por el proveedor					
AWS Lambda	Google Functions	Azure Functions	IBM Cloud Functions	Alibaba Functions	Oracle Functions
Enfoque: Muy fáciles de usar					
Vercel Functions		Netlify Functions		Fly.io*	
Enfoque: Computación de borde (Edge Computing)					
Cloudflare Workers		Edge Engine	Fly.io	Lambda @Edge	Vercel Functions + Edge Cache*

Tabla 2: Comparativa de plataformas FaaS según el enfoque [73].

2.3.10.6.2. Tabla comparativa de límites de las plataformas FaaS

	Memoria (MB)		Tiempo de ejecución		Datos de Contexto[8] (MB)	
	Por defecto	Máximo	Por defecto	Máximo	Peticiones	Respuestas
Vercel Functions	1024	3008	10s-900s*2	10s-900s	5	5
Netlify Functions	1024	1024*3	10s*3	10s*3	6	6
IBM Cloud Functions	256	2048	1m	10m	5	5
Google Cloud Functions	128	2048	N/A	540s	10	10
Oracle Functions	128	1024	30s	120s	6	6
Azure Functions	N/A	1536	5m	10m	No limit	No limit
AWS Lambda	128	3008	3s	15m	6*1	6*1
Alibaba Functions	?	?	?	600s	6*1	6*1
Cloudflare Workers	128	128	No limit 10ms CPU*6	No limit 10ms CPU	?	?
Edge Engine	128	128	15s 5ms CPU*6	15s 5ms CPU	?	?
AWS Lambda@Edge	128	128	30s	5s	50 Mb	40 KB
Fly.io	N/A	N/A	No limit	No limit	No limit	No limit

Tabla 3: Comparativa de límites de plataformas FaaS [73].

[8] Conocido también como “payload” en inglés.

2.3.10.6.3. Tablas comparativas de precios de las plataformas FaaS

	Capa Gratuita		
	Peticiones	**Uso de GB por segundo**	**Horas / Meses**
Netlify Functions	125K	N/A	100
Vercel Functions	N/A	N/A	20
IBM Cloud Functions	N/A	400K	N/A
Oracle Functions	2M	400K	N/A
Alibaba Functions	1M	400K	N/A
Google Cloud Functions	2M	400K	N/A
AWS Lambda	1M	400K	N/A
Azure Functions	1M	400K	N/A
Cloudflare Workers	N/A	N/A	N/A
Edge Engine	N/A	N/A	N/A
Fly.io	$10 crédito mensual en uso del servicio		
AWS Lambda@Edge	N/A	N/A	N/A

Tabla 4: Comparativa de límites de la capa gratuita de plataformas FaaS [73].

	Precios en producción		
	Peticiones	**Uso de GB por segundo**	**Horas / Meses**
Netlify Functions	$0.000038	N/A	1h por 1000 peticiones
Vercel Functions	N/A	N/A	$0.2 / h
IBM Cloud Functions	N/A	$0.000017	N/A
Oracle Functions	$0.0000002	$0.00001417	N/A
Alibaba Functions	$0.0000002	$0.00001668	N/A
Google Cloud Functions	$0.0000004	$0.0000025	N/A
AWS Lambda	$0.0000002	$0.00001667	N/A
Azure Functions	$0.0000002	$0.000016	N/A
Cloudflare Workers	$0.0000005	N/A	N/A
Edge Engine	$0.0000006	N/A	N/A
Fly.io	$0.000001015 / sec	N/A	N/A
AWS Lambda@Edge	$0.0000006	$0.00005001	N/A

Tabla 5: Comparativa de límites de la capa gratuita de plataformas FaaS [73].

3. CAPÍTULO III: ESTADO DEL ARTE

3.1. Arquitecturas de Software

Debido a la naturaleza macro que puede significar el diseño de una arquitectura con descripciones detalladas del manejo de datos y la complejidad de interoperabilidad de las diferentes partes, se plantea realizar una recopilación exhaustiva de teorías, métodos y metodologías para el análisis, diseño y evaluación de arquitecturas de software.

Es fundamental entender cómo las diferentes arquitecturas y métodos de evaluación se relacionan para dar como resultado a través del análisis del estado del arte una aproximación de que debe ser considerado para diseñar una arquitectura. Es importante aclarar que en partes del presente capítulo se darán ejemplos usando un sistema cliente-servidor, ya que es el más similar al requerido por un chatbot, para dar una descripción general; es por ello que este capítulo no busca explicar una arquitectura cliente-servidor completa debido a que no hace parte del objetivo de este:

3.1.1. Diseño y elementos de la Arquitectura de software

Se entiende como arquitectura de software al modelo que describe un diseño de más alto nivel de las estructuras de un sistema de software en términos de componentes computacionales, las relaciones entre los componentes y las restricciones para ensamblar los elementos del software. Esta ha sido la visión que se tiene de arquitectura de software y como esta es ejecutada en muchos campos, pero en la disciplina del desarrollo de software y su impacto en la industria ha evolucionado al punto de hablar de arquitecturas de software que deben ser no solamente diseñadas teniendo en cuenta el software en sí o más importante visualizando un sistema, sino más allá.

La arquitectura se aplica cada vez más a cosas que normalmente no se consideran sistemas, incluidas entidades con estructura y comportamiento similares a los de un sistema, como empresas, servicios, datos, funciones comerciales, áreas de misión, líneas de productos, familias de sistemas, elementos de software, etc. [31].

Diseñar una arquitectura implica entenderla en términos de que la compone, que es fundamental para que esta sea considerada como una buena arquitectura y que a su vez pueda ser eficaz en el cumplimiento de un mínimo de calidad posible. Para ello una arquitectura debe tener una serie de cualidades o características [32]:

- **Robustez:** la robustez en una arquitectura de software se refiere a qué tan sólida es esta, se busca a través de la robustez que una arquitectura no sea vulnerable a cambios menores en los sistemas de negocios, información, aplicaciones y tecnología.
- **Factibilidad:** Una arquitectura debe ser factible dentro de su contexto actual, no se puede desarrollar una arquitectura que no pueda ser viable en el momento de ser concebida, ya que

esto acarrea costos de implementación, es mejor identificar los requisitos lo antes posible para no generar expectativas infladas a las partes interesadas en el trabajo de arquitectura.

- **Utilidad:** la utilidad en una arquitectura es la forma de medir cual es el valor que esta aporta y qué resultados prácticos se consiguen con esta. Las arquitecturas que son elegantes pero que no brindan un valor demostrable y medible a las partes interesadas o las partes que las requirieron finalmente no tendrán éxito.
- **Durabilidad:** Las arquitecturas deben demostrar que son duraderas con el paso del tiempo y resistentes a los cambios en los entornos comerciales y técnicos que puedan ocurrir durante la **vida útil** de las arquitecturas. Esto implica que deben, en la medida de lo posible, adelantarse a las condiciones y entornos futuros. Siendo así fundamental tener presente cual es el objetivo de vida útil de una arquitectura.
- **Flexibilidad:** Se refiere a la capacidad de adaptación a las condiciones cambiantes y también proporcionar suficiente orientación para los equipos de implementación que tienen el conocimiento de su disciplina para tomar decisiones importantes y necesarias sobre los problemas técnicos y las oportunidades. Las arquitecturas que se crean con demasiado detalle a menudo generan diseños e implementaciones frágiles e inflexibles que dan como resultado sistemas que no se pueden adaptar a circunstancias y entornos cambiantes.
- **Verificable:** Una arquitectura debe ser fácilmente verificable, esto se refiere a que se pueda demostrar de forma simple que la arquitectura funcionará según lo diseñado y que no habrá efectos secundarios.
- **Estilización (Elegant):** Una arquitectura debe ser entendible y fácil de implementar, es una arquitectura bien diseñada tenderá a ser elegante y tendrá una simplicidad de forma que será obvia para aquellos que se toman el tiempo para estudiarla.
- **Trazable:** Una arquitectura de software en términos generales describe los procesos y medios de ejecución de un software para realizar los diferentes requisitos de negocios, es así como una arquitectura debe permitir entender fácilmente cómo trazar un proceso desde su concepción hasta su ejecución final.

La arquitectura de software de forma esencial como ya se explicó se basa en describir entidades y cómo estas se relacionan, considerando funciones o elementos totalmente aislados, o un sistema completo, así una arquitectura de software puede ser definida en término de los siguientes elementos [32]:

- **Componentes:** Los componentes son los elementos computacionales que colectivamente constituyen una arquitectura. Una arquitectura de software generalmente se descompone en subsistemas, que a su vez pueden descomponerse en módulos. También es posible una mayor descomposición. (Por ejemplo, en un diseño orientado a objetos, los módulos pueden descomponerse en clases). En el caso del proyecto actual como se habla de un sistema basado en web, se puede describir los componentes en términos de una arquitectura cliente-servidor, un ejemplo de esto incluyen:
    - **Clientes:** Cualquier elemento que se encuentre ajeno al sistema, elemento o función que consuma recursos de un servicio (los propios servicios pueden ser clientes de otros

servicios), pueden ser navegadores, aplicaciones nativas, sistemas embebidos, o en el caso de un bot un software de chat.
    - **Servicios:** Componentes encargados de la ejecución de recursos solicitados por clientes u otros servicios, ejemplo una base de datos, un servidor de archivos, entre otros.
    - **Módulos de seguridad:** Componente de filtrado de recursos para permitir accesos.
    - **Sistemas de autenticación:** Sistemas que pueden ser propios o terceros que permiten a los servicios identificar qué cliente está solicitando algo.

- **Relaciones:** Las relaciones son las conexiones lógicas entre componentes arquitectónicos, se usan para explicar la forma en la que estos se comunican y cómo interactúan, en este caso como es hablamos de interacción esta debe ser descrita bajo un lenguaje común a las partes del software, este lenguaje se conoce normalmente como protocolo y da las reglas sobre cómo los componentes deben comunicarse e interactuar. Las relaciones también son fundamentales para visualizar la dependencia de los componentes de una arquitectura dependen esto es esencial para la mantenibilidad del software ya que reemplazar, actualizar o remover componentes de la arquitectura tiene un impacto directo en como esta funcionara, esto es debido a la dependencia de los sistemas y cómo estos pueden estar fuerte o débilmente acoplados a los componentes, ejemplos de relaciones de componentes abstractos incluyen dependencia, agregación y composición. Ejemplos de relaciones de componentes concretas incluyen protocolos usados en cliente-servidor tales como:
    - TCP/IP
    - HTTP (Hypertext Transport Protocol)
    - SOAP (Simple Object Access Protocol)
- **Limitaciones (constraints):** Las restricciones proporcionan condiciones y restricciones para las relaciones de los componentes. Conectan la arquitectura a los requisitos del sistema. Los ejemplos de restricciones incluyen restricciones en los tipos de parámetros para protocolos de comunicación y requisitos de alta disponibilidad para tolerancia a fallas.

En la siguiente imagen se puede apreciar un ejemplo visual de una arquitectura simple cliente-servidor como complemento a los puntos anteriormente mencionados:

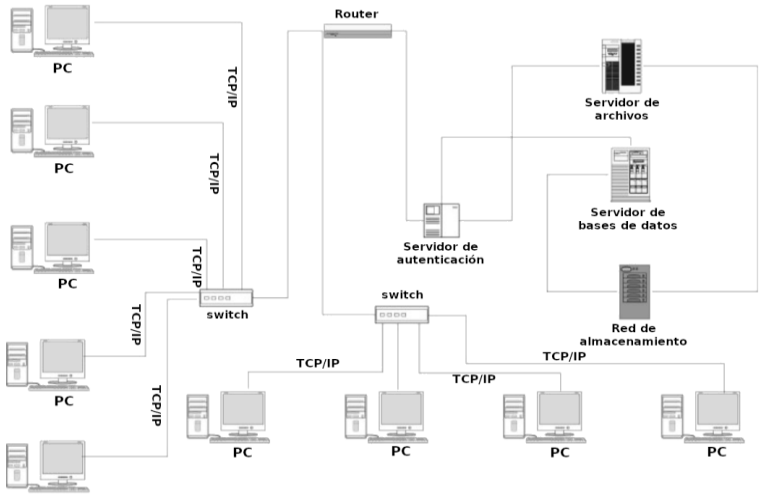

Figura 14: Ejemplo arquitectura cliente-servidor simple.

La arquitectura como se ve da un punto de referencia vital al software de como este debe ser construido. Sin embargo, la arquitectura de software no solo se basa en el desarrollo y mantenibilidad de un software sino también en términos de negocio, esto último siendo incluso más importante en ocasiones que el propio software, de hecho, conceptos como stakeholders (interesados) son fundamentales para conseguir implementar una arquitectura exitosamente. Partiendo de esto, se puede descomponer en términos de negocio y gobernanza según el estándar internacional de la IEEE [34] en lo siguiente:

- Tareas
- Procesos
- Actividades
- Etapa (stage)
- Fase
- Ciclo de vida
- **Stakeholder**
- Interés (concern)
- **Vista**
- Viewpoint
- Registro

Estos conceptos resumidos por la IEEE son de importancia en la implementación de una arquitectura y su ciclo de vida, por lo cual serán ampliados en los siguientes capítulos para dar un contexto más amplio concerniente a las bases de una buena arquitectura, estos conceptos son transversales a los diferentes tipos de arquitectura y/o métodos por lo que son conceptos claves para las explicaciones futuras.

3.1.2. Etapas del desarrollo de una arquitectura de software

Cuando se habla de arquitectura de software hay dos conceptos en las etapas de desarrollo para tener en cuenta: Arquitectura de Software y Diseño de Software. En Arquitectura, las decisiones no funcionales se emiten y separan por los requisitos funcionales. En Diseño, se cumplen los requisitos funcionales.

- **Arquitectura de software**

La arquitectura sirve como modelo para un sistema. Proporciona una abstracción para gestionar la complejidad del sistema y establecer un mecanismo de comunicación y coordinación entre los componentes.

Define una solución estructurada para cumplir con todos los requisitos técnicos y operativos, al tiempo que optimiza los atributos de calidad comunes como el rendimiento y la seguridad.

Además, implica un conjunto de decisiones importantes sobre la organización relacionadas con el desarrollo de software y cada una de estas decisiones puede tener un impacto considerable en

la calidad, la capacidad de mantenimiento, el rendimiento y el éxito general del producto final. Estas decisiones comprenden:

- Selección de elementos estructurales y sus interfaces que componen el sistema.
- Comportamiento según lo especificado en colaboraciones entre esos elementos.
- Composición de elementos estructurales y de comportamiento en un gran subsistema.
- Las decisiones arquitectónicas se alinean con los objetivos comerciales.
- Los estilos arquitectónicos guían la organización.

- **Diseño de software**

El diseño de software proporciona un plan de diseño que describe los elementos de un sistema, cómo encajan y cómo funcionan juntos para cumplir con los requisitos del sistema. Los objetivos de tener un plan de diseño son los siguientes:

- Negociar los requisitos del sistema y establecer expectativas con los clientes, el personal de marketing y de gestión.
- Actuar como modelo durante el proceso de desarrollo.
- Guíe las tareas de implementación, incluido el diseño detallado, la codificación, la integración y las pruebas.

Conseguir desarrollar una arquitectura es un esfuerzo complejo, que implica mucha experiencia y conocimiento técnico, pero también requiere entendimiento de cómo deben ser realizadas cada una de las etapas, esto para dar prioridad a cada uno de los sujetos/entidades que están involucradas en el proceso de implementación de la arquitectura, la IEEE en [34] divide el ciclo de desarrollo y gobernanza de una arquitectura de la siguiente forma:

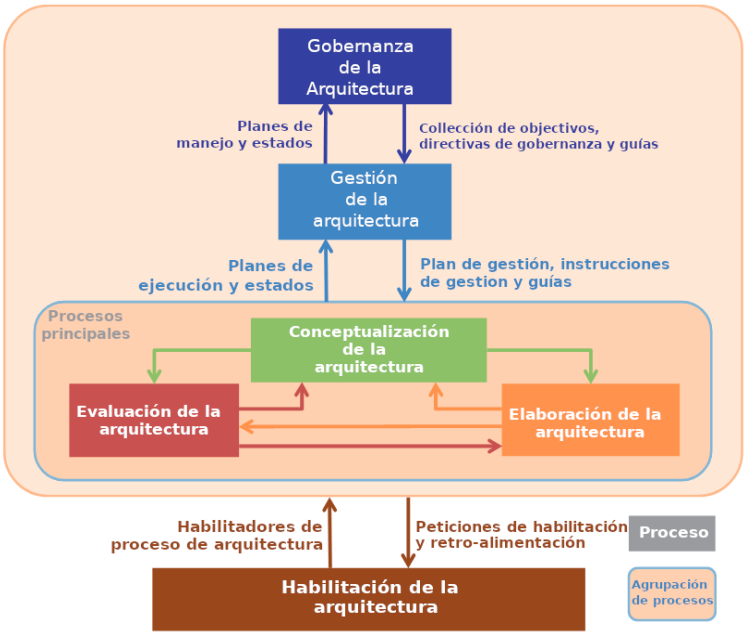

Figura 15: Procesos de arquitectura y sus interacciones [34]

El presente proyecto como se ha aclarado previamente busca desarrollar una arquitectura basada en serverless para la implementación de chatbots con servicios agnósticos, la naturaleza del proyecto y objetivos están enmarcados en lo que en el estándar se conoce como *core processes* y será en esta sección donde se pondrá mayor atención, sin dejar la gobernanza totalmente excluida, pero siendo claros en su papel secundario en este proyecto. En el *core process* del desarrollo de una arquitectura se explican las etapas y sus relaciones, estas son: Conceptualización/Análisis, Elaboración/Desarrollo y Evaluación para ampliar más la información sobre estos conceptos y cómo estos se relacionan se tiene la siguiente figura.

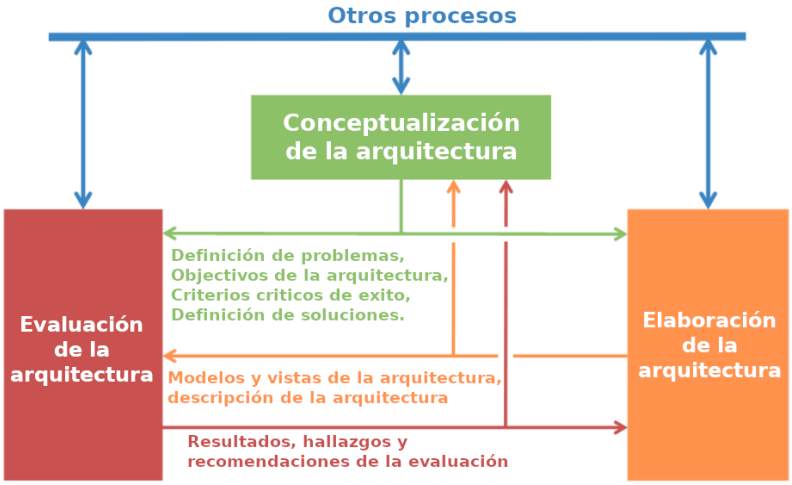

Figura 16: Relación de las diferentes etapas de una arquitectura [34]

El estándar internacional describe estas 3 etapas de diseño de una arquitectura como *core processes*, estos procesos son iterativos e interrelaciónales, lo cual significa que la ejecución de cada uno de estos está relacionada directamente con la ejecución de otro, se puede ver de hecho como la evaluación de la arquitectura es una etapa indispensable y transversal al proceso de análisis y elaboración.

La evaluación debe ser considerada así para permitir encontrar errores tempranos en el diseño y cómo estos pueden ser mejorados. Estas 3 etapas deben ser ampliadas ya que muchos métodos de desarrollo y evaluación de arquitecturas se basan en la interacción directa o indirecta de estas etapas.

3.1.2.1. Conceptualización/Análisis

Al igual que la construcción de un edificio o el desarrollo de un nuevo producto, el software y principalmente la arquitectura que lo acompaña requieren ser planificados de forma precisa. Es claro que para planear de forma adecuada lo que se quiere desarrollar es fundamental contar con una etapa de recolección de requisitos y/o análisis, en ese sentido la conceptualización en la arquitectura de software específica los objetivos de la arquitectura, los requisitos y las medidas de calidad que se pueden utilizar en la evaluación de su valor [34].

Sin embargo, no todos los requisitos son iguales, en "*Software Architecture in Practice*" [35] se describe cómo en esta etapa es destacable identificar algunos requisitos que tienen un efecto mucho más profundo en la arquitectura que otros. Estos se conocen como requisitos arquitectónicamente significativos (a partir de ahora **ASR,** *architecturally significant requirement*), los cuales tienen un impacto directo a la arquitectura, es decir, la arquitectura podría ser drásticamente diferente en ausencia de tales requisitos, estos se pueden identificar rápidamente leyendo las historias de usuario u observando el documento de requisitos del sistema ya que estos se generan a partir de la identificación y definición del problema / oportunidad que se produce en este proceso [35].

Los ASRs o en general los requisitos de una arquitectura aportan valor a la misma y se define en términos de la medida en que se abordan las preocupaciones de las partes interesadas (stakeholders). Los conceptos de arquitectura se generan teniendo en cuenta el valor y luego se evalúan utilizando estas medidas de calidad. Los valores de calidad son tan importantes que un ASRs puede tomar la forma de requisitos de atributos de calidad, tales como:

- El rendimiento
- La seguridad
- Modificabilidad
- Disponibilidad
- Usabilidad

En el caso del presente proyecto se ha hablado constantemente que la arquitectura debe proporcionar al sistema la capacidad de usar serverless y servicios cognitivos que se traduce en los atributos de calidad extensibilidad, escalabilidad y mantenibilidad; los cuales han sido mencionados previamente en el objetivo general.

Los ASRs tienen gran incidencia en la arquitectura final, ya que cuanto más difícil e importante sea el requisito de garantía de calidad, es más probable que afecte significativamente a la arquitectura [35], las siguientes definiciones son necesarias antes de extender aún más la información, ya que se usan frecuentemente:

3.1.2.1.1. Atributos y escenarios de calidad

Los "Quality Attribute" como requerimientos no funcionales de un sistema (RNF), son un rasgo o característica que afectan la calidad de este, e indican el grado en el cual el sistema debe exhibir varias propiedades. Los atributos de calidad son derivados de los objetivos de negocio. Estos criterios pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos, ya que éstos corresponden a los requisitos funcionales (RF). Estos requisitos son fundamentales ya que como se ha explicado previamente estos pueden afectar significativamente a la arquitectura debido a que al derivarse de los objetivos principales esto impactan directamente en el diseño e implementación posterior de la misma.

Es necesario expresar los atributos de calidad en términos de consideraciones o eventos específicos sobre un sistema particular, a esto se le conoce como escenarios de calidad los cuales constan de seis partes:

1. **Fuente de Estímulo:** Una entidad (una persona, un computador, o alguna otra entidad) genera un estímulo.
2. **Estímulo:** Un agente, factor o acción que a ser considerada al iniciar una reacción sobre el sistema.
3. **Ambiente (Entorno):** Representa las condiciones o estados bajo el cual el estímulo sucede.
4. **Artefacto:** Componente que es estimulado puede ser todo el sistema o ciertas partes de este.
5. **Respuesta:** Reacción al estímulo recibido.
6. **Respuesta de Medición:** Cuando la respuesta ocurre, debe ser medible a través de algún criterio de medida.

3.1.2.1.2. Stakeholder (partes de interés o involucrados) [36]

Un interesado es una persona u organización que tiene derechos, acciones, reclamos o intereses relacionados con el sistema o sus propiedades que satisfacen sus necesidades y expectativas.

En términos más simples, los intereses de las partes interesadas tienen cierta influencia en el proyecto, por lo que siempre se debe tener en cuenta su opinión. Si no hace esto y pasa por alto a una de las partes interesadas clave, puede arruinar todo el proyecto y será mucho más costoso que simplemente permitir un error de desarrollo en el proyecto. Las partes interesadas brindan oportunidades y limitaciones para el sistema y son la fuente de requisitos, los stakeholders se pueden clasificar en las siguientes categorías:

- **Equipo de trabajo:** Quienes implementarán la arquitectura, estas son las personas que tienen los conocimientos técnicos para desarrollarla.
- **Equipo directivo:** Este es el equipo que dirige y sabe cómo tomar resultados productivos de la implementación.
- **Empresas de terceros:** Aquellos que prestan servicios dando soporte o asesoramiento.
- **Clientes:** Partes interesadas en usar el resultado de implementar nuestra arquitectura.
- **Usuarios finales:** A diferencia de los clientes que están interesados en usar nuestra implementación como un producto interno, los usuarios finales son aquellos que realmente usan de forma cotidiana el producto resultante.

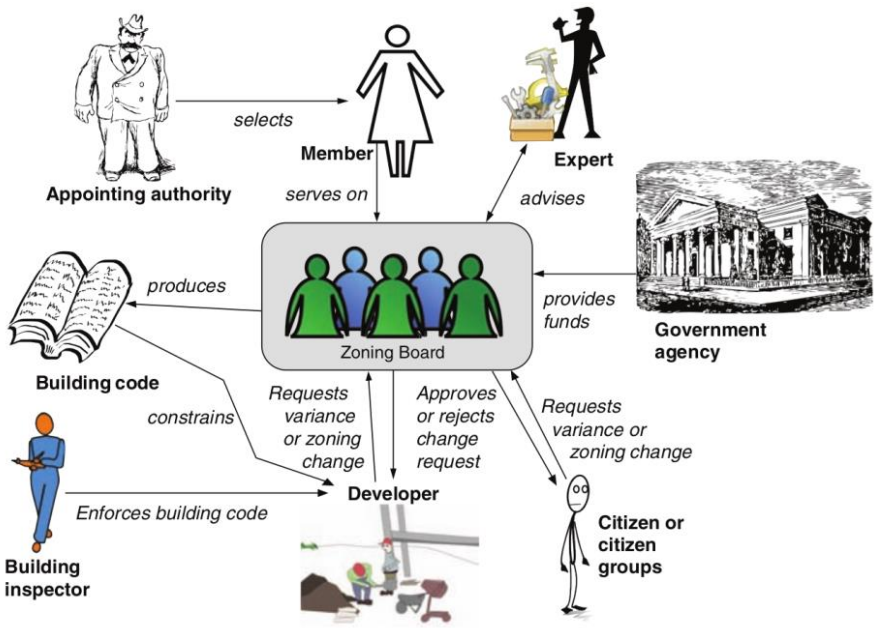

Figura 17: Relación de stakeholders [35].

Los stakeholders pueden involucrar más tipos de entidades, un ejemplo de esto son entes regulatorios o accionistas, así que siempre deben de tenerse en cuenta la identificación de cualquier individuo u organización que esté relacionada de forma directa o indirecta, ya que la acción de cualquiera de estos puede tener efectos determinantes en la arquitectura del sistema.

El estándar internacional [34] describe a la conceptualización como un proceso que permite caracterizar el espacio del problema, sintetizar soluciones potenciales, formular arquitecturas candidatas y expresar estas arquitecturas en una forma que sea adecuada para los usos previstos, es así como la identificación de los diferentes requisitos del sistema son parte esencial de esta etapa. Se puede notar que se hace énfasis en la generación de arquitecturas candidatas, esto es debido a que la conceptualización y elaboración de una arquitectura en ocasiones se maneja como una sola etapa ya que no significa que los resultados obtenidos en esta etapa estén necesariamente en el nivel "conceptual" [33].

En muchos métodos la etapa de conceptualización es obviada y se habla completamente de los procesos de recolección de requerimientos, análisis de atributos de calidad o incluso entrevistas a stakeholders dentro de la etapa de elaboración [31]; se considera sin embargo valioso en este proyecto discernir estas etapas ya que dan un punto de vista más claro del proceso de una arquitectura, además de estar descrito en el estándar internacional.

Se muestra particularmente que esta etapa puede incluir dentro de sus actividades el desarrollo de una arquitectura "lógica" o una arquitectura "física", dependiendo de la naturaleza de la situación [34], pero algunas de estas primeras descripciones de la arquitectura son poco más que bocetos. Una forma más completa de describir la arquitectura se desarrollaría durante la elaboración de esta y posteriormente la conceptualización puede utilizar los resultados de la elaboración de la arquitectura, cuando sea apropiado.

La conceptualización de la arquitectura solo necesita describir la arquitectura al nivel de especificidad y granularidad que sea adecuado para sus usuarios previstos, lo que en muchos casos no requiere una elaboración significativa. En este punto como resultado se puede utilizar la arquitectura generada para la elaboración de vistas, modelos y descripciones, esto a menudo puede ocurrir más adelante en el ciclo de vida de la arquitectura, después de evaluar si el esfuerzo adicional de elaboración vale la pena.

La elaboración a menudo se puede aplazar hasta después de que se hayan examinado varias alternativas de arquitectura para determinar su idoneidad, y se seleccionan hasta una o varias alternativas para un examen más detallado y un uso final en el esfuerzo de ingeniería, es por ello por lo que estas tres etapas son interrelacionadas ya que evaluar los resultados de la conceptualización de la arquitectura es fundamental para su posterior elaboración [34].

3.1.2.2. Elaboración

La elaboración como lo describe el estándar internacional [34] es un paso que surge al completar elementos del diseño de la arquitectura y cuando estos requieren una mayor descripción. Los objetivos, conceptos y propiedades de una arquitectura pueden ser expresados de una manera suficientemente completa y correcta haciendo uso como base el modelado, la delimitación y la descomposición de cada uno de estos elementos y siempre comunicando esto a los stakeholders, que incluyen al equipo de desarrollo, administradores de sistemas, operadores, propietarios de negocios y otras partes interesadas.

Los modelos y vistas elaborados, junto con los materiales de apoyo, se pueden utilizar para proporcionar una comprensión más detallada de la arquitectura y para verificar la coherencia con lo que se describió y planteó originalmente y su alineación con los objetivos.

Al igual que en el trabajo de recopilación y definición de requisitos y/o atributos de calidad, la elaboración de las diferentes vistas y modelos requieren ser evaluados constantemente para recibir retroalimentación, durante esta evaluación a veces puede haber la necesidad de modelos más completos y puntos de vista. En estos casos, se podría solicitar una elaboración para generar modelos y vistas adicionales. Durante la evaluación, estos modelos y vistas se pueden anotar con los resultados de la evaluación y con comentarios sobre las fortalezas y debilidades de la arquitectura o su descripción.

Quizás el concepto más importante asociado con la elaboración de una arquitectura de software y su documentación es la vista. Una arquitectura de software es una entidad compleja que no se puede describir de una sola manera, particularmente cuando en ella se encuentran relacionados muchos escenarios y actores para su funcionamiento, es por ello por lo que múltiples vistas y modos de representación son necesarias para comunicar de forma exitosa lo que se desea lograr con esta y por lo cual es necesario ampliar más el concepto de vistas y cómo se relacionan en la elaboración de una arquitectura.

3.1.2.2.1. Vistas de una arquitectura: Formas de comunicar y representar

Una vista es una representación de un conjunto de elementos del sistema y relaciones que nos permiten mostrar el comportamiento y objetivo principal de la arquitectura, cabe recordar que una arquitectura puede ser una recopilación de arquitecturas más pequeñas que se interrelacionan, dando pie a poder obviar en ocasiones elementos de estas sub-arquitecturas para enfocarse en las relaciones de nuestra arquitectura principal. Como ejemplo podemos tomar una arquitectura cliente-servidor, en la cual una vista puede mostrar los servicios del sistema (propios o de terceros) de forma superficial o no como estos están compuestos, para enfocarse en describir de manera fehaciente pero no completamente detalla la forma en la que estos servicios son usados, el tipo de protocolo, el medio de comunicación o la relación que existe entre ellos, esto incluso permite desacoplar cualquier dependencia a un módulo que requiera ser reemplazado[33].

Es por ello por lo que las vistas nos permiten dividir una arquitectura de software en varias representaciones interesantes y manejables del sistema. Pero el concepto de vistas no se limita en describirla como una representación maleable de la arquitectura, estas también entregan una forma muy confiable y útil de documentación de arquitectura y cómo lograr comunicar. En [35] se habla de la importancia de documentar nuestra arquitectura y de cómo "Documentar una arquitectura se puede resumir en documentar las vistas relevantes y luego agregar documentación que se aplique a más de una vista".

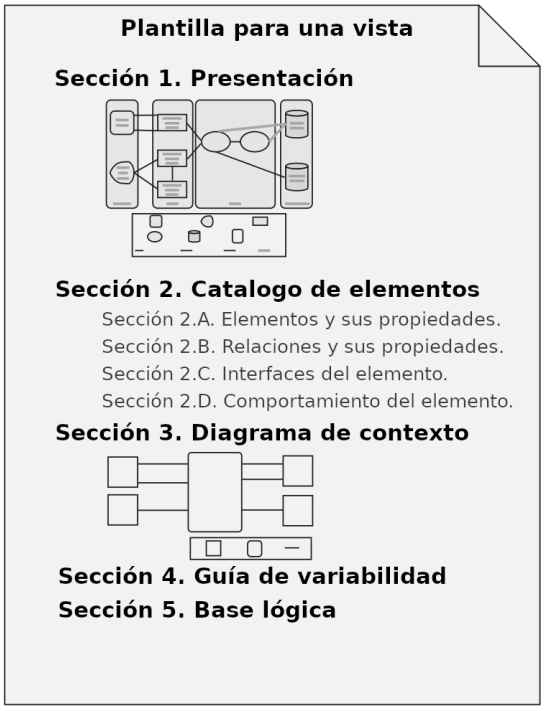

Figura 18: Plantilla de la documentación de una vista [35].

La documentación es imprescindible para dar a entender la arquitectura y esta debe ser explicada destacando sus correspondientes puntos de interés, es por ello por lo que se debe tener claridad qué es lo que se desea comunicar ya que diferentes vistas destacan diferentes elementos y

relaciones del sistema. La intención de crear una vista u otra está normalmente impulsada por la necesidad de documentar un patrón particular en el diseño, es por eso por lo que documentarlas dependen de los usos que se espera hacer de la documentación.

La cantidad de vistas diferentes generadas es el resultado de una decisión de costo / beneficio, cada vista tiene un costo y un beneficio, y debe asegurarse de que los beneficios de mantener una vista en particular superen sus costos.

Dada la importancia que tienen las vistas como medio para comunicar y documentar la arquitectura a las diferentes partes de interés, es relevante conocer varios métodos que permitan describir la arquitectura a otros, entre ellos los más destacados son:

3.1.2.2.1.1. Modelo de vistas 4 + 1

Es un modelo de vistas diseñado por Kruchten que encaja con el estándar "IEEE 1471-2000" [38], utilizado para la descripción de arquitecturas de software intensivo. Cada vista representa dentro de la arquitectura que se esté documentando, aspectos diferentes del sistema software.

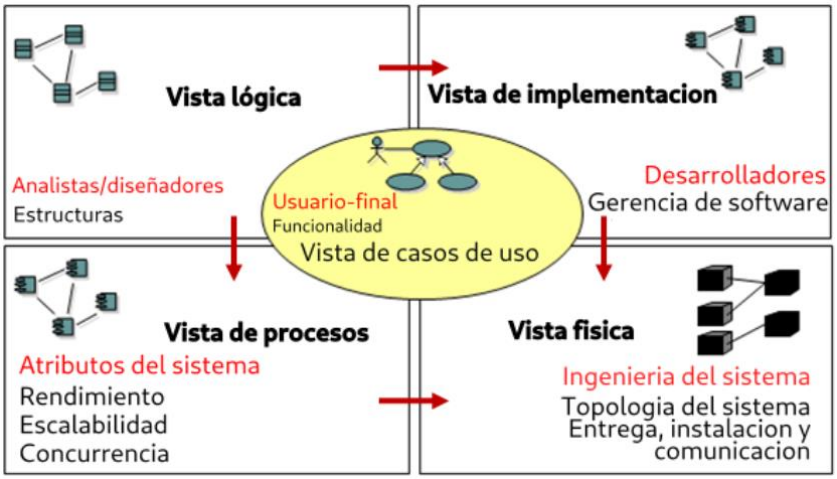

Figura 19: Modelo de vistas 4+1.

La figura ilustra la composición de las vistas 4+1, las cuales se describen a continuación:

1. **Vista Lógica:** Representa la funcionalidad que el sistema proporcionará a los usuarios finales, por lo tanto, describe lo que el sistema debe hacer, las funciones y servicios que ofrece. Incluye diagramas de clases, de comunicación o de secuencia de UML.
2. **Vista de despliegue:** Desde la perspectiva de un programador, se ocupa de la gestión del software; muestra cómo está dividido el sistema software en componentes y las dependencias que hay entre esos componentes. Incluye diagramas de componentes y de paquetes de UML.
3. **Vista de Procesos:** Muestra los procesos que hay en el sistema y la forma en la que se comunican estos procesos; representa desde la perspectiva de un integrador de sistemas, el flujo de trabajo paso a paso de negocio y operacionales de los componentes que conforman el sistema. Incluye el diagrama de actividades del UML.
4. **Vista Física:** Desde la perspectiva de un ingeniero de sistemas, muestra todos los

componentes físicos del sistema, así como las conexiones físicas entre esos componentes que conforman la solución (incluyendo los servicios). Incluye el diagrama de despliegue de UML.

5. **"+1" Escenarios:** Tiene la función de unir y relacionar las otras 4 vistas, a través de casos de uso, permitiendo una trazabilidad de componentes, clases, equipos, paquetes, etc, para realizar cada caso de uso. Incluye el diagrama de casos de uso de UML.

Como nota final, Kruchten no dice de qué manera se ha de documentar cada vista**; sino** que es lo que hay que documentar en cada vista.

Modelo Ambiental

Es un proceso de análisis estructurado de comportamiento que es parte del Modelo Esencial de un sistema. El modelo ambiental define la frontera entre el sistema y el mundo exterior, generalmente empleando un diagrama de flujo de datos de contexto (además de otros componentes). En otras palabras, dice que forma parte del sistema y qué cosas no:

- **Fronteras:** determina hasta dónde llega el sistema.
- **Ambiente:** grupo de sistemas, personas u organizaciones con los cuales un sistema interactúa.
- **Interfaces:** muestra el intercambio de datos entre el sistema y el ambiente.
- **Acontecimientos:** determina los acontecimientos que ocurren en el ambiente a los cuales el sistema debe reaccionar.

3.1.2.2.1.2. UML

UML son una serie de especificaciones para el modelamiento cuyo objetivo es proporcionar a los arquitectos de sistemas, ingenieros de software y desarrolladores de software herramientas para el análisis, diseño e implementación de sistemas basados en software, así como para modelar negocios y procesos similares [37]. Al ser usados para el modelamiento esto quiere decir que permiten representar el sistema usando elementos gráficos, pero que como se explicó previamente no buscan capturar todos los detalles de un sistema complejo en un solo diagrama, ya que representan aquellos detalles que nos permiten mostrar el comportamiento y objetivo principal de la arquitectura.

UML especifica numerosos tipos de diagrama, en [37] capítulo 7.4 se habla de cerca de 14, cada uno provee una vista del sistema, ya que al igual que en el mundo real, diferentes ángulos de cámara son requeridos para proveer un entendimiento de la acción que está tomando lugar.

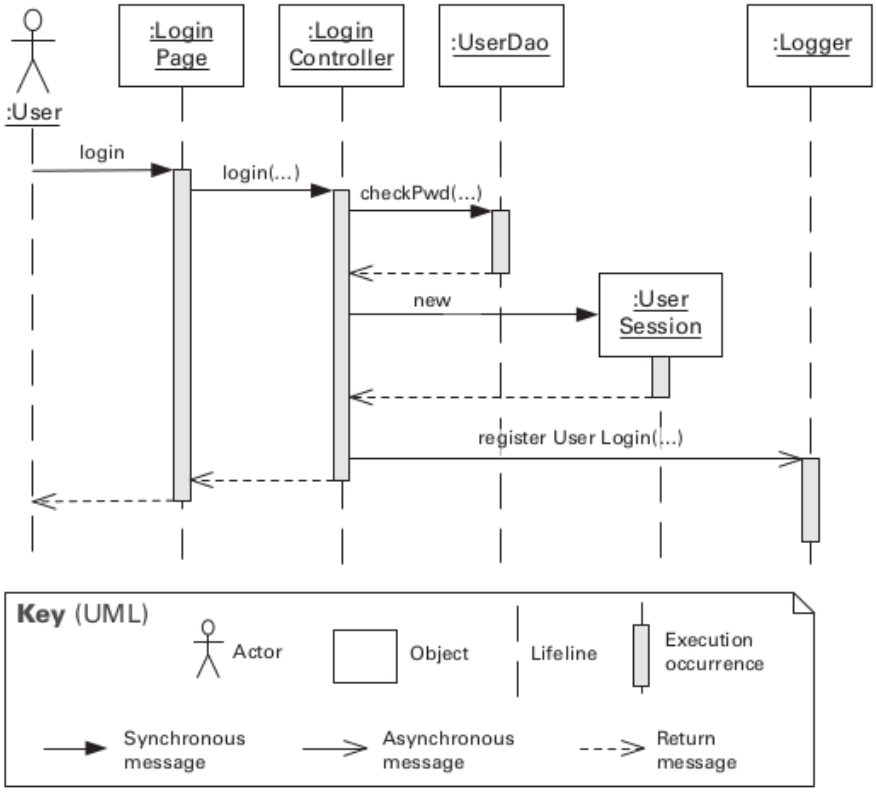

Figura 20: Ejemplo simple de UML [35]

Los diagramas UML pueden ser clasificados en dos grupos:

- Diagramas de estructura (organización de elementos en el sistema)
- Diagramas de comportamiento (manera como todos estos elementos colaboran para realizar su función).

El modelo del sistema presenta varios niveles de detalle:

- **Modelo Conceptual:** Captura el sistema en términos de las entidades dominio que existen o existirán, y su asociación con otras entidades del sistema. El nivel conceptual de modelado se lleva a cabo utilizando la terminología de dominio del negocio.
- **Modelo Lógico:** La vista lógica de un sistema toma el concepto creado en el modelo conceptual y establece la existencia y significado de las abstracciones clave y mecanismos que determinarán la arquitectura del sistema y el diseño general.
- **Modelo Físico:** Describe la composición concreta de software y hardware de la implementación del sistema. Y obviamente, el modelo físico y la especificación tecnológica.

En UML los modelos expresan lo suficiente para permitir a un desarrollador capturar todas las decisiones estratégicas y tácticas que necesita durante el análisis y diseño del sistema y durante la formulación de la arquitectura, y son lo suficientemente completos para servir de modelo de aplicación en casi cualquier lenguaje de programación orientado a objetos.

3.1.3. Architecture Description Language (ADL)

Este enfoque se utiliza para describir la arquitectura del software antes de la implementación del sistema. Aborda los siguientes puntos de interés: comportamiento, protocolo y conector. La principal ventaja de ADL es que podemos analizar la arquitectura en cuanto a integridad, coherencia, ambigüedad y rendimiento antes de comenzar formalmente el uso del diseño [38].

Los ADL proporciona uno o más tipos de modelos como un medio para enmarcar algunas preocupaciones para los stakeholders, y que son lenguajes formales para describir la arquitectura de un sistema de software, permitiendo definir una notación con sintaxis y semántica precisas en las que se pueden expresar los modelos de arquitectura, y proporciona un conjunto de herramientas correspondiente para trabajar con el lenguaje [39]. La siguiente figura es un buen resumen de cómo los ADL se relacionan con otros componentes en los escenarios del desarrollo de una arquitectura de software:

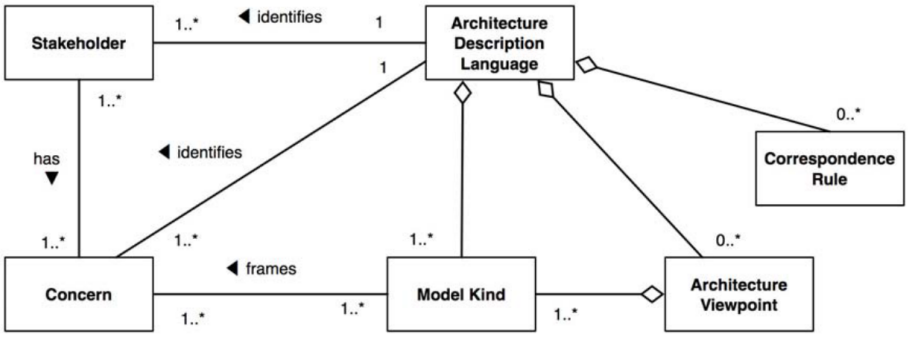

Figura 21: ADL en los escenarios de elaboración de arquitectura de software [38].

Una ADL puede tener un enfoque limitado, definiendo un solo tipo de modelo, o bien enfocado ampliamente para proporcionar varios tipos de modelos, opcionalmente organizados en puntos de vista. El estándar internacional [38] hace énfasis en cómo los ADL se apoyan principalmente en la evaluación de la arquitectura en base a sus atributos de calidad seleccionados, esto es fundamental ya que toma en cuenta el valor que un ASR tiene e impacta incluso la forma en la que se comunica la arquitectura. Algunos ejemplos de ADL incluyen:

- xADL, lenguaje de propósito general
- ACME, lenguaje de propósito general.
- Koala, lenguajes específicos de dominio (DSL)
- AUTOSAR.

Una ADL describe un sistema en el nivel de abstracción de componentes y conectores. Los componentes son unidades de almacenamiento de datos y cálculo. Los conectores describen interacciones entre componentes y las reglas que gobiernan estas interacciones [39].

3.1.3.1. Evaluación

Los procesos de creación de software desde sus etapas más tempranas deben garantizar la calidad y el cumplimiento de sus objetivos [31], para ello es necesario realizar evaluaciones constantes a cada proceso involucrado en la generación de un producto y la arquitectura no es la excepción. Por el contrario, la arquitectura del software es una parte fundamental que debe ser evaluada en todos sus ciclos de vida, las evaluaciones de la arquitectura se pueden realizar en una o más etapas del proceso de desarrollo de software. Se pueden usar para comparar e identificar fortalezas y debilidades en diferentes alternativas de arquitectura durante las primeras etapas de diseño. También se pueden utilizar para la evaluación de sistemas existentes antes del futuro mantenimiento o mejora del sistema, así como para identificar la visión y la vigencia arquitectónica [90].

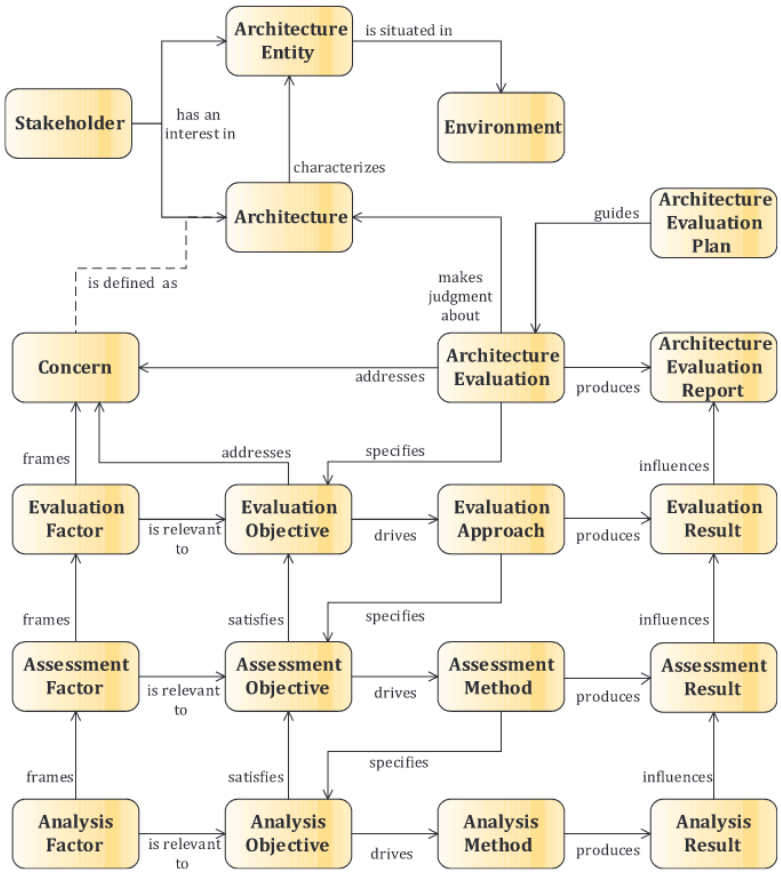

Figura 22: Evaluación en la arquitectura de software [31].

La evaluación toma un rol fundamental ya que durante esta podría reconocerse que se necesitan conceptos de arquitectura alternativa para conseguir cumplir los objetivos. Sin embargo, en algunos casos puede haber una única arquitectura que se esté evaluando para determinar su idoneidad. Las evaluaciones pueden basarse en el análisis de los atributos de arquitectura relevantes, los ASRs, las propiedades de la situación, o en incluso en otra evaluación como resultado de analizar cuánto valor se entrega, por ejemplo, a un entorno operativo a usuarios individuales [88]. Los resultados de la evaluación y el análisis, junto con las estimaciones de la

incertidumbre de la evaluación, se devuelven junto con los hallazgos y recomendaciones clave, para determinar si la arquitectura propuesta aborda suficientemente las preocupaciones de las partes interesadas. De lo contrario, se persiguen ciclos adicionales entre la conceptualización y la evaluación.

La evaluación de la arquitectura hace un juicio con respecto a qué tan bien se han logrado o se lograrán los objetivos de la arquitectura. Puede proporcionar respuestas a un conjunto identificado de preguntas para, por ejemplo, proporcionar insumos para la toma de decisiones estratégicas o para producir una nueva arquitectura que aborde mejor las necesidades actuales y futuras de las partes interesadas [90].

En [47] se hace un estudio comparativo de las formas más comunes de evaluación de una arquitectura de software, se menciona que hay principalmente 4 formas de evaluación, las cuales son:

- **Basadas en la experiencia:** son aquellas que usan la experiencia y conocimiento de dominio de los desarrolladores o consultores. Las personas que han encontrado los requisitos y el dominio del sistema de software antes pueden, basándose en la experiencia previa, decir si una arquitectura de software será lo suficientemente buena [47].
- **Basadas en simulación:** se hace uso de una implementación de alto nivel de algunos o todos los componentes de la arquitectura del software. La simulación se puede utilizar para evaluar los requisitos de calidad, como el rendimiento y la corrección de la arquitectura. La simulación también se puede combinar con la creación de prototipos, por lo tanto, los prototipos de una arquitectura.
- El **modelado matemático** utiliza pruebas y métodos matemáticos para evaluar principalmente los requisitos de calidad operativa, como el rendimiento y la fiabilidad de los componentes de la arquitectura. El modelado matemático se puede combinar con la simulación para estimar con mayor precisión el rendimiento de los componentes de un sistema [47].
- **Basada en escenarios:** Estas intenta evaluar un atributo de calidad particular creando un perfil de escenario que fuerza una descripción muy concreta del requisito de calidad. Los escenarios del perfil se utilizan luego para recorrer la arquitectura del software y se documentan las consecuencias del escenario [47].

El objetivo por ende de una evaluación es generar revisiones y evaluaciones, que pueden identificar las consecuencias de propiedades emergentes potenciales de una entidad de arquitectura que son indeseables, en un escenario como este la evaluación debe, como resultado, recomendar acciones explícitas para abordarlas. Cuando estas consecuencias son relativamente obvias, los evaluadores deben declarar en el informe de la evaluación los riesgos y las posibles consecuencias no deseadas, o la consideración incompleta de las posibles propiedades emergentes de una entidad de arquitectura, a fin de brindar a los arquitectos la oportunidad de mitigar esos riesgos.

Las evaluaciones son oportunidades de mejoras y de conseguir encontrar errores en el diseño, incluso puede ayudar a identificar si un supuesto en la planificación es falso, en este caso los evaluadores que entreguen los resultados deben usar sus conocimientos y habilidades para declarar cualquier riesgos y posibles consecuencias no deseadas de las suposiciones falsas con el fin de brindar a los arquitectos la oportunidad de mitigar esos riesgos, esto proporciona información y recomendaciones para ayudar en la toma de decisiones futuras y puede indicar tendencias y recomendaciones, esto afecta la decisiones futuras a nivel técnico y de tendencias para situaciones comparables en el futuro [31].

Evaluar la arquitectura es un trabajo complejo que puede resumirse en validar el cumplimiento de los objetivos y atributos de calidad [31], pero esto depende completamente del enfoque que se le dé a la evaluación. Debido a la necesidad de entender cómo valorar de manera adecuada si una arquitectura cumple con lo que se espera, en el presente trabajo se documentarán diferentes métodos que permitan tener una visión y un contexto más claro sobre cómo enfocar la evaluación e integrar la de forma orgánica al proceso de desarrollo de la arquitectura. Estos métodos serán ampliados en la sección de "Métodos de evaluación de arquitecturas de software".

3.1.4. Métodos para el análisis y elaboración de arquitecturas

3.1.4.1. Attribute-Driven Design (ADD) [88]

El diseño basado en atributos (ADD) es un método sistemático paso a paso que le ayuda a diseñar una arquitectura eficaz para sistemas con uso intensivo de software. ADD se basa en los requisitos arquitectónicamente significativos (ASR) del sistema, que incluyen requisitos funcionales, requisitos de atributos de calidad y restricciones.

ADD se puede utilizar para dominios de aplicaciones que van desde sistemas de información hasta sistemas integrados. El resultado de usar ADD es un conjunto de bocetos de vistas arquitectónicas con las que un diseñador de sistemas puede trabajar para desarrollar la arquitectura detallada.

ADD sigue un proceso de descomposición iterativo donde, en cada etapa de la descomposición, usted elige tácticas y patrones arquitectónicos para satisfacer los ASR. Hay cinco pasos en este proceso:

1. Seleccione un componente del sistema que aún no haya sido diseñado. En la primera iteración, este será el propio sistema. Con cada iteración sucesiva, examina todos los componentes del sistema que está diseñando.
2. Identificar y priorizar los ASR relevantes que son más importantes para el componente seleccionado. Esto significa que usted selecciona los requisitos funcionales, los requisitos de calidad y las restricciones que podrían afectar la arquitectura del sistema del componente que seleccionó para esta iteración del proceso.
3. Cree un boceto arquitectónico para el componente que cumpla con los ASR. Este paso es el corazón de ADD desde los bocetos arquitectónicos. Los bocetos que crea reflejan los ASR

que seleccionó. Utiliza patrones o tácticas arquitectónicas para guiar su desarrollo y, a menudo, hace concesiones entre tácticas y patrones, y los ASR que satisfacen.

4. Haga un inventario de los ASR restantes y seleccione un subconjunto para el componente que se diseñará en la siguiente iteración.

Repita los pasos 1-4 hasta que se resuelvan todos los ASR y tenga un boceto arquitectónico para cada componente.

3.1.4.2. Architecture Based Design Method (ABD) [75,76]

El método de diseño basado en arquitectura o ABD (de sus siglas en inglés Architecture Based Design Method) describe el sistema o los sistemas que se están diseñando en términos de los principales elementos de diseño y las relaciones entre ellos. La arquitectura conceptual representa las primeras elecciones de diseño realizadas durante un proceso de desarrollo. En consecuencia, es fundamental para lograr los objetivos comerciales y de calidad para el sistema y en proporcionar una base para el logro de la funcionalidad deseada.

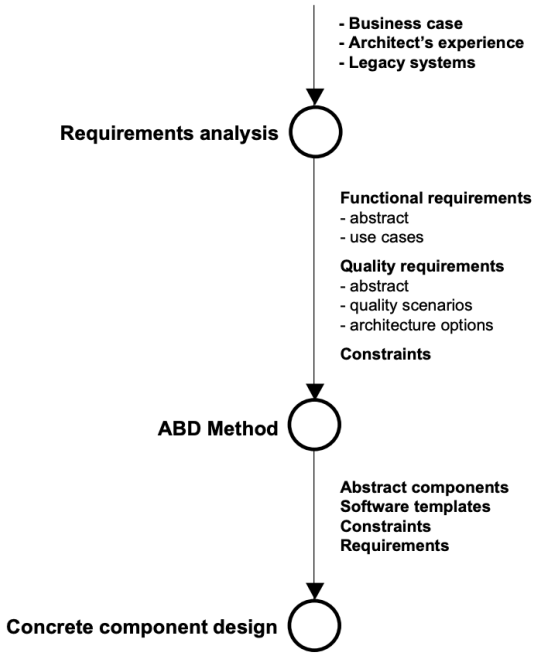

Figura 23: ABD dentro del ciclo de vida [76].

El método ABD tiene tres principios, primero está la descomposición de la función por este motivo el método utiliza técnicas bien establecidas basadas en el acoplamiento y la cohesión, el segundo es la realización de los requisitos de calidad y negocio mediante la elección del estilo de arquitectura. La externa es el uso de plantillas de software. Las plantillas de software son muy importantes ya que facilitan la integración de componentes y subsistemas teniendo en cuenta que todos los elementos de diseño que compartan un determinado patrón pueden

integrarse de la misma forma, también sirven para identificar patrones de comportamiento de diferentes elementos de diseño.

Las entradas del método están constituidas por una serie de requisitos y restricciones que deben de cumplir y de este modo satisfacer la arquitectura. Los requisitos se clasifican en tres: requisitos funcionales, requisitos de calidad y requisitos del negocio.
El método ABD es recursivo y el paso a paso dentro de cada iteración están claramente definidos. Por lo tanto, siempre queda claro durante el uso qué pasos de diseño se han logrado y cuáles quedan, de este modo ayudando a que el proceso de diseño arquitectónico sea menos misterioso. Este método usa el término elemento de diseño para referirse generalmente al sistema, un subsistema o un componente conceptual.

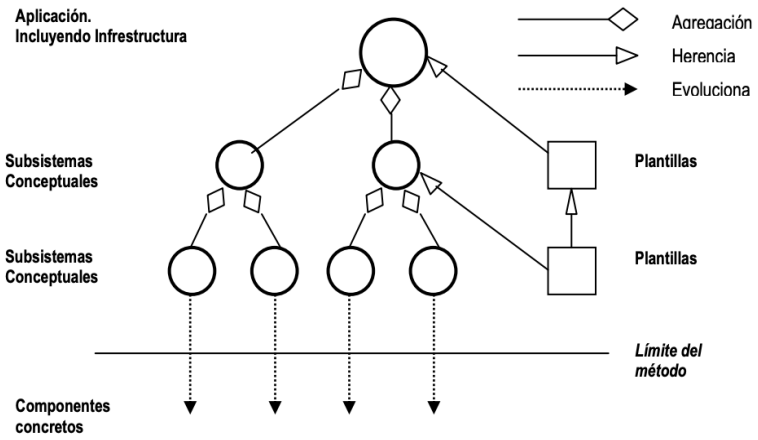

Figura 24: Descomposición del sistema en elementos de diseño [76].

De acuerdo con el método ABD cada elemento de diseño tiene su propia característica la cual es satisfacer los requisitos funcionales y de calidad, la plantilla y una serie de restricciones de diseño.

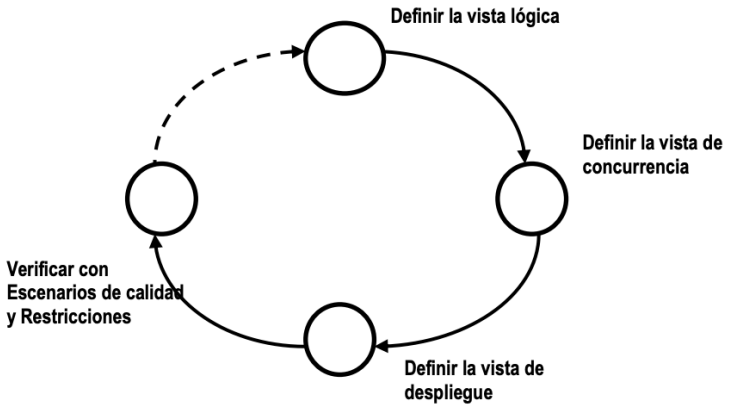

Figura 25: Pasos para la descomposición de un elemento de diseño [76].

Cuando un elemento se descompone el resultado son dos o más elementos, cada uno con sus propios requerimientos, plantillas y restricciones. Los pasos del método se mencionan a continuación:

- **Definición de la vista lógica:** Como se puede observar en la figura 4 se muestra el paso a paso para la definición de la vista lógica: descomposición de la funcionalidad, seleccionar el estilo arquitectónico, asignar la funcionalidad a los componentes que ya fueron definidos, mejoramiento de plantillas y realizar verificación con casos de uso.

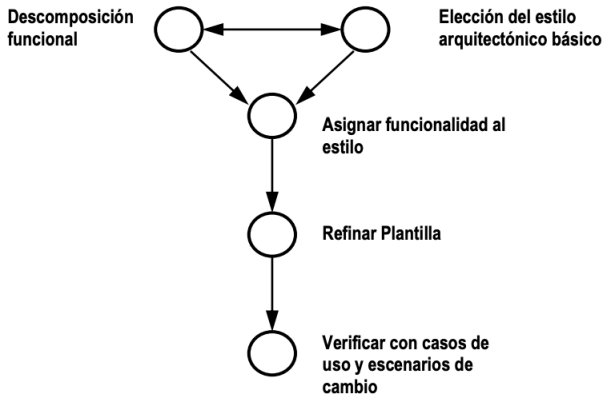

Figura 26: Definición de la vista lógica [76].

- **División de la funcionalidad.** Cada elemento de diseño tiene asignadas unas responsabilidades para poder cumplir con los requisitos de diseño. Dichas responsabilidades, las cuales están constituidas por la funcionalidad que realiza cada elemento y los atributos de calidad, pueden dividirse en diversos grupos.
- **Elección del estilo arquitectónico:** Cada elemento de diseño debe de tener un estilo arquitectónico, que debe de definir la forma en que cada elemento de diseño realiza sus responsabilidades. La forma de seleccionar dicho estilo debe de estar basada en las directrices arquitectónicas correspondientes a cada elemento. Una vez el estilo esté seleccionado este debe de adaptarse a los requisitos de calidad, para ello debe de hacerse una revisión de cada uno de estos requisitos y de este modo determinar si tiene alguna relevancia en el elemento que va a ser descompuesto.
- **Definición de la vista de concurrencia:** El objetivo de describir una vista de concurrencia es determinar aquellas actividades del sistema que deben ejecutarse al mismo tiempo, con el fin de descubrir los puntos de sincronización y de acceso a recursos compartidos.
- **Definición de la vista despliegue:** Si el sistema incluye más de un procesador es necesario describir cómo pueden asignarse elementos de diseño a los mismos. Un thread virtual no tiene una ubicación determinada y puede trasladarse a través de una red de un procesador a otro. ABD denomina thread físico a aquel que se ejecuta en un procesador determinado.

3.1.4.3. The Open Group Architecture Framework (TOGAF) [79, 80, 81]

TOGAF es un marco de trabajo propuesto por el OpenGroup que está basado en los principios de la arquitectura empresarial. Es un modelo iterativo que establece un conjunto de buenas prácticas para llevar a la organización a una alineación tecnológica de los procesos y objetivos estratégicos. La propuesta metodológica de TOGAF está centralizada en el Método de desarrollo de arquitectura (ADM), este presenta un ciclo de fases que permite definir un modelo de negocio alineando los objetivos, estrategias, procesos y tecnología de la organización (ver figura 27).

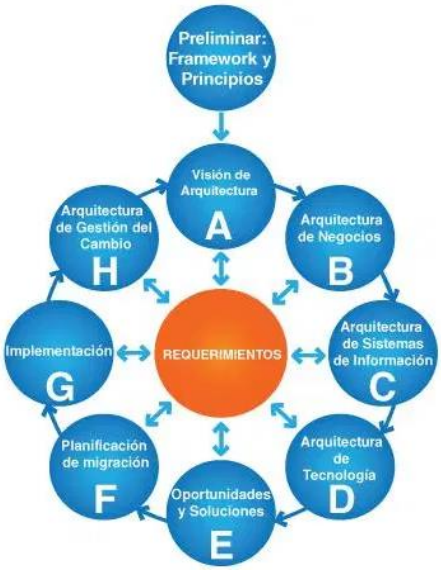

Figura 27: Estructura ADM-TOGAF, por ares (2013) Recuperado.

La parte que puede considerarse de suma importancia de este framework está compuesta por su "Architecture Development Method" o ADM. Siendo este el que define el proceso a realizar en la arquitectura, partiendo desde lo más simple hasta lo más concreto. Este ADM está conformado por las siguientes fases y se organiza de modo repetido y cíclico de la siguiente manera:

- **FASE A**: Es denominada visión de la arquitectura, delimita el alcance del proyecto y la estrategia para llevarla a cabo.
- **FASE B:** Es llamada arquitectura del negocio, busca obtener una clara arquitectura del negocio y las metas que quiere cumplir para revisar si es viable o no complementar con TI.
- **FASE C:** Esta fase se denomina, arquitectura de sistemas de información, contemplando las arquitecturas particulares para datos y aplicaciones, los cuales pueden ser desarrollados de manera simultánea o secuencial.
- **FASE D:** Se denomina arquitectura tecnológica, define la arquitectura que se integrará para el avance en las fases posteriores, aquí se aborda la documentación de la organización

esencial de sistemas de TI, representada en hardware, software y tecnologías de la información.

- **FASE E:** Se determina el inventario de elementos con los cuales se cuentan para montar la fase D. En ella se establece cuáles componentes son necesarios comprar, modificar o arreglar para que pueda servir en la arquitectura.
- **FASE F:** Es nombrada plan de migración, se coloca la prioridad a los proyectos paralelos y se gestiona un plan que permitirá llevar a cabo la migración de la empresa al sistema desarrollado.
- **FASE G:** Se denomina control de la implementación, es donde se lleva a cabo la realización de los proyectos que permitirán el desarrollo de las soluciones de TI.
- **FASE H:** Es llamada administración del cambio de la arquitectura, se monitorea y evalúa los sistemas existentes a fin de determinar cuándo se debe iniciar un nuevo ciclo de ADM.

3.1.4.4. Quality Attribute Workshop (QAW) [82, 83, 89]

En esencia, QAW es una versión ligera de una evaluación ATAM. Como una evaluación de ATAM, no apunta a una medida absoluta de la calidad arquitectónica. Más bien, el objetivo es identificar:

- Puntos de sensibilidad de los atributos de calidad, compensaciones y riesgos.
- Posibles estrategias de mitigación.

En un QAW, la carga real del análisis recae en los contratistas, con el SEI facilitando la revisión del análisis. Las partes interesadas suelen ser arquitectos, promotores y gerentes, representantes de los patrocinadores, ingenieros de sistemas y software, personal de logística, usuarios finales y otros que tienen un interés personal en el sistema.

El proceso de QAW se utiliza para descubrir y documentar los riesgos de los atributos de calidad, los puntos sensibles, y compensaciones, donde: Los riesgos de los atributos de calidad son decisiones arquitectónicas que pueden crear problemas futuros para algún requisito de atributos de calidad. Los puntos de sensibilidad son parámetros arquitectónicos para los que al realizar un pequeño cambio hace una diferencia significativa en algún atributo de calidad. Las compensaciones son parámetros arquitectónicos que afectan a más de un atributo de calidad.

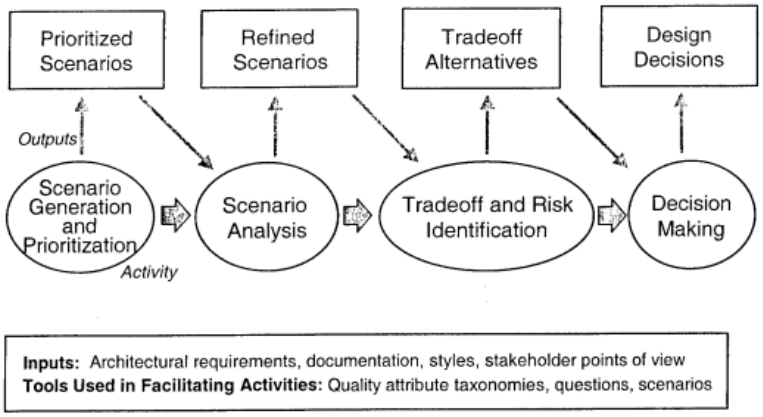

Figura 28: Actividades principales de QAW.

Como se puede observar en la figura anterior el proceso de QAW consiste en la realización de cuatro actividades principales divididas en 8 pasos que se mencionan a continuación [89]:

1. **Presentación e introducciones de QAW:** describe la motivación para QAW y explican cada paso del método.
2. **Presentación comercial / programática:** Un Stakeholder presenta los objetivos de negocio y/o programáticos del sistema.
3. **Presentación del plan arquitectónico:** Un Stakeholder técnico presenta los planes arquitectónicos del sistema tal como están con respecto a los documentos anteriores, gráficos para describir los detalles técnicos del sistema.
4. **Identificación de impulsores arquitectónicos:** Se incluyen requisitos de alto nivel, preocupaciones comerciales / de misión y varios atributos de calidad. Durante este paso, los facilitadores y los stakeholders llegan a un consenso sobre qué impulsores son clave para el sistema.
5. **Lluvia de ideas de escenarios:** Los stakeholders generan escenarios del mundo real para el sistema. Los escenarios comprenden un estímulo relacionado, una condición ambiental y una respuesta. Los facilitadores se aseguran de que al menos un escenario aborde cada uno de los impulsores arquitectónicos identificados en el Paso 4.
6. **Consolidación de escenarios:** que son similares en contenido.
7. **Priorización de escenarios:** Los stakeholders priorizan los escenarios a través de un proceso de votación.
8. **Refinamiento del escenario:** Para los cuatro o cinco escenarios principales, se describe lo siguiente:
   a. Los objetivos comerciales / programáticos que se ven afectados por estos escenarios.
   b. los atributos de calidad relevantes asociados con esos escenarios.

3.1.4.5. Rational Unified Process (RUB) [84, 85, 86, 87]

El Proceso Unificado de Rational (RUP) es un proceso de ingeniería del software el cual proporciona un acercamiento a la asignación de tareas y responsabilidades en una organización

de desarrollo. Su propósito es asegurar la producción de software de alta calidad que se ajuste a las necesidades de sus usuarios finales [87].

Las características principales de RUP son:

- **Dirigido por casos de uso:** Los casos de uso son una técnica de recolección de requisitos de un sistema desde el punto de vista del usuario. Los Casos de Uso constituyen un elemento integrador y una guía del trabajo como se muestra en la siguiente figura:

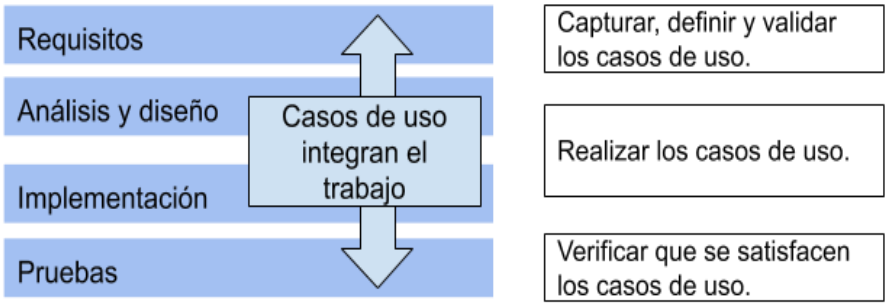

Figura 29: Los casos de uno integran el trabajo.

- **Centrado en arquitectura:** Se representa mediante varias vistas que se centran en aspectos concretos del sistema, abstrayéndose de lo demás. Todas las vistas juntas forman el llamado modelo 4+1 de la arquitectura, recibe este nombre porque lo forman las vistas lógicas, de implementación, proceso y despliegue, más la de casos de uso que es la que da adhesión a todas. Como resumen se muestra la siguiente figura.

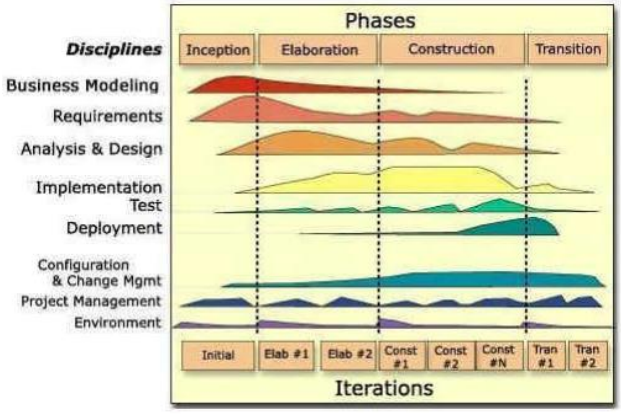

Figura 30: Fases, iteraciones y disciplinas [87].

- **Iterativo e Incremental:** Para hacer más manejable un proyecto se recomienda dividirlo en ciclos. Para cada ciclo se establecen fases de referencia, donde cada una debe ser considerada como un mini proyecto cuyo núcleo fundamental está constituido por una o más iteraciones de las actividades principales básicas de cualquier proceso de desarrollo.

3.1.5. Métodos de evaluación de arquitecturas de software

3.1.5.1. Software Architecture Analysis Method (SAAM) [47]

Es un método de evaluación de la arquitectura de software basado en escenarios, destinado a evaluar una arquitectura única o hacer comparables varias arquitecturas utilizando métricas como el acoplamiento entre componentes de la arquitectura. SAAM ha evolucionado a un método estructurado para la evaluación de la arquitectura de software basada en escenarios. Se pueden abordar varios atributos de calidad, según el tipo de escenarios que se creen durante el proceso de evaluación. El método consta de cinco pasos:

1. Comienza con la documentación de la arquitectura de una manera que todos los participantes de la evaluación puedan entender.
2. Luego, se desarrollan escenarios que describen el uso previsto del sistema. Los escenarios deben representar a todas las partes interesadas que utilizarán el sistema.
3. Luego se evalúan los escenarios y se selecciona un conjunto de escenarios que representa el aspecto que queremos evaluar.
4. Los escenarios interactivos se identifican luego como una medida del modularidad de la arquitectura.
5. Luego, los escenarios se ordenan según la prioridad y su impacto esperado en la arquitectura.

3.1.5.2. Attribute-Based Architectural Styles ABAS [47]

Attribute-Based Architectural Styles (ABASs) se define a partir del concepto de estilos arquitectónicos y se extiende asociando un marco de razonamiento con un estilo arquitectónico. El método se puede utilizar para evaluar varios atributos de calidad, por ejemplo, rendimiento o capacidad de mantenimiento, y por lo tanto no está dirigido a un conjunto específico de atributos de calidad. El marco de razonamiento para un estilo arquitectónico puede ser cualitativo o cuantitativo, y se basa en modelos para atributos de calidad específicos. Por lo tanto, los ABAS permiten el análisis de diferentes aspectos de calidad de las arquitecturas de software basadas en ABAS. El método es general y se pueden analizar varios atributos de calidad al mismo tiempo, dado que se proporcionan modelos de calidad para los atributos de calidad relevantes. Uno de los puntos fuertes de los ABAS es que también se pueden utilizar para el diseño arquitectónico.

3.1.5.3. Architecture Tradeoff Analysis Method (ATAM) [47, 77, 78, 90]

El Architecture Tradeoff Analysis Method (ATAM) es un método para evaluar Arquitecturas de Software que principalmente evalúa la adecuación de la Arquitectura de Software definida con respecto a los atributos de calidad especificados para el sistema. Surge reuniendo ideas y técnicas de tres áreas: la noción de estilos o patrones de arquitectura, el análisis de atributos de calidad y el método Software Architecture Analysis Method (SAAM) que es el predecesor del ATAM. El ATAM puede usarse para crear definiciones preliminares de la arquitectura, analizar las decisiones arquitectónicas sin necesidad de tener el código y

analizar sistemas que ya existen. Siendo su propósito principal descubrir las consecuencias de las decisiones arquitectónicas a la luz de los requisitos de los atributos de calidad. (kazman et al 2000).

El análisis de los atributos de calidad específicos puede desarrollarse teniendo en cuenta su dependencia o relación con los demás atributos del sistema. Dichos atributos de calidad no son independientes, sino que interactúan entre sí a través de las relaciones de estructura impuestas por la arquitectura.

Las bases en las que se sustenta el método pueden ser

- No evalúa la arquitectura respecto a los atributos de calidad abstractos, sino respecto de requisitos concretos
- Toma en consideración tanto los requisitos técnicos, como aspectos sociales y de negocio.
- Su ejecución debe consumir pocos recursos y realizarse en un lapso relativamente corto.
- Noción del estilo arquitectónico
- Caracterización de los atributos de calidad.

Para realizar la evaluación de un diseño arquitectónico es necesario contar con una caracterización de los atributos de calidad. Por ejemplo, para entender una arquitectura desde el punto de vista de la usabilidad requiere entender cómo observar y medir este atributo, así como entender de qué forma impacta este atributo los distintos tipos de decisiones arquitectónicas. Dicho proceso de evaluación permitirá descubrir los riesgos, puntos sensibles y puntos de compromiso asociados a las decisiones arquitectónicas:

- **Riesgos:** Se generan cuando hay que elegir entre varias decisiones de diseño arquitectónico o cuando una vez tomada la decisión no pueden determinarse claramente los efectos que produce sobre algún atributo de calidad.
- **Los puntos sensibles** de una arquitectura son aquellas propiedades de sus componentes y su relación con el cumplimiento de un determinado atributo de calidad y que por tanto son críticas para el cumplimiento de este.
- **Los puntos de compromiso** son los lugares donde se producen los conflictos entre atributos, o, dicho de otra manera, los lugares en los cuales los atributos interaccionan y no pueden considerarse por separado.

Con objeto de identificar los riesgos, puntos sensibles y puntos de compromiso, ATAM utiliza tres elementos: los escenarios de alta prioridad, las cuestiones específicas de atributo y los estilos arquitectónicos. Durante la evaluación, el arquitecto realiza los escenarios a través de las estructuras definidas en la arquitectura. Para ello deben identificarse los componentes y conectores involucrados en su realización y plantearse las cuestiones necesarias para identificar los riesgos, puntos sensibles y puntos de compromiso.

El propósito del método no es realizar análisis precisos de los atributos de calidad, sino identificar tendencias que permitan determinar si el enfoque arquitectónico adoptado es el correcto para descubrir en qué lugares y de qué manera se afectan las decisiones de diseño arquitectónicos. Una vez encontrados los puntos sensibles y de compromiso se puede determinar la necesidad de modelar con mayor exactitud los componentes y analizar su comportamiento en relación con los atributos de calidad.

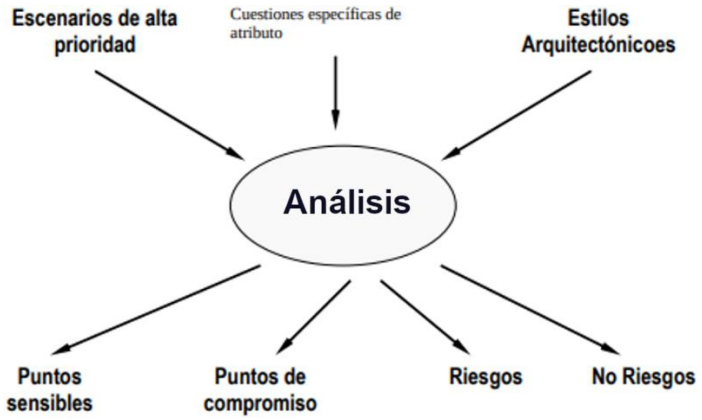

Figura 31: ATAM: Entradas y salidas.

3.1.6. Metodología SEI (software Engineering Institute) para la creación de arquitecturas de software

El Software Engineering Institute (SEI) de Carnegie Mellon, es una entidad académica de software de mucho renombre en el desarrollo e investigación de técnicas relacionadas a la arquitectura de software [49], el SEI a través de los años ha enfocado gran parte de sus esfuerzos en establecer una metodología orientada a generar soluciones arquitectónicas empresariales de calidad.

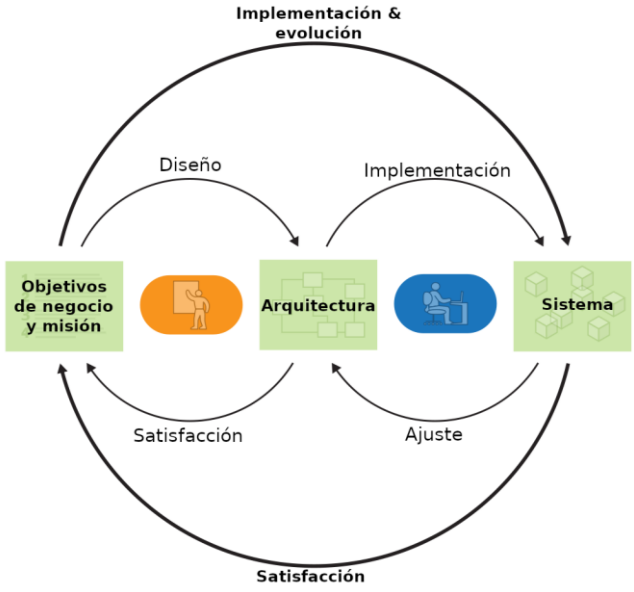

Figura 32: Enfoque del SEI para la calidad [49].

En su artículo de historia de innovación [49], el SEI explica cómo por cerca de más de 15 años de innovación y liderazgo su trabajo se ha enfocado en gran parte a la definición, evaluación, análisis y documentación de la arquitectura de software. El trabajo de SEI ha incluido el desarrollo de diferente material bibliográfico, el primer libro de arquitectura de software para profesionales, Software Architecture in Practice, ganador del prestigioso premio JOLT de la revista Software Development. Siguieron otros tres libros igualmente seminales. La relevancia del material bibliográfico para la arquitectura de software del SEI se puede ver en lo frecuente que este es citado, los libros se han actualizado en varias ediciones y en conjunto se han vendido más de 150.000 copias [49].

Se justifica por lo anterior la relevancia que tiene el SEI, de la cual gracias a sus investigaciones ha emergido una metodología de proceso de desarrollo unificado. Esta metodología puede ser explicada en cuatro métodos enfocados a requerimientos, diseño, documentación y evaluación arquitectónica. En "Software Architectural in practice" él SEI hace uso de la siguiente figura para explicar su visión de los principios y prácticas de las arquitecturas de software:

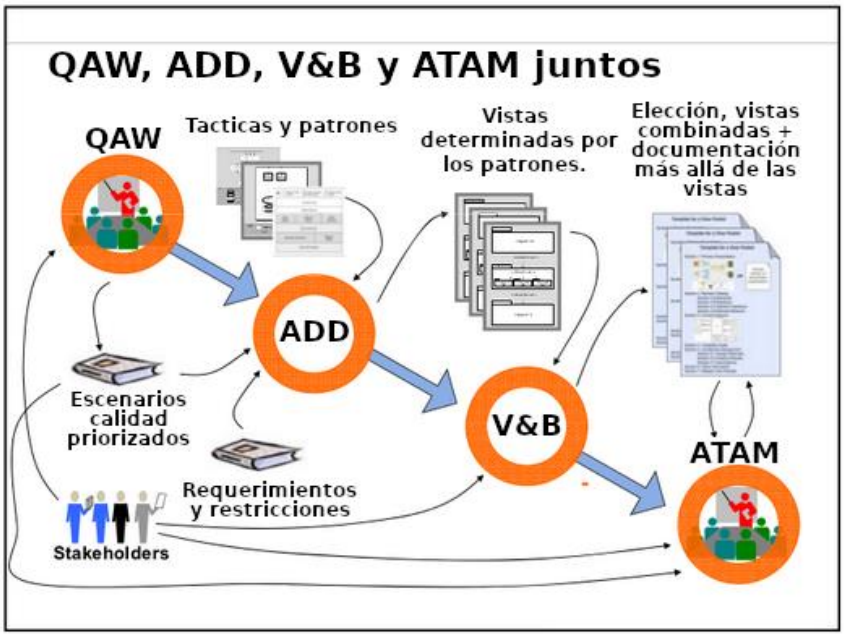

Figura 33: Representación visual de la metodología del SEI [45].

Como se puede ver el SEI basa su metodología en la implementación de 4 métodos para cada una de las fases de la arquitectura de software, como se ha explicado en los anteriores capítulos, el estándar internacional de la IEEE [31] habla de 3 etapas conceptualización, elaboración y evaluación. La metodología no se separa mucho de esta visión y estas etapas del estándar internacional en términos del SEI están enmarcadas en el mismo concepto.

Métodos para la arquitectura de software

El SEI tiene una variedad de métodos y prácticas probados que pueden ayudar a las organizaciones a utilizar la arquitectura de software para construir y desarrollar sistemas que satisfagan sus objetivos comerciales y de misión. Estos métodos y prácticas, que se aplican en

diferentes etapas del ciclo de vida del desarrollo, han evolucionado con la llegada de los enfoques de desarrollo ágiles.

- **Análisis de los ASRs:** analizando los impulsores comerciales, el contexto del sistema y los factores que los stakeholders del sistema consideran críticos para el éxito. El SEI usa dos métodos para identificar importantes atributos de calidad no funcionales del sistema (por ejemplo, rendimiento, confiabilidad, seguridad, protección) y aclarar los requisitos del sistema: **QAW** y ***Mission Thread Workshop***. En este proyecto nos enfocaremos en **QAW,** método que se enfoca en obtener los atributos de calidad descritos con ayuda de los stakeholders y escenarios relacionados con la arquitectura, para más información de QAW se tiene el capítulo **3.1.4**.
- **Diseñar una arquitectura:** con el SEI se plantea el desarrollo de estructuras arquitectónicas y estrategias de coordinación que satisfagan los requisitos. El SEI propone el método **ADD** como una ayuda a las organizaciones a diseñar arquitecturas efectivas de forma iterativa de descomposición del sistema en componentes cada vez más pequeños. Principalmente **ADD** toma los escenarios ya priorizados para diseñar la arquitectura realizando elecciones de diseño (usando soluciones conceptuales llamadas "patrones" o "tácticas") que permitan satisfacer los requerimientos descritos por los escenarios.
- **Documentación de la arquitectura:** con suficiente detalle y en una forma de fácil acceso para facilitar la comunicación con los desarrolladores y otras partes interesadas y para respaldar el análisis. Nuestro enfoque de Vistas y más allá captura múltiples vistas de arquitectura, cada una de las cuales aborda las preocupaciones de partes interesadas particulares. El SEI propone hacer uso de vistas con un método que ellos llaman el Views and Beyond (V&B) que se utilizan para analizar el rendimiento, la seguridad y la protección de los sistemas permitiendo comunicar como la arquitectura está implementada.
- **Evaluación de la arquitectura** determinando su capacidad para respaldar los atributos de calidad que cumplen con los objetivos comerciales y de misión de una organización. Nuestro método de análisis de compensación de arquitectura (ATAM) y las prácticas de revisión de diseño incrementales ayudan a las organizaciones a obtener una visión temprana y continua de sus arquitecturas de software.

Se considera al SEI como una opción óptima para para el desarrollo de este proyecto y su arquitectura resultante. Esto se justifica con lo anteriormente explicado y en cómo los métodos anteriormente documentados encajan de forma orgánica con la metodología del SEI. No obstante, si bien la metodología propuesta por el SEI ha recibido gran aceptación en el área de la Ingeniería del Software, y se considera como una gran candidata para su implementación en este proyecto, hay un punto crítico para que su uso sea viable, y es que, dado su enfoque a proyectos de gran envergadura, estos métodos tienden a ser costosos en términos de tiempo de adaptación y capacitación. El SEI es un ente que está principalmente enfocado a soluciones empresariales [49] con equipos grandes de cerca de 20 o más personas [48], esto es una muy buena noticia para muchas empresas que deseen mejorar sus procesos y trabajar con arquitecturas más fiables y mejor documentadas, pero esto tiene el efecto contrario cuando los equipos son pequeños. Los equipos pequeños tienen necesidades diferentes en algunos aspectos

en comparación a equipos grandes, esto debido generalmente a la falta de recursos, personal y tiempo. Ya que este proyecto está limitado y cuenta con un poco número de actores, se considera necesario ampliar más cómo flexibilizar un poco la metodología del SEI y adaptarla para equipos pequeños:

3.1.7. Metodología SEI para equipos pequeños de desarrollo [48]

Como se explicó anteriormente debido a la naturaleza empresarial y macro de la metodología que propone el SEI, es necesario adaptar esta al contexto del proyecto actual. La cantidad de recursos que se tiene son limitados al igual que el número de personas disponibles para la ejecución de este, por lo que la metodología debe ser puesta en visión de ser adaptable a un equipo pequeño de desarrollo que permita mantener la calidad y que a su vez de la posibilidad de ser ágiles.

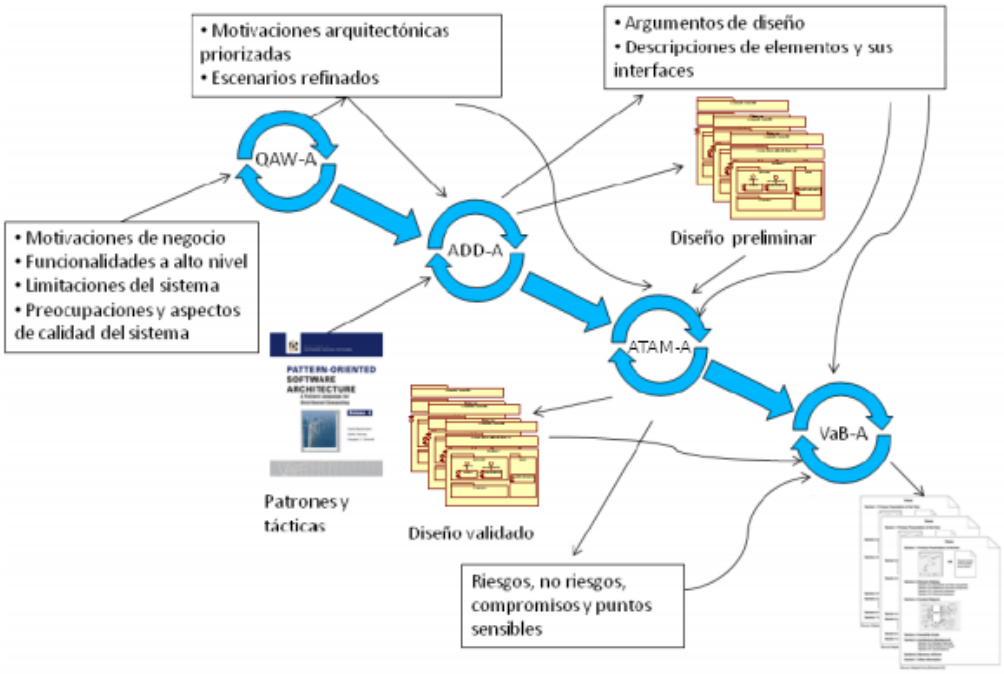

Figura 34: Representación de la metodología SEI adaptada.

En [48] los autores hacen énfasis en cómo esta metodología a pesar de ser muy aceptada y tener grandes beneficios [49], resulta ser un poco difícil de implementar en equipos de desarrollos como latinoamericanos (en el caso concreto del artículo son de México) que se caracterizan por ser equipos pequeños. Debido a esta necesidad se habla de cómo poder usar el SEI en equipos que no sean muy numerosos. Como resultado tenemos la siguiente metodología adapta que podemos ver en esta figura:

En la figura 34 se pueden ver un par de cambios, los cuales están enmarcados en alterar el orden de las etapas, agregar dependencias de resultados y finalmente remover ATAM en pro de usar una metodología llamada ATAM-A como reemplazo, de igual forma los métodos como se puede denotar son un tanto distintos ya que se les agrega una A para denotar que estos han sido

adaptados en algunos aspectos, por lo tanto, podemos nombrar a esta metodología como SEI-A y expandir un poco más su definición.

3.1.8. Metodología SEI-A, adaptación del SEI [51]

En trabajos anteriores [51] se demostró la efectividad de la metodología denominada SEI-A, esta metodología es la adaptación del SEI que se ve en [48], la cual propone hacer una serie de modificaciones sobre la metodología original del SEI de tal forma que sea más flexible, está adaptada cada uno de los métodos originales del SEI [48] se pueden hacer las siguientes observaciones de cada uno de los métodos adaptados:

- **QAW-A:** Las adaptaciones se centran en reducir el número de participantes (stakeholders) y en preparar la primera parte del **paquete de evaluación**. Las razones de la reducción son que las adaptaciones presentadas aquí están orientadas a pequeños equipos de desarrollo [53].

  La siguiente lista presenta los pasos de QAW-A y analiza cómo se adaptaron del QAW original, para aquellos métodos que se mantuvieron igual se omite la explicación:

  1. **Preparación del taller:** Se elabora una guía de participación con la siguiente información: Qué son los atributos de calidad, qué escenarios son y cómo se estructuran, quiénes son los participantes del taller. Este paso no existe en el método original.
  2. **Presentación del taller**.
  3. **Presentación de los objetivos comerciales.**
  4. **Presentación del plan arquitectónico.**
  5. **Identificación de controladores arquitectónicos.**
  6. **Generación de escenarios de atributos de calidad:** En este paso, se sugiere limitar la cantidad de escenarios que se generan.
  7. **Consolidación de escenarios.**
  8. **Priorización de escenarios.**
  9. **Refinamiento del escenario.**
  10. **Preparación del paquete de evaluación**: en preparación para el diseño y la evaluación, las partes interesadas también priorizan los requisitos funcionales primarios (casos de uso). Estos casos de uso se eligen porque son críticos o porque "ejercitan" muchos elementos arquitectónicos.

      Las limitaciones técnicas también se recopilan de la documentación existente. El conjunto de escenarios de atributos de calidad, requisitos funcionales primarios y limitaciones técnicas componen la primera parte del paquete de evaluación que se completará en el siguiente método.

- **ADD-A:** El objetivo de este es producir una documentación preliminar que se utilizará para evaluar el diseño.

La idea es agilizar el diseño arquitectónico, tomando como entrada las salidas de QAW-A y producir una documentación preliminar, no tan detallada como la propone VaB, que se pueda usar para evaluar el diseño propuesto. Las adaptaciones a ADD son las siguientes:

1. **Verificar que las entradas estén completas:** Esta es solo una verificación rápida para verificar que una lista priorizada de atributos de calidad, requisitos funcionales de alto nivel y restricciones de diseño según lo producido por QAW-A está disponible y completa.
2. **Elegir un elemento para descomponer.**
3. **Identifique la lista de controladores arquitectónicos candidatos.**
4. **Seleccione conceptos de diseño que satisfagan los impulsores arquitectónicos candidatos.**
5. **Instanciar elementos arquitectónicos y asignar responsabilidades.**
6. **Definir interfaces para los elementos instanciados.**
7. **Verificar y perfeccionar los requisitos.**
8. Si es necesario, se repiten los pasos 2 a 7 para el siguiente elemento de la descomposición.
9. **Preparación del paquete de evaluación:** Una vez terminado el diseño, se prepara el resto del paquete de evaluación. El paquete se compone de la lista de requisitos derivados de QAW-A y que sirvió como entrada para ADD-A, y el conjunto de plantillas de diseño que se produjeron durante ADD-A. (Este paso no está incluido en el ADD original).

- **ATAM vs ATAM-A:** Uno de los cambios más importantes que se propone por parte de esta adaptación es el uso de ATAM-A en lugar de ATAM como método para la evaluación de la arquitectura (además de esta ya no ser la etapa final).
  La motivación de esto se debe principalmente a que **ATAM-A no es realizado por un equipo de evaluadores externos, es realizada por miembros de la organización** de desarrollo con la documentación preliminar entregada por ADD-A, esta es imprescindible para el desarrollo de este proyecto y que no se cuenta con una entidad externa que pueda evaluar la arquitectura. De igual forma ATAM-A debe asegurar que el diseño propuesto satisfaga las motivaciones arquitectónicas identificadas en QAW-A, el cual especifica las motivaciones y objetivos que deben ser cumplidos.

- **Documentación vs Evaluación, VaB-A:** Otro cambio importante en la metodología adaptada se encuentra en el orden en el que las etapas de evaluación y documentación son ejecutadas. El SEI propone usar la evaluación como una etapa final, permitiendo entregar por lo tanto un producto casi terminado para de este modo ser evaluado junto a su documentación.
  Ciertamente tener un resultado completo es una solución que sirve para evaluar la arquitectura como un todo pero como se hace énfasis en [48] esto tiene un par de problemas particularmente cuando no podemos depender de evaluadores externos ni de mucho personal, otra consideración para alterar este orden es debido a que evaluar el paquete de documentación puede representar retrabajo en el caso de que la evaluación no sea

satisfactoria.

La alteración de la evaluación en pro de hacerse antes de la documentación elimina el trabajo extra que puede significar el encontrar problemas de diseño en las etapas previas y de este modo generar una documentación a partir de una arquitectura ya evaluada y aprobada. Como aclaración esto no significa que la arquitectura no pueda ser documentada previamente, de hecho, se aconseja generar documentaciones pequeñas que permitan comunicar la arquitectura con el equipo.

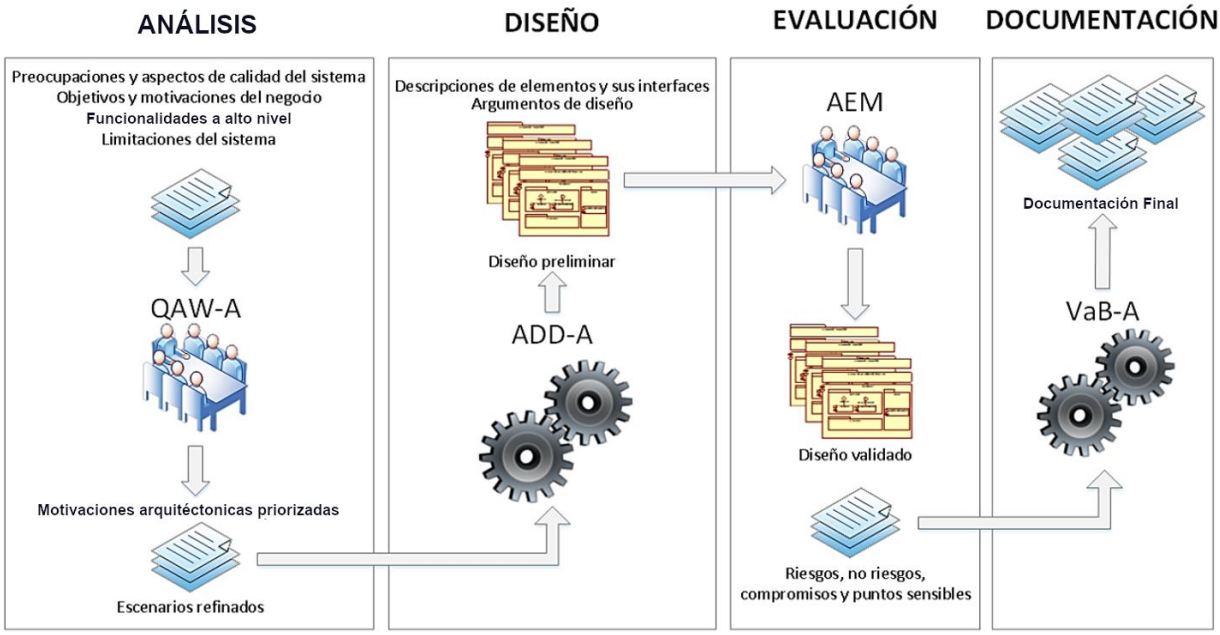

Figura 35: Metodología SEI-A [51].

3.2. NLP y NLU en el desarrollo de chatbots

Dentro del desarrollo del proyecto se hace énfasis sobre el uso de servicios cognitivos para la identificación de intenciones. Estos servicios en su gran mayoría consisten en modelos de aprendizaje automático pre entrenados que hacen uso de técnicas de NLP para el manejo del texto [50] [68]. Se hizo fundamental por ende entender el estado del arte del NLP, sus conceptos, cuáles son sus motivaciones, las tendencias y que aspectos son necesarios en la toma de decisiones al realizar el diseño de una arquitectura que realice una implementación de este.

Una acotación necesaria para tomar en cuenta al leer este capítulo es que se tomaron los conceptos de NLP basados en el entendimiento y comprensión de los textos. Esto último es necesario tomarlo en cuenta para entender por qué temas como la generación de lenguaje natural (NLG) o generadores de flujo de conversación, no hicieron parte del desarrollo del proyecto y por el contrario hacen parte de los trabajos futuros.

3.2.1. ¿Qué es el NLP?

Para comprender que es NLP se debe entender primero las características del lenguaje. En términos generales el lenguaje se utiliza como un medio de comunicación que permite a un emisor enviar un mensaje que será interpretado por un receptor en un contexto dado. Se habla entonces de que para que haya una conversación fluida se debe contar con la capacidad de generar y entender mensajes. Esto cambia cuando se trata de una conversación humano-maquina, las maquinas son sistemas lógicos que requieren de una fuente medible y fácilmente replicable para la ejecución de un recurso con un fin.

En el caso del lenguaje, este comprende un numero significativos de variables que hacen compleja su medición. Trabajos como la comprensión de voz a texto, la traducción de máquina, la comprensión de intenciones, análisis de sentimiento y/o el reconocimiento de entidades son solo unos pocos de los ejemplos y retos en los que se trabaja alrededor del lenguaje y su interpretación [101]. Es así como podemos entender que, para dotar a una máquina de las capacidades de entender el leguaje de forma natural, esta requiere de una serie de técnicas que permitan formalizar el lenguaje, que sea fácilmente medible y replicable [59].

Es de este modo que cualquier tipo de técnica o manipulación sobre un contenido idiomático (sea visual, auditivo o de texto) que permita estructurar y formalizar el lenguaje con un fin se puede considerar como procesamiento de lenguaje natural (NLP) [59]. En este capítulo se hace una descripción de algunas de las técnicas más relevantes que se han usado en él. El procesamiento del lenguaje natural (NLP) es toda aquella técnica computacional motivadas por la teoría para el análisis automático de realizar operaciones y representación al lenguaje humano [59].

3.2.2. Objetivos del NLP [57, 58]

Al NLP ser cualquier tipo de técnica aplicada a formalizar y entender el lenguaje humano de forma automatizada, este tiene un campo muy amplio de aplicación y cada uno de estos se relacionan entre sí de un modo u otro, los principales objetivos de aplicación del NLP son:

3.2.2.1. Objetivos técnicos [57, 58]

- Detección de límites de oraciones: las abreviaturas y los títulos ("m.g.", "Dr.") complican esta tarea, al igual que los elementos de una lista o expresiones con plantilla.
- Tokenización: identificación de tokens individuales (palabra, puntuación) dentro de una oración. Un lexer juega un papel fundamental para esta tarea y la anterior. En el texto biomédico, los tokens suelen contener caracteres que se utilizan normalmente como límites del token, por ejemplo, guiones, barras diagonales.
- Asignación de parte del discurso a palabras individuales ("etiquetado POS"): en inglés, los homógrafos ("set") y gerundios (verbos terminados en "ing" que se utilizan como sustantivos) complican esta tarea.
- Descomposición morfológica de palabras compuestas: muchos términos médicos, por ejemplo, "nasogástrico", necesitan descomposición para comprenderlos. Una subtarea útil es la lematización: conversión de una palabra a raíz eliminando sufijos. El NLP clínico no ingles enfatiza la descomposición; en idiomas altamente sintéticos (p. ej., alemán, húngaro), las palabras compuestas recién acuñadas pueden reemplazar frases enteras. Las aplicaciones de revisión ortográfica y la preparación de texto para indexación / búsqueda (en IR) también emplean análisis morfológico.
- Análisis poco profundo (fragmentación): identificación de frases de tokens etiquetados de parte del discurso constituyente. Por ejemplo, una frase nominal puede comprender una secuencia de adjetivos seguida de un sustantivo.
- Segmentación por problemas específicos: segmentar el texto en grupos significativos, como secciones, incluida la queja principal, el historial médico anterior, HEENT, etc.

3.2.2.2. Objetivos de nivel superior [57, 58]

Las tareas de nivel superior se basan en tareas de nivel inferior y suelen ser específicas de un problema. Incluyen:

- **Identificación y recuperación de errores ortográficos / gramaticales:** esta tarea es principalmente interactiva porque, como saben los usuarios de procesamiento de texto, está lejos de ser perfecta. Las frases altamente sintéticas predisponen a falsos positivos (palabras correctas marcadas como errores), y los homófonos incorrectamente usados (palabras que suenan idénticamente, palabras de diferente ortografía, por ejemplo, único / alma, su / allí) a falsos negativos.
- **Reconocimiento de entidades nombradas:** identificar palabras o frases específicas ("entidades") y categorizarlas, por ejemplo, como personas, ubicaciones, enfermedades, genes o medicamentos. Una tarea común de NER (Named Entity Recognition) es mapear

entidades con nombre a conceptos en un vocabulario. Esta tarea a menudo aprovecha el análisis superficial de entidades candidatas (por ejemplo, la frase nominal "dolor en el pecho"); sin embargo, a veces el concepto se divide en varias frases (p. ej., "la pared torácica muestra una ligera sensibilidad a la presión ...").

- **Traducción de máquina:** identificar palabras o frases específicas ("entidades") para realizar traducciones de un idioma a otro.

3.2.3. Natural Language Understanding (NLU) [2]

La comprensión de lenguaje natural es una rama del NLP que se enfoca en **comprender la intención** detrás de una frase y que utiliza el aprendizaje activo para mejorar continuamente el entendimiento. Es muy normal ver el NLU relacionado con chatbots [2] , ya que los estudios que se realizan en este campo van dirigidos en dotar a la máquina con la capacidad de tomar un mensaje y poder entenderlo.

Una vez que el sistema ha conseguido entender la intención del cliente gracias al NLU, el NLP utiliza esta información para generar la respuesta adecuada. El NLU es por tanto el campo del NLP en el cual se implementa el Machine Learning para dar independencia a la máquina en la comprensión del lenguaje humano, ya que la gran diferencia con el NLP es que este último busca de forma general la formalización del lenguaje a través de la conversión del texto en datos estructurados.

3.2.4. Técnicas usadas en el NLP

Como parte de los objetivos del proyecto se dará una breve explicación de las técnicas más comúnmente usadas en el NLP. Se aclara que no se desea hacer una explicación exhaustiva de cada una de estas ya que el objetivo del proyecto está enmarcado en el manejo del NLP y principalmente en servicios de una arquitectura basada en eventos.

- **Redes Semánticas y Árboles sintácticos**

En su esfuerzo por formalizar cada vez más el lenguaje y conseguir mejores resultados los investigadores propusieron resolver el problema de la sintaxis haciendo uso de redes semánticas y árboles sintácticos. Estas soluciones a pesar de ser simples en su concepción son bastante poderosas ya que permiten resolver problemas sintácticos y a su vez el relacionar las palabras de forma adecuada [70]. La idea en el caso de una red semántica es tener un grafo de palabras, que este etiquetado y cada una de sus palabras se encuentra relacionada de alguna forma. Ejemplo de esto está en los sinónimos, que palabras se relacionan directa o indirectamente con otra:

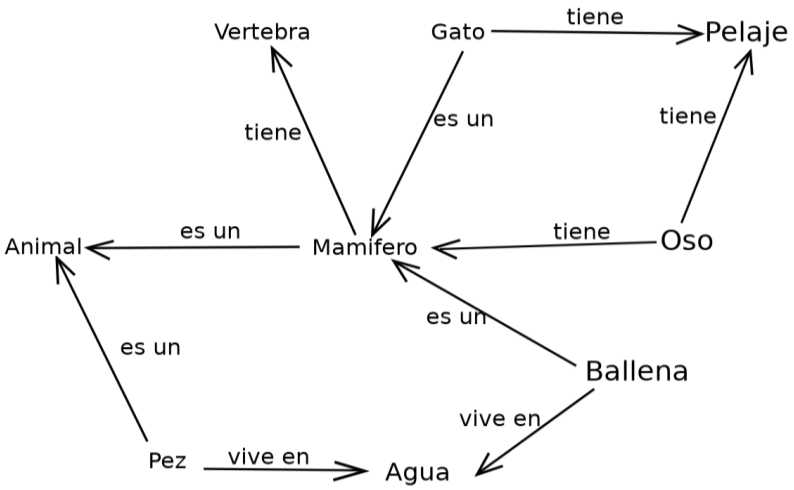

Figura 36: Ejemplo de red semántica.

Tomando como base la red semántica y el etiquetado de palabras/frases se pueden hacer análisis sobre frases que permitan descomponerla, esto con el objetivo de procesarla y entenderla. Haciendo uso de una oración preestablecida esta se descompone y se busca a través del etiquetado previo darle un valor y rol a cada una de las frases/palabras de esta forma se establecen los pronombres, los verbos y el rol que se cumplen en la oración. Uno de los temas más complejos de establecer en los árboles de sintaxis es el discernir a los sujetos ya que abreviaturas como “Dr.” son difíciles para descomponer en tokens.

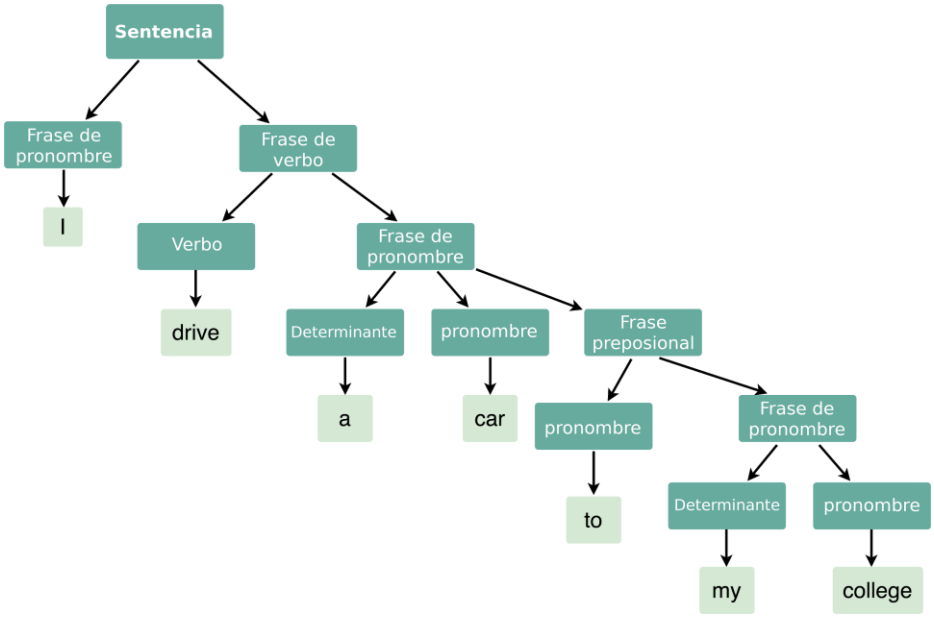

Figura 37: Ejemplo Árbol de sintaxis en Python [70].

- **Sistemas de estadística para traducción**

Como se vio en el capítulo de la historia del NLP este tuvo un estancamiento ya que los sistemas basados en semántica y sintaxis no permiten avanzar tanto como se deseaba, parte de esto es

que, aunque una palabra se pueda traducir como otra, esta puede ser vista en diferentes contextos que cambian su significado.

En el caso de la traducción de máquina, se buscaba de algún modo poder traducir entre idiomas de la forma más fielmente posible. Haciendo uso de estadística esto se consigue a través de la etiquetación basada en valores previamente ponderados en el cual se busca establecer correlaciones entre las palabras y cómo estas pueden tener un significado dependiendo de un contexto. Es así como, si una frase/palabra es usualmente traducida de un modo, ésta tendrá más posibilidades de ser traducida así.

Cabe aclarar que esto en sus inicios como se explica ayudó a poner nuevamente el interés en el NLP pero que tenía grandes limitantes ya que en sus inicios se buscaba traducir palabras basadas en el número de veces que esta se podría traducir siempre a otra. Es así como si la palabra "close" se traduce como "cerrado" era más probable que la traducción fuese "cerrado" en lugar de "cerca" que vendría siendo una opción válida dependiendo del contexto. Parte de estos problemas son correcciones y esto puede ser resuelto con la estadística para el NLP y los N-gramas.

- **N-gramas [57]**

Los N-gramas son en su concepto más simple una subsecuencia contigua de elementos [56]. Estos pueden ser tanto letras, palabras o frases. La parte más destacable de los N-gramas es que estos por si mismos solo permiten dividir las frases en conjuntos de N-valores contiguos, de este modo se puede tener 2-gramas que sería la cantidad de conjuntos de dos elementos contiguos en un universo(frases) o 3-gramas que seria los mismo, pero para 3 y de esto modo hasta el N-ésimo valor.

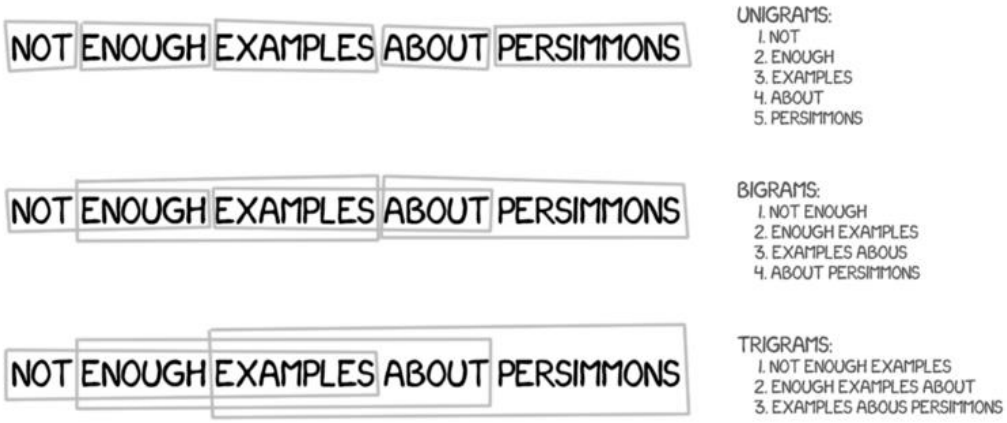

Figura 38: Visualización de N-gramas en una frase [57].

Los N-gramas permiten entender qué tan probable es que una palabra o letra está después de otra. Un ejemplo práctico de esto es tomar un libro y leer este en base a N-gramas para crear un algoritmo de calificación basado en la probabilidad de que una letra está junto a otra. Los N-gramas pueden ser usado en:

Autocompletado sugerido de palabras y frases para el usuario durante la búsqueda, como se ve en la propia interfaz de Google.

- **Corrección ortográfica:** una palabra mal escrita en una frase puede marcarse y sugerirse una ortografía correcta en función de las palabras vecinas correctamente escritas, como hace Google.
- **Reconocimiento del habla:** los homófonos ("dos" frente a "demasiado") se pueden desambiguar probabilísticamente basándose en palabras vecinas reconocidas correctamente.
- **Desambiguación de palabras:** si construimos N-gramas de "significado de palabras" a partir de un corpus anotado donde los homógrafos están etiquetados con sus significados correctos, podemos usar las palabras vecinas no ambiguas para adivinar el significado correcto de un homógrafo en un documento de prueba.

- **Algoritmos evolutivos [63]**

Los algoritmos evolutivos son aquellos que basan su comportamiento en los conceptos de población y como estos se generan a través de la mutación y la selección de una nueva generación. Estos algoritmos son parte de la introducción a los algoritmos de redes Neuronales ya que basan su corrección en el resultado de una función juez que permite saber el desempeño de cada nueva iteración. Es de este modo que cada nueva generación deberá ser igual o mejor que la pasada, permitiendo que se generen valores cada vez más acertados.

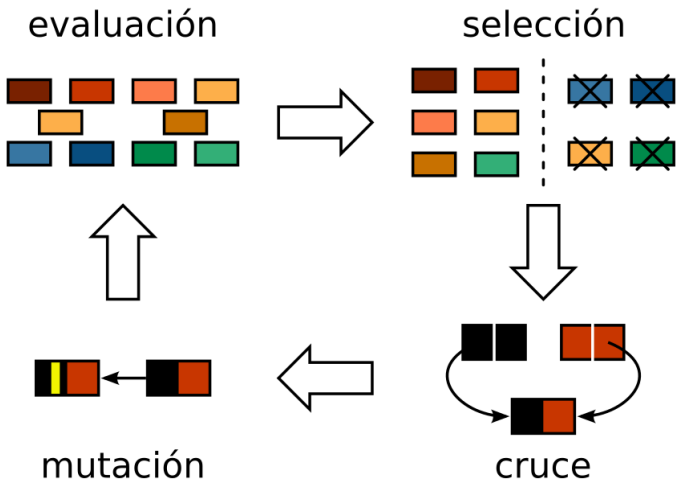

Figura 39: Funcionamiento de un algoritmo evolutivo [63].

En el NLP los algoritmos genéticos y evolutivos pueden ser usados para descifrar mensajes codificados. Es así como se puede generar una llave aleatoria para descifrar un mensaje y a través de mutaciones y evaluar la coherencia del mensaje cada generación tendrá un desempeño mejor hasta tener un resultado que permita encontrar la llave del mensaje original.

- **Modelo oculto de Markov [56, 58]**

Un Modelo oculto de Markov, es un modelo estadístico que basa su premisa en encontrar los valores ocultos de salidas observables conocidas. Podemos observar las salidas, pero los

componentes internos del sistema (es decir, las probabilidades de cambio de estado y las probabilidades de salida) están "ocultos".

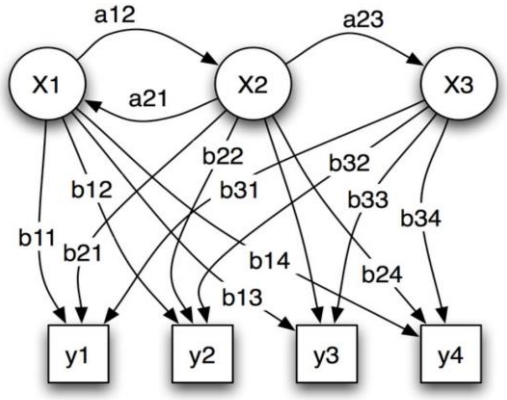

Figura 40: Representación de un modelo oculto de Márkov.

Los modelos ocultos de Márkov son especialmente aplicados en:

- Reconocimiento del habla
- Reconocimiento de escritura manual y gestos.
- Etiquetado gramatical.

Los modelos ocultos de Markov son muy usados especialmente en el reconocimiento de habla, muchos sistemas basados en estos modelos que se usan comercialmente ahora son lo suficientemente sólidos como para haber acabado con los esfuerzos de investigación académica. Esto se debe a que el reconocimiento de habla es un ejemplo de problemas que pueden ser resueltos usando estos modelos, donde tenemos una salida conocida con unas probabilidades (el sonido generado al hablar) y unos valores ocultos de transición que se desean conocer (las palabras).

- **Máquinas de soporte vectorial (SVMs) [56]**

Las máquinas de soporte vectorial son uno de los primeros pasos para la formalización del aprendizaje de máquina, estas tienen un enfoque de aprendizaje discriminativo, ya que su estrategia para el aprendizaje se basa en la clasificación de las entradas, entradas que pueden ser cualquier cosa como palabras, en categorías según un conjunto de características.

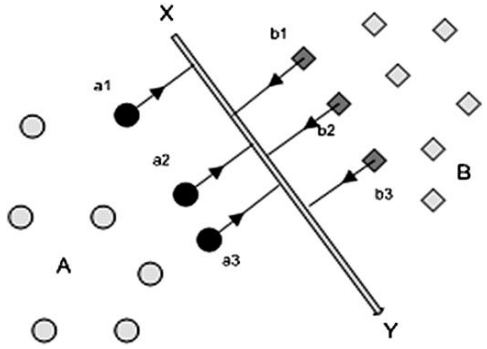

Figura 41: Muestra de un sistema simple vectorial para clasificación [56]

La entrada se puede transformar matemáticamente utilizando una "función de kernel" para permitir la separación lineal de los puntos de datos de diferentes categorías. Es decir, en el caso más simple de dos características, una línea recta los separaría en una gráfica X – Y. La función de kernel más utilizada es la gaussiana. El proceso de separación selecciona un subconjunto de los datos de entrenamiento (los "vectores de apoyo", puntos de datos más cercanos al hiperplano) que mejor diferencia las categorías.

- **Modelos basados en redes Neuronales y deep Learning [59]**

En la última década el desarrollo del NLP (y muchos otros campos) se ha visto volcado en dirección del uso de técnicas de Machine Learning para su mejoramiento, principalmente aquellas que hacen uso del deep Learning [59]. Las redes neuronales en su descripción más general se basan en una colección de unidades o nodos conectados llamados neuronas artificiales, que modelan libremente las neuronas en un cerebro biológico.

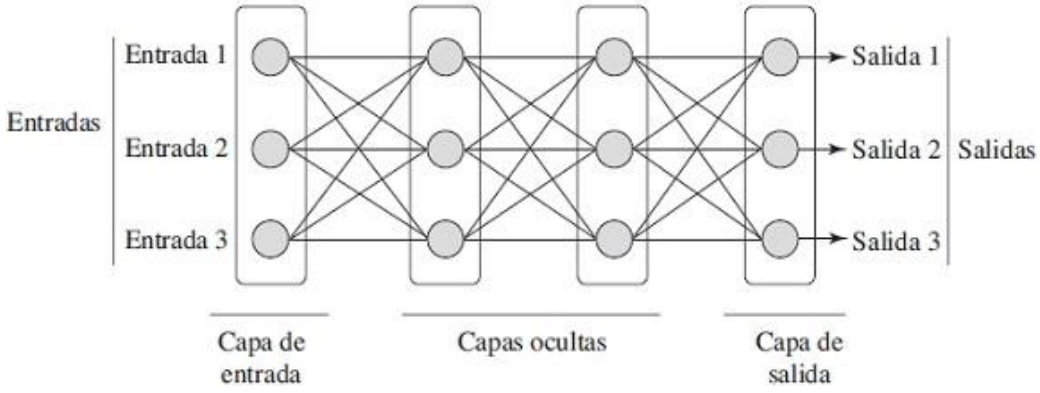

Figura 42: Explicación de una red neuronal.

Cada conexión, tiene una entrada y una salida puede transmitir un valor a otras neuronas, cambiando los pesos y valores resultantes a partir de las conexiones de esta. Normalmente, las neuronas se agrupan en capas. Las diferentes capas pueden realizar diferentes transformaciones en sus entradas. Las señales viajan desde la primera capa (la capa de entrada) hasta la última capa (la capa de salida), posiblemente después de atravesar las capas varias veces, esto último se implementa con el deep learning agregando capas ocultas.

En los sistemas de NLP basados en redes neuronales se pueden destacar especialmente los modelos de **word2vec**. Este busca dado el vocabulario generado con las palabras de un conjunto de frases (también llamado **corpus**), se entrena a la red neuronal con las sentencias del corpus para que, para una palabra dada, nos diga la probabilidad de que cada palabra del vocabulario sea vecina de la primera. Esto genera que nuevas palabras que se analicen a partir del entrenamiento previo pueden ser predichas de forma más precisa.

- **Redes Neuronales Recurrentes (RNN) [57, 59]**

Las RNN representan una forma natural de modelar secuencias, se toma el concepto de red neuronal y se adicionan nuevas redes que permiten tomar el resultado de la salida de una red

previa. En el caso del NLP las RNN constituyen un conjunto de técnicas que permiten obtener resultados muy buenos debido a la naturaleza del lenguaje en el cual cada palabra depende de su entorno.

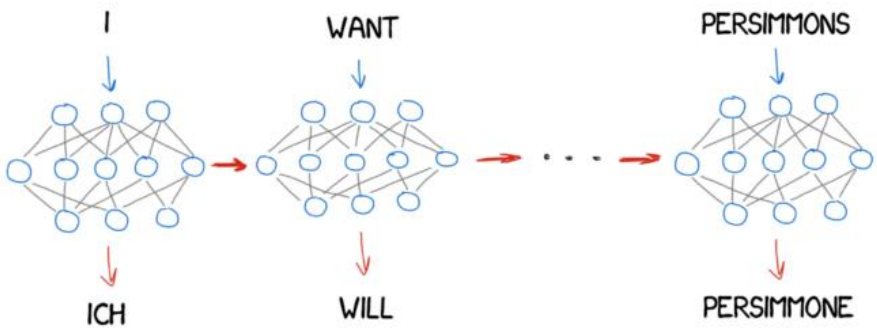

Figura 43: Redes neuronales recurrentes en la traducción [57].

Las redes neuronales recurrentes (RNN) serían la mejor opción para manejarlas, ya que recuerdan el resultado anterior, en el caso del NLP las palabras y sub-frases se combinan en frases de manera jerárquica. Dicha estructura se puede representar mediante un árbol de análisis de constituyentes. Específicamente, en una red neuronal recursiva, la representación de cada nodo no termina en un árbol de análisis está determinada por las representaciones de todos sus hijos.

3.2.5. Arquitecturas basadas en NLP

Actualmente el NLP y especialmente el NLU lidian con algunos problemas, en este caso se orientarán estos problemas en el contexto de lo chatbots ya que es el tema de interés de este proyecto [66]:

- **Estructura del idioma:** cada idioma tiene una estructura diferente para la elaboración de oraciones. Por ejemplo, la estructura de los textos, la puntuación y el uso de espacios difieren entre idiomas. Los chatbots existentes no pueden diferenciarlo.
- **Semántica**: La semántica es el significado de oraciones o palabras en forma de lenguaje natural humano. Los chatbots tradicionales basados en reglas no se enfrentan al procesamiento del lenguaje natural, ya sea para crear una respuesta o analizar preguntas.
- **Análisis de sentimientos:** los chatbots existentes no son capaces de detectar la emoción del sujeto del que habla el ser humano. El chatbot debe poder distinguir por la forma en que se presenta el texto o el patrón de voz si el humano está enojado, triste o feliz. Simplemente recopilan información y brindan respuestas a partir de la base de conocimientos.
- **Precisión:** los chatbots están programados para tener una conversación como humana para realizar una tarea. Sin embargo, los chatbots existentes tienen una mala tendencia a cambiar repentinamente de tema y generar respuestas impredecibles. A veces responde sin contexto. Por tanto, la precisión no alcanza un nivel satisfactorio.

- **Autoaprendizaje:** los chatbots anteriores no utilizan enfoques de aprendizaje automático supervisados. Son incapaces de aprender nuevos patrones de habla o palabras, descubren el contexto a través de la interacción y el razonamiento lógico. La mayoría de ellos no pueden entrenar a un clasificador para mapear de oraciones a intenciones y modelo de secuencia a filtros de ranuras.
- **Admite la integración de terceros:** La mayoría del software en NLP solo admiten el idioma inglés. También es difícil de incrustar en una página web debido al flujo de proceso crítico para la integración web.
- **Procesamiento de datos:** Procesar datos de estructura directamente y el uso de bases de datos relacionales.

El procesamiento del lenguaje natural considera relaciones lineales entre palabras; en casos simples de comprensión, podemos diseñar extractores de características y seleccionar las entidades para que coincidan con la intención dada. La parte compleja de la extracción tediosa de características de gran volumen implica redes neuronales y, a menudo, es más precisa que la extracción de características simples. El uso de redes neuronales de varias capas para el aprendizaje automático se denomina aprendizaje profundo.

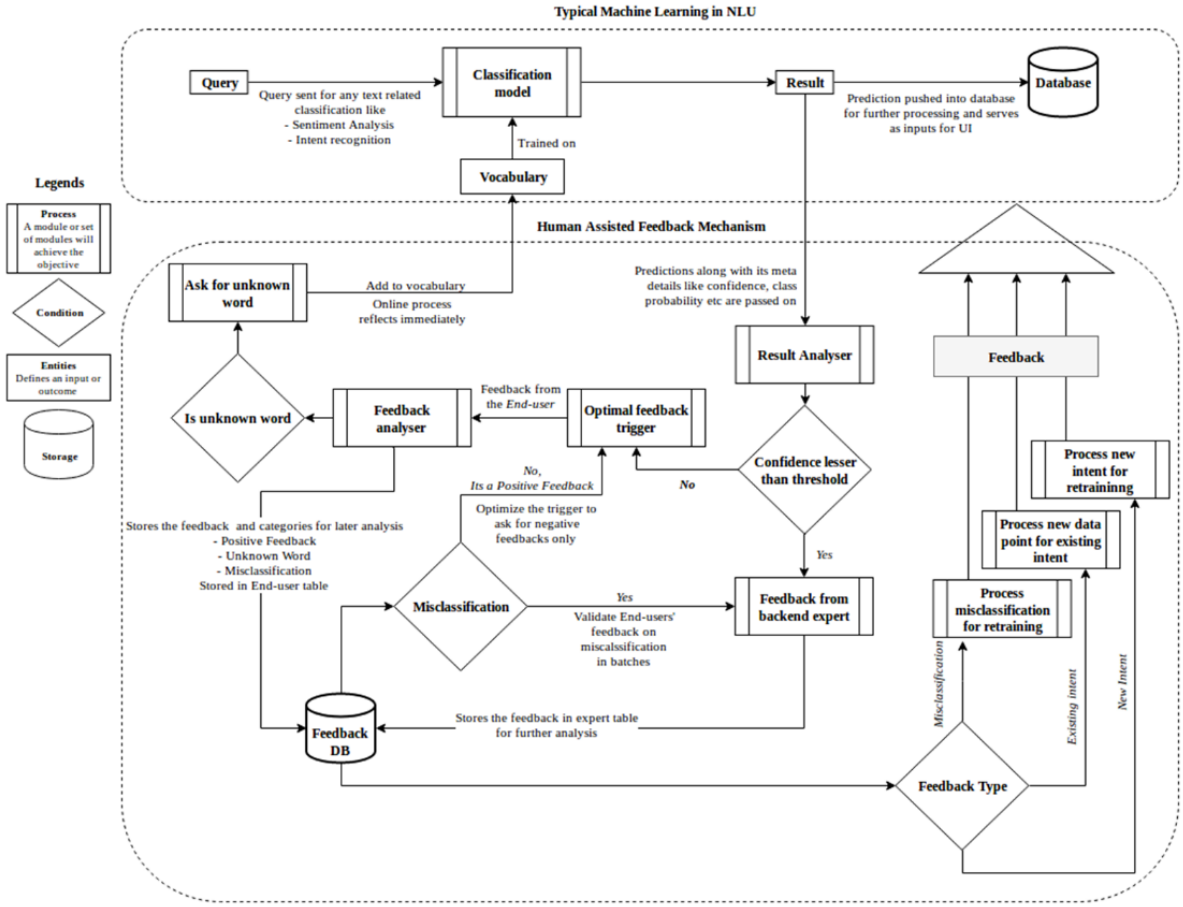

Figura 44: Ejemplo de framework para el uso de NLP con retroalimentación de usuarios [66].

La colección de algoritmos implementados bajo aprendizaje profundo tiene grandes similitudes con la relación entre estímulos y neuronas en el cerebro humano. Un sistema de NLP tiene un

léxico (vocabulario) y un conjunto de reglas gramaticales codificadas en el sistema. El procesamiento del lenguaje natural utiliza máquinas para ejecutar algoritmos de aprendizaje profundo para comprender las comunicaciones de texto de los usuarios y responder de manera inteligente a ellas en función de las configuraciones de flujo. Las aplicaciones de Respuesta Inteligente se crean con estos sistemas de PNL para automatizar las comunicaciones del Agente y sustituir a los seres humanos, manejando así los costos asociados con la escalabilidad del Agente.

De igual forma modelar una arquitectura debe entender la clasificación que el NLP tiene en términos de objetivos técnicos y de negocios, así el NLP puede verse de la siguiente forma [67]:

- NLP, refiriéndose a todo el sistema de estructuración y formalización del lenguaje.
- NLU, que como se ha explicado previamente busca que la máquina comprenda lo que se le está diciendo.
- NLG (Natural Language Generation), se enfoca en permitir a la maquina generar palabras de forma coherente.

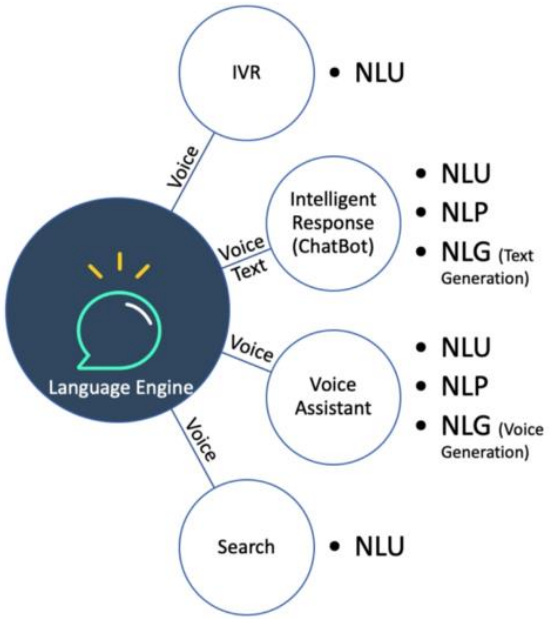

Figura 45: Clasificación de sistemas basados en NLP [67].

En el presente proyecto se busca dotar de entendimiento al chatbot por ende el NLG no es una característica esencial para el desarrollo de este. Es necesario por ende explicar cómo el NLP es usado a nivel de implementación de servicios y como este se relaciona con los componentes del software.

NLTK, Intel AI, SPacy, Allen NLP

Parte de las necesidades que surgen al trabajar con NLP se debe en gran medida a la acelerada evolución de este y sus técnicas, en este sentido muchas herramientas y sistemas basados en NLP han surgido. Muchas de estas herramientas y sistemas usan conceptos y técnicas muy

similares para el manejo de NLP con redes semánticas y generación de relaciones de palabras [69].

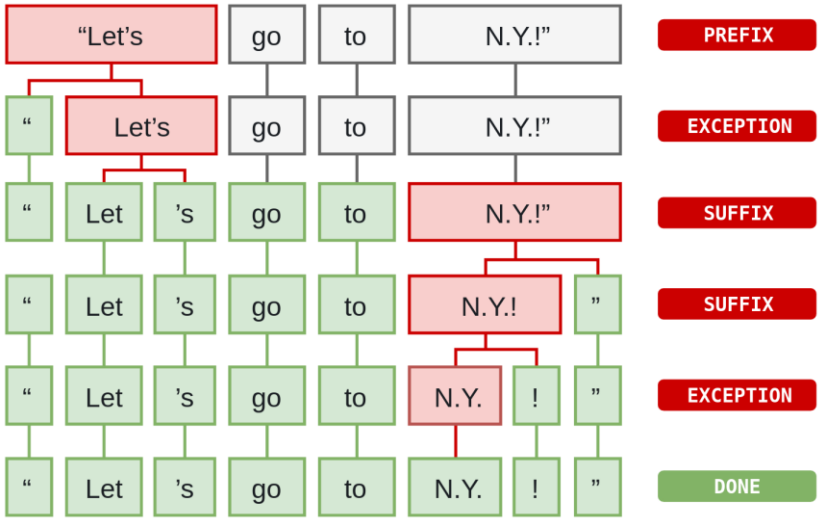

Figura 46: Sistema de etiquetación de una frase [69].

En el caso del uso de toolkits y sistemas en general como módulos de procesamiento de lenguaje natural estos deben permitir la clasificación y tokenización de una frase de forma adecuada. Los módulos de NLP deben permitir por ende formalizar la frase de forma que podamos identificar información relevante para posteriormente usar esta información en un análisis más profundo empleando técnicas de NLU para análisis de sentimientos o intenciones del usuario:

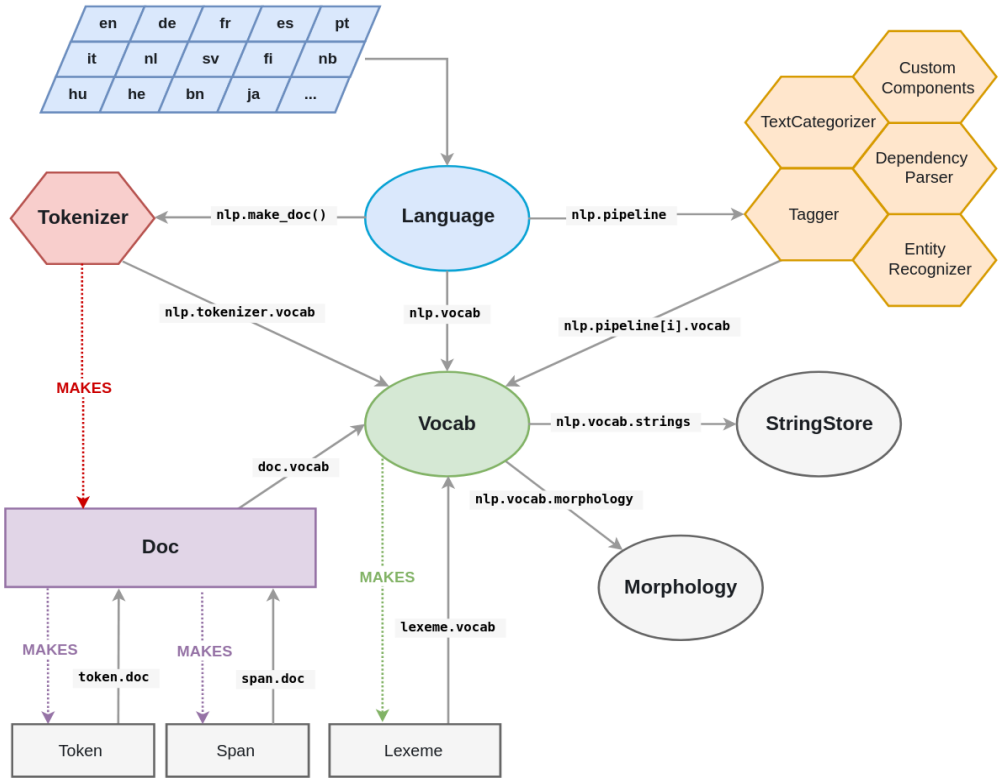

Figura 47: arquitectura propuesta por spacy [69].

En los sistemas de NLP se busca formalizar y estructurar de forma lógica las relaciones y roles de cada palabras y frase que se compone en un texto. Es así como se deben identificar las responsabilidades de una palabra y como esta es usada en el contexto de una frase, en las siguientes imágenes se pueden ver unos ejemplos de cómo las frases son tratadas y visualizadas por los diferentes softwares de NLP [68].

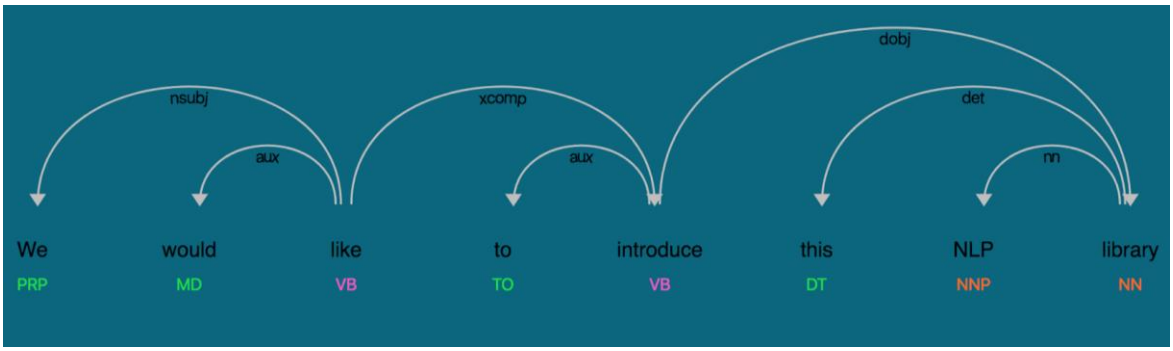

Figura 48: Estructuración de una frase por Intel AI [68].

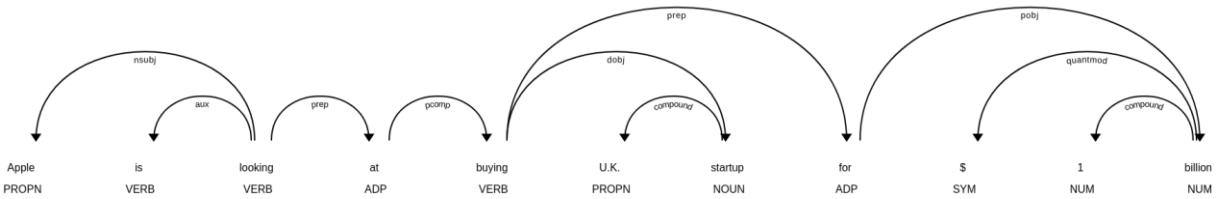

Figura 49: Estructuración de una frase por Spacy [69].

La velocidad en el desarrollo de herramientas ha generado que gran parte de la implementación del NLP en las arquitecturas modernas está enfocada en la extensión de toolkits y servicios agnósticos, motores de NLP son la nueva clase de software en el que todos los principales jugadores de la nube (Google, Microsoft, Amazon) han estado invirtiendo fuertemente para construir un sistema de eco alrededor de los motores de procesamiento de lenguaje.

Algo que se puede destacar en la forma en la que el NLP se ha implementado en los últimos años es que este es usado de igual forma por casi todos los proveedores actuales, los proveedores de la nube tienen los mismos algoritmos que utilizan tecnologías de deep learning para el aprendizaje automático [67]. Se puede por ende restar importancia a preocupaciones como el bloqueo de proveedores para sistemas de NLP, ya que no hay nada exclusivo y patentado en el algoritmo entre varios proveedores sobre la implementación de "redes neuronales de aprendizaje profundo" para el manejo y procesamiento de idiomas.

Cuando se habla de servicios para NLP, el único diferenciador es el Lexicón (Cantidad de datos de entrenamiento que está disponible con el proveedor y el vocabulario configurado en sus sistemas en varios idiomas). Las preguntas que se puedan resolver se harán a través de un conjunto preconfigurado. El Lexicón juega un papel más importante al implementar algo como la búsqueda por voz como una funcionalidad en nuestros canales. Mientras que en el caso concreto de este proyecto es más importante la identificación de intención, para la cual los proveedores generan resultados similares [68].

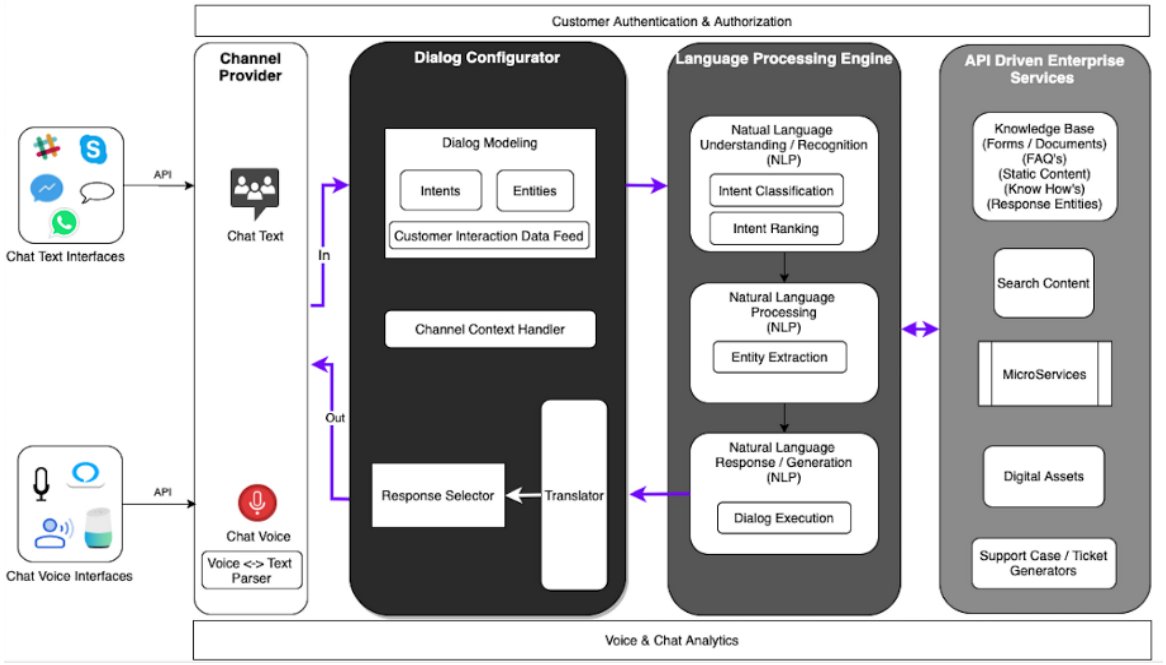

Figura 50: Arquitectura de ejemplo de NLP en un sistema de chat [67].

Ya que los motores y servicios para el uso de NLP son relevantes y necesarios para el desarrollo de un chatbot, se hace necesario dar una breve explicación de estos. Sus características y ventajas. Qué beneficios aportan y de qué modo estos pueden ser integrados en la arquitectura de un chatbot de tal forma que permita la identificación de intenciones.

3.2.6. Frameworks y plataformas NLP como servicio para el desarrollo de chatbots [6, 65]

En el mundo del desarrollo de chatbots es importante hacer una diferenciación entre dos tipos de herramientas clave entre las que se encuentran los "**Bot Frameworks"** y los "**Bot Platforms",** a los cuales se les seguirá haciendo referencia por su terminología en inglés.

Los **Bot Frameworks** son las plataformas enfocadas a la creación y alojamiento de bots, de los cuales encontramos una gran variedad dependiendo del nivel de complejidad, funcionalidad y capacidad de integración.

Por otro lado, los **Bot Platforms** son los entornos y aplicaciones donde los chatbots pueden ser desplegados, cómo es el caso de plataformas de chat externas cómo Slack, Google Chat, Microsoft Teams, Messenger, Whatsapp, widgets de chat en páginas web y entre otros.

3.2.6.1. Bot Frameworks

Dentro de la familia de Bot Frameworks se pueden encontrar plataformas que se agrupan en tres tipos: Visuales, conversacionales y programables.

3.2.6.2. Plataformas Visuales

Son plataformas donde se pueden construir chatbots sin conocimientos técnicos de programación. Son intuitivos, fáciles de aprender y permiten construir chatbots sencillos orientados a resolver comandos o peticiones concretas. Por otro lado, son bastante limitados ya que no permiten integración con APIs externas, no se pueden crear conversaciones complejas basadas en condicionales y solo se puede desarrollar orientado a plataformas de chat externas específicas que estén soportadas.

En esta categoría encajan plataformas cómo Landbot.io, Gorgias.com, Chatfuel, ManyChat, Octane.ai, Motion.ai, TARS, Landbot.io, BotMyWork y entre otras.

3.2.6.3. Plataformas Conversacionales

Permiten desarrollar chatbots capaces de mantener una conversación sobre temas triviales sin que tenga que existir una meta u objetivo específico. Son ideales para construir chatbots orientados a la publicidad, entretenimiento, e-learning o educación.

Estas plataformas usan lenguajes de marcado tipo XML para construir modelos de interacción dentro de los que se encuentra AIML (Artificial Intelligence Markup Language), en conjunto con servicios cognitivos de machine learning y NLP.

Dentro de este grupo destacan plataformas cómo Pandorabots, Botsify, Rebot.me, MobileMokey, Chatscript y entre otros.

3.2.6.4. Plataformas Programables

Son plataformas avanzadas para el desarrollo de chatbots complejos y personalizados en términos de programación, entendimiento y acciones que pueden realizar o procesar.

Son flexibles al permitir la integración con diferentes lenguajes de programación, servicios cognitivos de machine learning y NLP, bases de datos, servicios de infraestructura en la nube, APIs, bases de datos de conocimientos externos, etc.

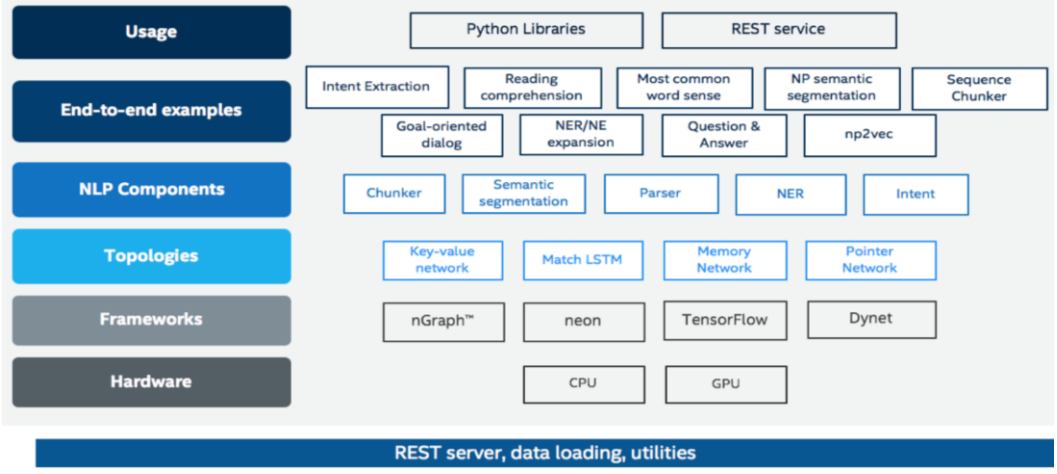

Figura 51: Arquitectura de NLP de intel AI lab [68].

Un detalle relevante a tener en cuenta es que algunas plataformas programables se pueden considerar en sí mismas cómo servicios cognitivos, ya que poseen componentes que exponen vía API herramientas de machine learning y NLP para el reconocimiento de intenciones y entidades, análisis de sentimientos y entre otras funciones avanzadas.

A continuación, se presentan algunas de las plataformas programables más relevantes en la actualidad:

3.2.6.4.1. DialogFlow

Antes conocido cómo api.ai, fue adquirido por Google en el 2016 y provee a los desarrolladores una plataforma que permite diseñar e implementar interfaces conversacionales que pueden ser embebidas en aplicaciones externas. La intención y el contexto son conceptos clave para modelar el comportamiento de un chatbot en DialogFlow. Las intenciones crean un enlace entre lo que el usuario dice y la acción que debe realizar el chatbot y posteriormente retornar una respuesta. Los contextos son registros usados para diferenciar cada petición, la cual puede variar su significado incluso si la intención es la misma dependiendo del contexto.

Lenguajes soportados: Node.js, .NET, C++, Python, Ruby, PHP, Java, Android, Xamarin e iOs.

Plataformas soportadas: Skype, Telegram, Slack, Cortana, Alexa y Facebook Messenger.

Ventajas:

- Provee un potente motor para modelar largos y complejos flujos conversacionales usando conceptos clave cómo lo son agente, intención, contexto y acción.
- Ofrece un servicio que permite entrenar chatbots de forma constante con mucha información de oraciones e historias para mejorar la detección de intenciones.
- Se puede integrar con aplicaciones externas usando webhooks, lo cual permite desarrollar widgets de chatbots de manera muy sencilla.

Limitaciones:

- La configuración y manejo de intenciones y contextos al mismo tiempo es una tarea compleja.

3.2.6.4.2. Facebook Bot Engine

El ecosistema de Facebook ofrece herramientas que permiten desarrollar bots de manera sencilla usando Wit.ai como servicio cognitivo, lo cual permite crear chatbots para Facebook Messenger con poco esfuerzo y gran capacidad para mantener una conversación.

Plataformas soportadas: Facebook Messenger.

Ventajas:

- Gracias a wit.ai la detección de intenciones es muy potente, lo cual permite relacionar muy bien la acción o respuesta del chatbot.

- Permite controlar el flujo de la conversación usando condicionales sobre las acciones, lo cual permite modelar fácilmente un árbol de decisión.
- Ofrece una interfaz visual que permite crear los flujos conversacionales de manera sencilla.

Desventajas:

- A pesar del poderoso servicio cognitivo y uso condicionales, es difícil controlar el flujo conversacional, ya que se suele malentender algunas intenciones debido al contexto.

3.2.6.4.3. Microsoft Bot Framework

Es una de las mejores plataformas disponibles para el desarrollo de chatbots, el cual integra de manera nativa servicios de machine learning. El entendimiento del lenguaje e intenciones es muy similar al de DialogFlow, se evalúa la semántica del mensaje enviado por el usuario y se detecta la intención y entidades de la conversación.

Lenguajes soportados: Soporta .NET, NodeJs y C#.

Plataformas soportadas: Microsoft Teams, Skype, Office 365 mail y muchos otros servicios vía API.

Ventajas:

- Sus servicios cognitivos están entre los más avanzados del mercado.
- Su servicio de entendimiento de lenguaje natural ofrece soporte para detección y evaluación de entidades compuestas.

Desventajas:

- No permite a los desarrolladores manejar parámetros de contexto.
- No ofrece una amplia gama de integración con otras plataformas de chat, cómo si las ofrece DialogFlow.

3.2.6.4.4. Botpress.io

Framework de desarrollo fácil de usar y de aprender, dispone un dashboard que permite crear flujos conversacionales mediante una interfaz visual, ofrece una versión open source bajo licencia GPL, pero requiere una versión de pago para acceder a todas sus capacidades.

Lenguajes soportados: NodeJs

Plataformas soportadas: Facebook Messenger, Slack, Microsoft Teams, Skype, WeChat y Telegram.

Ventajas:

- Permite crear flujos conversacionales con condicionales mediante un dashboard con interfaz gráfica.

Desventajas:

- Requiere licencia para desarrollos comerciales.

3.2.6.4.5. Botkit

Es un framework de desarrollo de código abierto, fácil de aprender y con una gran comunidad de desarrolladores.

Lenguajes soportados: NodeJs.

Plataformas soportadas: Facebook Messenger, Slack, Twilio, Facebook For Work, Microsoft Teams, Cisco Spark y Web Chat.

Ventajas:

- Ofrece kits de desarrollo para cada plataforma externa muy fáciles de usar y bien documentados.

Desventajas:

- A pesar de ofrecer una amplia gama de soporte multiplataformas, muchas de sus capacidades son limitadas.
- Puede ser un dolor de cabeza el soporte y mantenimiento, cada que una de las plataformas externas se actualiza.

3.2.6.4.6. Amazon Lex

Es un framework versátil que por sí mismo se puede considerar cómo un servicio cognitivo de NLP y machine learning, el cual es compatible con asistentes de voz y texto, ya que usa el mismo motor de conversación de Alexa, dado que ofrece un servicio de traducción de voz a texto y también de texto a voz.

Lenguajes soportados: Node.js, Python, .NET, Android, iOS, Java, Javascript, PHP y Ruby.

Plataformas soportadas: Amazon Alexa.

Ventajas:

- Utiliza conceptos clave cómo intención, locución, contexto y acciones para construir bots muy completos.
- Provee un servicio cognitivo avanzado para reconocimiento y detección de intenciones.
- Se pueden crear bots de plataformas externas consumiendo los servicios de Amazon Lex vía API.
- No está limitado solo a texto, también se puede trabajar con voz tanto en la entrada cómo la salida.

- Se puede integrar fácilmente con otros servicios de Amazon cómo DynamoDB, RDS, Amazon Cognito, AWS Lambda, entre otros.
- Ofrece una lista larga de entidades preconstruidas.

Desventajas:

- Dado que es un servicio que ofrece muchas capacidades requiere mucho conocimiento y tiempo para desarrollar un chatbot.

3.2.6.4.7. IBM Watson

Ofrece un servicio conversacional que permite desarrollar chatbots de manera sencilla mediante una interfaz web sin necesidad de programar, pero a la vez también los desarrolladores pueden codificar flujos complejos e integrar de manera versátil otros servicios cognitivos.

El NLP de IBM está basado en conceptos claves como son el uso de intenciones, entidades y sinónimos, con una interesante capacidad llamada "Fuzzy Matching", la cual permite reconocer palabras mal escritas o parecidas. Por ejemplo, si en el mensaje recibido se esperaba la palabra pizza y se recibe piza, el sistema podrá emparejar una coincidencia.

Lenguajes soportados: Node.js, Python, .Net, Android, iOS.

Plataformas soportadas: Slack, Facebook Messenger, Twillo, chatbots en la web y entre otros.

Ventajas:

- Permite monitorizar fácilmente el uso del chatbot mediante herramientas de analítica e integración son otras APIs de IBM Watson, bases de datos y entre otros.
- Gracias a los sinónimos se pueden construir entidades relacionadas, similar a cómo lo hace Amazon Alexa o DialogFlow. Por ejemplo, las palabras arroz, huevo, carne, se pueden relacionar mediante una entidad llamada "comida".

Desventajas:

- Puede ser un poco costoso.

3.2.6.4.8. Microsoft LUIS

Lanzado en 2015, es el servicio de reconocimiento de intenciones y entidades usado por Microsoft Bot Framework, el cual expone una poderosa API para crear chatbots propios de manera versátil desde cualquier herramienta de programación.

Lenguajes soportados: Node.js, Python, .NET y entre otros.

Plataformas soportadas: Microsoft Teams.

Ventajas:

- Se encuentra entre las herramientas más avanzadas en cuanto a servicios de NLP y machine learning en conjunto con IBM Watson.
- El servicio de LUIS está en constante aprendizaje de refuerzo, mejorando la calidad de los modelos de procesamiento de lenguaje natural.
- Se integra fácilmente con los servicios de Microsoft Azure.

Desventajas:

- Puede ser un poco costoso.

3.2.6.4.9. Wit.ai

Servicio cognitivo adquirido por Facebook a principios del 2015 para el desarrollo de chatbots, ofreciendo vía API una interfaz intermediaria para el reconocimiento de intenciones y entidades.

Lenguajes soportados: Node.js, Python y entre otros.

Plataformas soportadas: Facebook Messenger

Ventajas:

- Ofrece una interfaz web para configurar fácilmente intenciones y entidades que se quieran reconocer.
- También se puede usar para reconocimiento de voz.
- Admite más de 50 idiomas, incluido el español.
- Se puede importar y exportar intenciones y entidades en formato JSON.

Desventajas:

- No ofrece intenciones preconstruidas.
- No ofrece integración con plataformas externas cómo Google Home, Alexa y entre otros.

3.2.6.4.10. Aspect Customer Experience Platform (Aspect CXP)

El diseño y desarrollo de chatbots se hace a través de una herramienta propietaria llamada Aspect CXP Designer, la cual es un híbrido entre una interfaz gráfica y un lenguaje de programación, y para el NLP ofrece un componente llamado Aspect NLU.

Plataformas soportadas: SMS, voz, web, Skype y redes sociales cómo Facebook y Twitter.

Ventajas:

- Permite integración sencilla con servicios terceros, cómo APIs, bases de datos, servicios cognitivos y entre otros.

Desventajas:

- Con Aspect NLU se pueden hacer implementaciones robustas**,** pero son **menos flexibles** y **más complejas** de construir que otros competidores.
- La documentación es bastante pobre, no se encuentran fácilmente ejemplos para el desarrollo de chatbots y su código no es accesible.

3.2.6.4.11. Gupshup

Ofrece un kit de desarrollo con dos herramientas: Flow Bot Builder y Bot Builder IDE. Flow Bot Builder ofrece un editor virtual orientado a usuarios sin conocimientos técnicos en programación, permitiendo la creación y despliegue de chatbots sencillos.

Bot Builder IDE es una herramienta de desarrollo avanzado para la construcción de chatbots complejos.

Lenguajes soportados: Node.js, Python, Ruby, PHP, Java, C#, Android y iOS.

Plataformas soportadas: Facebook Messenger, SMS, Twitter, Telegram, Slack, Hipchat, Skype, Kik, Twillio, Line, Cisco Spark, Teamchat, etc

Ventajas:

- Incluye herramientas de analítica y monitoreo.
- Los servicios de NLP son sencillos y fáciles de implementar.
- Se puede integrar con otros motores de NLP más completos cómo DialogFlow y Wit.ai.

Desventajas:

- Si bien su servicio de NLP es fácil de usar e implementar, no es tan avanzado.

3.2.6.4.12. Ask SDK [74]

Ask SDK es el kit de desarrollo oficial de Amazon para su asistente virtual Alexa. Ofrece una manera fácil e intuitiva de desarrollar aplicaciones dentro del asistente llamadas “Skills”. Una skill no es más que la capacidad de implementar sobre Alexa un conjunto de instrucciones y comandos de voz que Alexa puede procesar y darnos una respuesta.

En Alexa el reconocimiento de intenciones se define mediante un diccionario de intenciones, entidades y frases sinónimas, las cuales se pueden configurar mediante una interfaz visual o vía código se puede desplegar usando Ask SDK.

Lenguajes soportados: Node.js, Java, Python.

Plataformas soportadas: Amazon Alexa.

Ventajas:

- Incluye herramientas de analítica y monitoreo.
- Soporte multilenguaje de intenciones y entidades.
- Su motor conversacional es intuitivo y fácil de implementar, se pueden crear aplicaciones que reconozcan comando sencillo en un par de minutos.
- Alexa se encarga del reconocimiento de entidades y de la conversión de voz a texto y de texto a voz.

Desventajas:

- Alexa no entrega al desarrollador todo el texto procesado, solo las intenciones reconocidas, y entidades, lo cual hace más complejo poder procesar comandos avanzados basados en el mensaje recibido.

4. CAPÍTULO V: DESARROLLO METODOLÓGICO

FASE 1 - ANÁLISIS (QAW-A)

Como etapa inicial según lo establecido por la metodología SEI-A, la cual se usará cómo base para el diseño de la arquitectura que se espera cómo resultado de este proyecto, se presenta el desarrollo de la fase de análisis que permitirá llevar a cabo un levantamiento y definición de objetivos, requerimientos, restricciones, motivaciones, intereses y aspectos de calidad de la arquitectura de un sistema chatbot. Esta arquitectura deberá satisfacer las necesidades de un agente conversacional mínimo viable con capacidades de interacción mediante el consumo de servicios cognitivos y acceso a un conjunto de recursos a través de la interacción a una base de conocimientos interna o externa vía API.

En las secciones posteriores se definirá un caso de estudio enfocado en presentar un diseño de arquitectura genérico**,** que permita ofrecer un amplio rango de maniobra al momento de satisfacer los casos de uso clave y necesidades en tendencia de los chatbot analizados en la sección **2.3.1**.

4.1. Presentación de “LaraBot” cómo caso del estudio

Se establece como caso de estudio el desarrollo de un asistente conversacional basado en texto llamado **LaraBot**, que estará inspirado en la arquitectura de chatbot presentada en la referencia [41] y en el motor conversacional del asistente de Amazon Alexa; el chatbot podrá interactuar con un usuario y ser capaz de responder a oraciones básicas cómo ¿qué hora es?, ¿qué día es hoy?, ¿quién eres?, ¿cómo estás? o un simple saludo.

Figura 52: Logo de LaraBot.

Se plantea además que el chatbot tenga un conjunto de capacidades, referenciadas de ahora en adelante cómo “habilidades” o “acciones”, las cuales permitirán interactuar con este de forma más amena. Una habilidad se entiende cómo la capacidad de dar al chatbot una orden para acceder a un dato, recurso o ejecutar un proceso.

La arquitectura de LaraBot permitirá extender las habilidades al desarrollador de manera flexible, pero cómo caso de estudio se limitarán las habilidades disponibles a las 6 listadas a continuación:

- **Noticias:** Se conectará a una API de un servicio de noticias para listar las más importantes del día actual.
- **Chistes:** Retorna un chiste aleatorio.

- **Gifs:** Genera un gif aleatorio basado en un mensaje.
- ***Timezones*:** Permitirá saber la hora basada en una zona horaria o nombre de país proporcionado.
- **Películas:** Devolverá una recomendación de películas basada en un género o autor.
- **Música:** Genera una recomendación de canciones basada en un género o autor.
- **Imágenes:** Retorna una imagen basada en un texto dado.

4.2. Objetivos de negocio

A continuación, se presentan los intereses y motivaciones principales de los involucrados del sistema, entre los que se encuentran tanto desarrolladores cómo usuarios finales.

- **OBJN-01:** Brindar una respuesta al usuario en un tiempo oportuno.
- **OBJN-02:** Presentarse como un chatbot cordial ante el usuario, mediante recursos como un saludo o una despedida.
- **OBJN-03:** Proporcionar un conjunto mínimo de atributos configurables en el *bot*, como acciones permitidas o valores predeterminados.
- **OBJN-04:** Entender la intención del mensaje enviado e intentar siempre dar una respuesta concisa y precisa al usuario incluso en caso de error o falta de entendimiento.
- **OBJN-05:** Disponer de un comando de ayuda que permita al chatbot explicar sus funciones, limitaciones y capacidades; lo que brinda al usuario una idea clara del alcance que posee el chatbot a la hora de brindar ayuda.
- **OBJN-06:** Si la intención del usuario no se entiende ofrecer comandos de ayuda o recomendaciones sobre qué puede hacer el chatbot.
- **OBJN-07:** Se debe poder mantener una conversación muy simple que tenga sentido sobre temas triviales respondiendo a preguntas simples cómo ¿qué hora es?, ¿qué día es hoy?, ¿quién eres?, ¿cómo estás?
- **OBJN-08:** Ofrecer la capacidad de dar retroalimentación a los desarrolladores vía chatbot que permita canalizar quejas, reclamos o sugerencias.
- **OBJN-09:** Recopilar analíticas de uso del chatbot sobre los comandos, preguntas, palabras, oraciones o errores más comunes capturados a partir de mensajes enviados por los usuarios.
- **OBJN-10:** El chatbot debe poder hablar en inglés y español.
- **OBJN-11:** El chatbot debe poder recibir mensajes desde diferentes canales de comunicación cómo Slack, Microsoft Teams, Google Chat o Chat de voz desde una petición HTTP vía terminal.
- **OBJN-12:** El chatbot debe permitir integrar nuevos canales de comunicación de manera flexible.
- **OBJN-13:** Se debe proveer una página de aterrizaje[9] que permita a los usuarios descubrir a LaraBot vía web y conocer sus bondades.

[9] Término conocido como “Landing Page” en inglés.

- **OBJN-14:** La comunicación con el chatbot, los servicios e interfaces web deben usar encriptación segura HTTPS.
- **OBJN-15:** Ofrecer al cliente la capacidad de autenticarse con plataformas externas mediante una interfaz web si es requerido.
- **OBJN-16:** El desarrollador debe poder agregar nuevas habilidades al sistema de manera flexible.
- **OBJN-17:** El chatbot debe poder almacenar contextos de las conversaciones con los usuarios en alguna base de datos.
- **OBJN-18:** El sistema debe tener la capacidad de enviar una notificación global a todos los usuarios sobre novedades o actualizaciones.
- **OBJN-19:** El sistema debe poder registrar errores que entreguen una trazabilidad al desarrollador sobre excepciones y su contexto.
- **OBJN-20:** El sistema debe integrar herramientas que permitan la integración y despliegue continuo.
- **OBJN-21:** Se deben utilizar tecnologías que permitan la menor cantidad de acoplamiento posible con proveedores de infraestructura en la nube.
- **OBJN-22:** Utilizar tecnologías que permitan desarrollar y desplegar el sistema en cualquier sistema operativo.
- **OBJN-23:** Integrar sistema de respaldos de información para las bases de datos.
- **OBJN-24:** La información en las bases de datos debe estar encriptada en disco.
- **OBJN-25:** Los comandos del chatbot deben implementar una buena experiencia conversacional por medio de texto simple o botones interactivos si la plataforma de chat lo permite y si el tipo de comando lo requiere.
- **OBJN-26:** El sistema debe estar en la capacidad de escalar sus recursos de memoria, procesamiento y almacenamiento en caso de presentarse una alta demanda de peticiones de usuarios.
- **OBJN-27:** El sistema debe de concebir desde su diseño las pruebas unitarias y pruebas de integración.
- **OBJN-28:** En plataformas de chat externas debe ser posible interactuar con LaraBot vía mensaje directo o en canales públicos.
- **OBJN-29:** El chatbot debe ofrecer un comando que permita deshabilitar/habilitar las comunicaciones si es requerido.
- **OBJN-30:** Estar disponible 24x7 consumiendo la menor cantidad de recursos posibles.

4.3. Descripción del sistema

Partiendo del discurso del proyecto, los objetivos de negocio, el objetivo general y el caso de estudio de este, se plantea un sistema chatbot al cual será referenciado cómo **#LaraBot** a partir de este punto**.** Este sistema contará con la capacidad de recibir un mensaje desde una interfaz de chat en texto interna o externa para posteriormente interpretar la intención de este y su descomposición en entidades; finalmente dada una base de conocimientos y/o habilidades relacionadas con la intención del mensaje recibido se buscará entregar una respuesta adecuada.

La arquitectura de **#LaraBot** buscará proponer un diseño modular y extensible que le permita integrarse de manera agnóstica con diferentes interfaces de chat y servicios cognitivos para potenciar sus capacidades. Respecto a esto último, se justifica que el NLP es implementado de forma muy similar por casi todos los proveedores actuales, más información en la teoría de la sección 3.2.1.4, por lo cual usar un servicio u otro no es relevante ya que no hay nada exclusivo y patentado en el algoritmo entre varios proveedores sobre la implementación de *deep learning* para el manejo y procesamiento de idiomas. El sistema por lo tanto tomará las preguntas o peticiones que el usuario puede realizar y estas se resolverán a través de un conjunto de servicios cognitivos preconfigurados para la identificación de intención.

4.3.1. Presentación de componentes del sistema

4.3.1.1. Diagrama de Contexto

A continuación, se presenta el diagrama de contexto el cual tiene como fin permitir visualizar los componentes del sistema y la relación entre ellos.

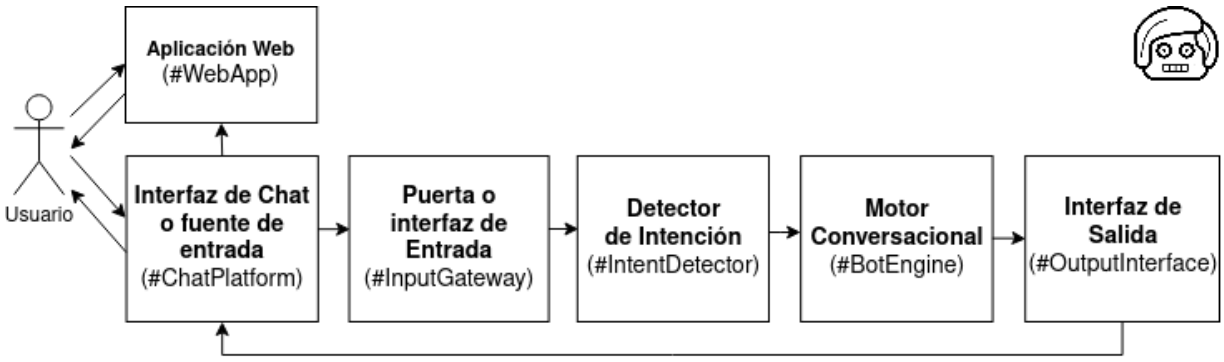

Figura 53: Componentes del sistema

4.3.1.2. Descripción de componentes del sistema

El sistema se descompondrá a nivel de un conjunto de subsistemas de ejecución, tal y cómo han sido presentados en la figura 53:

- **Interfaz de chat o fuente de entrada (#ChatPlatform)**

  Representa el medio por el cual el usuario podrá interactuar con el sistema para el envío de mensajes. La fuente de entrada se puede lograr involucrando software de terceros con métodos que garanticen llamadas a la API de entrada al sistema del chatbot; la interacción con estas plataformas externas se logra por medio de los diferentes SDKs y APIs de distintos proveedores. También el diseño contempla el uso de peticiones directas al sistema mediante un cliente HTTP seguro, un widget de chat, o el uso de un asistente de voz de terceros al cual se le delega la generación de voz a texto para su posterior uso en el sistema.

  Algunas plataformas de chat externas cómo Slack suelen requerir una autenticación previa antes de permitir él envió de mensajes, por lo cual más adelante se tendrá en cuenta un subsistema web para suplir esta necesidad.

- **Aplicación web (#WebApp)**

  Si bien el sistema propuesto a desarrollar es una experiencia enteramente conversacional, algunas de las plataformas en **#ChatPlatform** pueden requerir de una interfaz web para completar un paso de autorización y validación de permisos, esto será un valor para tener en cuenta para poder integrar plataformas futuras.

  Por otro lado, el diseño de la arquitectura contempla la posible existencia de un portal web donde los usuarios puedan administrar preferencias del sistema del chatbot y visualizar información sobre cada una de las habilidades de dicho sistema, idealmente esto se hará sobre un subcomponente de página de aterrizaje.

  Finalmente, para los usuarios administradores, desarrolladores y stakeholders del sistema **#LaraBot** implementará un panel de administración interno que permita visualizar información de usuarios registrados en base de datos y el histórico de eventos realizados.

- **Puerta o interfaz de entrada del sistema (#InputGateway)**

  Expone una interfaz de comunicación vía API para el paso de mensajes entre el **#ChatPlatform** y el sistema del chatbot. Dependiendo del tipo **#ChatPlatform**, la puerta de entrada puede operar en modo síncrono o asíncrono.

  - Puerta de entrada síncrona: Modo de comunicación en el cual el **#ChatPlatform** espera una respuesta inmediata sobre la interpretación del mensaje recibido. Este modo suele ser requerido cuando la comunicación es iniciada desde un cliente HTTP, widget de chat tercero o asistente de voz.
  - Puerta de entrada asíncrona: Se diferencia del modo síncrono ya que requiere una confirmación de recibido (ACK[10]) por parte del sistema del chatbot sobre el mensaje

[10] ACK (acknowledgement): Señal de confirmación que espera un emisor sobre un mensaje enviado.

enviado. La respuesta sobre la interpretación del mensaje recibido será enviada eventualmente por el sistema del chatbot a **#ChatPlatform** haciendo uso de las APIs expuestas por el mismo. Este tipo de comunicación es el más utilizado por las plataformas de chat externas cómo Slack, Google Chat y Microsoft Teams, y dado que la interacción suele ser iniciada por un usuario, la expectativa de respuesta para ofrecer una experiencia conversacional aceptable suele estar en un rango máximo de 3 a 5 segundos.

- **Detector de intención (#IntentDetector)**

  Módulo encargado de tomar el mensaje y contexto recibido por el **#InputGateway** e interpretar una intención a partir de este junto a un conjunto de entidades tomando como referencia una base de conocimientos y habilidades/acciones preestablecidas. Este módulo en su etapa inicial usará los servicios cognitivos determinados en pasos posteriores. En un futuro está abierta la posibilidad de realizar una implementación propia basada en una arquitectura de microservicios para la ejecución de modelos locales. Los valores generados por este módulo como el contexto, la intención y la entidad serán usados por el motor conversacional.

- **Motor conversacional (#BotEngine)**

  Componente que recibe cómo entrada las salidas generadas por **#IntentDetector**, las cuales están conformadas por una intención, un conjunto de entidades y el contexto. A partir de estos parámetros se ejecuta un controlador (*handler*) asociado a la intención detectada.

  El controlador se encarga de realizar acciones, consumir/consultar APIs o ejecutar alguna instrucción relacionada con la intención recibida, apoyándose de las entidades y el contexto recibido. Finalmente, el módulo genera una respuesta que se enviará de vuelta al usuario.

- **Interfaz de Salida (#OutputInterface)**

  Modulo encargado de tomar el mensaje de respuesta generado por el **#BotEngine** y prepararlo con base al formato esperado por **#ChatPlatform**, las plataformas externas aceptan formatos conocidos como texto rico, el cual puede ser presentado con formatos tipo *markdown*, HTML, texto plano, objetos JSON, entre otras estructuras de datos especificadas por la interfaz del chat.

  Posteriormente el mensaje formateado se envía de vuelta tomando como referencia a los metadatos de contexto del usuario del **#ChatPlatform**, usando los SDKs y/o APIs disponibles según la plataforma.

4.4. Usuarios del sistema

4.4.1. Usuario Chat

Tipo de usuario	Usuario final que interactúa con alguna interfaz de chat soportada por **#ChatPlatform**.
Formación	No aplica.
Habilidades	Tener conocimientos mínimos para interactuar con la alguna de las interfaces de chat soportadas por **#ChatPlatform**.
Actividades	Interactuar con **#LaraBot** mediante alguna interfaz de chat soportada, enviando mensajes relacionados con algunas de las habilidades del sistema cómo: ● Mantener una conversación con oraciones básicas tales cómo ¿qué hora es?, ¿qué día es hoy?, ¿quién eres?, ¿cómo estás? o un simple saludo. ● Obtener información de noticias. ● Solicitar un chiste. ● Solicitar un gif aleatorio basado en un mensaje. ● Permitir obtener la hora basada en una zona horaria o nombre de país. ● Obtener una recomendación de películas. ● Obtener una recomendación de música. ● Pedir una imagen aleatoria.

4.4.2. Usuario Web

Tipo de usuario	Usuario final que interactúa con el sistema a través de una interfaz web como **#WebApp**.
Formación	No aplica
Habilidades	Tener conocimientos mínimos para interactuar con páginas y navegadores web.
Actividades	● Los usuarios serán redirigidos al **#WebApp** desde el **#ChatPlatform** para completar un flujo de autorización requerido por algunas plataformas de chat. ● Permitir visualizar información sobre el sistema **#LaraBot** mediante una página de aterrizaje.

4.4.3. Usuario Desarrollador o Administrador

Tipo de usuario	Usuario desarrollador o Stakeholder del sistema **#LaraBot.**
Formación	Desarrollador o persona con conocimientos técnicos básicos sobre chatbots.
Habilidades	● Estar familiarizado con los componentes **#ChatPlatform** y **#WebApp** del sistema **#LaraBot**. ● Saber interpretar un gráfico o los históricos de eventos arrojados por el sistema.
Actividades	● Interactuar directamente con el sistema a través de una página de administración interna de la **#WebApp** para visualizar, editar o eliminar registros de usuarios en base de datos. ● Visualizar e interpretar históricos de eventos realizados por los usuarios. ● Visualizar e interpretar históricos de eventos generados por errores o excepciones capturadas en el sistema.

4.5. Requisitos específicos del sistema

En la presente sección se procede a exhibir los requisitos funcionales, atributos de calidad y restricciones de diseño del sistema.

4.5.1. Requisitos Funcionales

4.5.1.1. Instalar #LaraBot en interfaces de chats externas

Número del requisito	RF-01
Nombre del requisito	Instalar **#LaraBot** en interfaces de chats externas
Fuente del requisito	Plataformas externas soportadas por el **#ChatPlatform**
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

Descripción:

Como usuario debo tener la capacidad de instalar **#LaraBot** como una aplicación o extensión en plataformas externas cómo Slack, Microsoft Teams y/o Google Chat.

Entradas:
Plataforma de chat externa e información del usuario.

Proceso:

- Buscar a **#LaraBot** en el directorio de aplicaciones de la plataforma externa.
- Seguir los pasos de instalación que la plataforma requiera. Puede que en algunas plataformas sea necesario seguir un proceso de autenticación.
- Tras el paso anterior el usuario debe poder visualizar una instancia de **#LaraBot** instalada dentro de sus aplicaciones en la interfaz de la plataforma externa.

Salida:
Aplicación instalada en la plataforma del proveedor tercero.

4.5.1.2. Autenticación del usuario en plataforma de chat externa

Número del requisito	RF-02
Nombre del requisito	Autenticación de usuario en plataforma de chat externa
Fuente del requisito	Proceso de instalación que requiere autorización en plataforma de chat externa especificado en RF-01.
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

Descripción:

Como usuario debo tener la capacidad de autenticarme a través de los medios proporcionados por las plataformas externas soportadas en **#ChatPlatform** y **#WebApp**; se espera que el formulario de autenticación desplegado corresponda al proveedor de servicio de la cuenta con la que se desea autorizar.

Entradas:
Información del usuario obtenida a través de la plataforma externa en **#ChatPlatform** que requiere autenticación.

Proceso:

- Siguiendo el proceso de instalación de **#LaraBot** especificado en RF-01 el usuario es redirigido desde el **#ChatPlatform** hacia la **#WebApp**, que a su vez aplica una redirección hacia un formulario de autorización del proveedor externo con el que se desea autenticar.
- El usuario se autoriza aceptando los permisos de interacción que la plataforma requiera y se retorna información de autenticación hacia la **#WebApp** donde es enviada y almacenada en base de datos.
- El usuario retorna al **#ChatPlatform** con la aplicación autorizada por medio de un token.

Salida:

- La aplicación autorizada en la plataforma externa.
- Información de contexto del usuario autorizado es almacenada en base de datos de **#LaraBot**.

4.5.1.3. Interactuar con #LaraBot en interfaces de chats externas

Número del requisito	RF-03
Nombre del requisito	Interactuar con **#LaraBot** en interfaces de chats externas
Fuente del requisito	Usuario del sistema
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

Descripción:
Como usuario final del sistema debo poder enviar mensajes a **#Larabot** y recibir la respuesta de dichos mensajes desde la interfaz de chat de las plataformas externas soportadas en **#ChatPlatform**.

Entradas:
Mensaje del usuario enviado desde plataforma de chat externa.

Proceso:
- Abrir un mensaje directo con **#LaraBot** una vez instalada la aplicación siguiendo los pasos en RF-01 y RF-02.
- Enviar mensaje vía chat a **#LaraBot**.
- Recibir mensaje de respuesta de vuelta.

Salida:
Respuesta de **#LaraBot** enviada de regreso hacia el usuario.

4.5.1.4. Interactuar con #LaraBot desde una interfaz de chat interna

Número del requisito	RF-04
Nombre del requisito	Interactuar con **#LaraBot** desde una interfaz de chat HTTP
Fuente del requisito	Usuarios del sistema
Prioridad del requisito	▢ Alta/Esencial ▣ Media/Deseado ▢ Baja/Opcional

Descripción:
Como usuario debo poder enviar y recibir mensajes desde una interfaz de chat interna, cómo un widget de chat embebido en la **#WebApp** o directamente enviando una petición desde un cliente HTTP.

Entradas:
Mensaje del usuario enviado desde interfaz de chat interna.

Proceso:

- Enviar mensaje a **#LaraBot** desde interfaz de chat interna.
- Recibir mensaje de respuesta de vuelta.

Salida:

Respuesta de **#LaraBot** enviada de vuelta hacia el usuario.

4.5.1.5. Uso del sistema por usuarios anónimos

Número del requisito	RF-05
Nombre del requisito	Uso del sistema por usuarios anónimos
Fuente del requisito	Peticiones web y/o plataformas de terceros con RF-02
Prioridad del requisito	▢ Alta/Esencial ▢ Media/Deseado ▣ Baja/Opcional

Descripción:
El sistema debe permitir que usuarios sin autenticación puedan interactuar con **#LaraBot**. Esto dependerá enteramente en que la plataforma externa lo permita, dado que en algunas de ellas la autenticación es un paso obligatorio.

Entradas:
Plataformas externas que permitan la interacción de usuarios anónimos con la aplicación o directamente alguna fuente de peticiones web.

Proceso:
Habilitar en el sistema la omisión de contexto de usuario en caso de que este no se encuentre autenticado. Permitiendo a los usuarios anónimos acceder a características definidas para estos en el sistema del *bot*.

Salida:
Usuario anónimo puede enviar y recibir mensajes de **#LaraBot**.

4.5.1.6. Almacenamiento de información básica del usuario

Número del requisito	RF-06
Nombre del requisito	Almacenamiento de información básica del usuario
Fuente del requisito	RF-01 y RF-02
Prioridad del requisito	▢ Alta/Esencial ▢ Media/Deseado ▣ Baja/Opcional

Descripción:
El sistema debe contar con la capacidad de almacenar de forma persistente la información correspondiente al usuario y su contexto, en caso de que este esté se encuentre autenticado. Es deseable que la información del usuario corresponda a:

- Nombre.
- Correo electrónico.
- Organización (opcional).
- Historial de comandos y/o chats (opcional).
- Idioma del usuario.

Entradas:
La información es obtenida de los sistemas de registro y autenticación de las plataformas de terceros en RF-01 y RF-02.

Proceso:
Se obtiene la información correspondiente del usuario a través de las plataformas externas, posteriormente esta información se almacena de forma estructurada en algún sistema persistente de información que facilite su consulta posterior para mantener el contexto de los mensajes enviados por el usuario en cuestión.

Salida:
Sistema para almacenamiento de información estructurada.

4.5.1.7. Creación y eliminación de comando/acciones

Número del requisito	RF-07
Nombre del requisito	Creación y eliminación de comando/acciones (habilidades)
Fuente del requisito	Desarrolladores y administradores
Prioridad del requisito	☐ Alta/Esencial ▣ Media/Deseado ☐ Baja/Opcional

Descripción:
El sistema debe de contar con una estructura simple que facilite la creación, integración y eliminación de comandos/acciones al sistema. Estas acciones usan recursos específicos y generan respuestas a partir de la ejecución de estos mismos. Se debe permitir visualizar las acciones disponibles en el sistema. Adicional a esto se espera que las acciones se compongan de:

- Una frase o token de identificación.
- Parámetros que representen las entidades de la acción.
- Un controlador establecido para la ejecución de recursos.

- Una respuesta preestablecida para:
  - Errores en la ejecución.
  - Éxito en la ejecución.

Entradas:
Los desarrolladores pueden implementar nuevas acciones o eliminar acciones antiguas ya preestablecidas de forma simple. Los administradores pueden solicitar y sugerir nuevas acciones.

Proceso:
El sistema contará con un número de acciones predeterminadas que el usuario podrá ejecutar a partir de los mensajes estructurados. Estas acciones cuentan con una estructura y formato bien definidos que permiten a los desarrolladores integrarlas al sistema. Las acciones se ejecutan con las estructuras generadas en los mensajes del usuario, hacen uso de los recursos del sistema y estas pueden generar una respuesta a partir del recurso ejecutado.

Salida:
Lista de acciones disponibles en el sistema que ejecutan los recursos y características del sistema para generar un mensaje.

4.5.1.8. Configuración y parametrización del sistema

Número del requisito	RF-08
Nombre del requisito	Configuración y parametrización del sistema
Fuente del requisito	**#WebApp** y usuario administrador
Prioridad del requisito	☐ Alta/Esencial ▣ Media/Deseado ☐ Baja/Opcional

Descripción:
El sistema debe permitir a los administradores modificar valores configurables para parametrizar el chatbot en su organización. Debe existir un módulo web que habilite estas capacidades y el sistema debe persistir estos cambios para su uso posterior.

Entradas:
Datos de las órdenes ingresadas por un usuario.

Proceso:
El módulo de configuración en **#WebApp** se mantiene en un estado de reposo en tanto se ingresan los parámetros de configuración, el módulo de configuración sale de este estado cuando una orden indica que se ha terminado la parametrización. Se realiza el envío de los valores capturados en las entradas de la interfaz en **#WebApp**, estos son enviados al sistema a través de la interfaz de comunicación.

Salida:
Sistema configurado de acuerdo con las modificaciones establecidas por el usuario administrador.

4.5.1.9. Identificación de la intención del mensaje

Número del requisito	RF-09
Nombre del requisito	Identificación de la intención del mensaje
Fuente del requisito	Mensaje proveniente de **#InputGateway**
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

Descripción:
El *bot* internamente debe contar con la capacidad de identificar los mensajes del usuario. A partir de los mensajes estructurados se debe encontrar una intención para asociarla con un conjunto de acciones preestablecidas, esto por medio de frases claves de las acciones.

Entradas:
Se requiere el mensaje estructurado junto a las acciones preestablecidas y los servicios previamente implementados.

Proceso:
#InputGateway entrega los mensajes e información de contexto correspondiente para el módulo **#IntentDetector** éste toma las acciones que se han establecido previamente por los desarrolladores para asociar dichas acciones a la intención adecuadamente.

Salida:
Intención y acción asociadas con su correspondiente controlador (*handler*) para su ejecución.

4.5.1.10. Identificación de las entidades del mensaje

Número del requisito	RF-10
Nombre del requisito	Identificación de las entidades del mensaje
Fuente del requisito	Mensaje proveniente de **#InputGateway**
Prioridad del requisito	▢Alta/Esencial ▣ Media/Deseado ▢ Baja/Opcional

Descripción:
El *bot* internamente debe contar con la capacidad de identificar las entidades correspondientes a los mensajes del usuario. A partir de estos mensajes estructurados se deben encontrar dichas entidades, las cuales deben ser almacenadas y posteriormente usadas en la ejecución de la acción asociada previamente.

Entradas:
Se requiere el mensaje estructurado junto a las acciones preestablecidas y los servicios previamente implementados.

Proceso:
#InputGateway entrega los mensajes e información de contexto correspondiente para el módulo **#IntentDetector** éste usa los servicios de NLP y descompone el mensaje con sus respectivas entidades.

Salida:
Entidades correspondientes al mensaje analizado.

4.5.1.11. Ejecución de comandos/acciones

Número del requisito	RF-11
Nombre del requisito	Ejecución de comandos/acciones
Fuente del requisito	Lista de acciones RF-06, intención de RF-09 y entidades de RF-10
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

Descripción:
El sistema debe ejecutar las acciones correspondientes a la intención y entidades identificadas previamente. Estas acciones deben generar un resultado que se usará para comunicar el éxito o falla en su ejecución.

Entradas:
Lista de acciones con sus correspondientes *handlers*, intención y entidades asociadas previamente.

Proceso:
Se envía la intención asociada y sus entidades a una acción correspondiente. El sistema ejecuta la acción y genera un resultado.

Salida:
Estructura con el resultado de la ejecución de la acción.

4.5.1.12. Generación de respuestas

Número del requisito	RF-12
Nombre del requisito	Generación de respuestas según resultados obtenidos
Fuente del requisito	Lista de acciones y resultados de las acciones RF-11

Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

Descripción:
Se debe generar una respuesta a partir de los resultados de ejecutar una acción, esta respuesta se obtiene por medio de respuestas preestablecidas por los desarrolladores.

Entradas:
Lista de acciones y resultado de la acción ejecutada

Proceso:
La acción al ser ejecutada genera un resultado, dependiendo de este resultado las acciones tienen respuestas preestablecidas para comunicar si la ejecución fue exitosa o no. Esta respuesta debe comunicar al usuario los resultados con retroalimentación respectiva a los parámetros establecidos en el mensaje.

Salida:
Respuesta con los valores correspondientes de la retroalimentación de la ejecución de una acción.

4.5.2. Requisitos no Funcionales o Atributos de Calidad

4.5.2.1. Interoperabilidad

Se desea lograr a través de una estructura de alta cohesión y bajo acoplamiento, una comunicación eficaz entre los subsistemas, esto a partir de los siguientes requisitos:

★ **Recepción y estructuración del mensaje**

Número del requisito	RNF-01
Nombre del requisito	Recepción y estructuración del mensaje
Fuente del requisito	Mensaje proveniente de **#ChatPlatform**
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

Teniendo en consideración las características del mensaje entregado por el usuario, se debe analizar qué procesos son necesarios para que este esté formalizado en una estructura transversal a cualquiera de las fuentes de entrada en **#ChatPlatform**. El mensaje debe tener una estructura estándar para ser enviado junta al contexto para la ejecución de acciones y generación de respuestas.

El sistema recibe el mensaje a través de una interfaz intermedia que permite generar una estructura estándar. Esta estructura es agnóstica y garantiza el funcionamiento de los procesos

internos del *bot*. Este mensaje cuenta con los valores de contexto correspondiente al usuario dueño del mensaje.

★ **Implementación de servicios para NLP**

Número del requisito	RNF-02
Nombre del requisito	Implementación de servicios para NLP
Fuente del requisito	El mensaje en **#InputGateway** y las acciones de RF-06
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

El sistema debe implementar de forma extensible y con poca dependencia servicios cognitivos que permitan realizar procesos de formalización a los mensajes de los usuarios. Tomando los requisitos de los diferentes servicios cognitivos, se genera una interfaz que facilite el envío y recepción de información a estos servicios. Estos servicios usan el mensaje que ha sido estructurado previamente se analizan y se recibe una respuesta a partir de las acciones para su posterior uso y relación con los recursos.

★ **Envió de respuesta al usuario**

Número del requisito	RNF-03
Nombre del requisito	Envió de respuesta al usuario
Fuente del requisito	Respuesta generada en **#BotEngine**
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

Se debe comunicar la respuesta obtenida en el sistema al usuario. El sistema debe contar con la capacidad de enviar esta respuesta sin importar la interfaz o la fuente de salida.

Se obtiene una respuesta generada por **#BotEngine** en el módulo de **#OutputInterface.** Esta respuesta se estructura en el formato del sistema correspondiente al que el usuario envió el mensaje original y haciendo uso de las APIs de las plataformas externas en **#ChatPlatform** o una interfaz web **#WebApp** se comunica del resultado de la acción al usuario.

4.5.2.2. Mantenibilidad

★ **Uso de un proveedor de servicio serverless**

Número del requisito	RNF-04
Nombre del requisito	Uso de un proveedor de servicio
Fuente del requisito	Desarrollador

Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

Para garantizar la mantenibilidad del sistema, se requiere el uso de un proveedor de servicio con rasgos de los sistemas serverless, esto ligado a las características y ventajas de los servicios serverless en la sección **2.3.10.3**.

★ **Uso de contenedores**

Número del requisito	RNF-05
Nombre del requisito	Uso de contenedores
Fuente del requisito	Sistemas de contenerización
Prioridad del requisito	▢ Alta/Esencial ▣ Media/Deseado ▢ Baja/Opcional

El sistema requiere ser desarrollado con el uso de contenedores como herramienta principal para facilitar su mantenibilidad cuando deba desplegarse en un servidor. Los contenedores garantizan un entorno consistente y sin cambios para la ejecución del software [92, 93], principalmente para los servidores, lo cual es un punto fundamental para el posterior proceso de mantenimiento y actualización del software.

★ **Herramientas de integración continua**

Número del requisito	RNF-06
Nombre del requisito	Herramientas de integración continua
Fuente del requisito	Servidor de despliegue
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

El sistema debe contar con herramientas de integración continua para facilitar la mantenibilidad de este. Se requiere que estas herramientas puedan detectar los cambios y procesos de liberación del producto para hacer los correspondientes despliegues automatizados del código.

★ **Pruebas unitarias y de integración**

Número del requisito	RNF-07
Nombre del requisito	Pruebas unitarias y de integración
Fuente del requisito	Código del sistema
Prioridad del requisito	▢ Alta/Esencial ▢ Media/Deseado ▣ Baja/Opcional

El sistema debe contar con pruebas que garanticen una cobertura alta, que justifique y mantenga la calidad del software. Las pruebas unitarias y de integración permiten mantener el software seguro de que ningún cambio genere daños o reversiones cuando una nueva característica o arreglo es adicionado.

4.5.2.3. Escalabilidad

★ Uso de funciones en la nube (*cloud*) o lambdas

Número del requisito	RNF-08
Nombre del requisito	Uso de funciones *cloud* o lambdas
Fuente del requisito	Proveedor de servicio en RNF-05
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

Crear aplicaciones serverless que sean las entradas de ejecución a los servicios de sistema. De esta forma permitir escalar los recursos como sea requerido.

★ Versionamiento de repositorios de software

Número del requisito	RNF-09
Nombre del requisito	Versionamiento de repositorios de software
Fuente del requisito	Código del sistema
Prioridad del requisito	▢ Alta/Esencial ▣ Media/Deseado ▢ Baja/Opcional

El código del sistema debe estar debidamente versionado y tener un repositorio de software que permita realizar un seguimiento a los cambios y el proceso de desarrollo del *bot*. Se recomienda el uso de GitHub y git para el manejo del repositorio del código y versionamiento respectivamente.

★ Tiempo de respuesta menor a 5 segundos

Número del requisito	RNF-10
Nombre del requisito	Tiempo de respuesta menor a 5 segundos
Fuente del requisito	Sistema
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

El sistema debe tener la capacidad de escalar y mantener recursos que permitan generar respuestas en un tiempo máximo de 5 segundos, dándole al usuario la capacidad de gozar de una experiencia fluida con el sistema.

4.5.2.4. Extensibilidad

★ Usar serverless framework o herramientas similares

Número del requisito	RNF-11
Nombre del requisito	Usar serverless framework o herramientas similares
Fuente del requisito	RNF-08
Prioridad del requisito	▢ Alta/Esencial ▣ Media/Deseado ▢ Baja/Opcional

Para garantizar que la implementación sea lo más transversal y agnóstica posible hacia un proveedor de servicios, se requiere que la creación de funciones de serverless están hechas con un sistema agnóstico como lo es serverless framework [91].

★ Usar patrón de diseño Strategy

Número del requisito	RNF-12
Nombre del requisito	Usar patrón de diseño strategy
Fuente del requisito	Código de implementación y desarrolladores
Prioridad del requisito	▢ Alta/Esencial ▣ Media/Deseado ▢ Baja/Opcional

Hacer uso de múltiples plataformas de chats es un proceso complejo que puede conducir al sistema a ser poco extensible al requerir la creación de código específico para la interacción del *bot* con cada una de estas plataformas de terceros. Es por esto por lo que se requiere que el sistema implemente patrones de diseño que garanticen la extensibilidad del mismo.

Se aconseja principalmente hacer uso del patrón de diseño strategy para tomar el código de interacción o implementación específica diferentes y se extraigan todos esos comportamientos para ser estructurados en clases separadas llamadas estrategias [94].

★ Implementación de servicios en API Rest

Número del requisito	RNF-13
Nombre del requisito	Implementación de servicios en API Rest
Fuente del requisito	Servicios del sistema

Prioridad del requisito	▢ Alta/Esencial ▣ Media/Deseado ▢ Baja/Opcional

Se requiere que el sistema implemente una arquitectura basada en microservicios que garanticen el bajo acoplamiento y la extensibilidad del código con el uso de una arquitectura API Rest.

4.5.2.5. Usabilidad y Accesibilidad

★ Internacionalización

Número del requisito	RNF-14
Nombre del requisito	Internacionalización
Fuente del requisito	Plantillas y texto del sistema
Prioridad del requisito	▢ Alta/Esencial ▣ Media/Deseado ▢ Baja/Opcional

El sistema debe estar construido con reglas y manejo de internacionalización que faciliten la integración de nuevos idiomas, al menos en lo que se refiere a los mensajes desplegados a los usuarios.

★ Soporte de navegadores web

Número del requisito	RNF-15
Nombre del requisito	Soporte de navegadores web
Fuente del requisito	**#WebApp**
Prioridad del requisito	▢ Alta/Esencial ▣ Media/Deseado ▢ Baja/Opcional

Se debe garantizar el ingreso correcto al módulo de #WebApp y la página de aterrizaje. Es deseable que la interacción con el sistema web sea realizada desde cualquiera de los siguientes navegadores:

- Mozilla Firefox 83 o superior.
- Google Chrome 88 o superior.

4.5.2.6. Seguridad

★ Integridad de la información

Número del requisito	RNF-16
Nombre del requisito	Integridad de la información

Fuente del requisito	#ChatPlatform, #WebApp, #InputGateway, #IntentDetector, #BotEngine y #OutputInterface
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

El sistema garantiza que la información usada es tratada de forma adecuada, protegida y encriptada permitiendo que dicha información se mantenga relevante, completa, consistente y oportuna.

★ **Encriptar la base de datos**

Número del requisito	RNF-17
Nombre del requisito	Encriptar la base de datos
Fuente del requisito	Base de datos
Prioridad del requisito	▢ Alta/Esencial ▢ Media/Deseado ▣ Baja/Opcional

La información almacenada en las bases de datos debe contar con sistemas de encriptación que garantice el almacenamiento de esta sin haber riesgos a que pueda filtrarse información sensible del usuario.

★ **Respaldo de las bases de datos**

Número del requisito	RNF-18
Nombre del requisito	Respaldo de las bases de datos
Fuente del requisito	Base de datos
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

La información almacenada en las bases de datos debe contar con un sistema de respaldo que permita mantener la base de datos respaldada en caso de que esta tenga algún tipo de problema de integridad, estos respaldos deben permitir repone la base de datos y deben realizarse de forma periódica.

4.5.2.7. Disponibilidad

★ **Disponibilidad**

Número del requisito	RNF-19
Nombre del requisito	Disponibilidad

Fuente del requisito	**#Larabot**
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

El tiempo de actividad de todos los subsistemas debe estar por encima de un margen o porcentaje de disponibilidad mayor o igual a 99%; dicho porcentaje puede ser estimado a partir de la siguiente ecuación:

PDS = Porcentaje de disponibilidad.
TTT = Total tiempo activo transcurrido desde el inicio.
TTI = Total tiempo acumulado de inactividad por fallos.

$$\textbf{PDS} = (TTT - TTI) / TTT$$

4.5.3. Restricciones de diseño

4.5.3.1. Desarrollo web

Número del requisito	RD-01
Nombre del requisito	Desarrollo web
Fuente del requisito	**#WebApp**
Prioridad del requisito	▢ Alta/Esencial ▣ Media/Deseado ▢ Baja/Opcional

La implementación del *landing page* y del **#WebApp** debe realizarse con herramientas y frameworks de HTML 5 y CSS 3.

4.5.3.2. Texto como única fuente

Número del requisito	RD-02
Nombre del requisito	Texto como única fuente
Fuente del requisito	Usuarios y **#OutputInterface**
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

El sistema puede habilitar un módulo de conversión de voz a texto, pero todos los procesos posteriores deben ser realizados solamente con un formato basado en texto. No se aceptará contenido multimedia de ningún tipo solo el mensaje en texto para la identificación de intencionalidad.

4.5.3.3. Soporte multilenguaje, solo de español e inglés

Número del requisito	RD-03
Nombre del requisito	Soporte multilenguaje, solo español e inglés
Fuente del requisito	Usuarios y **#OutputInterface**
Prioridad del requisito	▢ Alta/Esencial ▢ Media/Deseado ▣ Baja/Opcional

El sistema solo debe permitir el uso del inglés y español como idiomas de interacción con los recursos. Otros idiomas no hacen parte de la visión del software y no serán considerados válidos para la ejecución de este. Sin embargo, en futuras revisiones o implementaciones de la arquitectura se podría considerar nuevos idiomas.

4.5.3.4. Solo canales de comunicación HTTPS

Número del requisito	RD-04
Nombre del requisito	Solo canales de comunicación HTTPS
Fuente del requisito	**#OutputInterface**
Prioridad del requisito	▣ Alta/Esencial ▢ Media/Deseado ▢ Baja/Opcional

El sistema solo debe permitir el uso de HTTPS como protocolo de comunicación. Esto se requiere para mantener una comunicación segura con los clientes y que la información de los mensajes no se encuentre comprometida por un protocolo con seguridad débil.

4.6. Aplicando QAW-A

QAW-A es un método que se utiliza para descubrir y documentar los escenarios y riesgos de los atributos de calidad, los puntos sensibles, y compensaciones. En este se priorizan y refinan escenarios de atributos de calidad antes de que la arquitectura se comience a diseñar formalmente, donde:

- Los riesgos de los atributos de calidad son decisiones arquitectónicas que pueden crear problemas futuros para algún requisito de atributos de calidad.
- Los puntos de sensibilidad son parámetros arquitectónicos para los que al realizar un pequeño cambio hace una diferencia significativa en algún atributo de calidad.
- Los escenarios son utilizados por el arquitecto para analizar a la arquitectura e identificar intereses y posibles estrategias de mitigación.

Para la realización de QAW se requiere de la participación los stakeholders del proyecto que permitan analizar y generar estos diferentes escenarios y análisis de riesgos.

4.6.1. Involucrados en QAW-A

★ **Arquitectos**

La elaboración de la arquitectura del proyecto estará dirigida por profesionales en el área del desarrollo de software. Son dos profesionales aspirantes al magister de Ingeniería en Sistemas y computación de la Universidad Tecnológica de Pereira. Estos están dotados del conocimiento y las herramientas para el diseño arquitectónico en la Ingeniería del Software.

- **Arquitecto líder:** Sergio Alexander Florez Galeano. Ingeniero en Sistemas y computación. Con más de 7 años de experiencia en el desarrollo de software.
  Rol: Captura escenarios crudos, su priorización, refinamiento y facilitar las discusiones durante el taller.
- **Arquitecto líder:** Nelson Orlan Escobar Ceballos. Ingeniero en Sistemas y computación. Con más de 7 años de experiencia en el desarrollo de software.
  Rol: Captura escenarios crudos, guía la priorización de estos y asegurar que los pasos de este método se lleven a cabo.

★ **Stakeholders**

El proyecto requiere de dos tipos de roles en términos de partes de interesadas que participen en el taller, estos son:

- **Expertos en el área de software:** Con conocimientos técnicos afines al área del desarrollo de software. Los expertos en software son aquellos que tienen conocimiento suficiente para entender las necesidades del desarrollo de un producto de software. Es fundamental la participación de los roles de estos actores expertos ya que representan a la población principal a la que el proyecto se encuentra orientado, estos siendo los desarrolladores que

pueden implementar la arquitectura que está enfocada en la extensibilidad, escalabilidad y mantenibilidad de un proyecto de chatbot.

- **Expertos o no de otras áreas:** actores de otros campos que no poseen un conocimiento tan detallado sobre el desarrollo de software, con el rol de un usuario final. Los participantes que no tienen conocimiento técnico del proyecto son profesionales afines de otras áreas de conocimiento, los cuales entienden el funcionamiento básico de un software pero que su rol principal está enmarcado en la validación del producto y como este se orienta a nivel de usuarios.

Estos roles son fundamentales para dar una visión global del público que hará uso de esta arquitectura de forma directa e indirecta. Se cuenta en este punto con dos desarrolladores afines al manejo de infraestructura y creación de software en la nube y de igual forma con dos profesionales afines a áreas ajenas al desarrollo de software:

Expertos en el área de software:

- **Julián Andrés Florez Galeano:** Ingeniero de Software y experto en el desarrollo de aplicaciones móviles. Egresado de la carrera de Ingeniería de Sistemas y Computación de la Universidad Tecnológica de Pereira.
- **Mauricio Morales:** Ingeniero de software y emprendedor serial experto en el desarrollo de productos digitales. Fundador y CEO de Rocka.

Expertos o no de otras áreas:

- **Laura Juliana Ruiz Sanchez:** Analista líder y coordinadora de laboratorios internacionales Merieux Nutriscience. Egresada de la carrera de Química industrial de la Universidad Tecnológica de Pereira.
- **Astrid Katherine López Vanegas:** Psicóloga con certificación para intervención en Terapia ABA basada en la lista de tareas de la BACB. Con experiencia en la elaboración, ejecución y evaluación de proyectos de intervención en comunidades e instituciones, con conocimiento en el trabajo con comunidades vulnerables.
- **Julio Hernando Vargas Moreno:** Magister en la enseñanza de las matemáticas y docente de la Universidad Tecnológica de Pereira.

4.6.2. Despliegue del método QAW-A

QAW-A cuenta con 10 pasos enfocados en la priorización de escenarios, este es un método adaptado del original QAW y se centra en reducir el número de participantes (stakeholders) para preparar la primera parte de los valores que pueden entrar en una futura evaluación. Es importante recordar que las razones por las que se hacen estas reducciones y el uso de las adaptaciones son debido a la necesidad y enfoque a equipos pequeños de desarrollo [53].

En caso de querer profundizar más sobre las adaptaciones y diferencias de QAW y QAW-A se recomienda hacer una lectura de las secciones **3.2.4** y **3.6**. De igual forma se puede

complementar la lectura del discurso y adaptaciones del método QAW-A por medio de [53] que están basadas en del desarrollo y adaptación del SEI para equipos pequeños.

4.6.2.1. Identificación de motivaciones arquitectónicas

Las motivaciones arquitectónicas surgen de las diferentes necesidades y justificaciones de desarrollar un sistema que resuelva un problema específico, en este caso, relacionar los mensajes con una intención y una acción que genere una respuesta adecuada a un usuario. Estas motivaciones han sido expuestas en la lista de motivaciones arquitectónicas identificadas a lo largo del análisis desarrollado en el presente capítulo.

- Objetivos de negocio (Ver sección 4.2).
- Requisitos funcionales de alto nivel (Ver sección 4.4.2).
- Requisitos Atributos de calidad (Ver sección 4.4.3).
- Restricciones de diseño (Ver sección 4.4.4).

4.6.2.2. Lluvia de ideas sobre escenarios

Se aclara que parte de los escenarios del mundo real son generados por los stakeholders para el sistema. Los escenarios comprenden un estímulo relacionado, una condición ambiental y una respuesta. Los facilitadores se aseguran de que al menos un escenario aborde cada uno de los impulsores arquitectónicos identificados. A continuación, se muestra la lista de escenarios en crudo generados:

1. El usuario desea usar el *bot* en una plataforma de chat externa nueva, los desarrolladores evalúan e implementan una nueva extensión para el uso de *bot* en la nueva plataforma.
2. El usuario desea usar el *bot* en una plataforma de chat externa nueva, los desarrolladores evalúan y descartan la implementación del *bot* en esta plataforma.
3. El administrador de la organización realiza la instalación de la instancia de *bot* en una plataforma externa, el *bot* es adicionado al chat de la organización y se presenta.
4. Un Stakeholder solicita la adición de una nueva acción, los desarrolladores integran la nueva acción al *bot*.
5. Un Stakeholder solicita la adición de una nueva acción no válida, los desarrolladores validan la acción y comunican que no es adecuada para ser integrada al *bot*.
6. Un Stakeholder solicita la eliminación de una acción, los desarrolladores validan la acción y remueven la acción del sistema del *bot*.
7. El usuario ingresa datos inválidos de autenticación, el sistema rechaza el ingreso y maneja al usuario como un usuario anónimo hasta que este se autentique correctamente.
8. El usuario ingresa datos válidos de autenticación, el sistema acepta al usuario y lo identifica con su correspondiente información de contexto y organización si es necesario.
9. Un administrador con un idioma español o inglés instala la aplicación en su organización, el *bot* notifica al administrador en el idioma correspondiente que la aplicación con el *bot* ha sido exitosamente adicionada.

10. Un administrador con un idioma distinto al español e inglés instala la aplicación en su organización, el bot notifica al administrador de que su idioma no es soportado por el sistema.
11. El usuario hace un saludo a **#LaraBot**, el sistema responde de forma cordial y amigable al usuario.
12. El usuario pide al bot ejecutar una acción que no existe, el sistema no es capaz de reconocer la acción solicitada, notifica que la acción no existe y da una sugerencia de acciones posibles.
13. El usuario pide al bot ejecutar una acción con palabras y formas diferentes de expresar la acción preestablecida en el sistema, el sistema no es capaz de reconocer la acción solicitada y da una sugerencia de acciones posibles.
14. El bot ejecuta una acción errónea y el usuario pide corregirlo, el sistema recibe la solicitud y ejecuta la acción correspondiente.
15. El usuario pide al *bot* ejecutar una acción con palabras y formas diferentes de expresar la acción preestablecida en el sistema, el sistema reconoce la intención del mensaje con la acción solicitada y la ejecuta generando una respuesta apropiada.
16. El usuario intenta ejecutar una acción previamente removida, el bot notifica al usuario que la acción ya no existe y que ha sido previamente removida.
17. El administrador hace una modificación invalida en la configuración del *bot*, el sistema rechaza el cambio y notifica el error.
18. El administrador modifica de forma válida la configuración del *bot*, el sistema acepta esta modificación y cambia la configuración del bot notificando al administrador de que la modificación fue exitosa.
19. El usuario hace mención del *bot* en un canal público de un chat, el bot detecta la mención del usuario, se presenta y solicita de forma amable al canal como puede ser de utilidad.
20. En un canal público se le pide al bot ejecutar una acción invalida, el sistema rechaza la acción en un canal público y le comunica al canal de la invalidez del comando en este contexto.
21. En un canal público se le pide al bot ejecutar una acción permitida, el sistema analiza el mensaje lo asocia a la acción correspondiente, se ejecuta y genera una respuesta que se comunica al canal.
22. El usuario menciona a otro usuario a **#LaraBot**, el sistema obtiene la información del otro usuario y muestra acciones que pueden ser soportadas con otro usuario.
23. El usuario utiliza un sistema de chat por voz aceptado por **#LaraBot**, el bot recibe el mensaje convertido en texto y genera una respuesta de acuerdo con lo que se le solicitó en el mensaje.
24. El usuario utiliza un sistema de chat por voz no soportado por **#LaraBot**, no hay respuesta ya que el bot no recibe el mensaje.
25. Un cliente envía una petición HTTP sin ser segura, el sistema rechaza toda petición HTTP que no sea segura.
26. El usuario termina la sesión con el *bot*, el sistema acepta la salida del usuario y remueve la sesión.
27. El administrador de la organización remueve la aplicación del *bot* del chat, la plataforma acepta la solicitud y elimina la instancia del *bot* en el chat.

28. Los desarrolladores realizan pruebas sobre los respaldos de la base de datos, los respaldos no son satisfactorios y los desarrolladores corrigen el problema
29. Los desarrolladores realizan pruebas sobre los respaldos de la base de datos, los respaldos funcionan de forma satisfactoria y los desarrolladores mantienen el respaldo y el sistema intactos.
30. Un fallo en el sistema ocasiona la corrupción de la base de datos, se toma el respaldo de la base de datos y se recupera la información.

4.6.2.3. Consolidación de escenarios

En la sección presente se realiza un análisis de los escenarios crudos; estos son reescritos, filtrados y ajustados según sea necesario. Los escenarios serán agrupados y se les asignará un identificador con base a los atributos de calidad que satisfacen. El objetivo es generar una base de escenarios que permitan ser priorizados posteriormente.

Interoperabilidad

1. **INT-01:** El usuario establece una conexión con el bot al autenticarse y usar **#ChatPlatform**. El bot recibe los mensajes y despliega las opciones que este posee de forma íntegra.
2. **INT-02:** Durante el proceso de envío y recepción de un mensaje por parte del sistema los diferentes módulos **#ChatPlatform, #WebApp, #InputGateway** deben mantener la integridad del mensaje y relacionarlo a un controlador preestablecido en el sistema. El sistema debe recibir el mensaje de forma exitosa en **#IntentDetector** y **#BotEngine** para reconocer la intención del mensaje con la acción solicitada, ejecutarla para generar una respuesta apropiada y enviarla al **#OutputInterface**.

Mantenibilidad

1. **MNT-01:** Un desarrollador realiza la liberación de código nuevo en un módulo del sistema; una o más pruebas unitarias y de integración nuevas y/o antiguas fallan y evitan la integración de este código al módulo. El desarrollador corrige el error en su código y actualiza las pruebas, el código es aceptado y se integra al módulo.
2. **MNT-02:** Una nueva versión de **#LaraBot** es liberada en un ciclo de desarrollo. Cuando esta liberación ocurre los procesos de integración continua toman el código de los repositorios de software de los componentes liberados de **#LaraBot** y realizan un despliegue del código hacia un contenedor o plataforma (**#ChatPlatform)**, etiquetando el software con la nueva versión.
3. **MNT-03:** Una nueva versión de un componente de **#LaraBot** es liberada. Los sistemas de integración continua (CI) ponen al sistema en un estado de mantenimiento. En cuanto el proceso de mantenimiento está abierto el software de CI hace el despliegue del contenedor o software en la plataforma correspondiente:
    a. En caso de ser exitoso se finaliza el proceso y se pone la nueva versión a disposición de los usuarios.

b. En caso de haber una falla en el despliegue del software se hace un paso atrás se pone la versión antigua y se notifica de la falla para desplegar.

Extensibilidad

1. **EXT-01:** Un Stakeholder desea realizar la integración de una nueva acción/comando a **#LaraBot**, esta acción no se encuentra soportada por el sistema. Los desarrolladores analizan las necesidades de la nueva acción y el esfuerzo que esta requiera para ser añadida. Los desarrolladores integran la nueva acción en el **#BotEngine** y la habilitan para que esta pueda ser usada por los usuarios en conjunto con el **#IntentDetector**.
2. **EXT-02:** Un Stakeholder desea integrar un nuevo motor de intenciones en el sistema en **#BotEngine**. Los desarrolladores usando un patrón como el Strategy implementan una estrategia nueva para la ejecución de código específico del nuevo motor de intenciones.
3. **EXT-03:** Un Stakeholder desea usar el *bot* en una plataforma de chat externa nueva. Esta plataforma es aceptada por los desarrolladores que evalúan sus necesidades específicas. Los desarrolladores usando un patrón como el Strategy implementan una estrategia nueva para la ejecución de código específico de la nueva plataforma.
4. **EXT-04:** El administrador modifica de forma válida la configuración del *bot* en **#WebApp**. El sistema acepta esta modificación y cambia la configuración del bot notificando al administrador de que la modificación fue exitosa.

Escalabilidad

1. **ESC-01:** 400 o más mensajes son recibidos en el sistema provenientes de **#ChatPlatform**, si el valor de peticiones por segundo supera a los mil o el límite que sea establecido de peticiones, el sistema escala automáticamente y genera costos con respecto a los recursos accedidos
2. **ESC-02:** Los mensajes recibidos por el sistema son estructurados y relacionados con una intención, este proceso posteriormente genera una respuesta con resultados en un tiempo menor a 5 segundos.

Usabilidad y Accesibilidad

1. **UAC-01:** El usuario solicita al *bot* a través de un mensaje en #**ChatPlatform** ejecutar una acción, este mensaje se compone de palabras y formas diferentes de expresar una acción preestablecida en el sistema, el sistema reconoce la intención del mensaje en **#IntentDetector** con la acción solicitada y ejecuta su controlador generando una respuesta apropiada en **#BotEngine** y la envía al usuario en **#OutputInterface**.
2. **UAC-02:** El usuario utiliza un sistema de chat por voz aceptado por **#LaraBot** para realizar el envío de un mensaje al sistema. El *bot* recibe el mensaje que ha sido convertido en texto previamente y genera una respuesta de acuerdo con lo que se le solicitó en el mismo.
3. **UAC-03:** En un canal público se le pide al *bot* ejecutar una acción permitida, el sistema analiza el mensaje lo asocia a la acción correspondiente, se ejecuta y genera una respuesta que se comunica al canal.

4. **UAC-04:** El usuario menciona a otro usuario a **#LaraBot**, el sistema obtiene la información del otro usuario y muestra los diferentes comandos/acciones que pueden ser soportadas con otro usuario. El usuario A ejecuta la acción y el usuario B recibe la notificación.
5. **UAC-05:** Un administrador con un idioma español o inglés instala la aplicación en su organización, el *bot* notifica al administrador en el idioma correspondiente que el bot ha sido exitosamente adicionado
6. **UAC-06:** El usuario solicita listar las acciones/comandos disponibles en el sistema. El sistema retorna una respuesta con las diferentes opciones de comando que pueden ser ejecutadas.
7. **UAC-07:** El usuario ejecuta una acción que no puede ser asociada o que no existe. El sistema retorna una respuesta con diferentes opciones que pueden ser ejecutadas.

Seguridad

1. **SEG-01:** El sistema semanalmente genera un respaldo de la base de datos encriptada del sistema, este respaldo es validado y comprobado para analizar su integridad. El respaldo es almacenado para su posterior uso.
2. **SEG-02:** Un fallo en el sistema ocasiona la corrupción de la base de datos. Se analiza la integridad de la base de datos y esta es irrecuperable. Se toma un respaldo de la base de datos no mayor a una semana y se recupera la información.
3. **SEG-03:** Un cliente genera una petición HTTP sin el protocolo de seguridad HTTPS, el sistema rechaza la petición y no permite la ejecución de ningún recurso.
4. **SEG-04:** El administrador de la organización realiza la instalación de la instancia de bot en una plataforma externa. El *bot* verifica la información de la organización y su veracidad. El bot es adicionado al chat de la organización y se presenta.

Disponibilidad

1. **DIS-01:** Más de 100 usuarios usan el bot en diferentes horarios y días de la semana. El bot funciona con normalidad y siempre se encuentra disponible.

4.6.2.4. Priorización de escenarios

Teniendo en cuenta los objetivos del proyecto, se hace una priorización de cada uno de los escenarios según su importancia y efecto en el cumplimiento de dichos objetivos. A continuación, se presentan este escenario ordenándolos de mayor a menor según su prioridad:

1. **UAC-01:** El usuario solicita al bot una acción.
2. **EXT-02:** Un Stakeholder desea integrar un nuevo motor de intenciones.
3. **EXT-01:** Un Stakeholder desea realizar la integración de una nueva acción/comando.
4. **ESC-01:** 400 o más mensajes son recibidos y el sistema escala los recursos.
5. **MNT-01:** Pruebas unitarias y de integración evitan la integración código nuevo.
6. **INT-02:** Integridad del mensaje a través de todos los módulos del sistema se mantiene.
7. **MNT-02:** Una nueva versión de **#LaraBot** se libera y se despliega automáticamente.
8. **MNT-03:** Un componente es liberado en un contenedor o plataforma externa.

9. **EXT-03:** Un Stakeholder desea usar el bot en una plataforma de chat externa nueva.
10. **DIS-01:** Más de 100 usuarios usan el bot en diferentes horarios y días de la semana.
11. **UAC-06:** El usuario solicita listar las acciones/comandos disponibles en el sistema.
12. **UAC-07:** El usuario ejecuta una acción que no puede ser asociada o que no existe.
13. **SEG-01:** El sistema semanalmente respalda la DB encriptada y la verifica.
14. **SEG-02:** Un fallo ocasiona la corrupción de la base de datos y se usa el respaldo.
15. **SEG-04:** El administrador de la organización realiza la instalación **#LaraBot**.
16. **INT-01:** El usuario se autentica y usa **#ChatPlatform**.
17. **EXT-04:** El administrador modifica la configuración del bot en **#WebApp**.
18. **ESC-02:** El sistema genera una respuesta en un tiempo menor a 5 segundos.
19. **UAC-03:** Se le pide al bot ejecutar una acción permitida en un canal público.
20. **UAC-05:** Un administrador con un idioma español o inglés instala la aplicación.
21. **UAC-04:** El usuario menciona a otro usuario en un comando a **#LaraBot**.
22. **SEG-03:** Un cliente genera una petición HTTP sin el protocolo de seguridad HTTPS.
23. **UAC-02:** El usuario utiliza un sistema de chat por voz aceptado por **#LaraBot**.

4.6.2.5. Refinamiento de escenarios

Identificador: Id de escenario refinado *[colocar un identificador único para el escenario]*		
Escenario(s):		*[Lista de escenarios crudos que contempla este refinamiento]*
Objetivos de negocio:		*[Lista de objetivos de negocio que contempla este escenario refinado]*
Atributos de calidad relevantes:		*[Lista de atributos de calidad que contempla este escenario refinado]*
Componentes del escenario:	Estímulo:	*[Condición que el sistema debe considerar cuando llega a él]*
	Fuente del estímulo:	*[Entidad que genera el estímulo. Ej.: usuario, sistema externo]*
	Entorno:	*[Condiciones bajo las cuales el estímulo ocurre]*
	Artefacto:	*[Parte del sistema que es estimulada. Puede ser incluso todo el sistema]*
	Respuesta:	*[Actividad realizada después de que el estímulo llega al sistema]*
	Medida de la respuesta:	*[Forma cuantificable de medir la respuesta del sistema]*

Tabla 6: Plantilla de refinamiento de escenarios propuesta por Len Bass en [35].

En [53] se hace énfasis en que las plantillas y pasos de los métodos adaptados del SEI son muy similares a los originales. Tomando en cuenta que estos métodos pueden usar gran

parte del material original se plantea hacer uso de las plantillas diseñadas por el SEI como sugieren en [53] para la estructuración de la información. En *Software Architecture in Practice* [35] se plantea la siguiente plantilla para documentar y refinar los escenarios de calidad de la arquitectura:

Importancia Alta:

★ **Escenario Refinado 1**

<table>
<tr><td colspan="3">Identificador: ERF-01</td></tr>
<tr><td colspan="2">Escenario (s):</td><td>●UAC-01: El usuario solicita al bot a través de un mensaje en #ChatPlatform ejecutar una acción, este mensaje se compone de palabras y formas diferentes de expresar una acción preestablecida en el sistema.
●INT-02: Integridad del mensaje a través de todos los módulos del sistema se mantiene.</td></tr>
<tr><td colspan="2">Objetivos de negocio:</td><td>●OBJN-04: Entender la intención del mensaje enviado e intentar siempre dar una respuesta concisa y precisa al usuario incluso en caso de error o falta de entendimiento.
●OBJN-07: Se debe poder mantener una conversación muy simple que tenga sentido sobre temas triviales respondiendo a preguntas simples cómo ¿qué hora es?, ¿qué día es hoy?, ¿quién eres?, ¿cómo estás?
●OBJN-08: Ofrecer la capacidad de dar retroalimentación a los desarrolladores vía chatbot que permita canalizar quejas, reclamos o sugerencias.
●OBJN-28: En plataformas de chat externas debe ser posible interactuar con #LaraBot vía mensaje directo o en canales públicos.</td></tr>
<tr><td colspan="2">Atributos de calidad:</td><td>Interoperabilidad, Usabilidad y accesibilidad.</td></tr>
<tr><td rowspan="6">Componentes</td><td>Estímulo:</td><td>Obtener Mensaje del usuario</td></tr>
<tr><td>Fuente del estímulo:</td><td>El usuario del sistema en #ChatPlatform</td></tr>
<tr><td>Entorno:</td><td>Sistema de detección de intención y plataforma de chat externa</td></tr>
<tr><td>Artefacto:</td><td>Sistema</td></tr>
<tr><td>Respuesta:</td><td>El sistema reconoce la intención del mensaje en #IntentDetector con la acción solicitada y ejecuta su controlador generando una respuesta apropiada en #BotEngine y la envía al usuario en #OutputInterface.</td></tr>
<tr><td>Medida de la respuesta:</td><td>La acción y la respuesta tenga sentido con la intención del mensaje del usuario</td></tr>
</table>

★ **Escenario Refinado 2**

Identificador: ERF-02		
Escenario (s):		**EXT-02:** Un Stakeholder desea integrar un nuevo motor de intenciones.
Objetivos de negocio:		● **OBJN-09:** Recopilar analíticas de uso del chatbot sobre los comandos, preguntas, palabras, oraciones o errores más comunes capturados a partir de mensajes enviados por los usuarios. ● **OBJN-16:** El desarrollador debe poder agregar nuevas habilidades al sistema de manera flexible. ● **OBJN-21:** Se deben utilizar tecnologías que permitan la menor cantidad de acoplamiento posible con proveedores de infraestructura en la nube.
Atributos de calidad:		Extensibilidad
Componentes	Estímulo:	Integrar nuevos motores de intenciones en **#IntentDetector**.
	Fuente del estímulo:	Stakeholders del proyecto y desarrolladores
	Entorno:	Componentes en **#IntentDetector** y **#BotEngine**
	Artefacto:	Servicio cognitivo
	Respuesta:	Se pudo integrar el servicio para la detección de intención
	Medida de la respuesta:	Que siempre se pueda hacer un cambio en el sistema de servicios de detección de intenciones de forma simple

★ **Escenario Refinado 3**

Identificador: ERF-03	
Escenario (s):	**EXT-01:** Un Stakeholder desea realizar la integración de una nueva acción/comando.
Objetivos de negocio:	● **OBJN-05:** Disponer de un comando de ayuda que permita al chatbot explicar sus funciones, limitaciones y capacidades; lo que brinda al usuario una idea clara del alcance que posee el chatbot a la hora de brindar ayuda. ● **OBJN-16:** El desarrollador debe poder agregar nuevas habilidades al sistema de manera flexible. ● **OBJN-25:** Los comandos del chatbot deben implementar una buena experiencia conversacional por medio de texto simple o botones interactivos si la plataforma de chat lo permite y el tipo de comando lo requiere.

		●**OBJN-29:** El chatbot debe ofrecer un comando que permita deshabilitar/habilitar las comunicaciones si es requerido.
Atributos de calidad:		Extensibilidad
Componentes	Estímulo:	Adicionar nuevas acciones/comandos al *bot*
	Fuente del estímulo:	Stakeholders del proyecto y desarrolladores
	Entorno:	Módulos de **#BotEngine** y **#OutputInterface**
	Artefacto:	Lista de acciones del sistema
	Respuesta:	Se pudo adicionar una nueva acción/comando al sistema.
	Medida de la respuesta:	Cuánto esfuerzo y tiempo requiere el desarrollo de una acción/comando nuevo y su posterior integración en el sistema sin afectar las ya existentes.

★ **Escenario Refinado 4**

Identificador: ERF-04		
Escenario (s):		●**ESC-01:** 400 o más mensajes son recibidos y el sistema escala los recursos. ●**ESC-02:** El sistema genera una respuesta en un tiempo menor a 5 segundos.
Objetivos de negocio:		●**OBJN-01:** Brindar una respuesta al usuario en un tiempo oportuno. ●**OBJN-26:** El sistema debe estar en la capacidad de escalar sus recursos de memoria, procesamiento y almacenamiento en caso de presentarse una alta demanda de peticiones de usuarios.
Atributos de calidad:		Escalabilidad
Componentes	Estímulo:	Tener un sistema escalable en recurso según su uso
	Fuente del estímulo:	Usuarios del sistema
	Entorno:	Sistema interno del bot, principalmente los módulos de **#InputGateway**, **#IntentDetector** y **#BotEngine**
	Artefacto:	Mensajes y peticiones enviadas al sistema.
	Respuesta:	El sistema pudo escalar los recursos y respondió a todos los mensajes y peticiones.

	Medida de la respuesta:	El sistema puede responder mensajes escalando los recursos y puede responder en un tiempo menor a 5 segundos

★ **Escenario Refinado 5**

Identificador: ERF-05		
Escenario (s):		**MNT-01:** Un desarrollador realiza la liberación de código nuevo en un módulo del sistema, una o más pruebas unitarias y de integración nuevas y/o antiguas fallan y evitan la integración de este código al módulo. El desarrollador corrige el error en su código y actualiza las pruebas, el código es aceptado y se integra al módulo.
Objetivos de negocio:		●**OBJN-19:** El sistema debe poder registrar errores que entreguen una trazabilidad al desarrollador sobre excepciones y su contexto. ●**OBJN-22:** Utilizar tecnologías que permitan desarrollar y desplegar el sistema en cualquier sistema operativo. ●**OBJN-27:** El sistema debe de concebir desde su diseño las pruebas unitarias y pruebas de integración.
Atributos de calidad:		Mantenibilidad
Componentes	Estímulo:	Mantener calidad en el código y evitar la introducción de errores en el software
	Fuente del estímulo:	Desarrolladores
	Entorno:	Repositorio de software y servidores de despliegue.
	Artefacto:	Código específico de los componentes del sistema
	Respuesta:	El código es aceptado.
	Medida de la respuesta:	El nuevo código no posee errores evidentes en la ejecución de las pruebas unitarias y de integración.

★ **Escenario Refinado 6**

Identificador: ERF-06	
Escenario (s):	●**MNT-03:** Un componente es liberado en un contenedor o plataforma externa. ●**MNT-02:** Una nueva versión de **#LaraBot** es liberada en un ciclo de desarrollo. Cuando esta liberación ocurre los procesos de integración continua toman el código de los repositorios de software de los componentes liberados de **#LaraBot** y realizan un despliegue del código

		hacia un contenedor o plataforma (**#ChatPlatform**), etiquetando el software con la nueva versión.
Objetivos de negocio:		●**OBJN-19:** El sistema debe poder registrar errores que entreguen una trazabilidad al desarrollador sobre excepciones y su contexto. ●**OBJN-14**: La comunicación con el chatbot, los servicios e interfaces web deben usar encriptación segura HTTPS. ●**OBJN-18:** El sistema debe tener la capacidad de enviar una notificación global a todos los usuarios sobre novedades o actualizaciones. ●**OBJN-20:** El sistema debe integrar herramientas que permitan la integración y despliegue continuo.
Atributos de calidad:		Mantenibilidad
Componentes	Estímulo:	Liberar versiones nuevas de **#LaraBot** de forma controlada y automática.
	Fuente del estímulo:	Usuarios y desarrolladores
	Entorno:	Repositorio de software y servidores de despliegue.
	Artefacto:	Código del bot
	Respuesta:	El software se pudo desplegar en los contenedores o plataformas externas.
	Medida de la respuesta:	Cuánto esfuerzo requiere hacer el despliegue de código nuevo en los contenedores o plataformas del sistema.

★ **Escenario Refinado 7**

Identificador: ERF-07	
Escenario (s):	**EXT-03:** Un Stakeholder desea usar el bot en una plataforma de chat externa nueva. Esta plataforma es aceptada por los desarrolladores que evalúan sus necesidades específicas. Los desarrolladores usando un patrón como el Strategy implementan una estrategia nueva para la ejecución de código específico de la nueva plataforma.
Objetivos de negocio:	●**OBJN-11**: El chatbot debe poder recibir mensajes desde diferentes canales de comunicación cómo Slack, Microsoft Teams, Google Chat o desde una petición HTTP vía terminal. ●**OBJN-12:** El chatbot debe permitir integrar nuevos canales de comunicación de manera flexible. ●**OBJN-15:** Ofrecer al cliente la capacidad de autenticarse con plataformas externas mediante una interfaz web si es requerido. ●**OBJN-28:** En plataformas de chat externas debe ser posible interactuar con LaraBot vía mensaje directo o en canales públicos.

Atributos de calidad:		Extensibilidad
Componentes	Estímulo:	Integrar plataformas y canales nuevos al bot
	Fuente del estímulo:	Usuarios administradores y organizaciones
	Entorno:	**#ChatPlatform** e **#InputGateway**
	Artefacto:	Instancias del bot en las plataformas
	Respuesta:	El soporte para la plataforma de chat aceptado es aceptado e implementado
	Medida de la respuesta:	Cuánto esfuerzo involucra la adición de una nueva plataforma de chat externa en el sistema

★ **Escenario Refinado 8**

Identificador: ERF-08		
Escenario (s):		**DIS-01:** Más de 100 usuarios usan el bot en diferentes horarios y días de la semana.
Objetivos de negocio:		**OBJN-30:** Estar disponible 24x7 consumiendo la menor cantidad de recursos posibles.
Atributos de calidad:		Disponibilidad
Componentes	Estímulo:	Tener un sistema con alta disponibilidad
	Fuente del estímulo:	Usuarios del sistema
	Entorno:	Sistema interno del bot, principalmente los módulos de **#InputGateway**, **#IntentDetector** y **#BotEngine.**
	Artefacto:	Sistema
	Respuesta:	El tiempo de actividad de todos los subsistemas tuvo disponibilidad mayor o igual a 99%.
	Medida de la respuesta:	**PDS=** Porcentaje de disponibilidad. **TTT=** Total tiempo activo transcurrido desde el inicio. **TTI=** Total tiempo acumulado de inactividad por fallos. **PDS=** (TTT –TTI) / TTT

Importancia Media:

★ **Escenario refinado 9**

<table>
<tr><td colspan="3">Identificador: ERF-09</td></tr>
<tr><td colspan="2">Escenario (s):</td><td>●UAC-06: El usuario solicita listar las acciones/comandos disponibles en el sistema.
●UAC-07: El usuario ejecuta una acción que no existe o no puede ser asociada.</td></tr>
<tr><td colspan="2">Objetivos de negocio:</td><td>●OBJN-02: Presentarse como un chatbot cordial ante el usuario, mediante recursos como un saludo o una despedida.
●OBJN-05: Disponer de un comando de ayuda que permita al chatbot explicar sus funciones, limitaciones y capacidades; lo que brinda al usuario una idea clara del alcance que posee el chatbot a la hora de brindar ayuda.
●OBJN-06: Si la intención del usuario no se entiende ofrecer comandos de ayuda o recomendaciones sobre qué puede hacer el chatbot.</td></tr>
<tr><td colspan="2">Atributos de calidad:</td><td>Usabilidad</td></tr>
<tr><td rowspan="6">Componentes</td><td>Estímulo:</td><td>Dar al usuario las herramientas para comprender las capacidades y acciones que se pueden usar con el bot.</td></tr>
<tr><td>Fuente del estímulo:</td><td>Usuarios del sistema</td></tr>
<tr><td>Entorno:</td><td>Ejecución de acciones</td></tr>
<tr><td>Artefacto:</td><td>El sistema</td></tr>
<tr><td>Respuesta:</td><td>El sistema retorna una respuesta con las diferentes opciones de comando que pueden ser ejecutadas.</td></tr>
<tr><td>Medida de la respuesta:</td><td>Capacidad del sistema de entregar alternativas adecuadas a lo que se le solicita.</td></tr>
</table>

★ E**scenario refinado 10**

<table>
<tr><td colspan="2">Identificador: ERF-10</td></tr>
<tr><td>Escenario (s):</td><td>SEG-01: El sistema genera semanalmente un respaldo de DB encriptada validado y comprobado.
SEG-02: Un fallo ocasiona la corrupción de la base de datos y se usa el respaldo.</td></tr>
</table>

Objetivos de negocio:		● **OBJN-17:** El chatbot debe poder almacenar contextos de las conversaciones con los usuarios en alguna base de datos. ● **OBJN-23:** Integrar sistema de respaldo (*backup*) para las bases de datos. ● **OBJN-24:** La información en las bases de datos debe estar encriptada en disco.
Atributos de calidad:		Seguridad e Integridad
Componentes	Estímulo:	Tener una base de datos segura y confiable con información de contexto
	Fuente del estímulo:	Usuarios del sistema y desarrolladores
	Entorno:	**#IntentDetector**
	Artefacto:	Base de datos e información de contexto del usuario
	Respuesta:	Base de datos integra con respaldos
	Medida de la respuesta:	Integridad de la información del usuario y de contexto almacenada en las bases de datos

★ **Escenario refinado 11**

Identificador: ERF-11		
Escenario (s):		**SEG-04:** El administrador de la organización realiza la instalación de la instancia del *bot* en una plataforma externa. El *bot* verifica la información de la organización y su veracidad.
Objetivos de negocio:		● **OBJN-02:** Presentarse como un chatbot cordial ante el usuario, mediante recursos como un saludo o una despedida. ● **OBJN-13:** Se debe proveer una página de aterrizaje que permita a los usuarios descubrir a **#LaraBot** vía web y conocer sus bondades. ● **OBJN-28:** En plataformas de chat externas debe ser posible interactuar con **#LaraBot** vía mensaje directo o en canales públicos.
Atributos de calidad:		Seguridad
	Estímulo:	Permitir a un usuario instalar **#LaraBot** en su plataforma de chat externa.
	Fuente del estímulo:	Usuario administrador del sistema.
	Entorno:	**#WebApp** y **#ChatPlatform**
	Artefacto:	Instancia del *bot* para la plataforma externa

Componentes	Respuesta:	El bot es adicionado al chat de la organización y se presenta.
	Medida de la respuesta:	Se puede adicionar una instancia del bot en un chat de forma simple

★ **Escenario refinado 12**

Identificador: ERF-12		
Escenario (s):		**INT-01:** El usuario establece una conexión con el bot al autenticarse y usar **#ChatPlatform**. El bot recibe los mensajes y despliega las opciones que este posee de forma íntegra.
Objetivos de negocio:		● **OBJN-05:** Disponer de un comando de ayuda que permita al chatbot explicar sus funciones, limitaciones y capacidades; lo que brinda al usuario una idea clara del alcance que posee el chatbot a la hora de brindar ayuda. ● **OBJN-08:** Ofrecer la capacidad de dar retroalimentación a los desarrolladores vía chatbot que permita canalizar quejas, reclamos o sugerencias. ● **OBJN-15:** Ofrecer al cliente la capacidad de autenticarse con plataformas externas mediante una interfaz web si es requerido. ● **OBJN-13:** Se debe proveer una página de aterrizaje que permita a los usuarios descubrir a **#LaraBot** vía web y conocer sus bondades. ● **OBJN-28:** En plataformas de chat externas debe ser posible interactuar con **#LaraBot** vía mensaje directo o en canales públicos.
Atributos de calidad:		Interoperabilidad y usabilidad del sistema
Componentes	Estímulo:	Permitir a un usuario autenticarse y usar el bot
	Fuente del estímulo:	Usuarios del sistema
	Entorno:	**#WebApp** y **#ChatPlatform**
	Artefacto:	Sistema web y chat
	Respuesta:	El usuario pudo autenticarse e interactuar con **#LaraBot**
	Medida de la respuesta:	Los accesos del sistema se adaptan a las necesidades o preferencias particulares de los usuarios permitiéndoles usar el bot.

★ **Escenario refinado 13**

Identificador: ERF-13		
Escenario (s):		**EXT-04:** El administrador modifica de forma válida la configuración del bot en **#WebApp**.
Objetivos de negocio:		●**OBJN-03:** Proporcionar un conjunto mínimo de atributos configurables en el bot, como acciones permitidas o valores predeterminados. ●**OBJN-08:** Ofrecer la capacidad de dar retroalimentación a los desarrolladores vía chatbot que permita canalizar quejas, reclamos o sugerencias. ●**OBJN-13:** Se debe proveer una página de aterrizaje que permita a los usuarios descubrir a **#LaraBot** vía web y conocer sus bondades.
Atributos de calidad:		Extensibilidad
Componentes	Estímulo:	Tener un sistema que posea un mínimo número de características configurables.
	Fuente del estímulo:	Usuario administrador
	Entorno:	**#WebApp**
	Artefacto:	Formularios de configuración
	Respuesta:	El sistema acepta esta modificación y cambia la configuración del bot notificando al administrador de que la modificación fue exitosa.
	Medida de la respuesta:	Los valores configurables del sistema se adaptan a las necesidades o preferencias particulares del administrador permitiéndoles cambiarlo exitosamente.

★ **Escenario Refinado 14**

Identificador: ERF-14	
Escenario (s):	**UAC-03:** En un canal público se le pide al *bot* ejecutar una acción permitida, el sistema analiza el mensaje lo asocia a la acción correspondiente y se ejecuta.
Objetivos de negocio:	**OBJN-28:** En plataformas de chat externas debe ser posible interactuar con LaraBot vía mensaje directo o en canales públicos.
Atributos de calidad:	Usabilidad y accesibilidad

Componentes	Estímulo:	Interactuar con **#LaraBot** en canales nuevos y públicos.
	Fuente del estímulo:	Usuarios en un canal
	Entorno:	**Sistema**
	Artefacto:	Canal público en **#ChatPlatform**
	Respuesta:	El bot pudo generar una respuesta que se comunica al canal.
	Medida de la respuesta:	La respuesta es satisfactoria y el usuario obtiene lo que espera

Importancia baja:

★ **Escenario refinado 15**

Identificador: ERF-15		
Escenario (s):		**UAC-04:** El usuario menciona a otro usuario a **#LaraBot**, el sistema obtiene la información del otro usuario y muestra los diferentes comandos/acciones que pueden ser soportadas con otro usuario.
Objetivos de negocio:		● **OBJN-06:** Si la intención del usuario no se entiende ofrecer comandos de ayuda o recomendaciones sobre qué puede hacer el chatbot. ● **OBJN-09:** Recopilar analíticas de uso del chatbot sobre los comandos, preguntas, palabras, oraciones o errores más comunes capturados a partir de mensajes enviados por los usuarios. ● **OBJN-18:** El sistema debe tener la capacidad de enviar una notificación global a todos los usuarios sobre novedades o actualizaciones.
Atributos de calidad:		Usabilidad y Accesibilidad
Componentes	Estímulo:	Permitir al usuario interactuar a través de acciones con otros usuarios
	Fuente del estímulo:	Usuarios del sistema
	Entorno:	Operación normal
	Artefacto:	**#ChatPlatform** mensajes directos y canales públicos.
	Respuesta:	El sistema pudo ejecutar la acción que el usuario A solicitó y el usuario B recibe la notificación.
	Medida de la respuesta:	Capacidad del sistema de relacionar la intención del usuario A e integrar al usuario B.

★ **Escenario Refinado 16**

Identificador: ERF-16		
Escenario (s):		**UAC-05:** Un administrador con un idioma español o inglés instala la aplicación en su organización.
Objetivos de negocio:		**OBJN-10:** El chatbot debe poder hablar en inglés y español.
Atributos de calidad:		Usabilidad y Accesibilidad
Componentes	Estímulo:	Usar el sistema con idiomas en inglés y español.
	Fuente del estímulo:	Usuario administrador
	Entorno:	El sistema
	Artefacto:	**#WebApp** y **#ChatPlatform**
	Respuesta:	El bot pudo ser instalado exitosamente y dio una notificación al administrador en el idioma correspondiente.
	Medida de la respuesta:	Facilidad de instalación del bot y fiabilidad de la respuesta en el idioma correspondiente.

★ **Escenario Refinado 17**

Identificador: ERF-17		
Escenario (s):		**SEG-03:** Un cliente genera una petición HTTP sin el protocolo de seguridad HTTPS.
Objetivos de negocio:		**OBJN-14:** La comunicación con el chatbot, los servicios e interfaces web deben usar encriptación segura HTTPS.
Atributos de calidad:		Seguridad
Componentes	Estímulo:	Restringir el uso del sistema a protocolos seguros
	Fuente del estímulo:	Stakeholders
	Entorno:	Operación normal
	Artefacto:	Sistema
	Respuesta:	El sistema rechazó la petición y no permite la ejecución de ningún recurso.
	Medida de la respuesta:	Cantidad de peticiones rechazadas que el sistema no acepta al no usar protocolos seguros

★ **Escenario Refinado 18**

Identificador: ERF-18		
Escenario (s):		**UAC-02:** El usuario utiliza un sistema de chat por voz aceptado por **#LaraBot.**
Objetivos de negocio:		**OBJN-11:** El chatbot debe poder recibir mensajes desde diferentes canales de comunicación cómo Slack, Microsoft Teams, Google Chat o Chat de voz desde una petición HTTP vía terminal.
Atributos de calidad:		Usabilidad
Componentes	Estímulo:	Permitir al usuario usar la voz para interactuar con el bot
	Fuente del estímulo:	Usuarios del sistema
	Entorno:	Operación normal de acción
	Artefacto:	Sistema voz a texto en **#ChatPlatform**
	Respuesta:	**#ChatPlatform** envió el texto del mensaje proveniente de la voz y el sistema lo aceptó de forma normal
	Medida de la respuesta:	Capacidad recibir el texto y enviar este como mensaje en un escenario normal del bot

4.6.2.6. Priorización de las motivaciones arquitectónicas

El arquitecto líder junto a los stakeholders priorizan la lista de requisitos funcionales y restricciones de diseño, dándoles un valor de importancia como: Alta, Media o Baja.

★ **Requisitos Funcionales:**

Importancia Alta:

- RF-09: Identificación de la intención del mensaje.
- RF-01: Instalar #LaraBot en interfaces de chats externas.
- RF-03: Interactuar con #LaraBot en interfaces de chats externas.
- RF-11: Ejecución de comandos/acciones.
- RF-12: Generación de respuestas del bot según resultados.
- RF-02: Autenticación del usuario en plataforma de chat externa.

Importancia Media:

- RF-04: Interactuar con **#LaraBot** desde una interfaz de chat HTTP.
- RF-07: Creación y eliminación de comando/acciones.
- RF-10: Identificación de las entidades del mensaje.

- RF-08: Configuración y parametrización del sistema.

Importancia Baja:

- RF-05: Uso del sistema por usuarios anónimos.
- RF-06: Almacenamiento de información básica del usuario.

★ **Requisitos no Funcionales:**

Importancia Alta:

- RNF-01: Recepción y estructuración del mensaje.
- RNF-02: Implementación de servicios para NLP.
- RNF-03: Envío de respuesta al usuario.
- RNF-04: Uso de un proveedor de servicio.
- RNF-06: Herramientas de integración continua.
- RNF-08: Uso de funciones cloud o lambdas.
- RNF-10: Tiempo de respuesta menor a 5 segundos.
- RNF-16: Integridad de la información.
- RNF-18: Respaldo de las bases de datos.
- RNF-19: Disponibilidad.

Importancia Media:

- RNF-05: Uso de contenedores.
- RNF-09: Versionamiento de repositorios de software.
- RNF-11: Usar serverless framework o herramientas similares.
- RNF-12: Usar patrón de diseño strategy.
- RNF-13: Implementación de servicios en API Rest.
- RNF-14: Internacionalización.
- RNF-15: Soporte de navegadores web.

Importancia Baja:

- RNF-07: Pruebas unitarias y de integración.
- RNF-17: Encriptar la base de datos.

★ **Restricciones de Diseño (Sección 4.5.3):**

Importancia Alta:

- RD-04 Solo canales de comunicación HTTPS
- RD-02: Texto como única fuente

Importancia Media:

- RD-01: Desarrollo web

Importancia Baja:

- RD-03: Soporte multi lenguaje a español e inglés.

4.6.3. Resultados de QAW-A

Siendo QAW-A un método para la obtención de escenarios de calidad, su aplicación permite obtener como salidas:

- Lista priorizada de escenarios consolidados (Sección 4.6.2.4).
- Lista priorizada de escenarios refinados (Sección 4.6.2.5).
- Lista priorizada de motivantes arquitectónicos (Sección 4.6.2.6).

Los escenarios de calidad obtenidos son un mecanismo de análisis arquitectónico que permiten expresar los objetivos los atributos de calidad en términos simples que proveen al arquitecto la comprensión necesaria para materializar de manera idónea, lo requerido por el sistema en el diseño de la arquitectura.

5. CAPÍTULO VI: DESARROLLO METODOLÓGICO

FASE 2 - DISEÑO (ADD-A)

Se procede a desarrollar el diseño del sistema del caso de estudio #LaraBot usando el método ADD-A según lo establece la metodología SEI-A.

En esta fase se toman los resultados obtenidos en el QAW-A y se procede a realizar un proceso de diseño basado en la descomposición iterativa del sistema. En cada iteración se aplican patrones y tácticas arquitectónicas con el fin de satisfacer los atributos de calidad.

5.1. Involucrados en ADD-A

Según la metodología SEI-A la fase de diseño es responsabilidad de los arquitectos, los cuales los cual son:

- **Arquitecto Líder:** Sergio Alexander Flórez Galeano.
- **Arquitecto Líder:** Nelson Orlan Escobar Ceballos.

5.2. Despliegue del método ADD-A

Según lo descrito por la metodología ADD-A en la sección **3.1.6**, se procede a realizar una descomposición del sistema de forma iterativa. Sin embargo, antes de iniciar las iteraciones es importante por parte de los arquitectos verificar que en la etapa de análisis en QAW-A se hayan obtenido cómo mínimo los siguientes resultados:

- Escenarios de atributos de calidad refinados y priorizados (sección **4.6.2.5**).
- Motivaciones arquitectónicas priorizadas (sección **4.6.2.6**).

Figura 54: Plantilla de la documentación de una vista [35].

Habiendo destacado estos elementos que han sido previamente consolidados, se aclara que parte de los entregables generados en estos procesos iterativos serán material contemplado como procesos de documentación de la arquitectura. Cada uno de estos procesos iterativos generarán elementos que pueden ser usados como documentación del sistema.

Las vistas generadas en esta etapa nos permiten dividir la arquitectura en varias representaciones interesantes y manejables del sistema. La intención de crear una vista u otra está normalmente impulsada por la necesidad de documentar un patrón particular en el diseño, en este caso se hace en base a las motivaciones arquitectónicas expuestas en la sección **4.6.2.6**.

Como se puede ver en la siguiente plantilla del SEI del libro *Views and Beyond* gran parte de los elementos de una vista documentada pueden ser generados en este proceso:

Esto se tomará como referencia en la futura etapa de documentación del sistema, que hará referencia a las vistas generadas a partir de los diferentes diagramas y catálogos de elementos priorizados en la etapa de ADD-A.

5.2.1. ITERACIÓN #1

La iteración inicial del método ADD-A indica que el primer elemento a descomponer debe ser el propio sistema que se está modelando. Tomando al sistema como punto de partida se puede descomponer este en diferentes módulos o subsistemas que permitirán generar componentes a través del proceso iterativo de cada uno de estas subelementos.

En esta primera iteración se le asignan todos los motivantes arquitectónicos al sistema, buscando como objetivo tener una visión más completa del Caso de Estudio.

5.2.1.1. Priorización de las motivaciones arquitectónicas

Previamente en la sección **4.6.2.6** se listaron las diferentes motivaciones arquitectónicas del sistema y se les asignaron unos valores de importancia (alto, medio y bajo) basados en el interés de implementación de cada uno de estos, de igual forma en la sección 4.6.2.5 se hizo el refinamiento de escenario con su correspondiente asignación de prioridad.

Como parte de la primera iteración se deben asignar todos los motivantes arquitectónicos y priorizarlos, ya que todos hacen parte del funcionamiento del sistema. En las siguientes iteraciones se dividirán estos motivantes por cada uno de sus correspondientes módulos de impacto.

Teniendo las motivantes arquitectónicos clasificados por valor de importancia se procede a asignar un valor de impacto. Esto es necesario ya que una motivación puede no ser tan importante en la conceptualización, pero por el contrario generar un impacto alto en el componente. Teniendo esto en cuenta se despliega la siguiente tabla con la asignación de valores de impacto y priorización de cada una de estas motivaciones.

Motivaciones Arquitectónicas	Valor de Importancia (Alta, Media, Baja)	Valor de Impacto (Alto, Medio, Bajo)
RF-09: Identificación de la intención del mensaje	Alta	Alto
RF-01: Instalar #LaraBot en interfaces de chats externas	Alta	Medio
RF-03: Interactuar con el bot en chats externos	Alta	Alto
RF-11: Ejecución de comandos/acciones	Alta	Alto
RF-12: Generación de respuestas del bot según resultados	Alta	Medio
RF-02: Autenticación del usuario en plataforma externa	Alta	Alto
RF-04: Interactuar con el bot en interfaz de chat HTTP	Media	Alto
RF-07: Creación y eliminación de comandos/acciones	Media	Alto
RF-10: Identificación de las entidades del mensaje	Media	Alto
RF-05: Uso del sistema por usuarios anónimos	Baja	Bajo
RF-06: Almacenamiento de información básica del usuario	Baja	Medio
RF-08: Configuración y parametrización del sistema	Media	Alto
RD-04: Solo canales de comunicación HTTPS	Alta	Medio
RD-02: Texto como única fuente de mensajes	Alta	Alto
RD-01: Desarrollo web	Media	Medio
RD-03: Soporte multi lenguaje a español e inglés.	Baja	Alta
ERF-01: Recepción del mensaje del usuario	Alta	Alto
ERF-02: Integración de nuevos motores de intenciones	Alta	Alto
ERF-03: Adición de nuevas acciones/comandos al bot	Alta	Alto
ERF-04: Escalar los recursos del sistema según su uso	Alta	Alto
ERF-05: Control de pruebas unitarias y de integración	Media	Alto
ERF-06: Liberación de versiones nuevas del software de forma controlada y automática.	Alta	Alto
ERF-07: Integración de plataformas/canales nuevos al bot	Alta	Medio
ERF-08: Sistema con alta disponibilidad	Alta	Alto

ERF-09: Dar al usuario las herramientas para comprender las capacidades y acciones que se pueden usar en el bot.	Media	Bajo
ERF-10: Base de datos segura y confiable con contexto.	Media	Alto
ERF-11: Instalación del bot en plataforma de chat externa.	Media	Medio
ERF-12: Permitir a un usuario autenticarse y usar el bot	Media	Alto
ERF-13: Sistema mínimo de características configurables	Media	Alto
ERF-14: Interactuar en canales nuevos y públicos	Media	Medio
ERF-15: Interacción multiusuario a través de acciones	Baja	Medio
ERF-16: Interacción con el sistema en inglés y/o español	Baja	Alto
ERF-17: Restringir el uso del sistema a protocolos seguros	Baja	Alto
ERF-18: El usuario interactúa con el sistema a través de aplicación de voz.	Baja	Bajo

Tabla 7: Motivaciones arquitectónicas priorizadas.

Con los valores priorizados se toma en cuenta que los grupos con más relevancia son aquellos con importancia alta, impacto alto o una combinación de ambos. Estas motivaciones son las cuales tendrán la priorización más alta en el sistema.

5.2.1.2. Descomposición del sistema

Elemento	Motivantes Arquitectónicos Asociados
Interfaz de chat o fuente de entrada (#ChatPlatform)	**Requisitos Funcionales (Ver sección 4.5.1):** RF-01, RF-03, RF-11, RF-02, RF-05 **Restricciones de Diseño (Ver sección 4.5.3):** RD-04, RD-02, RD-03 **Escenarios de Calidad (Ver sección 4.6.2.5):** ERF-01, ERF-05, ERF-06, ERF-07, ERF-11, ERF-12, ERF-14, ERF-15, ERF-16, ERF-17, ERF-18
Aplicación web (#WebApp)	**Requisitos Funcionales (Ver sección 4.5.1):** RF-11, RF-02, RF-04, RF-05, RF-08 **Restricciones de Diseño (Ver sección 4.5.3):** RD-04, RD-02, RD-01, RD-03 **Escenarios de Calidad (Ver sección 4.6.2.5):** ERF-05, ERF-06, ERF-07, ERF-08, ERF-09, ERF-12, ERF-13, ERF-16, ERF-17
Puerta o interfaz de entrada del sistema (#InputGateway)	**Requisitos Funcionales (Ver sección 4.5.1):** RF-03, RF-11, RF-02, RF-04, RF-05, RF-06 **Restricciones de Diseño (Ver sección 4.5.3):** RD-04, RD-02 **Escenarios de Calidad (Ver sección 4.6.2.5):** ERF-01, ERF-04, ERF-05, ERF-06, ERF-08, ERF-10, ERF-11, ERF-12,

	ERF-17
Detector de intención (#IntentDetector)	**Requisitos Funcionales (Ver sección 4.5.1):** RF-09, RF-11, RF-07, RF-10, RF-06 **Restricciones de Diseño (Ver sección 4.5.3):** RD-04, RD-02, RD-03 **Escenarios de Calidad (Ver sección 4.6.2.5):** ERF-01, ERF-02, ERF-03, ERF-05, ERF-06, ERF-09, ERF-10, ERF-12, ERF-15, ERF-16, ERF-17
Motor conversacional (#BotEngine)	**Requisitos Funcionales (Ver sección 4.5.1):** RF-11, RF-12, RF-07, RF-10, RF-06, RF-08 **Restricciones de Diseño (Ver sección 4.5.3):** RD-04, RD-02, RD-03 **Escenarios de Calidad (Ver sección 4.6.2.5):** ERF-01, ERF-03, ERF-05, ERF-06, ERF-08, ERF-09, ERF-10, ERF-12, ERF-13, ERF-15, ERF-16, ERF-17
Interfaz de Salida (#OutputInterface)	**Requisitos Funcionales (Ver sección 4.5.1):** RF-11, RF-12, RF-04, RF-08 **Restricciones de Diseño (Ver sección 4.5.3):** RD-04, RD-03 **Escenarios de Calidad (Ver sección 4.6.2.5):** ERF-01, ERF-04, ERF-05, ERF-06, ERF-07, ERF-08, ERF-09, ERF-11, ERF-12, ERF-13, ERF-14, ERF-15, ERF-16, ERF-17, ERF-18

Tabla 8: Elementos resultantes de la descomposición del sistema.

Teniendo cada uno de los módulos del sistema definidos con sus correspondientes motivantes asignados, se procede a hacer una presentación formal de sus características y comportamiento al igual que el rol que desempeñan en el sistema:

5.2.1.3. Módulos del sistema

- **Interfaz de chat o fuente de entrada (#ChatPlatform)**

  Este módulo representa la aplicación externa proveniente de otros proveedores, ejemplo Slack, Google Chat o Microsoft Teams. Representa el medio por el cual el usuario podrá interactuar con el sistema para el envío de mensajes a un API mediante el protocolo seguro de HTTP.

- **Aplicación web (#WebApp)**

  Este módulo representa tanto la plataforma para la configuración y autenticación del sistema por parte de un usuario administrador como la página de presentación del bot. Estas páginas son conocidas como páginas de aterrizaje o *landing pages* y permiten describir las características de un producto al ser la carta de presentación a un usuario. Tomando lo anterior este módulo tiene 3 responsabilidades en el sistema:

  - Habilitar la autenticación de usuarios ya sea una autenticación interna con el sistema o una autenticación en una plataforma externa de ser necesario.

- Dar una landing page permitiendo captar usuarios para presentar el bot, al igual que un pequeño módulo de chat-web.
- Brindar un panel de administración interno que permita visualizar información de usuarios registrados y la configuración de características básicas.

- **Puerta o interfaz de entrada del sistema (#InputGateway)**

  Es la interfaz de entrada al sistema interno del bot, esta se encarga de recibir los mensajes provenientes de los diferentes clientes a través de peticiones web y un sistema de API web, esta puede operar en dos estados:

  - Síncrono: El cliente es bloqueado en espera de respuesta, normalmente requerido cuando la comunicación es iniciada desde un cliente HTTP, widget de chat tercero o asistente de voz.
  - Asíncrono: El cliente no se bloquea ya que la respuesta es enviada eventualmente por el sistema del chatbot haciendo uso de las APIs en las plataformas externas. Este tipo de comunicación es usualmente usado por chat externos cómo Slack, Google Chat o Microsoft Teams. Estas respuestas suelen estar en un rango máximo de 3 a 5 segundos.

- **Detector de intención (#IntentDetector)**

  Este es el módulo *core* del sistema en el contexto de este proyecto ya que se encarga de tomar los valores de un mensaje y hacer uso de uno o más servicios agnósticos para poder identificar intencionalidad en lo que el usuario desea. Su rol principal es encontrar la intención y asociarla con una a acción en el sistema, de igual forma esta debe obtener las entidades del mensaje para entregar este valor.

- **Motor conversacional (#BotEngine)**

  Se encarga de reunir la intención, un conjunto de entidades y contexto, y a partir de estos parámetros ejecuta un controlador (handler) asociado a la intención detectada para generar una respuesta.

- **Interfaz de Salida (#OutputInterface)**

  Se encarga de ser el puente de comunicación para enviar las respuestas del sistema abstrayendo los formatos específicos de cada plataforma externa.

5.2.1.4. Definición de la interacción de módulos

- **#ChatPlatform <==> #WebApp**

  Dependiendo de las necesidades especiales de la plataforma puede ser necesaria la comunicación con **#WebApp** para realizar la autenticación del usuario y permitirle usar el bot.

- **#ChatPlatform --> #InputGateway**

Teniendo la instancia del bot instalada **#ChatPlatform** procede a hacer peticiones HTTP con protocolo seguro a un *endpoint* en **#InputGateway**. La información que el chat enviará puede contener información de sesión dependiendo de si el usuario está o no autenticado.

- **#WebApp --> #InputGateway**

  La aplicación web **#WebApp** realiza todas sus interacciones con el sistema a través de la API expuesta por **#InputGateway** éstas se hacen a través de petición HTTP seguras, estas pueden ser:

  - Autenticación de un usuario.
  - Cambiar valores de configuración del sistema.
  - visualizar información de usuarios registrados en base de datos.
  - Logs de eventos realizados.
  - Envío de retroalimentación.
  - Interacciones básicas con el bot.

- **#InputGateway --> #IntentDetector**

  Como se explicó anteriormente **#InputGateway** opera de forma síncrona o asíncrona dependiendo de la naturaleza de la petición recibida. Idealmente este módulo de entrada estará orientado al funcionamiento por eventos (serverless) y esta estructura el mensaje antes de ser entregado en tiempo de ejecución a **#IntentDetector** esta estructuración del mensaje contemplada:

  - Formalización del formato recibido por un plataforma específica o petición HTTP.
  - Adición del contexto del mensaje como información del usuario si esto aplica.
  - Menciones a otros usuarios.
  - Elementos y multimedia en caso de que sea necesario (opcional).
  - Idioma del mensaje.
  - Mensaje en texto plano limpio.

- **#IntentDetector --> #BotEngine**

  El módulo de detección de intención toma el mensaje estructurado y formaliza esta información en un componente que se encarga de traducir los valores de envío y respuesta de los servicios cognitivos, se toma un conjunto de frases para relacionar el mensaje con una acción se envían dos elementos al **#BotEngine**:

  - Intención relacionada (entidad) a la acción correspondiente.
  - Entidades identificadas en el mensaje.

- **#BotEngine --> #OutputInterface**

  El motor conversacional es un conjunto de componentes que toman la entidad identificada y sus entidades para la ejecución de un controlador(handler). Los resultados de la ejecución

de la acción en el controlador se usan para generar una respuesta enviada por el **#OutputInterface**

- **#OutputInterface --> #ChatPlatform**

  El rol de **#OutputInterface** es similar al módulo de entrada del sistema. En este caso el módulo se encarga de convertir las respuestas del sistema del bot a una en el formato aceptado por la plataforma externa.

  Estos formatos pueden incluir elementos específicos del sistema como un gráfico, imágenes, botones entre otros. Estas respuestas pueden ser enviadas en un formato tipo markdown, HTML, texto plano, objetos JSON, entre otras estructuras de datos especificadas por **#ChatPlatform.**

5.2.2. ITERACIÓN #2: #ChatPlatform

A Partir de este punto se realizarán iteraciones por cada uno de los subsistemas/módulos identificados en la iteración #1, esto permitirá que en cada iteración se explique el flujo de interacción con el sistema desde los cambios realizados en el cliente hasta la recepción y posterior envío de respuestas de los módulos internos del sistema.

5.2.2.1. Elegir un elemento del sistema a descomponer

La elección del elemento a descomponer en esta iteración se hace en base en priorizar la interacción con el usuario como primera fuente de estímulos del sistema. Esto está vinculado a los atributos de calidad de Usabilidad y accesibilidad. Teniendo en cuenta estos criterios el principal elemento interacción del usuario con el sistema es la Interfaz de chat o fuente de entrada **#ChatPlatform**, que es un módulo de tecnologías externas y específicas de un proveedor de chat.

Se desea formalizar los componentes e interacciones del elemento **#ChatPlatform** para comprender las necesidades y características de estos sistemas de chat externos. Para posteriormente vincular su interacción con los otros módulos del sistema.

5.2.2.2. Identificar candidatos a motivantes arquitectónicos principales

En la sección **5.2.1.1** de la iteración #1 se hizo una priorización inicial de los motivantes arquitectónicos del sistema y descomponiendo cada uno de estos motivantes según su respectivo módulo. De esta iteración se obtuvo el siguiente resultado de motivantes arquitectónicos con respecto a **#ChatPlatform:**

Interfaz de chat o fuente de entrada (#ChatPlatform)	**Requisitos Funcionales (Ver sección 4.5.1):** RF-01, RF-03, RF-11, RF-02, RF-05 **Restricciones de Diseño (Ver sección 4.5.3):** RD-04, RD-02, RD-03 **Escenarios de Calidad (Ver sección 4.6.2.5):** ERF-01, ERF-05, ERF-06, ERF-07, ERF-11, ERF-12, ERF-14,

	ERF-15, ERF-16, ERF-17, ERF-18

Se procede a la priorización de cada uno de estos motivantes en **#Chatplatform** para posteriormente identificar los motivantes principales, estos se ordenan de mayor a menor prioridad:

Motivaciones Arquitectónicas	**Valor de Importancia (Alta, Media, Baja)**	**Valor de Impacto (Alto, Medio, Bajo)**
RF-01: Instalar #LaraBot en interfaces de chats externas	Alta	Alto
RF-03: Interactuar con el bot en chats externos	Alta	Alto
ERF-01: Recepción del mensaje del usuario	Alta	Alto
RF-11: Ejecución de comandos/acciones	Alta	Alto
RD-02: Texto como única fuente de mensajes	Alta	Alto
RF-02: Autenticación del usuario en plataforma externa	Alta	Alto
ERF-06: Liberación de versiones controlada y automática.	Alta	Alto
ERF-07: Integración de plataformas/canales nuevos al bot	Alta	Medio
RD-04: Solo canales de comunicación HTTPS	Alta	Medio
ERF-05: Control de pruebas unitarias y de integración	Media	Alto
ERF-12: Permitir a un usuario autenticarse y usar el bot	Media	Alto
ERF-11: Instalación del bot en plataforma de chat externa.	Media	Medio
ERF-14: Interactuar en canales nuevos y públicos	Media	Medio
RD-03: Soporte multi lenguaje a español e inglés.	Baja	Alto
ERF-16: Interacción con el sistema en inglés y/o español	Baja	Alto
ERF-17: Restringir el uso del sistema a protocolos seguros	Baja	Alto
ERF-15: Interacción multiusuario a través de acciones	Baja	Medio
RF-05: Uso del sistema por usuarios anónimos	Baja	Bajo
ERF-18: El usuario interactúa con aplicación de voz	Baja	Bajo

Tabla 9: Motivantes arquitectónicas de #ChatPlatform.

Con base a la tabla anterior y la priorización de estos motivantes se procede a enumerar cada uno de los candidatos a motivantes arquitectónicos principales del módulo:

Indicador	Referencia	Criterio
RF-01	4.5.1.1	Basándose en las necesidades que posee **#ChatPlatform** se llega a la conclusión de cuales motivantes son más relevantes para que este módulo sea implementado y en especial el impacto que generan en el módulo.
RF-03	4.5.1.3	
RF-02	4.5.1.2	
ERF-12	4.6.2.5	
ERF-14	4.6.2.5	

Tabla 10: Candidatos a motivantes arquitectónicos principales para el #ChatPlatform.

5.2.2.3. Especificación estructural del #ChatPlatform

El **#ChatPlaform** como se identifica previamente es una interfaz de chat que está pensada en ser la fuente principal de interacción del usuario con el sistema. Este módulo es una implementación realizada en un ecosistema externo al sistema ya que este funciona directamente con las tecnologías que provee la plataforma de chat nativa o web.

La razón principal de generar un sistema de chat es darle las herramientas al usuario para que este pueda interactuar con el bot. De igual forma la arquitectura del bot está pensada para funcionar con escenarios del mundo real por lo que es importante concebir la instanciación de y acceso al sistema por parte plataformas de chat populares ya que en términos de objetivos de negocios en el motivante principal para la creación de un chatbot.

Finalmente, este módulo permite entender cómo afecta al sistema la creación de software alejado de un ecosistema sus necesidades. Generando una abstracción de las interfaces de chat más comúnmente usadas se identifican los siguientes elementos:

- **Plataforma de chat terceras**

  Son las plataformas que proveen el ecosistema en el cual el bot será instanciado. Este elemento en particular hace referencia a las herramientas para la instalación del bot en la plataforma específica, estos ecosistemas requieren:

  - Nombre
  - Descripción
  - Logotipo
  - Token del bot

  Los siguientes componentes del sistema hacen referencia a cada uno de esos componentes necesarios para que el bot funcione de forma apropiada.

- **Suscripción al sistema (URL de entrada)**

  Cuando se desea instalar un bot en una plataforma externa esta requiere una función de *CallBack*, que se resume en una dirección web que le permita a la plataforma hacer llamadas

al servidor del sistema. Las llamadas realizadas al sistema son los mensajes y estos son enviados a la entrada especificada en la URL.

- **Configuración de permisos**

  Algunas plataformas necesitan que se especifiquen los permisos que requiere el bot para poder funcionar, estos permisos son aquellos que le permiten al bot acceder a información o parámetros del chat, algunos ejemplos son:

  - Acceso a contenido multimedia
  - Acceso a canales
  - Acceso a las listas de usuarios
  - Escuchar llamadas o eventos de sonidos
  - Otros

  Estos permisos dependen completamente de la plataforma y las características que brinde. Para el caso concreto de **#LaraBot** basándonos en los objetivos de negocio, los permisos principales con los que debe contar son aquellos que le permitan acceder a canales públicos, responder mensajes y hacer uso de menciones.

- **Eventos de interés**

  En las plataformas se debe de especificar qué tipos de eventos pueden desencadenar una acción en el bot. Los eventos son específicos para cada plataforma como, por ejemplo, la eliminación de un canal, agregar a un usuario nuevo en el espacio de trabajo, la creación de un nuevo canal, realizar una mención, entre otros. El evento principal del cual #Larabot puede hacer uso es el evento de mención al bot que le permitan responder al usuario cuando éste lo requiera.

- **Componente de OAuth y Token**

  Algunos sistemas de chats requieren que el usuario que realiza la instalación del bot en un espacio de trabajo se autentique a través de un sistema de OAuth, un ejemplo de esto es Slack. Cuando el usuario se encuentra autenticado el bot se comunica con el usuario a través del uso de tokens que permitan dar respuestas o enviar información al usuario.

- **Mensajes y peticiones HTTP**

  El mensaje es la abstracción de los eventos y peticiones que se realizan al punto de entrada del sistema. Es el componente principal con el que se interactúa en el sistema y que posteriormente se hará todo el procesamiento correspondiente para la generación de respuestas, esta petición generada hacia el sistema contiene información relevante que de forma transversal en las diferentes plataformas debe contener los siguientes elementos:

  - Texto del mensaje.
  - Información del usuario.
  - Token.

- Canal de origen.
- Metadatos de contexto.
- Intenciones pre-procesadas.

- **Canales públicos**

  Los Canales son aquellos chats multiusuario, en los cuales se ocurren múltiples interacciones de muchos usuarios de forma simultánea. Ya que el principio de los canales es el envío de mensajes que pueden ser recibidos por cualquier miembro del canal y por ende no es un mensaje directo **#LaraBot** debe poder interactuar y solo recibir los mensajes que correspondan a un evento de interés para el *bot*.

- **Chats privados**

  Son los espacios en los cuales se envían mensajes directos hacia el sistema, cuando el usuario quiere realizar el envío de un mensaje a un bot, las plataformas crean una opción que permite escribir directamente al bot y este obtiene los mensajes directos del usuario.

- **Asistentes de voz terceros**

  Interfaces de audio a voz que permiten realizar el envío de mensajes o la ejecución de acciones sobre una aplicación. Ejemplo de esto es Amazon Alexa o Google Assistance que tiene una API que permite recibir mensajes de voz convertidos a texto para posteriormente responder sobre la misma API al software. La conversión del audio a texto es completamente ajena al sistema del bot y el software de asistencia es el encargado de implementarlo.

- **Menciones**

  En las plataformas de chat normalmente en el envío de mensajes se pueden mencionar usuarios, canales u otros elementos específicos del sistema para enriquecer la experiencia de la acción que se desea realizar al enviar un mensaje.

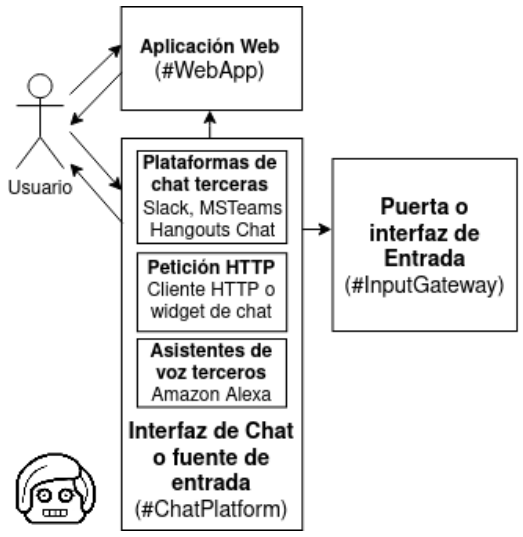

Figura 55: Descomposición de componentes del #ChatPlatform.

En el caso de la implementación, se debe tener en cuenta que al mencionar algún usuario o del mismo bot esto acciona un evento en el bot que permite recopilar la información adicional. La información adicional que se obtiene en estos casos es conocida como metadatos de contexto y puede contener información relevante sobre lo que se está mencionando. A continuación, se presenta el #ChatPlatform y sus subcomponentes internos principales, así como su relación con los demás componentes del sistema.

5.2.2.4. Especificación comportamental del #ChatPlatform

El rol principal del **#ChatPlatform** es permitir a los usuarios interactuar con el bot haciendo uso de una interfaz amigable y dinámica. El usuario utiliza **#ChatPlatform** para la ejecución de las acciones específicas que se desea realizar, a continuación, se hace una descomposición de estas diferentes acciones, así como sus respectivos gestos de interacción en cada uno de los componentes de **#ChatPlatform**:

Tabla 11: Acciones permitidas por el usuario en #ChatPlatform

Acción	Descripción	Comportamiento de los componentes del módulo (Figura 56)
Hacer la instalación de la aplicación de #LaraBot en la plataforma externa	Permite entender el proceso de adición de la instancia del bot en la plataforma de chat de un usuario.	El usuario ingresa a una plataforma de chat o directamente a #WebApp el usuario seguirá los siguientes pasos: 1. Seleccionar la instalación del bot en la plataforma 2. Se realiza el proceso de adición de bot en la plataforma y en caso de requerir una autorización se conecta con **#WebApp**. 3. Se obtiene un token de la aplicación y se almacena la instancia del bot en el chat del espacio de trabajo del usuario
Autenticar a #LaraBot en la plataforma externa	Permite al usuario realizar una autenticación para la obtención de información de permisos de ser solicitada por el chat.	Por medio de la instalación del bot o su posterior instancia se puede realizar una autenticación a la plataforma de la siguiente forma: 1. El usuario va a autenticarse por medio de OAuth. 2. Se despliega un formulario de autenticación de **#WebApp**. 3. El usuario usa sus credenciales y #WebApp retorna un token de autorización.
Visualizar la lista de acciones del sistema	Da al usuario las herramientas para comprender las capacidades y acciones que se pueden usar en el bot.	El usuario puede observar un catálogo de acciones disponibles que se pueden ver en la página de aterrizaje.
Enviar mensajes directos al bot y	Da al usuario la capacidad de	Dependiendo de la plataforma el cliente debe de realizar el envío de algún token que permita autorizar al usuario,

ejecutar acción	interactuar con el bot de forma directa a través del envío de texto	para todos los casos concretos funciona de forma similar: 1. El usuario envía un mensaje al bot por chat privado directo. 2. El mensaje es formateado por la plataforma con su estructura específica. 3. Se usa el *Callback* especificado en el componente de suscripción del sistema. 4. Se envía el mensaje hacia la URL especificada por medio de una petición HTTP segura. 5. Se obtiene una respuesta de forma síncrona o asíncrona dependiendo de la plataforma externa.
Mencionar a otro usuario en una acción	Integra las acciones multiusuario con el bot, permitiendo realizar cambios o comportamientos específicos	La implementación de acciones en el bot se realiza bajo demanda, integrando una acción a la vez, pero de forma general las acciones multiusuario funcionan siguiendo el siguiente comportamiento: 1. Se menciona a un usuario en un contexto con el bot. 2. En caso de no poseer información suficiente, se muestra un listado con las acciones disponibles que involucran otros usuarios. 3. Se envía por medio de una petición HTTP segura al sistema y ejecuta una acción con otro usuario. 4. Se obtiene una respuesta y si la acción lo amerita se le notifica al otro usuario.
Mencionar al bot en un mensaje dentro de un canal público.	Permite al bot interactuar con diferentes usuarios en un canal público y ejecutar acciones específicas de los canales	1. El usuario menciona al bot en un canal público 2. El sistema del chat ejecuta un evento en caso de que la instancia del bot tenga permisos se continúa con normalidad. 3. El evento dispara un llamado al *Callback* del sistema y se envía una petición HTTP segura con el contexto del usuario y el identificador del canal. 4. El sistema responde con el identificador del canal y el resultado de la acción.
Ejecutar acciones en plataforma externa que disparan eventos al bot	Da al bot la capacidad de ejecutar acciones relaciona a eventos específicos de la plataforma	En caso de poseer los permisos necesarios sobre un evento (ejemplo la eliminación de un canal), el bot será notificado con la información del evento para poder ejecutar alguna acción específica. Algunos de los eventos más usados son: ● Bot es añadido a un canal. ● Bot es eliminado de un canal. ● Un usuario es invitado a un canal. ● Un usuario es eliminado de un canal. ● Un canal es renombrado. ● Un canal es eliminado o archivado. ● Usuario edita mensaje enviado al bot.
Remover el bot del chat	Permite al usuario remover la instancia del bot de su espacio de trabajo	La plataforma externa se encarga de remover la instancia del bot del espacio de trabajo de usuario, en caso del bot contar con los permisos se dispara un evento notificando al sistema de la eliminación de la instancia.

5.2.2.5. Diagrama de Casos de Uso

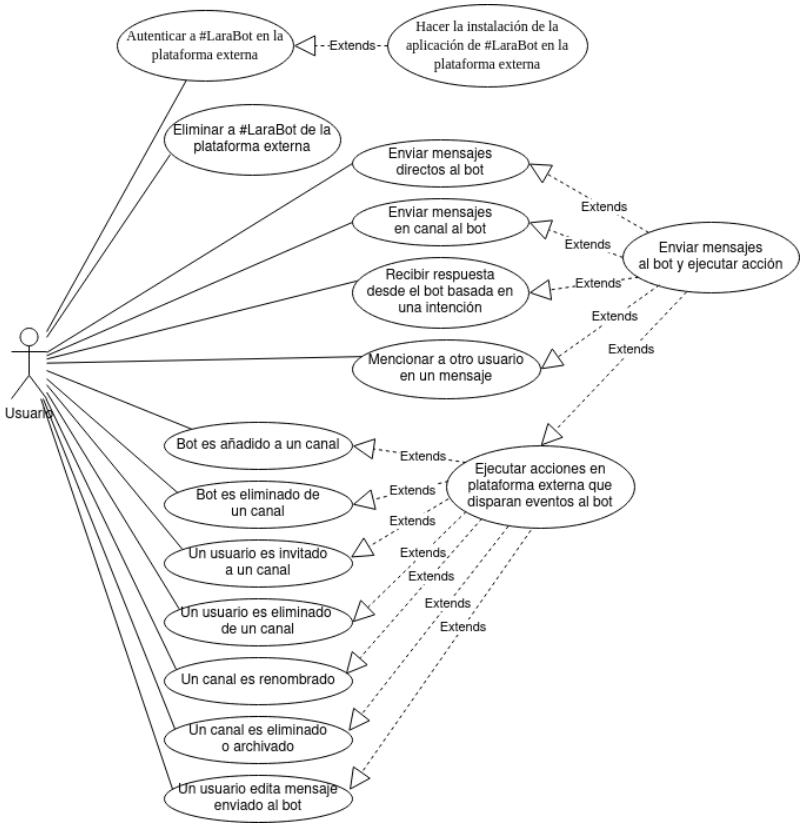

Figura 56: Diagrama de casos de uso del #ChatPlatform.

5.2.3. ITERACIÓN #3: #WebApp

5.2.3.1. Elegir un elemento del sistema a descomponer

Para conservar la consistencia y el flujo de interacción del usuario con el sistema, se elige continuar en esta iteración con el módulo web **#WebApp** que permite darle al usuario acceso al bot y conocer sus características.

5.2.3.2. Identificar candidatos a motivantes arquitectónicos principales

Similar a la iteración anterior en esta se toman los resultados de los motivantes priorizados en la iteración #1 y se listan de la siguiente forma:

Aplicación web (#WebApp)	**Requisitos Funcionales (Ver sección 4.5.1):** RF-11, RF-02, RF-04, RF-05, RF-08 **Restricciones de Diseño (Ver sección 4.5.3):** RD-04, RD-02, RD-01, RD-03 **Escenarios de Calidad (Ver sección 4.6.2.5):** ERF-05, ERF-06, ERF-07, ERF-08, ERF-09, ERF-12, ERF-13, ERF-16, ERF-17

Realizando el desglose de estos motivantes arquitectónicos según la priorización en **#WebApp** se obtiene la siguiente tabla:

Motivaciones Arquitectónicas	**Valor de Importancia (Alta, Media, Baja)**	**Valor de Impacto (Alto, Medio, Bajo)**
RD-02: Texto como única fuente de mensajes	Alta	Alto
RF-02: Autenticación del usuario en plataforma externa	Alta	Alto
RF-11: Ejecución de comandos/acciones	Alta	Alto
ERF-06: Liberación de versiones nuevas del software de forma controlada y automática.	Alta	Alto
ERF-08: Sistema con alta disponibilidad	Alta	Alto
ERF-09: Dar al usuario las herramientas para comprender las capacidades y acciones que se pueden usar en el bot.	Media	Alto
RF-04: Interactuar con el bot en interfaz de chat HTTP	Media	Alto
RD-04: Solo canales de comunicación HTTPS	Alta	Medio
RF-08: Configuración y parametrización del sistema	Media	Alto
ERF-07: Integración de plataformas/canales nuevos al bot	Alta	Medio
ERF-12: Permitir a un usuario autenticarse y usar el bot	Media	Alto
ERF-13: Sistema mínimo de características configurables	Media	Alto
RD-01: Desarrollo web	Media	Medio
ERF-05: Control de pruebas unitarias y de integración	Media	Alto
RD-03: Soporte multi lenguaje a Español e Inglés.	Baja	Alta
ERF-16: Interacción con el sistema en inglés y/o español	Baja	Alto
ERF-17: Restringir el uso del sistema a protocolos seguros	Baja	Alto
RF-05: Uso del sistema por usuarios anónimos	Baja	Bajo

Tabla 12: Motivantes arquitectónicas de #WebApp.

Se procede a hacer un análisis de los motivantes arquitectónicos más relevantes para encontrar aquellos que aportan e impactan fuertemente el módulo de #WebApp y que son fundamentales para su posterior implementación y descomposición:

Indicador	Referencia	Criterio
RF-02	5.5.1.2	Según los valores que #WebApp aporta al sistema, se identifican necesidades como la captación de público, configuración del sistema y el acceso a la bot de forma consistente y ergonómica, basándose en esto se procedieron a elegir los respectivos motivantes arquitectónicos.
RF-11	5.5.1.11	
RF-08	5.5.1.8	
ERF-09	5.6.2.5	
ERF-07	5.6.2.5	

Tabla 13: Candidatos a motivantes arquitectónicos principales para el #WebApp.

Con los ASRs identificados en este módulo según el criterio especificado en la tabla anterior se procede a realizar una descomposición del módulo según esta priorización.

5.2.3.3. Especificación estructural del #WebApp

El **#WebPlaform** es el punto de entrada al bot de cara al usuario, este es un módulo web que puede ser accedido a través de www.larabot.com. El objetivo principal de este es permitirle al usuario entender que es el bot, cuáles son sus características y brindarle herramientas para que este se guie en cómo puede usarlo de forma adecuada.

Este módulo puede ser accedido a través de un navegador web y de peticiones directas del chat externo del sistema. Está implementado con tecnologías basadas en web ya que son estas las que funcionan de mejor forma en los navegadores. Tomando los valores principales y ASRs que se esperan de este módulo se llega a los siguientes componentes:

- **Portal Web o landing page**

Página principal de entrada de **#WebApp**, esta es una página web con contenido estático. Solo posee la información relevante para que el usuario comprenda las capacidades del sistema, en esta debe poderse visualizar información como la última versión del *bot*, datos de éxito, lista de acciones y una numeración de las distintas plataformas aceptadas en el su momento por el bot

- **Formularios o chat simple**

Formularios específicos del sistema que se pueden usar para la ejecución de un recurso en el sistema o almacenamiento de información, esto incluye:

- Formulario de registro de usuarios nuevos
- Formulario de retroalimentación del sistema
- Chat con características pequeñas para enviar mensajes al bot
- Cerrar sesión en el sistema
- Otras futuras especificaciones

- **Interfaz de autenticación**

Por medio de **#WebApp** se puede realizar la autenticación de un usuario, ya sea de forma directa o por medio de un proceso de OAuth. Este es un formulario que permite la integración de plataformas externas en **#ChatPlatform** al sistema de autenticación de bot habilitando un proceso de *OAuth* que debe generar un token de autorización para la posterior identificación del usuario. Este token es retornado a la plataforma que lo está requiriendo o almacenado en el navegador para habilitar una sesión en **#WebApp** con el sistema.

- **Panel de administración**

Componente de #WebApp que permite realizar modificaciones específicas del sistema y visualizar información de las instancias del bot instaladas por un usuario administrador. Este panel incluye los módulos de configuración y parametrización del bot para un usuario específico, así como su historial de acción ejecutadas. Este módulo está diseñado para soportar formularios de configuración o sistemas sensibles para el funcionamiento del bot. A continuación, se presenta el #WebApp y sus componentes internos principales, así como su relación con los demás componentes del sistema.

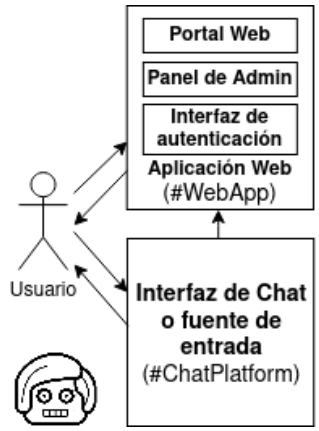

Figura 57: Descomposición de componentes de la #WebApp.

5.2.3.4. Especificación comportamental del #WebApp

El rol principal de #WebApp en términos de comportamiento es brindar al usuario un punto de referencia para conocer al bot y habilitar una interfaz de autenticación a los sistemas externos que requieran un resultado de autenticación con el sistema y permite la ejecución de formulario específicos. Teniendo esto en cuenta se llega a la conclusión de las siguientes acciones permitidas en #WebApp:

Acción	Descripción	Comportamiento de los componentes del módulo (Figura 58)
Visualizar la landing page del bot	Da al usuario las herramientas para comprender las	El usuario ingresa a través del navegador a la dirección www.larabot.com, la página se carga y despliega la landing page con información relevante del bot como

	capacidades y acciones que se pueden usar en el bot.	sus características y acciones que pueden ser usadas.
Registrar usuario antes de autenticar	Da al usuario las herramientas para crear un perfil en #WebApp para su posterior autenticación.	El usuario desea crear una cuenta para autenticarse: 1. Ingresa al formulario de autenticación que permite realizar el registro. 2. El usuario ingresa los datos del formulario. 3. El sistema envía información de notificación al correo electrónico del usuario sobre la creación de la cuenta.
Autenticarse directamente en #WebApp	Permite al usuario identificarse en la #WebApp.	El usuario desea autenticarse: 1. Ingresa al formulario de autenticación. 2. El usuario ingresa la información de autenticación (nombre de usuario y contraseña). 3. El sistema autentica al usuario que ahora tiene una sesión activa en #WebApp.
Iniciar proceso de autenticación por OAuth	Integra aplicaciones externas para que obtengan acceso a la plataforma con un usuario y contraseña.	El proceso de autenticación dependerá de la plataforma ya que esta puede o no requerir de autenticación, para aquellas que lo requieren el proceso es similar al siguiente: 1. La plataforma de chat despliega el formulario de autenticación en www.larabot.com. 2. El usuario ingresa la información de autenticación (nombre de usuario y contraseña). 3. El sistema autentica al usuario y retorna un token de autorización a la plataforma de chat externa
Cambiar parámetros y configuraciones en el panel de administración	Da las capacidades a un administrador de configurar elementos y características del sistema.	Un usuario con sesión activa en #WebApp ingresa al módulo de administración, visualiza los formularios de configuración del sistema y modifica valores que posteriormente son aceptados por el sistema
Ver historial de actividad	Permite que el usuario pueda ver su historial de interacciones.	Un usuario con sesión activa en #WebApp ingresa al módulo de administración, selecciona la opción de historial y visualiza historial del sistema.

Enviar retroalimentación o mensajes al sistema	Permite interactuar con el bot de forma sencilla	El usuario selecciona el módulo de chat simple puede tomar dos: 1. El usuario envía un mensaje pidiendo dar una retroalimentación del sistema. 2. El usuario selecciona la opción de retroalimentación, un formulario se despliega y el usuario envía su retroalimentación. En los dos casos la retroalimentación es recibida en el sistema y almacenada para su posterior lectura por los responsables de calidad del producto.
Cerrar sesión del usuario	Permite a un usuario terminar y remover el token con el que este estaba autenticado	El usuario usa la opción de cerrar sesión en #WebApp, el sistema remueve el token de autorización y elimina la sesión.
Visualizar panel de administración y herramientas de analíticas	Como usuario administrador puede visualizar modelos de la base de datos y herramientas de seguimiento	1. El usuario administrador ingresa al panel de administración y puede visualizar los diferentes *CRUDs* de los modelos/tablas de la base de datos. 2. Como usuario administrador también tiene acceso a herramientas de monitoreo de analíticas disponibles por el sistema que le permiten visualizar el registro de actividad.

Tabla 14: Acciones permitidas por el usuario en #WebApp.

5.2.3.5. Diagrama de Casos de Uso

Figura 58: Casos de uso de la #WebApp.

5.2.4. ITERACIÓN #4: #InputGateway

5.2.4.1. Elegir un elemento del sistema a descomponer

En la iteración #3 se finalizó la especificación de las interfaces que permiten al usuario interactuar con el sistema de forma visual y ya que estas interfaces requieren un punto de entrada para ejecutar y acceder a cualquier tipo de recurso en el sistema es imprescindible continuar con **#InputGateway**.

Antes de poder continuar con la descomposición de los módulos internos del sistema es necesario describir como el punto de entrada del sistema puede lidiar con los diferentes recursos del sistema recibiendo peticiones de un número escalable de plataformas externas.

5.2.4.2. Identificar candidatos a motivantes arquitectónicos principales

Se procede a tomar los motivantes arquitectónicos del módulo de #InputGateway, especificado en interacciones previas:

Puerta o interfaz de entrada del sistema (#InputGateway)	**Requisitos Funcionales (Ver sección 4.5.1):** RF-03, RF-11, RF-02, RF-04, RF-05, RF-06 **Restricciones de Diseño (Ver sección 4.5.3):** RD-04, RD-02 **Escenarios de Calidad (Ver sección 4.6.2.5):**

	ERF-01, ERF-04, ERF-05, ERF-06, ERF-07, ERF-08, ERF-10, ERF-11, ERF-12, ERF-17

Teniendo los motivantes arquitectónicos identificados se procede a hacer la priorización de estos y organizarlos de mayor a menor:

Motivaciones Arquitectónicas	**Valor de Importancia (Alta, Media, Baja)**	**Valor de Impacto (Alto, Medio, Bajo)**
ERF-01: Recepción del mensaje del usuario	Alta	Alto
RF-02: Autenticación del usuario en plataforma externa	Alta	Alto
RF-03: Interactuar con el bot en chats externos	Alta	Alto
RF-11: Ejecución de comandos/acciones	Alta	Alto
ERF-08: Sistema con alta disponibilidad	Alta	Alto
RF-04: Interactuar con el bot en interfaz de chat HTTP	Alta	Alto
ERF-07: Integración de plataformas/canales nuevos al bot	Alta	Alto
RD-02: Texto como única fuente de mensajes	Alta	Alto
ERF-04: Escalar los recursos del sistema según su uso	Alta	Alto
ERF-06: Liberación de versiones nuevas del software de forma controlada y automática.	Alta	Alto
ERF-05: Control de pruebas unitarias y de integración	Medio	Alto
RD-04: Solo canales de comunicación HTTPS	Alta	Medio
ERF-12: Permitir a un usuario autenticarse y usar el bot	Media	Alto
ERF-10: Base de datos segura y confiable con contexto	Media	Alto
ERF-11: Instalación del bot en plataforma de chat externa.	Media	Medio
RF-06: Almacenamiento de información básica del usuario	Baja	Medio
ERF-17: Restringir el uso del sistema a protocolos seguros	Baja	Alto
RF-05: Uso del sistema por usuarios anónimos	Baja	Bajo

Tabla 15: Motivaciones arquitectónicas de #InputGateway.

Los motivantes arquitectónicos de **#InputGateway** deben dirigir la implementación de este módulo teniendo en cuenta que este debe ser extensible y escalable ya que lidia con la

integración de muchas fuentes de información variadas por parte de plataformas externas específicas, teniendo esto en cuenta se consigue la siguiente tabla con los ASRs del módulo:

Indicador	Referencia	Criterio
RF-02	4.5.1.2	Los motivantes arquitectónicos seleccionados en **#InputGateway** buscan dar prioridad a los atributos de calidad en los cuales el proyecto está enfocado. Estos atributos impactan directamente los objetivos del proyecto por lo que los motivantes elegidos permiten justificar los complimientos de dichos atributos.
RF-03	4.5.1.3	
RF-04	4.5.1.4	
ERF-01	4.6.2.5	
ERF-04	4.6.2.5	
ERF-08	4.6.2.5	

Tabla 16: Candidatos a motivantes arquitectónicos principales (ASRs) para el #InputGateway.

5.2.4.3. Especificación estructural del #InputGateway

Este módulo específico las implementaciones necesarias para que el sistema pueda recibir información de diferentes fuentes de información. La arquitectura del sistema está contemplada para que esta funcione de forma nativa por ejecución bajo demanda, esto en términos prácticos significa que la entrada del sistema debe funcionar a través de la interacción de peticiones http que ejecuten un evento que despierte al sistema. Como se explicó en la sección de estado del arte de serverless, el funcionamiento de este se hace basado en eventos que despliegan el sistema para ejecutar un evento.

Ya qué **#InputGateway** es la entrada del sistema es el primer módulo que debe ser implementado y que esté orientado en una arquitectura serverless que le permita ser escalable ante el número elevado de fuentes y plataformas externas. Teniendo este concepto arquitectónico por eventos presente se obtuvieron los siguientes componentes a partir del módulo de entrada del sistema:

- **Función principal de entrada**

  Gran parte de las implementaciones actuales de serverless están orientadas en la creación de lambdas y/o funciones que ejecutan un recurso interno y retornan una respuesta. En el caso de **#InputGateway** esta es la función que debe entregar toda la información proveniente de un evento que despierte la ejecución de los módulos del sistema.

  El rol principal de esta función es desplegar el punto de entrada del sistema, entregar la información de una petición recibida y retornar una respuesta. Concretamente en serverless el sistema al ser desplegado por esta función tarda un tiempo pequeño en “despertarse” posteriormente al finalizar dando una respuesta el sistema se mantiene activo durante un tiempo prudente esperando peticiones/eventos nuevos y si estos no llegan se apaga.

Adicional a esto, el proveedor escala las funciones principales de forma automática ejecutándolas en múltiples procesos al mismo tiempo dependiendo completamente de la concurrencia del sistema, por lo cual la escalabilidad del sistema está asegurada, para ampliar más estos conceptos se puede referenciar la sección **2.3.3**.

- **Mensaje crudo del usuario**

  Las peticiones recibidas en la función principal del sistema reciben una *payload*, en el formato especificado por la plataforma que envía el mensaje, esto se conocerá en el sistema como mensaje crudo o formato origen. Este concretamente son las estructuras originales en las cuales el mensaje fue enviado y que posteriormente deben ser respetadas cuando se desee responder al usuario. Cada plataforma tiene un formato específico por lo que se deben tratar estos mensajes crudos antes de poderse usar en el sistema.

- **Transformador de mensaje**

  Este es el componentes o estrategia específica que permite realizar la conversión de un mensaje crudo de una plataforma específica en una estructura estándar y manejable para el sistema. Este componente es fundamental para mantener una extensibilidad y mantenibilidad adecuada para el sistema ya que este permite realizar la implementación de código específico de las plataformas en un módulo especial para cada una de estas y no lidiar con especificidades en los módulos principales del sistema.

- **Tipo de mensaje**

  Los eventos recibidos pueden ser de muchos tipos, dado que el bot no solo recibe interacciones de tipo mensaje, sino que también se reciben eventos por acciones que ocurren en la plataforma externa. Algunas de estas acciones pueden ser:

  - El bot ha sido invitado a un canal.
  - El bot ha sido removido de un canal.
  - Un canal al que el bot pertenece ha sido renombrado, eliminado o archivado.
  - El bot ha sido mencionado en un canal.
  - Un canal ha sido creado, eliminado o archivado.

- **Base de datos**

  Instancia de base de datos para almacenar información de forma persistente, esta se puede utilizar para validaciones como a la autenticación de usuarios o guardar un histórico de accesos a la plataforma, en el caso de registro del usuario, se debe contar principalmente con:

  - Nombre de usuario.
  - Correo Electrónico.
  - Contraseña.
  - Idioma.

- **Contexto del mensaje de un usuario**

  En el proceso de recepción de un mensaje, este puede contener información relevante adjuntada que debe ser manejada para su posterior uso en la generación de mensajes y principalmente para responder apropiadamente a la API del chat externo. Esta información de contexto puede contener la siguiente información:

  - Información especial del formato del mensaje o evento de la plataforma.
  - Información relevante del usuario.
  - Canal origen junto a un identificador de canal.
  - Usuarios mencionados en el mensaje.
  - Metadatos extra de la plataforma como tokens o puntos de acceso.
  - Intención pre-procesada, las plataformas podrían entregar una información más neutral del mensaje.

Teniendo especificados los componentes relevantes del módulo, a continuación, se presenta el **#InputGateway** y sus componentes internos principales, así como su relación con los demás componentes del sistema:

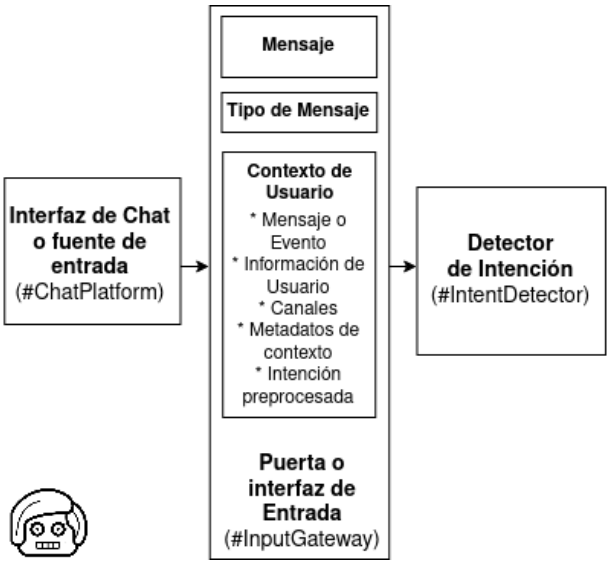

Figura 59: Descomposición de componentes de la #InputGateway.

5.2.4.4. Especificación comportamental del #InputGateway

El rol principal de #InputGateway en su comportamiento es permitir ser un puente entre los eventos de entrada en su función principal y los módulos internos del sistema. Las acciones que se pueden ejecutar en este son todas las acciones permitidas en las interfaces de chat externa **#ChatPlatform** y la plataforma web **#WebApp**. Con estos criterios y objetivos de comportamiento en el presente módulo se genera la siguiente tabla de acciones permitidas:

Acción	Descripción	Comportamiento de los componentes del módulo
Registrar un usuario en plataforma	Permite registrar usuarios en el sistema	Hay acciones específicas del sistema que se ejecutan directamente sin un proceso de identificación de acciones, en este caso se almacena la información del usuario y se encripta la contraseña.
Autenticar usuario en el sistema	Da la capacidad a las peticiones http de autenticar un usuario para obtener token de autorización	Se verifica que la información del usuario corresponda con los datos almacenados, para la autenticación se usa un sistema compatible con Oauth2 como la generación de un token con JWT.
Configurar parámetros del sistema	Modificaciones sobre parámetros del sistema para un usuario	Se hace la modificación correspondiente de los valores en base de datos para las acciones o valores personalizados del sistema.
Recibir retroalimentación	Permite almacenar información de retroalimentación para su posterior lectura	Se recibe la petición con la retroalimentación y se almacena en la base de datos del sistema, la retroalimentación tiene asociada una acción en el sistema para permitir identificar la acción en caso de que esta sea enviada a través de un mensaje en una plataforma.
Recibir un mensaje directo en petición HTTP segura	Permite recibir un mensaje en una petición http plana sin ningún tipo de formato	1. El módulo de **#InputGateWay** recibe el mensaje crudo del usuario en el formato específico de la plataforma o como una petición plana. 2. Este mensaje es transformado en el componente transformador, se traduce el formato del mensaje en el formato estándar del sistema. 3. Se entrega el mensaje con el nuevo formato al módulo de #IntentDetector.
Recibir un mensaje con información de contexto, como menciones	Habilita la capacidad del sistema de tomar mensajes en un formato específico con información de contexto y transformarlo	Como proceso adicional al recibir un mensaje se agrega cualquier información de contexto relevante en un formato estándar en el mensaje que se envía al módulo de #IntentDetector.
Traducir mensaje de una plataforma específica a una estructura estándar para el sistema	Maneja los mensajes de plataformas externas por parte del sistema, de esta forma permite que el sistema tenga las capacidades de recibir información de una plataforma específica	Dentro del proceso para usar un mensaje se usa un patrón como strategy que permita la traducción de un sistema específico al estándar de la plataforma.
Recibir mensaje proveniente de un canal público	Da la capacidad al sistema de interactuar con un contexto específico de	Los mensajes recibidos por un canal público cuentan con una característica especial, todo el proceso de traducción y envío del mensaje al

	canales	módulo de identificación es el mismo, pero se adiciona un parámetro con el identificador del canal y un valor booleano que identifique que es un mensaje de canal.
Recibir un evento de una plataforma externa	Permite que se puedan recibir peticiones que no requieren identifica intenciones si no la ejecución de un recurso específico.	Los eventos son acciones específicas del sistema que no requieren de la detección de una intención, ya que estos son valores conocidos y estáticos que no cambian como el de un idioma.
Ejecutar acciones directas	Habilita al módulo a realizar una interacción rápida con el módulo de intenciones y mensajes para ejecutar una acción sin necesidad de usar recursos extras	Escenario en el cual cualquier tipo de ejecución proveniente de un evento, punto final de un API o código específico, es ejecutado sin realizar la identificación de una intención al ser valores que no lo requieren.
Terminar la sesión de un usuario	Deshabilita un token de autorización para que no tenga acceso al sistema	Se valida el token de autorización y se borra rastros de sesión de usuario.

Tabla 17: Acciones permitidas por el usuario en #InputGateway.

5.2.5. ITERACIÓN #5: #IntentDetector

5.2.5.1. Elegir un elemento del sistema a descomponer

Habiendo definido los procesos de entrada de eventos y peticiones al sistema, se procede a descomponer el módulo que toma estos mensajes para la posterior ejecución de acciones, este es el **#IntentDetector**. El criterio de elección de este módulo se enmarca en descomponer el módulo que debe usar los mensajes y peticiones transformados por #**InputGateway**. Este módulo es fundamental para el cumplimiento de los objetivos de este proyecto.

5.2.5.2. Identificar candidatos a motivantes arquitectónicos principales

Siguiendo el proceso ya establecido en iteraciones anteriores se procede a tomar los resultados de las asignaciones de motivantes del **#IntentDetector** clasificado en la iteración #1, estos son:

Detector de intención (#IntentDetector)	**Requisitos Funcionales (Ver sección 4.5.1):** RF-09, RF-11, RF-07, RF-10, RF-06 **Restricciones de Diseño (Ver sección 4.5.3):** RD-04, RD-02, RD-03 **Escenarios de Calidad (Ver sección 4.6.2.5):** ERF-01, ERF-02, ERF-03, ERF-05, ERF-06, ERF-09, ERF-10, ERF-12, ERF-15, ERF-16, ERF-17

Como parte complementaria del proceso de clasificación de estos motivantes en la presente iteración se procede a priorizar los motivantes para encontrar los ASRs del módulo:

Motivaciones Arquitectónicas	Valor de Importancia (Alta, Media, Baja)	Valor de Impacto (Alto, Medio, Bajo)
RF-09: Identificación de la intención del mensaje	Alta	Alto
RF-11: Ejecución de comandos/acciones	Alta	Alto
RF-07: Creación y eliminación de comandos/acciones	Alta	Alto
RD-02: Texto como única fuente de mensajes	Alta	Alto
ERF-03: Adición de nuevas acciones/comandos al bot	Alta	Alto
ERF-01: Recepción del mensaje del usuario	Alta	Alto
ERF-02: Integración de nuevos motores de intenciones	Alta	Alto
RF-10: Identificación de las entidades del mensaje	Media	Alto
RF-06: Almacenamiento de información básica del usuario	Media	Alta
RD-04: Solo canales de comunicación HTTPS	Alta	Medio
ERF-05: Control de pruebas unitarias y de integración	Media	Alto
ERF-06: Liberación de versiones nuevas del software de forma controlada y automática.	Alta	Alto
ERF-10: Base de datos segura y confiable con contexto	Media	Alto
ERF-12: Permitir a un usuario autenticarse y usar el bot	Media	Alto
ERF-13: Sistema mínimo de características configurables	Media	Alto
ERF-14: Interactuar en canales nuevos y públicos	Media	Medio
ERF-16: Interacción con el sistema en inglés y/o español	Baja	Alto
ERF-17: Restringir el uso del sistema a protocolos seguros	Baja	Alto
ERF-15: Interacción multiusuario a través de acciones	Baja	Medio
ERF-09: Dar al usuario las herramientas para comprender las capacidades y acciones que se pueden usar en el bot.	Media	Bajo
RD-03: Soporte multi lenguaje a español e inglés.	Baja	Alta

Tabla 18: Motivaciones arquitectónicas priorizadas.

A partir de los resultados priorizados se procede a obtener los ASRs de **#IntentDetector,** estos permiten enfocarse en la implementación de servicios cognitivos de forma extensible y escalable para la identificación de intenciones.

Indicador	Referencia	Criterio
RF-09	4.5.1.9	Elegir los motivantes que se enfocarán en el objetivo principal de la identificación de intenciones y que permitan mantener la extensibilidad de este módulo con sus respectivas restricciones.
RF-11	4.5.1.11	
RF-10	4.5.1.10	
RD-02	4.5.3.2	
ERF-02	4.6.2.5	
ERF-03	4.6.2.5	

Tabla 19: motivantes arquitectónicos principales candidatos (ASRs) para el #IntentDetector.

5.2.5.3. Especificación estructural del #IntentDetector

El módulo de **#IntenDetector** es el módulo central del sistema y en el cual parte de la arquitectura del sistema actual se diferencia de otras arquitecturas de chatbot tradicionales. Generalmente, como se explica en la sección 2.3.2, los chatbots tienen una clasificación de intenciones basadas en reglas, esto es muy limitante para encontrar de forma apropiada como un mensaje se relaciona con una intención y recursos establecidos en el sistema. Este módulo cambia esta idea y abstrae el proceso de identificación de intenciones de los controladores para permitirle hacer un proceso de formalización al mensaje usando técnicas de NLP por medio de implementación de servicios.

Este módulo es fundamental para detectar la intención, entidades e información relevante del mensaje para ser usada en #BotEndine. Considerando las características de este módulo y cómo deben relacionarse con los atributos de calidad establecidos en los objetivos del proyecto y de la arquitectura se establecen los siguientes componentes indispensables para su implementación:

- **Base de datos de conocimientos**

Este componente hace referencia a la información preestablecida en el sistema, es aquella usada para establecer una relación entre el mensaje y los recursos que se desean integrar en el sistema. Esta base de conocimientos contiene las frases sinónimas proveniente de las acciones implementadas en los controladores (*handlers*) para de esta forma encontrar un porcentaje de similitud y permitir relacionar el mensaje con una intención en el sistema.

- **Clasificador de intención**

Componente con el cual se busca extraer y normalizar toda la lógica referente a la identificación de una intención a través de una interfaz estándar. La idea principal del componente de clasificación es que se puede es tomar un mensaje en formato de texto plano, un cúmulo de

conocimiento en el cual se incluyen las frases sinónimas y la información de contexto relevante, para posteriormente de forma extensible implementar una estrategia específica, como un servicio cognitivo o interno. Al realizar esta ejecución el componente obtiene los resultados que le permiten clasificar la similaridad del mensaje con una frase preestablecida en el sistema, generando un resultado que contenga:

- Token o llave de la frase sinónima
- Entidades identificadas en el mensaje
- Información relevante del mensaje
- Y de ser aplicable la red semántica generada a partir del mensaje

- **Servicios cognitivos**

Estos son los servicios específicos de NLP/NLU que pueden ser implementados en el sistema. Actualmente se tiene en consideración Luis.ai, Wit.ai e IBM Watson como referentes de servicios cognitivos para la identificación de similitudes del mensaje y procesamiento de este con frases sinónimas en el sistema. Siempre que se desee implementar un servicio nuevo debe tenerse en cuenta que este debe contar con la capacidad de generar este porcentaje de similitud para identificar apropiadamente la intención y de igual forma las entidades del mensaje. Estos servicios requieren:

- Frase para analizar(mensaje)
- Conjunto de frases sinónimas a comparar
- Idioma sobre el cual se está trabajando

- **Microservicios internos de reconocimiento**

Como parte del proceso de permitir futuras implementaciones personalizadas sobre el resultado del proyecto se plantea crear un servicio interno de reconocimiento que cuente con características de una arquitectura serverless. La idea principal de este microservicio es dar las herramientas necesarias a los stakeholders que deseen implementar la arquitectura pero que a su vez no deseen crear una dependencia a un servicio externo de detección de intenciones. Este microservicio debe ser similar a los servicios cognitivos en términos de entradas y salidas. Como recomendación se puede tomar la arquitectura y herramientas generadas por Spacy [69] como referencia.

A continuación, se presenta el **#IntentDetector** y sus componentes internos principales, así como su relación con los demás componentes del sistema.

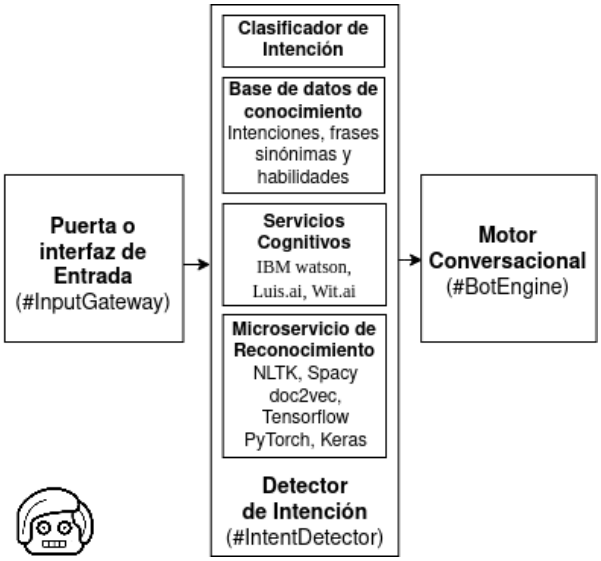

Figura 60: Descomposición de componentes de la #IntentDetector.

5.2.5.4. Especificación comportamental del #IntentDetector

A continuación, se hace la descomposición de las acciones permitidas en este módulo, enfocadas en entregar información relevante para la ejecución de recurso en #BotEngine:

Acción	Descripción	Comportamiento de los componentes del módulo
Recibir mensaje con formato estándar.	Permite al módulo recibir un mensaje con la estructura aceptada por el sistema.	El mensaje es recibido por el módulo y es preparado con la información relevante en el formato específico del servicio cognitivo a usar.
Recibir información de contexto.	Da la capacidad de usar la información de contexto del sistema y relacionarlo con una base de conocimiento.	La información de contexto en este módulo se usa principalmente para enriquecer la información enviada a los servicios de identificación de intenciones, esta información de contexto incluye el idioma del mensaje, el historial de conversación, usuarios mencionados en el mensaje, entre otros.
Recibir mensaje con menciones a otros usuarios.	Permite relacionar mensajes de texto plano con usuarios mencionados en este.	Los mensajes que poseen información de usuarios mencionados son utilizados para entregarse a los controladores de la acción específica.
Identificar intenciones y entidades.	Integra la información manejada por el módulo para el envío a un servicio que permita identificar la intención del mensaje sobre una base de datos de conocimiento.	Se procede a identificar las intenciones en los siguientes pasos: 1. Se toma el mensaje y se adiciona información relevante de contexto como historial.

		2. Se hace uso de la base de conocimientos para obtener las frases que funcionan como tokens en los controladores del sistema. 3. Esta información es enviada al servicio cognitivo con la llave de autenticación de la aplicación del sistema para identificar similitudes entre el mensaje, el idioma y la base de conocimientos, al igual que para obtener información relevante de las frases, como las entidades contenidas en esta. 4. Se recibe la información retornada por el servicio y se elige la acción que cumpla con el porcentaje más alto de intención. 5. Se ejecuta una acción en el #BotEngine.
Integrar motor de intenciones nuevo o servicios cognitivos nuevos.	Da las herramientas necesarias al desarrollador para que este tenga un panorama claro sobre las restricciones y necesidades en el sistema para adicionar un motor de intenciones nuevo.	En el caso de integrar un motor de intenciones nuevo el desarrollador cuenta con los siguientes pasos: 1. Crear un módulo de conversión del mensaje al formato estándar del servicio. 2. La acción en el servicio debe permitir la comparación del mensaje con frases sinónimas en la base de conocimientos. 3. Se debe obtener las entidades del sistema, como información relevante. 4. Este resultado debe responder con el token del controlador para ejecutar la acción y con las entidades identificadas.
Recibir un evento o acción directa.	Permite que el módulo no use los servicios de identificación si no ejecutar recurso de forma directa	Los eventos en el formato estándar ya poseen la acción correspondiente a ejecutar, la cual es ejecutada directamente en **#BotEngine**
Ejecutar una acción.	Habilita a #IntentDetector para realizar la ejecución de un recurso en #BotEngine.	Con el token del controlador correspondiente se procede a ejecutar una acción en **#BotEngine**, el cual recibe la información relevante entre ellas: ● Token de la intención identificada con su controlador. ● Entidades de mensaje. ● Identificador de un canal en caso de que el mensaje provenga de una canal.

		• Usuarios que hacen parte del mensaje (menciones). • Información extra de contexto. • El idioma del usuario. • En caso de que la implementación de NLP sea nativa o el servicio cognitivo lo permita, la red semántica generada por el mensaje.

Tabla 20: Acciones permitidas en #IntentDetector.

5.2.6. ITERACIÓN #6: #BotEngine

5.2.6.1. Elegir un elemento del sistema a descomponer

Se procede a tomar los resultados de la clasificación de la intención para ejecutar recursos en el sistema con el módulo de **#BotEngine**.

5.2.6.2. Identificar candidatos a motivantes arquitectónicos principales

A continuación, se muestran los motivantes arquitectónicos que hacen parte de **#BotEngine**:

Motor conversacional (#BotEngine)	**Requisitos Funcionales (Ver sección 4.5.1):** RF-11, RF-12, RF-07, RF-10, RF-06, RF-08 **Restricciones de Diseño (Ver sección 4.5.3):** RD-04, RD-02, RD-03 **Escenarios de Calidad (Ver sección 4.6.2.5):** ERF-01, ERF-03, ERF-05, ERF-06, ERF-08, ERF-09, ERF-10, ERF-12, ERF-13, ERF-15, ERF-16, ERF-17

A partir de los motivantes se realiza la priorización de estos en la siguiente tabla:

Motivaciones Arquitectónicas	**Valor de Importancia** **(Alta, Media, Baja)**	**Valor de Impacto** **(Alto, Medio, Bajo)**
RF-11: Ejecución de comandos/acciones	Alta	Alto
RF-12: Generación de respuestas del bot según resultados	Alta	Alto
RF-07: Creación y eliminación de comandos/acciones	Alta	Alto

RD-02: Texto como única fuente de mensajes	Alta	Alto
ERF-01: Recepción del mensaje del usuario	Alta	Alto
ERF-03: Adición de nuevas acciones/comandos al bot	Alta	Alto
ERF-06: Liberación de versiones nuevas del software de forma controlada y automática.	Alta	Alto
ERF-08: Sistema con alta disponibilidad	Alta	Alto
RF-08: Configuración y parametrización del sistema	Alta	Medio
RD-04: Solo canales de comunicación HTTPS	Alta	Medio
RD-03: Soporte multi lenguaje a Español e Inglés.	Baja	Alta
ERF-05: Control de pruebas unitarias y de integración	Media	Alto
ERF-10: Base de datos segura y confiable con contexto	Media	Alto
ERF-12: Permitir a un usuario autenticarse y usar el bot	Media	Alto
ERF-16: Interacción con el sistema en inglés y/o español	Baja	Alto
ERF-17: Restringir el uso del sistema a protocolos seguros	Baja	Alto
ERF-13: Sistema mínimo de características configurables	Media	Alto
ERF-15: Interacción multiusuario a través de acciones	Baja	Medio
RF-06: Almacenamiento de información básica del usuario	Baja	Medio
ERF-09: Dar al usuario las herramientas para comprender las capacidades y acciones que se pueden usar en el bot.	Media	Bajo

Tabla 21: Motivaciones arquitectónicas de #BotEngine.

Se procede a seleccionar los ASRs del módulo basados en un criterio de elección:

Indicador	Referencia	Criterio
RF-11	4.5.1.11	A pesar de haber motivantes arquitectónicos con más prioridad, el criterio está principalmente enfocado en los motivantes que específicamente afectan el diseño del módulo de **#BotEngine,** principalmente para la mantenibilidad y extensibilidad de este**.**
RF-12	4.5.1.12	
RF-07	4.5.1.7	
RF-08	4.5.1.8	
ERF-01	4.6.2.5	
ERF-08	4.6.2.5	

Tabla 22: Candidatos a motivantes arquitectónicos principales para el #BotEngine.

5.2.6.3. Especificación estructural del #BotEngine

El #BotEngine es el módulo encargado de ejecutar los recursos asociadas a una acción que el usuario solicita a través de un mensaje. Se busca generar respuesta a través de estos resultados para dar una retroalimentación al usuario. Este módulo es muy similar a como es implementado en los sistemas tradicionales de chatbots, la diferencia mayor es la extracción del análisis de la intencionalidad que anteriormente fue tratado en **#IntenDetector** al igual que la atomización de los componentes para la mantenibilidad del sistema. A continuación, se presentan los componentes de este módulo:

- **Intención del mensaje:** Este componente hace referencia a los tokens e información de relación del mensaje con una frase preestablecida en el sistema para ser usada en el controlador correspondiente
- **Entidades identificadas:** Las entidades son los parámetros que permiten cuantificar o clasificar elementos en una frase se usan para ejecutar acciones de forma cuantificada.
- **Contexto:** Componente con la información extra para la generación y envío de respuestas, este puede contener tanto datos sobre usuario mencionados o el canal de origen de un mensaje como sus identificadores.
- **Manejador de contexto:** Componente en los controladores que permiten obtener la información del contexto del componente anterior
- **Conector APIs:** Este elemento hace referencia a aquellos recursos específicos que pueden ser usados en la ejecución de una acción, como software especial de una plataforma o bases de datos del sistema.
- **Máquina de estados:** En conversaciones avanzadas que requieren un flujo continuo, es importante implementar estrategias que permitan identificar correctamente la intención del usuario, la cual puede cambiar dependiendo de contextos previos. El componente de máquina de estados suele implementarse a base de condicionales y árboles de decisión, pero

se recomienda seguir patrones de diseño que permitan hacer seguimiento a cada estado usando estructuras de datos.

- **Controladores (*Handlers*):** En la sección **2.3.3** se explica visualmente el accionar de una handler, para resumirlo es aquel que contiene la implementación de las acciones específicas del sistema. Este especifica la acción, sus entidades y el código a ser ejecutado para posteriormente usar un generador de respuestas.
- **Generador de respuestas:** Componente que permite generar respuestas estáticas internacionalizadas a partir del resultado de un controlador. Este módulo en la arquitectura actual es muy simple con respuestas estáticas almacenadas. En futuras implementaciones y revisiones de la arquitectura se puede abstraer este componente para realizar una implementación con NLG para la generación de respuestas personalizadas que hagan sentir al usuario más cómodo con el bot.

A continuación, se presenta el #BotEngine y sus componentes internos principales, así como su relación con los demás componentes del sistema.

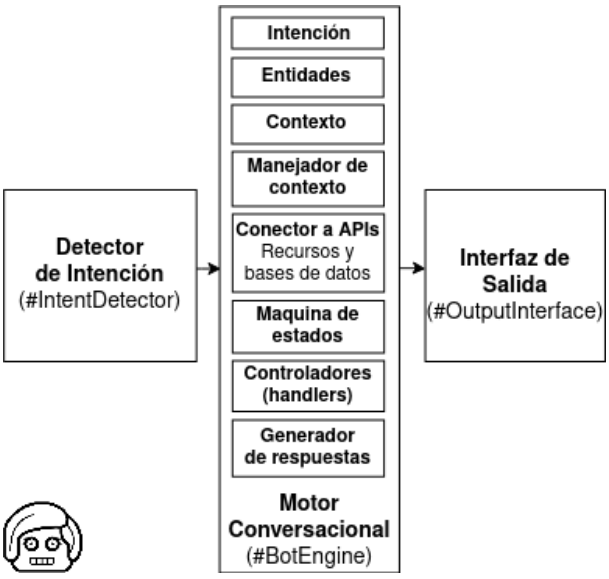

Figura 61: Descomposición de componentes de la #BotEngine.

5.2.6.4. Especificación comportamental del #BotEngine

Se procede a describir las acciones disponibles en este módulo:

Acción	Descripción	Comportamiento de los componentes del módulo (Figura X)
Recibir intención y entidades de un mensaje	Habilita al módulo para usar la información del mensaje para el uso de controladores	Con las intenciones identificadas y la información especificada por #IntenDetector, #BotEngine se prepara para la ejecución ordenada en el módulo de detección de intención.

Recibir información de contexto	Permite usar información relevante para la generación de la respuesta y su posterior envío en **#OutputInterface**	La información de contexto es relevante para el envío de respuesta generadas en la ejecución de una acción, ya que este puede contener el identificador de un canal para enviar la respuesta específica.
Ejecutar una acción	Permite que el módulo pueda recibir la ejecución directa de una acción	La acción por ejecutar puede estar directamente relacionada con un evento, ejecución directa con comandos o con la intención identificada, en todos estos casos el módulo se comporta de la siguiente forma: 1. Se entrega la información relevante al controlador (esto dependerá de cada acción). 2. Se ejecuta la acción con el token del controlador. 3. Se obtiene un resultado con los valores del controlador, ya sea un valor de una ejecución satisfactoria o con algún tipo de error. 4. En caso de que el #BotEngine no haya recibido un token con intención para el controlador, se ejecuta la acción para listar las acciones disponibles en el sistema adicionando un mensaje en caso de que esto sea por un error en la identificación del mensaje.
Generar un mensaje de respuesta con resultado	Permite al módulo generar una respuesta a partir de la ejecución de un recurso	Tomando el resultado obtenido en un controlador se procede a generar una respuesta personalizada al usuario con la información referente de la acción ejecutada.
Almacenar información del usuario	Da al sistema las herramientas para almacenar información de forma persistente	Como parte del proceso del historial de usuario en #WebApp, se procede a almacenar un registro de la ejecución de una acción en el sistema en la base de datos.
Listar acciones disponibles en el sistema	Habilita las opciones de listado de acciones de las acciones	Primera acción del sistema que toma la base de conocimientos y controladores, para generar una lista de acciones con sus respectivas descripciones.
Configurar parámetros del sistema	Permite modificar parámetros concretos como acceso a acciones.	Ejecutar código específico para alterar valores de base de datos para la ejecución de acciones basadas en los parámetros alterados

Integrar una acción nueva	Habilita la integración de acciones nuevas al sistema de forma extensible con su respectivo controlador	El desarrollador implementa una acción nueva de la siguiente forma: 1. Se crea una frase identificadora que será el token de controlador de la acción y que debe estar adiciona en la base de conocimientos del sistema para frases sinónimas en la identificación de intención. 2. Se le proporciona una descripción a la acción, que debe estar implementado bajo un sistema de internacionalización (i18n), para escribirla en múltiples idiomas, por defecto en inglés. 3. Se identifican y adicionan las entidades relevantes del controlador. 4. Se hace la implementación de la función correspondiente del controlador con un generador de resultados en formato estándar
Eliminar una acción.	Remueve alguna acción de un controlador para no ser usada	Se debe de remover las funciones y toda mención de la acción en los controladores y bases de conocimiento

Tabla 23: Acciones permitidas por el usuario en #BotEngine.

5.2.7. ITERACIÓN #7: #OutputInterface

5.2.7.1. Elegir un elemento del sistema a descomponer

A través de las iteraciones se logró descomponer los elementos necesarios para obtener una respuesta a lo que el usuario requiere, pero esta respuesta requiere ser enviada según una interfaz de salida ya que esta depende completamente de la plataforma la cual está interactuando con el sistema. Por lo anterior se hace necesaria y válida la descomposición de módulo **#OutputInterface** como módulo final del sistema previo a analizar el sistema como resultado en una iteración final.

5.2.7.2. Identificar candidatos a motivantes arquitectónicos principales

Se toman los motivantes arquitectónicos del presente módulo:

Interfaz de Salida (#OutputInterface)	**Requisitos Funcionales (Ver sección 4.5.1):** RF-11, RF-12, RF-04, RF-08 **Restricciones de Diseño (Ver sección 4.5.3):** RD-04, RD-03 **Escenarios de Calidad (Ver sección 4.6.2.5):** ERF-01, ERF-04, ERF-05, ERF-06, ERF-07, ERF-08, ERF-09, ERF-11, ERF-12, ERF-13, ERF-14, ERF-15, ERF-16, ERF-17, ERF-18

Se realiza la priorización de los motivantes para posteriormente clasificar los ASRs:

Motivaciones Arquitectónicas	Valor de Importancia (Alta, Media, Baja)	Valor de Impacto (Alto, Medio, Bajo)
RF-11: Ejecución de comandos/acciones	Alta	Alto
RF-12: Generación de respuestas del bot según resultados	Alta	Alto
RD-04: Solo canales de comunicación HTTPS	Alta	Alto
ERF-01: Recepción del mensaje del usuario	Alta	Alto
ERF-04: Escalar los recursos del sistema según su uso	Alta	Alto
ERF-05: Control de pruebas unitarias y de integración	Alta	Alto
ERF-06: Liberación de versiones nuevas del software	Alta	Alto
ERF-08: Sistema con alta disponibilidad	Alta	Alto
ERF-14: Interactuar en canales nuevos y públicos	Media	Alto
ERF-07: Integración de plataformas/canales nuevos al bot	Alta	Medio
RF-04: Interactuar con el bot en interfaz de chat HTTP	Media	Alto
ERF-12: Permitir a un usuario autenticarse y usar el bot	Media	Alto
ERF-13: Sistema mínimo de características configurables	Media	Alto
ERF-16: Interacción con el sistema en inglés y/o español	Baja	Alto
ERF-17: Restringir el uso del sistema a protocolos seguros	Baja	Alto
RD-03: Soporte multi lenguaje a Español e Inglés.	Baja	Alto
ERF-15: Interacción multiusuario a través de acciones	Baja	Medio
ERF-11: Instalación del bot en plataforma de chat externa.	Media	Medio
RF-08: Configuración y parametrización del sistema	Baja	Medio
ERF-09: Dar al usuario las herramientas para comprender las capacidades y acciones que se pueden usar en el bot.	Media	Bajo
ERF-18: El usuario interactúa con el sistema en aplicación de voz	Baja	Bajo

Tabla 24: Motivantes arquitectónicas de #OutputInterface.

Se obtienen los ASRs del módulo basados en un criterio de selección:

Indicador	Referencia	Criterio
RF-12	4.5.1.12	Motivantes que permitan enfocarse en generar una respuesta de forma apropiada a las peticiones hechas sobre el sistema y cómo mantener la integración de los diferentes formatos de salida de forma segura, escalable y mantenible.
RF-11	4.5.1.11	
RD-04	4.5.3.4	
ERF-05	4.6.2.5	
ERF-07	4.6.2.5	
ERF-14	4.6.2.5	

Tabla 25: Motivantes arquitectónicos principales (ASRs) de #OutputInterface.

5.2.7.3. Especificación estructural del #OutputInterface

El rol principal de **#OutputInterface** es prestar una interfaz de salida común al sistema para realizar el envío de respuesta al cliente a través de implementaciones específicas de cada plataforma. Este módulo especifica las reglas de cada plataforma y cómo realizar esta implementación de forma extensible y mantenible sin perder características en **#ChatPlatform** o **#WebApp.**

- **Mensaje de respuesta:** Componente resultado generado por #BotEngine, este es un objeto con la respuesta generada por la ejecución de un controlador. Este mensaje se usa para ser enviado como respuesta a una petición de un cliente.
- **Información de contexto usuario para salida:** Como se ha explicado previamente en iteraciones anteriores el contexto del usuario puede contener información relevante para el envío de mensajes al usuario. Esta información incluye el token del usuario o en caso de que el mensaje provenga de un canal, este contendrá un identificador del canal. Esta información es relevante para la interfaz de salida y poder hacer el envío correspondiente.
- **Interfaz de salida de chat:** Esta es la implementación del envío del mensaje de respuesta a la plataforma externa, aquí se retorna el mensaje como una respuesta http en el formato del sistema o por el contrario se usa el software especial de la plataforma si es necesario. Cabe aclarar que como se explica en la iteración #1 este envío de respuestas puede ser síncrono o asíncrono, lo cual dependiendo completamente de la plataforma.
- **Formateador de mensaje:** Se realiza el proceso inverso de **#InputGateway**, ya que se toma el mensaje respuesta que está en un formato estándar para el sistema y se crea una estructura entendible para la plataforma externa o el cliente http que haya interactuado con el sistema. Este formateador puede incluir valores para enriquecer la respuesta como formatos para especificar botones o elementos en el sistema.

A continuación, se presenta el #BotEngine y sus componentes internos principales, así como su relación con los demás componentes del sistema.

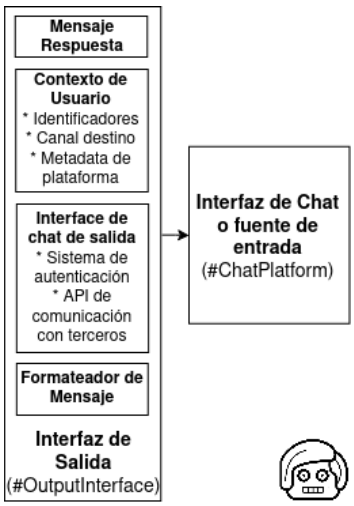

Figura 62: Descomposición de componentes de la #OutputInterface.

5.2.7.4. Especificación comportamental del #OutputInterface

El comportamiento de este módulo está enmarcado en la interacción del sistema con las plataformas externas y clientes http. Las acciones correspondientes por lo tanto hacen parte de todos los resultados esperados de interactuar con el sistema, de forma directa o indirecta:

Acción	Descripción	Comportamiento de los componentes del módulo
Recibir mensaje resultado	Permite recibir el resultado de la ejecución de una acción	Se obtiene el mensaje con la información relevante para preparar este para ser enviado en la interfaz de salida, esta debe especificar la plataforma a la cual se debe enviar el mensaje.
Formatear mensaje en la plataforma específica	Habilita al sistema para tener la capacidad de interactuar con diferentes plataformas externas de forma extensible	Haciendo uso de una estrategia específica para una plataforma implementada, como se especifica en el patrón strategy, se toma el mensaje y este es convertido en el formato específico para esta plataforma.
Integrar una plataforma de chat nueva	Permite que se puedan integrar plataformas nuevas en el sistema	Los desarrolladores implementan una nueva plataforma en la interfaz de salida de la siguiente forma: 1. Se adiciona cualquier tipo de SDK o paquete específico de software que sea requerido por la plataforma en el sistema. 2. Se implementa una clase que será instanciada en un objeto específico para la implementación de las estrategias necesarias para la ejecución específica en el sistema. 3. Se crean las funciones de formateador y envío de mensajes.

		4. Se implementa el código específico en cada uno y se adiciona la clase a las opciones de plataformas disponibles en el sistema.
Enviar mensaje a un canal público	Habilita los módulos para el envío de respuestas a canales distintos según el formato de respuesta	Se usa la interfaz de salida con el mensaje ya formateado por la estrategia específica y se envía el mensaje usando el identificador del canal público y el token del bot.
Enviar mensaje a un usuario directamente	Permite comunicarse directamente con el usuario	Se usa la interfaz de salida con el mensaje ya formateado por la estrategia específica y se envía el mensaje usando el token del usuario y del bot.
Ejecutar recursos en plataforma externa	Habilita al sistema para poder interactuar con recursos específicos de cada plataforma	Algunas acciones podrán realizar la ejecución de algún tipo de recurso en una plataforma si esta lo permite, **#OutputInterface** cumple con el rol de ejecutar estos recursos comunicando este accionar a la plataforma enviando el mensaje o ejecutando el software específico de la plataforma en la estrategia correspondiente.

Tabla 26: Acciones permitidas por el usuario en #OutputInterface.

5.2.8. ITERACIÓN #8 - Final

Habiendo generado una iteración por cada uno de los elementos del sistema y sus correspondientes acciones permitidas se procede a realizar una iteración final para retomar estos nuevos conceptos generados en las iteraciones anteriores. Esta iteración está orientada a hacer un despliegue de todos los resultados obtenidos en cada uno de los elementos del sistema para generar un entregable que sirva de documentación general del sistema. Teniendo en cuenta lo anterior se procede a especificar los comportamientos relevantes del sistema.

5.2.8.1. Especificación comportamental del sistema

En el objetivo general del proyecto se planteó el diseño de una arquitectura conversacional por texto, que sea extensible, escalable y mantenible mediante serverless, haciendo uso de servicios cognitivos de Machine Learning y NLP. Esta arquitectura debía facilitar la implementación de chatbots dándoles la capacidad de mantener conversaciones naturales a través de la generación de respuestas más precisas en el contexto de la intención del mensaje enviado al chatbot. En términos de comportamiento para el sistema se comprende en permitirle al usuario enviar mensajes y recibir respuestas coherentes a lo que solicita. Sin embargo, antes de especificar los casos de usos relevantes para cumplir con este objetivo se procede a desplegar un diagrama con todos los casos de uso y sus relaciones entre los módulos y componentes del sistema.

5.2.8.2. Diagramas de casos de uso

Se presenta el diagrama de casos de uso del sistema completo refinado, asignando identificadores y destacando las acciones principales del usuario desde las cuales las interfaces de entrada intervienen a través del sistema completo.

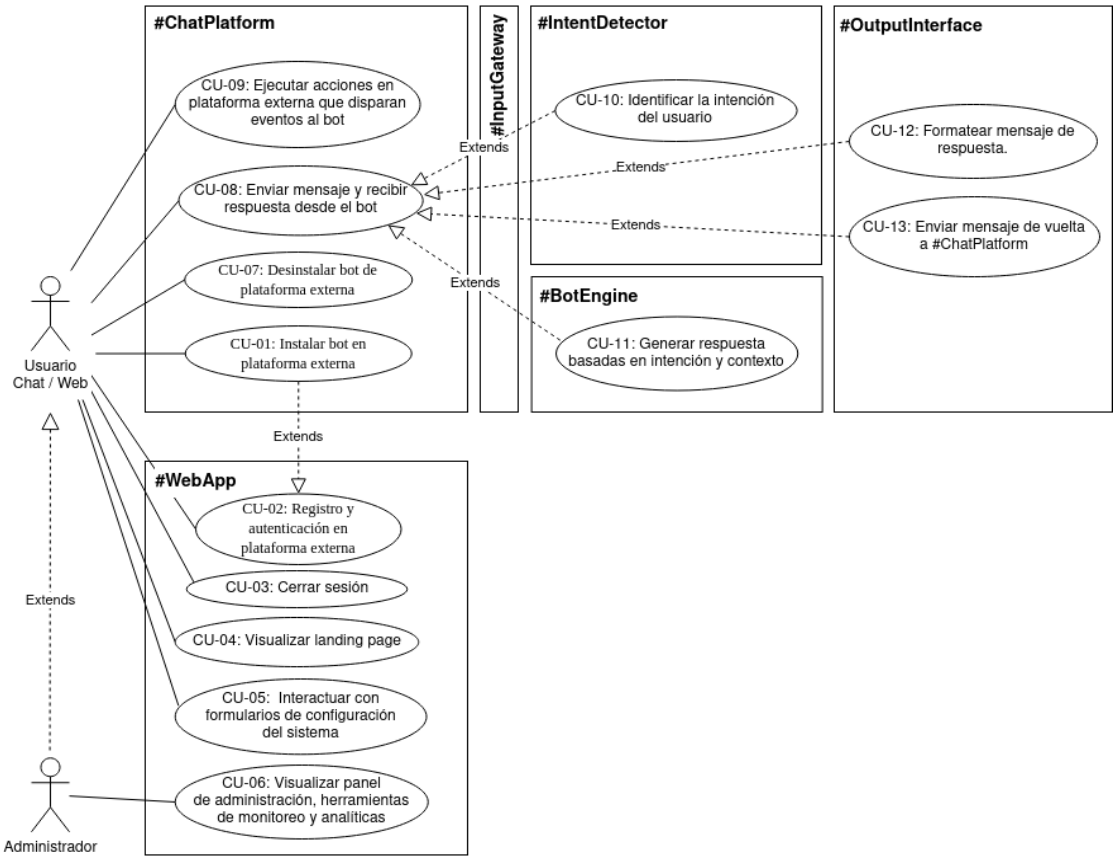

Figura 63: Diagrama de casos de uso del sistema #LaraBot.

A partir de este diagrama se seleccionarán los siguientes casos de uso. Cabe aclarar que se planea realizar la especificación de los casos de usos principales para evitar la extensión innecesaria de este proceso ya que muchos de estos casos tienen un accionar común por lo que es más productivo en términos de documentación generar un caso de uso principal que múltiples casos de usos innecesarios:

- ✓ **Instalar bot en plataforma externa**

Caso de uso	CU-01: Instalar bot en plataforma externa
Actores	Usuario Web
Propósito	Este caso de uso permite al usuario crear una instancia del bot en su plataforma de chat. Para poder interactuar posteriormente con el sistema del chatbot.
Resumen	El usuario ingresa a una plataforma de chat o directamente a #WebApp, el usuario seguirá los siguientes pasos: 1. Seleccionar la instalación del bot en la

	plataforma 2. Se realiza el proceso de adición de bot en la plataforma y en caso de requerir un logueo se conecta con **#WebApp**. 3. Se obtiene un token de la aplicación y se almacena la instancia del bot en el chat del espacio de trabajo del usuario.
Precondición	Creación de la aplicación del bot en plataforma específica por parte de los desarrolladores.
Tipo	Primario
Clase	Esencial
Curso Normal de los Eventos	
Acción de los actores	**Respuesta del Sistema**
1. El usuario ingresa a una plataforma de chat o directamente a #WebApp	
2. Seleccionar la instalación del bot en la plataforma	
	3. Se realiza el proceso de adición de bot en la plataforma y en caso de requerir una autenticación se conecta con **#WebApp**.
	4. Se obtiene un token de la aplicación y se almacena la instancia del bot en el chat del espacio de trabajo del usuario

✓ Registro y autenticación en plataforma externa o internas

Caso de uso	CU-02: Registro y autenticación en plataforma externa o internas
Actores	Usuario
Propósito	Este caso de uso permite al usuario crear una instancia del bot en su plataforma de chat. Para poder interactuar posteriormente con el sistema del chatbot.
Resumen	Por medio de la instalación del bot o su posterior instancia se puede realizar una autenticación a la plataforma de la siguiente forma: 1. El usuario va a autenticarse por medio de OAuth. 2. Se despliega un formulario de autenticación de **#WebApp**. 3. El usuario usa sus credenciales y **#WebApp** retorna un token de autorización.
Precondición	
Tipo	Primario
Clase	Esencial
Curso Normal de los Eventos	
Acción de los actores	**Respuesta del Sistema**
1. El usuario va a autenticarse por medio de OAuth	
	2. Se despliega un formulario de autenticación de **#WebApp**.
3. El usuario usa sus credenciales.	

	4. **#WebApp** retorna un token de autorización.

✓ Cerrar sesión

Caso de uso	CU-03: Cerrar sesión
Actores	Usuario
Propósito	Permitir que el usuario puede eliminar su sesión en una plataforma
Resumen	El usuario usa la opción de cerrar sesión en #WebApp, el sistema remueve el token de autorización y elimina la sesión.
Precondición	
Tipo	Primario
Clase	Esencial
Curso Normal de los Eventos	
Acción de los actores	**Respuesta del Sistema**
1. El usuario usa la opción de cerrar sesión en #WebApp.	
	2. El sistema remueve el token de autorización.
	3. El sistema elimina la sesión.
	4. El sistema responde al usuario exitosamente.

✓ Visualizar landing page

Caso de uso	CU-04: Visualizar landing page
Actores	Usuario
Propósito	Da al usuario las herramientas para comprender las capacidades y acciones que se pueden usar en el bot.
Resumen	El usuario ingresa a través del navegador a la dirección www.larabot.com, la página se carga y despliega la landing page con información relevante del bot como sus características y acciones que pueden ser usadas.
Precondición	
Tipo	Primario
Clase	Esencial
Curso Normal de los Eventos	
Acción de los actores	**Respuesta del Sistema**
1. El usuario ingresa a través del navegador a la dirección www.larabot.com.	
	2. La página se carga y despliega la landing page.
	3. La página muestra información relevante del bot, como sus características y acciones que pueden ser usadas

✓ Interactuar con formularios de configuración del sistema

Caso de uso	CU-05: Interactuar con formularios de configuración del sistema
Actores	Usuario Administrador

Propósito	Da las capacidades a un administrador de configurar elementos y características del sistema.
Resumen	Un usuario con sesión activa en #WebApp ingresa al módulo de administración, el sistema despliega el panel de administración, visualiza los formularios de configuración del sistema y modifica valores que posteriormente son aceptados por el sistema
Precondición	CU-02
Tipo	Secundario
Clase	Deseable
Curso Normal de los Eventos	
Acción de los actores	**Respuesta del Sistema**
1. El usuario accede al módulo de administración en #ChatPlatform.	
	2. El sistema despliega el panel de administración.
3. Visualiza los formularios de configuración del sistema.	
4. El usuario modifica los valores de configuración	
	5. **#WebApp** envía el formulario al sistema
	6. El sistema acepta la configuración y responde al usuario exitosamente.

✓ Visualizar panel de administración, herramientas de monitoreo y analíticas

Caso de uso	CU-06: Visualizar panel de administración, herramientas de monitoreo y analíticas
Actores	Usuario Administrador
Propósito	Permitir a un administrador de visualizar información sensible.
Resumen	Un usuario con sesión activa en #WebApp ingresa al módulo de administración, el sistema despliega el panel de administración, el usuario visualiza los valores de monitoreo y analítica
Precondición	CU-02
Tipo	Secundario
Clase	Deseable
Curso Normal de los Eventos	
Acción de los actores	**Respuesta del Sistema**
1. El usuario accede al módulo de administración.	
	2. #WebApp despliega el módulo de administración.
3. El usuario selecciona las secciones de monitoreo y analítica.	
	4. #WebApp renderiza la sección de monitoreo y analítica, solicitando la información al sistema

	5. El sistema consulta la información en la base de datos y retorna los resultados a #WebApp
	6. #WebApp despliega la información de monitoreo y analítica
7. Visualiza los valores de monitoreo y analítica.	

✓ Desinstalar bot de plataforma externa

Caso de uso	CU-07: Desinstalar bot de plataforma externa
Actores	Usuario
Propósito	Permite al usuario desinstalar la instancia del bot de su espacio de trabajo
Resumen	La plataforma externa #ChatPlatform se encarga de remover la instancia del bot del espacio de trabajo de usuario, en caso del bot contar con los permisos se dispara un evento notificando al sistema de la eliminación de la instancia.
Precondición	CU-01
Tipo	Primario
Clase	Esencial
Curso Normal de los Eventos	
Acción de los actores	**Respuesta del Sistema**
1. El usuario solicita remover la instancia del bot en sus chats.	
	2. **#ChatPlatform** notifica al sistema sobre la intención del usuario de remover.
	3. **#ChatPlatform** notifica al sistema sobre la intención del usuario de remover.
	4. **#ChatPlatform** notifica al usuario de la instalación exitosa
5. El usuario selecciona las secciones de monitoreo y analítica.	

✓ Enviar mensaje y recibir respuesta desde el bot

Caso de uso	CU-08: Enviar mensaje y recibir respuesta desde el bot
Actores	Usuario
Propósito	Da al usuario la capacidad de interactuar con el bot de forma directa a través del envío de texto
Resumen	Dependiendo de la plataforma el cliente debe de realizar el envío de algún token que permita autorizar al usuario, para todos los casos concretos funciona de forma similar: 1. El usuario envía un mensaje al bot por chat privado directo. 2. El mensaje es formateado por la plataforma con su estructura específica. 3. Se usa el Callback especificado en el componente de suscripción del sistema. 4. Se envía el mensaje hacia la URL

	especificada por medio de una petición HTTP segura. 5. Se obtiene una respuesta de forma síncrona o asíncrona dependiendo de la plataforma externa.
Precondición	CU-01
Tipo	Primario
Clase	Esencial
Curso Normal de los Eventos	
Acción de los actores	**Respuesta del Sistema**
1. El usuario envía un mensaje al bot por chat privado directo en **#ChatPlaform**.	
	2. El mensaje es formateado en #ChatPlatform por la plataforma con su estructura específica.
	3. Se usa el Callback especificado en el componente de suscripción del sistema y #ChatPlatform envía el mensaje a **#InputGateway** junto al token de autorización (si es necesario).
	4. **#InputGateway** convierte el mensaje en formato estándar del sistema.
	5. **#InputGateway** envía el mensaje a los módulos internos donde se procesa y se genera una respuesta que es enviada a **#OutputInterface.**
	6. **#OutputInterface** transforma la respuesta en el formato específico de la plataforma y la envía a **#ChatPlatform** por interfaz síncrona o asíncrona.
	7. **#ChatPlatform** recibe el mensaje y lo notifica al usuario en el canal correspondiente o en su chat.

✓ Ejecutar acciones en plataforma externa que disparan eventos al bot

Caso de uso	CU-09: Ejecutar acciones en plataforma externa que disparan eventos al bot
Actores	Usuario
Propósito	Da al bot la capacidad de ejecutar acciones relaciona a eventos específicos de la plataforma
Resumen	En caso de poseer los permisos necesarios sobre un evento (ejemplo la eliminación de un canal), el bot será notificado con la información del evento para poder ejecutar alguna acción específica. Algunos de los eventos más usados son: ● Bot es añadido a un canal. ● Bot es eliminado de un canal. ● Un usuario es invitado a un canal. ● Un usuario es eliminado de un canal. ● Un canal es renombrado. ● Un canal es eliminado o archivado.

	● Usuario edita mensaje enviado al bot.
Precondición	CU-01
Tipo	Primario
Clase	Esencial
Curso Normal de los Eventos	
Acción de los actores	**Respuesta del Sistema**
1. El usuario realiza una acción que desencadena un evento en #**ChatPlatform**.	
	2. El mensaje es formateado por la plataforma **#ChatPlatform** con su estructura específica.
	3. Se usa el Callback especificado en el componente de suscripción del sistema y **#ChatPlatform** envía el mensaje a **#InputGateway** junto al token de autorización (si es necesario).
	4. **#InputGateway** convierte la petición en formato estándar del sistema.
	5. **#InputGateway** envía la acción a los módulos internos donde se procesa y se genera una respuesta que es enviada a **#OutputInterface**
	6. **#OutputInterface** transforma la respuesta en el formato específico de la plataforma y la envía a **#ChatPlatform** por interfaz síncrona o asíncrona
	7. **#ChatPlatform** recibe el mensaje y lo notifica al usuario en el canal correspondiente o en su chat dependiendo de la naturaleza del evento

✓ Identificar la intención del usuario.

Caso de uso	CU-10: Identificar la intención del usuario.
Actores	Usuario
Propósito	Integra la información manejada por el módulo para el envío a un servicio que permita identificar la intención del mensaje sobre una base de datos de conocimiento.
Resumen	Se procede a identificar las intenciones en los siguientes pasos: 1. Se toma el mensaje y se adiciona información relevante de contexto como historial. 2. Se hace uso de la base de conocimientos para obtener las frases que funcionan como tokens en los controladores del sistema. 3. Esta información es enviada al servicio cognitivo con la llave de autenticación de la aplicación del sistema para identificar similitudes entre el mensaje, el idioma y la base de conocimientos, al igual que para

	obtener información relevante de las frases, como las entidades contenidas en esta. 4. Se recibe la información retornada por el servicio y se elige la acción que cumpla con el porcentaje más alto de intención. 5. Se ejecuta una acción en el #BotEngine.
Precondición	CU-08 o CU-09
Tipo	Primario
Clase	Esencial
Curso Normal de los Eventos	
Acción de los actores	**Respuesta del Sistema**
1. El usuario realiza el envío de un mensaje al sistema	
	2. **#InputGateway** convierte la petición en formato estándar del sistema.
	3. **#InputGateway** envía el mensaje a **#IntentDetector** con la información de contexto.
	4. En **#IntentDetector** Se toma el mensaje y se adiciona información relevante de contexto como historial.
	5. La información es enviada al servicio cognitivo con la llave de autenticación del sistema para identificar similitudes entre el mensaje, el idioma y la base de conocimientos.
	6. Se obtiene un valor de similitudes con la intención identificada al igual que las entidades del mensaje.
	7. Se genera una respuesta que es enviada a **#OutputInterface.**
	8. **#OutputInterface** transforma la respuesta en el formato específico de la plataforma y la envía al cliente por interfaz síncrona o asíncrona

- ✓ **Generar respuestas basadas en intención y contexto.**

Caso de uso	CU-11: Generar respuesta basadas en intención y contexto
Actores	Usuario
Propósito	Permite que el módulo pueda recibir acciones y generar una respuesta a partir de la ejecución de un recurso
Resumen	La acción por ejecutar puede estar directamente relacionada con un evento, ejecución directa con comandos o con la intención identificada en CU-10, en todos estos casos el módulo se comporta de la siguiente forma: 1. Se ejecuta la acción con el token del controlador. 2. Se obtiene un resultado con los valores del

	controlador. 3. Tomando el resultado obtenido en un controlador se procede a generar una respuesta personalizada al usuario con la información referente de la acción ejecutada.
Precondición	CU-08, CU-9 y/o CU-10
Tipo	Primario
Clase	Esencial
Curso Normal de los Eventos	
Acción de los actores	**Respuesta del Sistema**
1. El usuario realiza el envío de un mensaje al sistema	
	2. **#InputGateway** convierte la petición en formato estándar del sistema.
	3. **#InputGateway** envía el mensaje a **#IntentDetector** con la información de contexto.
	4. **#IntentDetector** genera un resultado con la intención identificada que se envían a #BotEngine**.**
	5. Se ejecuta la acción con el token del controlador junto a la información relevante para el controlador como las entidades y el contexto.
	6. Se obtiene un resultado con la ejecución de la acción en el controlador.
	7. Se procede a generar una respuesta personalizada al usuario con la información referente de la acción ejecutada y su resultado.
	8. **#BotEngine** envía la respuesta generada a **#OutputInterface.**
	9. **#OutputInterface** transforma la respuesta en el formato específico de la plataforma y la envía al cliente por interfaz síncrona o asíncrona.

✓ Formatear mensaje de respuesta.

Casos de usos	CU-12: Formatear mensaje de respuesta.
Actores	Usuario
Propósito	Habilita al sistema para tener la capacidad de interactuar con diferentes plataformas externas de forma extensible
Resumen	Haciendo uso de una estrategia específica para una plataforma implementada, como se especifica en el patrón strategy, se toma el mensaje y este es convertido en el formato específico para esta plataforma.
Precondición	CU-11
Tipo	Primario

Clase	Esencial
Curso Normal de los Eventos	
Acción de los actores	**Respuesta del Sistema**
1. El usuario realiza el envío de un mensaje al sistema	
	2. **#InputGateway** convierte la petición en formato estándar del sistema.
	3. **#IntentDetector** genera un resultado con la intención identificada que se envían a #BotEngine**.**
	4. **#InputGateway** envía el mensaje a los módulos internos donde se procesa y se genera una respuesta que es enviada a **#OutputInterface.**
	5. **#OutputInterface** transforma la respuesta en el formato específico de la plataforma
	6. Se usa la información de contexto para adicionar parámetros de envío en el sistema
	7. Se construye el objeto respuesta
	8. **#OutputInterface** envía el objeto con el mensaje formateado al cliente por interfaz síncrona o asíncrona .

- ✓ **Enviar mensaje de vuelta.**

Casos de usos	CU-13: Enviar mensaje de vuelta.
Actores	Usuario
Propósito	Habilita al sistema para tener la capacidad de interactuar con diferentes plataformas externas de forma extensible
Resumen	Haciendo uso de una estrategia específica para una plataforma implementada, como se especifica en el patrón strategy, se toma el mensaje y este es convertido en el formato específico para esta plataforma.
Precondición	CU-112
Tipo	Primario
Clase	Esencial
Curso Normal de los Eventos	
Acción de los actores	**Respuesta del Sistema**
1. El usuario realiza el envío de un mensaje al sistema	
	2. **#InputGateway** convierte la petición en formato estándar del sistema.
	3. **#IntentDetector** genera un resultado con la intención identificada que se envían a #BotEngine**.**
	4. **#InputGateway** envía el mensaje a los módulos internos donde se procesa y se genera una respuesta que es enviada a **#OutputInterface.**
	5. Se construye el objeto respuesta

	6. **#OutputInterface** envía el objeto con el mensaje formateado al cliente por interfaz síncrona o asíncrona

✓ Diagramas de secuencia del sistema

Se presentan los diagramas de secuencia de los casos de uso presentados en la sección anterior:

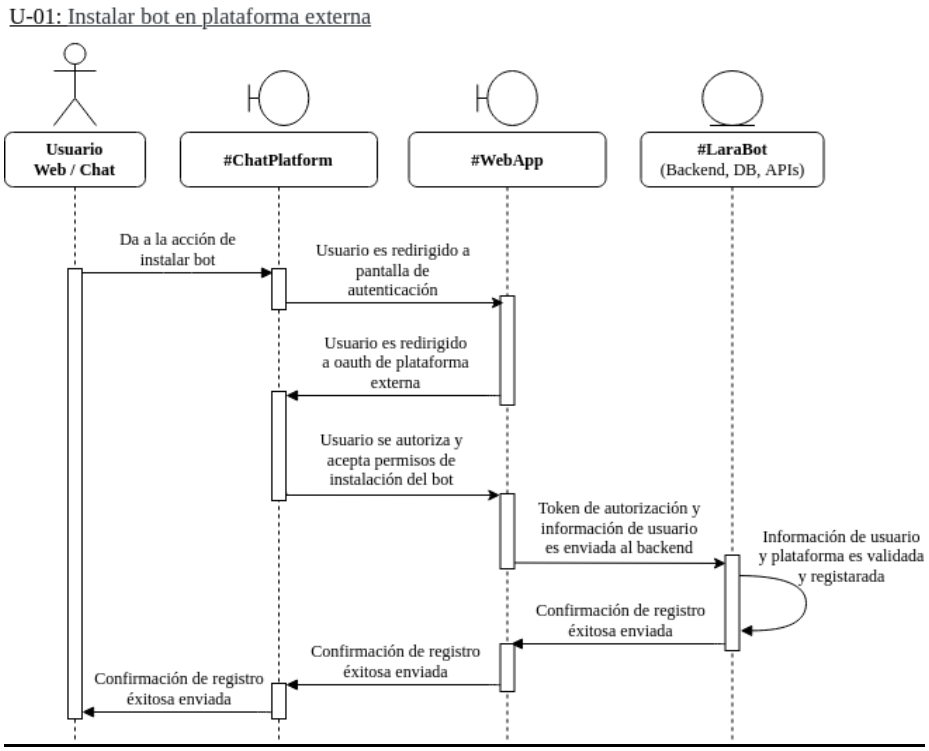

Figura 64: Diagrama de secuencia para el caso de uso CU-01.

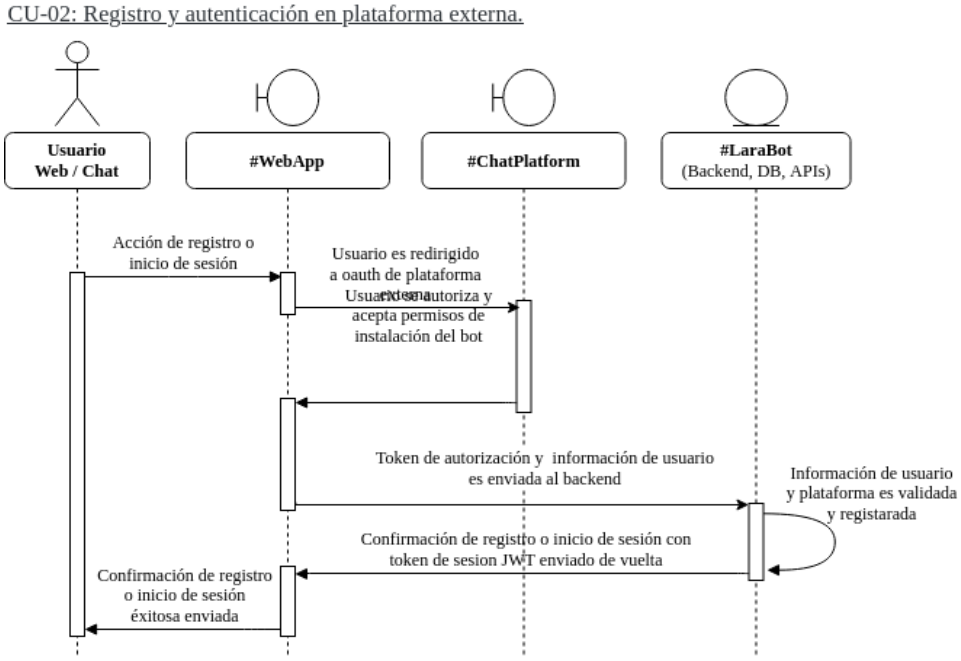

Figura 65: Diagrama de secuencia para el caso de uso CU-02.

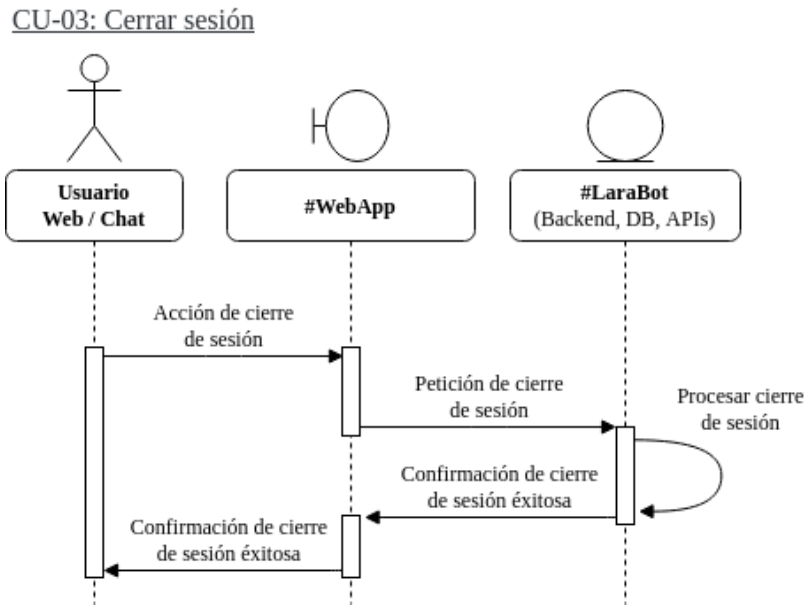

Figura 66: Diagrama de secuencia para el caso de uso CU-03.

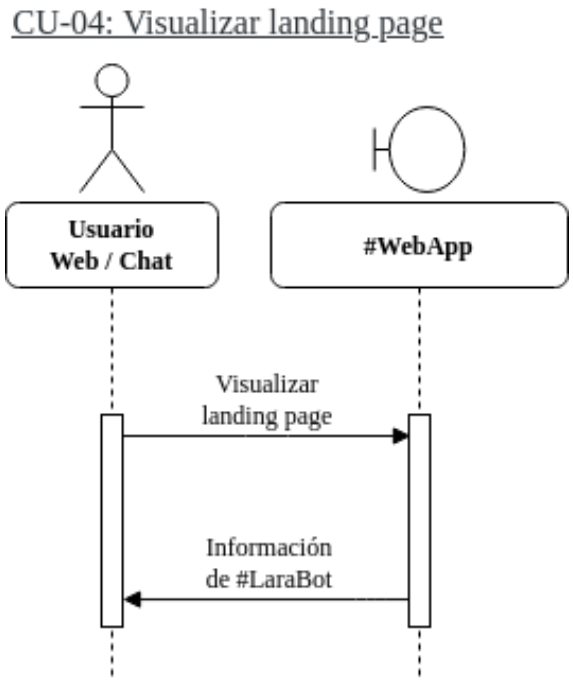

Figura 67: Diagrama de secuencia para el caso de uso CU-04.

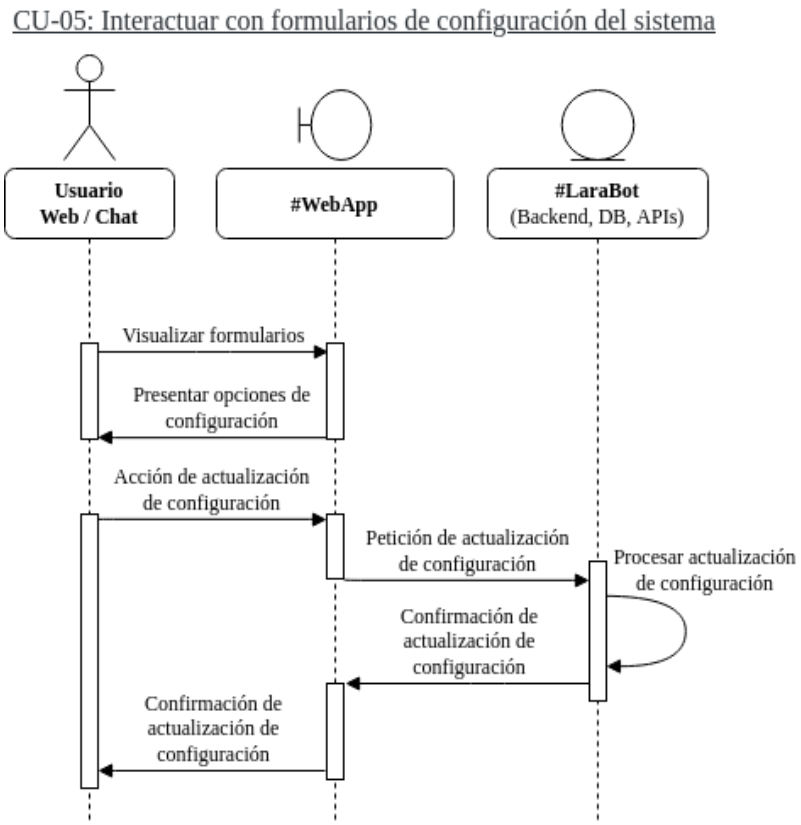

Figura 68: Diagrama de secuencia para el caso de uso CU-05.

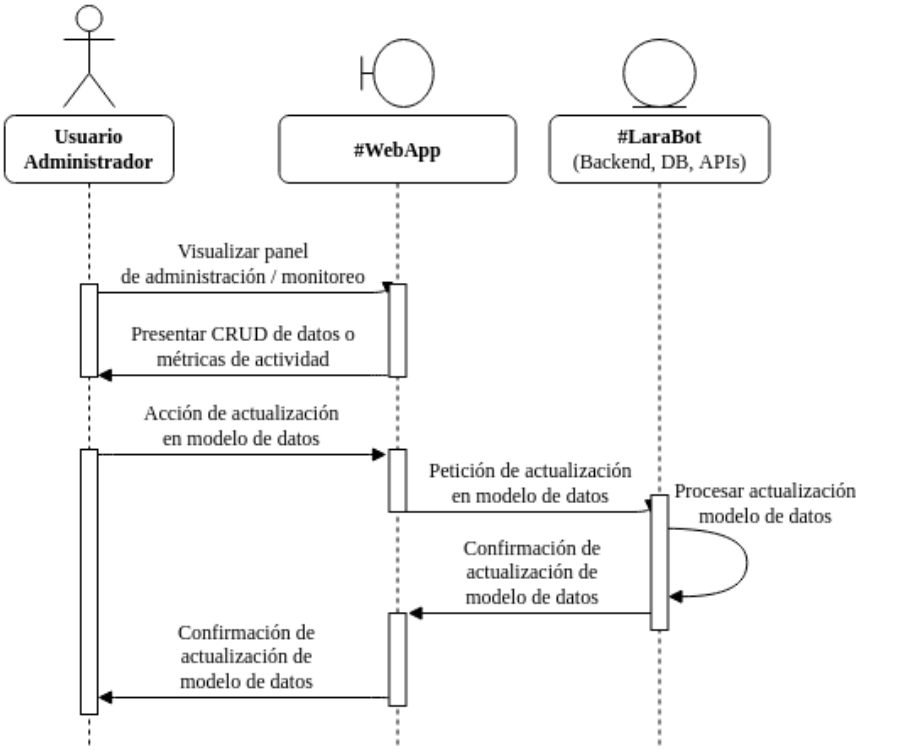

Figura 69: Diagrama de secuencia para el caso de uso CU-06.

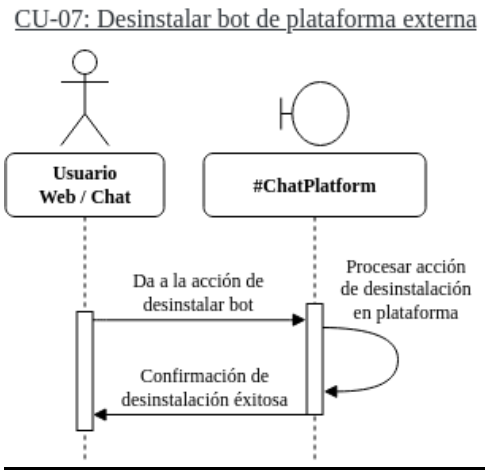

Figura 70:Diagrama de secuencia para el caso de uso CU-07.

CU-08: Enviar mensaje y recibir respuesta desde el bot
CU-09: Ejecutar acciones en plataforma externa que disparan eventos al bot
CU-10: Identificar la intención del usuario
CU-11: Generar respuestas basadas en intención y contexto
CU-12: Formatear mensaje de respuesta
CU-13: Enviar mensaje de vuelta a #ChatPlatform

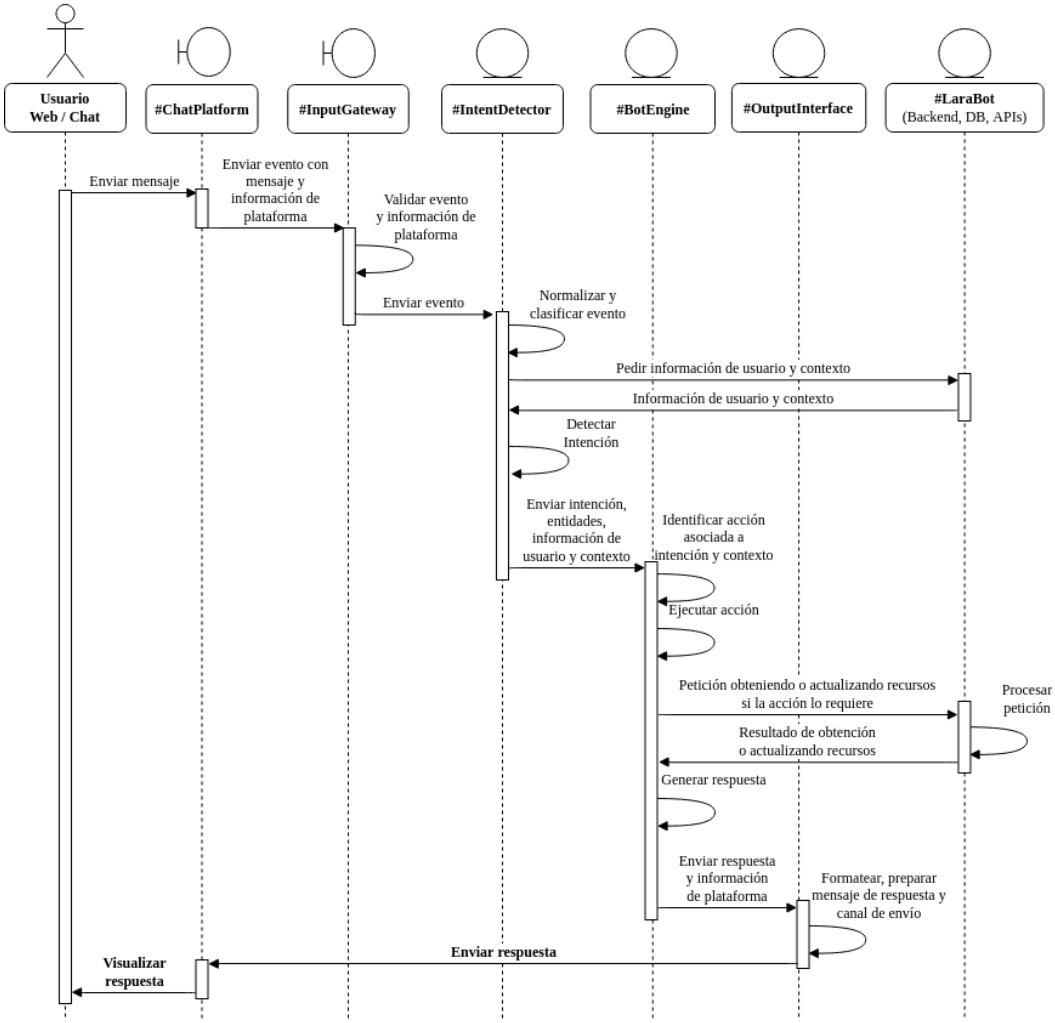

Figura 71: Diagrama de secuencia para los casos de uso del CU-08 a CU-13.

5.3. Resultados de ADD-A

Recapitulando cada uno de los ítems que se generaron en esta etapa de la metodología, se concluye que el método permitió guiar la descripción del sistema de una manera modular, se realizaron ocho iteraciones con los siguientes resultados:

- **Iteración #1 (sección 5.2.2):** Descripción y descomposición de todo el sistema (#ChatPlatform, #WebApp, #InputGateway, #IntentDetector, #BotEngine y #OutputInterface).
- **Iteración #2 (sección 5.2.3):** Descripción y descomposición de #ChatPlatform.
- **Iteración #3 (sección 5.2.4):** Descripción y descomposición de subsistema #WebApp.
- **Iteración #4 (sección 5.2.5):** Descripción y descomposición de #InputGateway.
- **Iteración #5 (sección 5.2.6):** Descripción y descomposición de #IntentDetector.
- **Iteración #6 (sección 5.2.4):** Descripción y descomposición de #BotEngine.
- **Iteración #7 (sección 5.2.5):** Descripción y descomposición de #OutputInterface.
- **Iteración #8 (sección 5.2.7):** Integración y descripción de todo el sistema.

El resultado de estas iteraciones permitió describir el sistema de manera tanto estructural como comportamental, dando como resultado el siguiente diagrama de contexto.

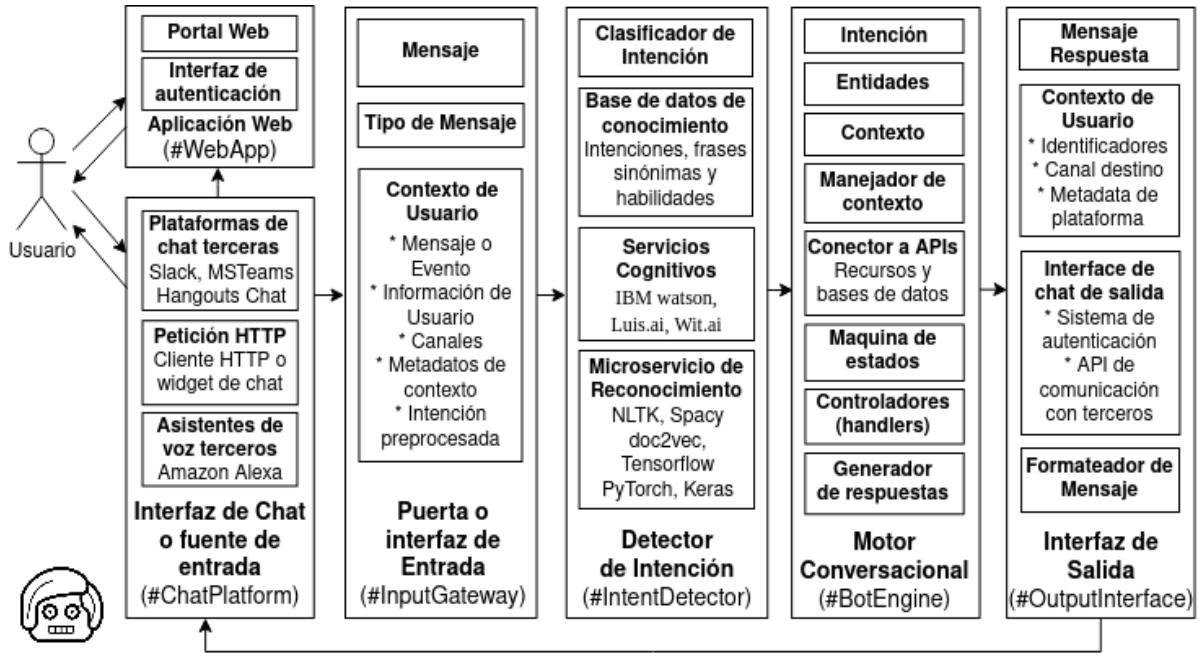

Figura 72: Diagrama de contexto del sistema #LaraBot.

Como último punto a destacar, se reafirma la aclaración hecha en la sección 5.2 y esta se basa en tomar muchos de los resultados obtenidos en este proceso como futura fuente de documentación para la finalización y entrega de una arquitectura en la etapa 4 de Views and Beyond.

6. CAPÍTULO VII: DESARROLLO METODOLÓGICO

FASE 3 - EVALUACIÓN (ATAM-A)

El objetivo de este capítulo es aplicar el método AEM, propuesto por la metodología SEI-A [75], para evaluar el diseño resultante de la aplicación de ADD-A en el capítulo anterior.

AEM es un método inspirado en ATAM, y es importante citar en este punto una pequeña parte de la definición presentada en la sección 3.3.3.

6.1. Involucrados en ATAM-A

A esta fase de evaluación además de los arquitectos, se suman los stakeholders como recurso importante:

- **Líder del Proyecto y Arquitecto:** Sergio Alexander Flórez Galeano.
- **Líder del Proyecto y Arquitecto:** Nelson Orlan Escobar Ceballos.
- **Stakeholder:** Julian Andres Florez Galeano.
- **Stakeholder:** Mauricio Morales.

El rol de los arquitectos en este punto consistirá en presentar a los stakeholders cada uno de los resultados obtenidos tras la aplicación de la metodología con el Caso de Estudio, con el fin de recibir retroalimentación de los stakeholders para determinar la validez de la arquitectura planteada.

6.2. Despliegue del método ATAM-A

Para el desarrollo del método, el grupo de evaluación propone evaluar la trazabilidad de la especificación de diseño realizada, a través de un conjunto de matrices que permitan verificar desde los objetivos de negocio, como estos articulan a lo largo del diseño.

✓ Matriz de trazabilidad entre objetivos de negocio y motivantes arquitectónicos

Dado que los objetivos de negocio son el recurso principal para expresar los intereses de los stakeholders, es importante evaluar cómo estos son expresados por los motivantes arquitectónicos:

- **Objetivos de negocio:** Sección 4.2.
- **Motivantes arquitectónicos iniciales:** Sección 4.6.2.6.

Motivante arquitectónico	Objetivos de negocios																														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
RF-01		*										*			*			*										*		
RF-02								*					*															*		
RF-03		*						*	*		*	*	*		*			*										*		
RF-04		*	*					*	*		*		*	*	*															
RF-05								*	*		*		*				*	*												
RF-06			*						*														*	*						
RF-07				*	*	*										*									*			*	*	
RF-08			*						*							*														
RF-09	*			*	*	*																			*				*	
RF-10	*			*	*	*																			*				*	
RF-11		*	*																						*			*		
RF-12	*	*		*						*															*			*		
RNF-01	*			*	*	*	*			*															*			*	*	
RNF-02	*			*	*	*				*															*				*	
RNF-03	*	*		*			*			*															*			*		
RNF-04																		*	*	*	*	*				*				*
RNF-05																		*	*	*										

RNF-06															*		*	*	*							*				
RNF-07															*					*						*				
RNF-08	*			*	*	*																			*			*	*	
RNF-09															*		*	*												
RNF-10	*			*	*	*	*													*								*		
RNF-11																			*	*					*				*	
RNF-12															*					*										
RNF-13											*				*															
RNF-14									*																					
RNF-15		*	*					*	*		*		*	*	*															
RNF-16									*																					
RNF-17									*							*							*						
RNF-18									*							*						*							
RNF-19								*				*																		*
RD-01		*	*					*	*		*		*	*	*															
RD-02	*			*																										
RD-03		*		*						*																				
RD-04									*		*			*																

Tabla 27: Matriz de trazabilidad entre los objetivos de negocio y motivantes arquitectónicos.

✓ **Matriz de trazabilidad entre atributos de calidad y escenarios de calidad**

Siendo los escenarios de calidad un mecanismo para expresar los objetivos de los atributos de calidad es importante poder observar como premisa logra satisfacerse:

- **Atributos de calidad**: Sección 4.5.2
- **Escenarios de calidad**: Sección 4.6.2.5

Escenarios de calidad	**Atributos de calidad**																		
	RNF 01	RNF 02	RNF 03	RNF 04	RNF 05	RNF 06	RNF 07	RNF 08	RNF 09	RNF 10	RNF 11	RNF 12	RNF 13	RNF 14	RNF 15	RNF 16	RNF 17	RNF 18	RNF 19
ERF-01	*		*													*			
ERF-02		*								*		*				*			
ERF-03												*	*			*			
ERF-04	*		*	*	*			*		*	*								*
ERF-05		*			*	*	*		*										
ERF-06				*	*				*										
ERF-07	*		*		*							*							
ERF-08	*			*				*		*	*								*
ERF-09													*						
ERF-10													*			*	*	*	
ERF-11	*		*																
ERF-12													*			*	*		
ERF-13													*						
ERF-14																*			
ERF-15													*			*			
ERF-16	*													*					
ERF-17															*	*			
ERF-18	*																		

Tabla 28: Matriz de trazabilidad entre atributos de calidad y escenarios de calidad.

- ✓ **Matriz de trazabilidad entre los motivantes arquitectónicos y los componentes del sistema**

A través del proceso iterativo generado en la etapa de ADD-A se realizó la descomposición del sistema en diferentes elementos que permiten explicar el comportamiento de este de forma estructurada y concisa. Estos elementos cumplen con la responsabilidad de satisfacer un conjunto de motivantes arquitectónicos específicos. Esta asociación se puede observar a continuación

- **Componentes del sistema:** #ChatPlatform (5.2.2), #WebApp (5.2.3), #InputGateway (5.2.4), #IntentDetector (5.2.5), #BotEngine (5.2.6), #OutputInterface (5.2.7).
- **Motivantes arquitectónicos:** Sección 4.6.2.6

Motivantes arquitectónicos	**Componentes**					
	# Chat Platform	**# WebApp**	**# Input Gateway**	**# Intent Detector**	**# Bot Engine**	**# Output Interface**
RF-01	*					
RF-02	*	*	*			
RF-03	*		*			
RF-04		*	*			*
RF-05	*	*	*			
RF-06			*	*	*	
RF-07				*	*	
RF-08		*			*	*
RF-09				*		
RF-10				*	*	
RF-11	*	*	*	*	*	*
RF-12					*	*
RD-01		*				
RD-02	*	*	*	*	*	
RD-03	*	*		*	*	*
RD-04	*	*	*	*	*	*
ERF-01	*		*	*	*	*
ERF-02				*		
ERF-03				*	*	
ERF-04			*			*

ERF-05	*	*	*	*	*	*
ERF-06	*	*	*	*	*	*
ERF-07	*	*				*
ERF-08		*	*		*	*
ERF-09		*		*	*	*
ERF-10			*	*	*	
ERF-11	*		*			*
ERF-12	*	*	*	*	*	*
ERF-13		*			*	*
ERF-14	*					*
ERF-15	*			*	*	*
ERF-16	*	*		*	*	*
ERF-17	*	*	*	*	*	*
ERF-18	*					*

Tabla 29: Matriz de trazabilidad entre los motivantes arquitectónicos y componentes.

✓ **Matriz de trazabilidad entre los casos de uso y los requerimientos funcionales**

Finalmente, como resultados de la labor iterativa de ADD se obtuvieron unos casos de uso que permiten describir las interacciones posibles con el sistema. Es necesario hacer una validación sobre el cumplimiento de los requerimientos funcionales y que estos se encuentren reflejados en los casos de uso. A continuación, se muestra el resultado de esta trazabilidad en una tabla:

- **Casos de uso:** Sección 5.2.8.1
- **Motivantes Arquitectónicos:** Sección 4.6.2.6

Casos de uso	**Motivantes arquitectónicos**											
	RF01	RF02	RF03	RF04	RF05	RF06	RF07	RF08	RF09	RF10	RF11	RF12
CU-01	*		*									
CU-02	*	*	*			*		*				
CU-03		*										
CU-04				*								
CU-05				*			*	*				

CU-06								*				
CU-07	*		*									
CU-08			*	*	*	*	*				*	
CU-09			*		*	*	*		*	*	*	*
CU-10									*	*	*	*
CU-11							*				*	*
CU-12							*					*
CU-13			*									*

Tabla 30: Matriz de trazabilidad entre los casos de uso y los requerimientos funcionales.

El proceso realizado en la evaluación ATAM y los resultados de las matrices fue ejecutados 2 veces en la elaboración de este proyecto y de igual forma estos resultados generaron cambios en el capítulo de ADD. Sin embargo, se decidió realizar un condensado de estos resultados para evitar la extensión innecesaria del proyecto, es por ello por lo que en este caso del paso de ATAM se hace referencia a un único proceso de evaluación, el cual fue el último. Se considera por ende que gracias a los resultados obtenidos en esta etapa se puede decir que los objetivos planteados para la aceptación de la arquitectura fueron cumplidos a cabalidad.

Con estos resultados del proceso de evaluación se procede a realizar la verificación sobre la documentación generada y si es necesario procesos o pasos extra en la documentación de la arquitectura creada.

7. CAPÍTULO VIII: DESARROLLO METODOLÓGICO

FASE 4 - DOCUMENTACIÓN (VAB-A)

En la sección 5 del capítulo correspondiente al despliegue del método ADD-A, se explicó cómo los elementos resultantes de este despliegue generaban una documentación cualificada que posteriormente sería retomada en la etapa de VAB-A. La metodología SEI-A [75], propone una fase final a lo largo del proceso de diseño arquitectónico a través del método VaB-A, el cual consiste en adicionar documentación a la que contiene la especificación de diseño arquitectónico del sistema diseñado.

Se aclara entonces que la documentación necesaria por el sistema ha sido generada ya en el diseño de la propia arquitectura y que el objetivo principal de esta fase es permitir justificar esto último

7.1. Involucrados en VaB-A

Similar a las etapas anteriores, se hace una definición de los involucrados en el despliegue del método, en este caso al ser la documentación generada anteriormente y posterior evaluación la responsabilidad recae en los roles de los arquitectos líderes:

- **Líder del Proyecto y Arquitecto:** Sergio Alexander Florez Galeano.
- **Líder del Proyecto y Arquitecto:** Nelson Orlan Escobar Ceballos.

El rol de los arquitectos en este punto consistirá en encontrar necesidades de documentación adicional y en caso de ser necesario generar nueva o documentación extra, para ayudar a la toma de decisiones sobre el sistema en instancias posteriores.

7.2. Despliegue del método VaB-A

VaB es un método que se desarrolla a lo largo del proceso de diseño de la arquitectura. Y como se explicó en ADD los resultados son documentación que puede ser usada como referencia. En VaB se define un primer paso antes de realizar el desarrollo de documentación adicional, este dice:

"Determinar las necesidades de información"

En el caso concreto de este proyecto se determina que no es necesaria la generación de ninguna documentación adicional. Sin embargo, se considera valioso realizar la implementación de la arquitectura en un mínimo que permita demostrar sus bondades y beneficios.

Es por esto último que como paso final y como documentación adicional se propone un capítulo extra en el proyecto con la información de una implementación de un prototipo. Este siguiente capítulo puede ser usado como documentación extra, para futuras implementaciones.

8. CAPÍTULO IX: IMPLEMENTACIÓN DE PROTOTIPO (VAB-A)

Este capítulo es una recopilación de resultados y de documentación adicional. Se considera este capítulo como un valor agregado al proyecto ya que este se realiza después de que los objetivos de este han sido cumplidos y que el proyecto determinó inicialmente el alcance de este en el diseño de la arquitectura del sistema de chatbot propuesto. Habiendo realizado la aclaración, a continuación, se presenta la documentación para una implementación de un prototipo de la arquitectura propuesta, que permita demostrar las bondades de esta a través de una demo. También se liberará cómo código abierto, lo cual ofrecerá un mayor aporte al entregable final del presente proyecto adicional a la documentación aquí generada.

8.1. Tecnologías de desarrollo

Con base a las necesidades de la arquitectura, a continuación, se enumeran las tecnologías, lenguajes de programación, frameworks, librerías y proveedores en la nube a usar para el desarrollo del prototipo.

- **JavaScript:** Lenguaje de programación multiparadigma interpretado, cuyo dialecto sigue el estándar ECMAScript.
- **Typescript:** Desarrollado y mantenido por Microsoft es un *superset* que permite extender la sintaxis de JavaScript para añadir tipado estático y objetos basados en clases, buscando la definición de un lenguaje mucho más robusto y mantenible.
- **Node.js:** Entorno de tiempo de ejecución de JavaScript en tiempo real para la implementación de servicios de Back-end.
- **Nuxt.js:** Framework de desarrollo Front-end basado en Vue.js y escrito en JavaScript totalmente modular y basado en componentes e interfaces web de sencillas.
- **Python:** Lenguaje de programación orientado al scripting, interpretado y multiparadigma cuya apuesta principal se enfoca en la simplicidad, versatilidad y rapidez de desarrollo.
- **Serverless Framework:** Herramienta CLI de código abierto que facilita mediante comandos la construcción, configuración, implementación y despliegue de funciones serverless de manera agnóstica en diversos proveedores (Amazon AWS, Microsoft Azure, Google Cloud, IBM Open Whisk, Kubeless, etc).
- **Django o FastAPI:** Frameworks de desarrollo web escrito en Python diseñado para implementar aplicaciones robustas de manera ágil. Ambos brindan un conjunto de herramientas simples para la implementación de APIs de manera ágil, lo cual permitirá exponer las interfaces de comunicación del sistema #LaraBot.
- **Docker:** Plataforma de contenedores basados en Unix desarrollada en GoLang. Es útil para tener entornos independientes para cada componente o artefacto de una aplicación. Permite tener un entorno de ejecución "único" tanto en local, como en servidores de desarrollo o de producción.
- **Amazon Web Services:** AWS será el proveedor en la nube principal en el desarrollo de este proyecto, usando diferentes servicios de IaaS, BaaS, FaaS y SaaS ellos:
    - **AWS Identity and Access Management (IAM):** Servicio core de AWS para manejo de roles y permisos.
    - **API Gateway:** Servicio BaaS que sirve cómo disparador de eventos vía API, muy comúnmente utilizado con AWS Lambda.

- **DynamoDB:** Base de datos no relacional documental. Usada para almacenar contextos.
- **Simple Queue Service (SQS):** Servicio de colas.
- **S3:** Servicio de almacenamiento de archivos.
- **EC2:** Es un servicio de IaaS que permite contratar servidores virtuales en la nube que pueden ser configurados y actualizados de forma escalable.
- **Amazon Alexa:** Asistente virtual de Amazon sobre el cual es posible construir aplicaciones llamadas *Skills*. En este proyecto se buscará construir una *skill* base que pueda ser invocada y conectada con **#LaraBot** para la generación de respuestas.
- **AWS Lambda:** Plataforma FaaS de AWS para el aprovisionamiento y ejecución de funciones en la nube.
- **Cloudwatch:** Provee herramientas de monitoreo para seguimiento a eventos y visualización de logs. También permite definir reglas para la ejecución de tareas periódicas.
- **Relational Database Service (RDS):** Permite operar y configurar fácilmente bases de datos relacionales. En este proyecto se usará para aprovisionar una base de datos principal con PostgreSQL.
- **CloudFormation:** Permite aprovisionar de manera rápida y consistente un conjunto de servicios de AWS mediante infraestructura cómo código. En este proyecto se utilizará en conjunto con Serverless Framework para describir los componentes de la infraestructura y desplegarlos de manera sencilla.

8.2. Arquitectura de chatbot basada en componentes de tecnología específica

A continuación, se presenta una versión actualizada del diagrama de contexto de la arquitectura de chatbots completa presentada en la sección 6.3 cómo resultado del ADD-A.

En su gran mayoría toda la implementación del prototipo propuesto se ha llevado a cabo usando servicios de AWS y el desarrollo se podrá encontrar como código abierto bajo licencia MIT en GitHub siguiendo la siguiente url: https://github.com/LaraBot-HQ.

Como una prueba de concepto se hizo la implementación del #IntentDetector para entender las necesidades y problemas que podían surgir para su posterior implementación. En este caso se creó un microservicio con FastAPI para simplificar su construcción, además de no depender de bases de datos u otros recursos y de esta forma solo especificar una API. Este servicio utiliza un motor local y una implementación con Luis.ai para la obtención de intenciones (y la identificación sencilla de entidades). El motor local utiliza un modelo Word2Vec [102] pre entrenado e implementado con la herramienta Spacy con NLTK.

Tener en cuenta que las tecnologías de desarrollo elegidas no son más que una recomendación por parte de los autores de este proyecto basada en la experiencia, gustos y en muchos casos facilidad de implementación; por lo cual tanto los lenguajes de programación, librerías, frameworks y proveedores podrían ser sustituidas por otra tecnología de similares características.

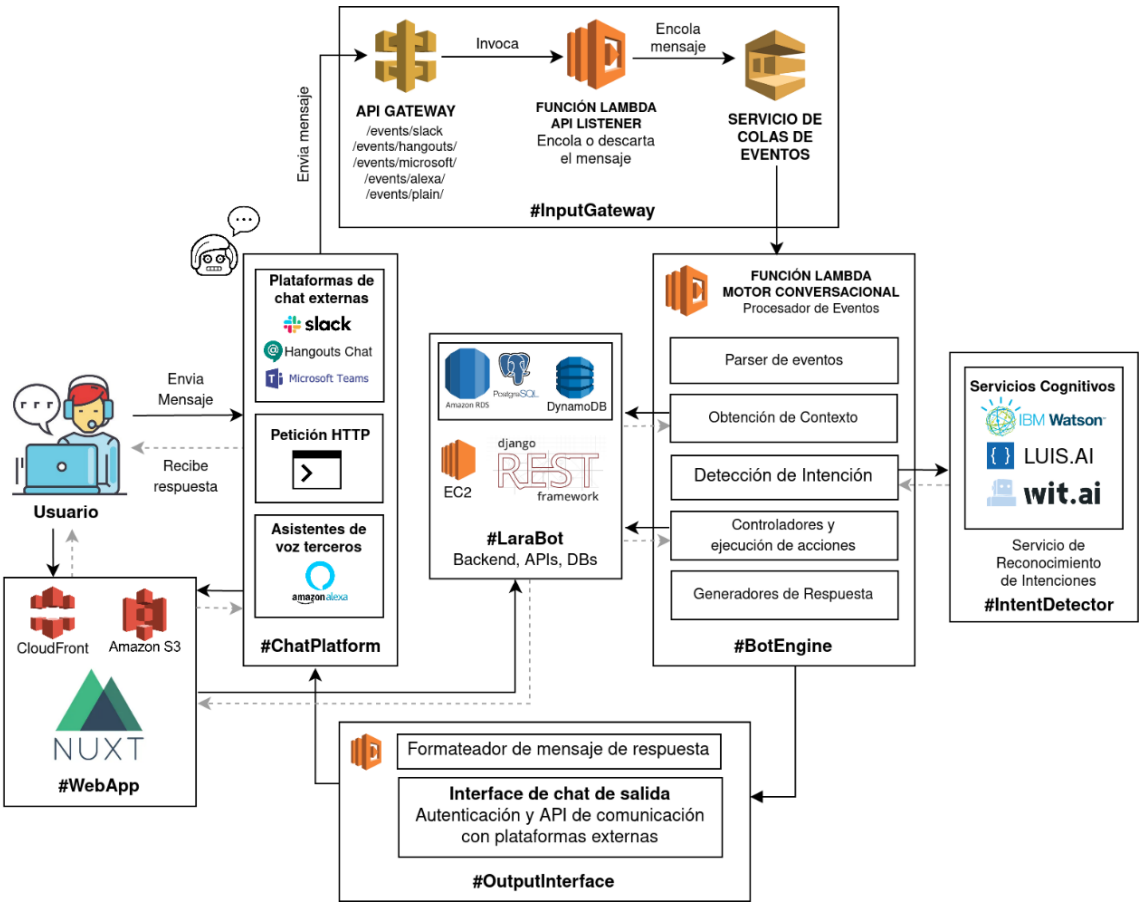

Figura 73: Arquitectura de chatbots prototipo.

9. CAPÍTULO X: CONCLUSIONES Y TRABAJO FUTURO

9.1. Conclusiones

El desarrollo y la ejecución del proyecto se llevaron a cabo buscando responder una serie de incógnitas relacionadas con los objetivos específicos planteados en la sección 1.3.2, las cuales servirán de base para establecer, clasificar y justificar las siguientes conclusiones:

¿Cuáles son las mejores metodologías y prácticas para diseñar una arquitectura de software?

- La metodología del SEI presentada en la sección 3.1.4 ha sido la más destacada tras una basta investigación académica y desarrollo de estado del arte. Esta metodología es un referente en la industria sobre cómo generar soluciones arquitectónicas empresariales de calidad, pero su ejecución en el contexto de equipos o proyectos pequeños no es práctica.
- Por lo anterior, la adaptación al SEI presentada en la sección 3.1.6, denominada como SEI-A, ha demostrado buenos resultados y se ha adoptado de manera ágil en la ejecución de este proyecto.
- Adoptar una metodología de diseño arquitectónico es de vital importancia y no debe dejarse en segundo plano en la implementación de un proyecto de software, por pequeño que sea.
- Las metodologías y prácticas tradicionales en el desarrollo de software suelen estar orientadas y pensadas hacia la satisfacción de requisitos funcionales, lo cual ha demostrado ser poco recomendado, pues dan como resultado soluciones inmaduras que demandan constantemente rediseños a corto plazo.
- Por lo anterior, metodologías como el SEI proponen orientar el diseño arquitectónico hacia la satisfacción de los atributos de calidad (requisitos no funcionales) que guíen la arquitectura hacia la implementación de soluciones robustas.

¿Qué atributos de calidad deberían satisfacer una arquitectura de índole conversacional?

- Entre los autores de este proyecto, se tuvieron involucrados a ingenieros de software con más de cinco años de experiencia desarrollando aplicaciones conversacionales, cuyo conocimiento ha sido clave para el desarrollo de este. Además de la experiencia, se tuvieron fuentes de consulta para definir la extensibilidad, mantenibilidad, escalabilidad e interoperabilidad como pilares de una arquitectura orientada al desarrollo de chatbots.
- Lo anterior se sustenta debido a la necesidad de las empresas y los usuarios que demandan la presencia de chatbots en múltiples canales de comunicación. Esto implica la integración del sistema a múltiples plataformas externas y servicios (cada uno de estos con diferentes protocolos, formatos, reglas, entre otras características). Los desarrolladores, al coordinar la integración con diferentes servicios cognitivos, permitirían la implementación de una arquitectura basada en componentes estandarizados y autónomos que pueden ser intercambiables entre sí. El resultado de esto es un diseño arquitectónico orientado hacia la extensibilidad y mantenibilidad.

- Adicionalmente, los chatbots han demostrado caracterizarse por tener un consumo de recursos muy variable. Por ejemplo, un agente conversacional que ayuda a una empresa a automatizar el *onboarding* de sus empleados tiene un uso muy escaso o nulo el resto del tiempo; o también, un chatbot orientado a la atención al cliente de un comercio electrónico puede tener picos de uso elevados en temporadas vacacionales, y decaer posteriormente. Teniendo en cuenta los picos de alta y baja concurrencia en un sistema la arquitectura debe estar preparada para usar recursos de forma sabia, lo cual se traduce idealmente en implementar una arquitectura que esté preparada para escalar recursos en demanda y, por consiguiente, en una alta disponibilidad y ahorro de costes a largo plazo.

¿Qué tipo de patrones de diseño y modelos de arquitectura de software son ideales para satisfacer los atributos de calidad mencionados?

- Tras experimentar con múltiples chatbots en el desarrollo de este proyecto, se ha encontrado un patrón en tendencia clave sobre el uso de modelos de computación en la nube basados en arquitecturas serverless (ver sección 2.3.3), las cuales han supuesto un cambio de paradigma para los desarrolladores en la forma en que se diseñan y ejecutan aplicaciones.
- Se puede concluir que serverless aporta un gran beneficio al área de desarrollo de chatbots, gracias a ser un modelo de computación en la nube orientado a eventos. Serverless es una arquitectura muy flexible y fácil de integrar a otros modelos, usa recursos en demanda y está diseñada para ser automáticamente escalable de forma predeterminada.
- A nivel de implementación durante el desarrollo del prototipo del caso de estudio, nos encontramos que el uso de patrones de diseño, como Strategy y Factory, juega un papel importante en cuanto a la permisión de la extensibilidad y mantenibilidad del código, dadas las necesidades de integración redundante a plataformas y servicios externos.
- Estudiar el funcionamiento de diversos chatbots ha aportado *insights* importantes en la toma de decisiones sobre el tipo de diseño que se quería lograr, en especial el asistente virtual de Amazon Alexa, que aportó información valiosa sobre cómo desarrollar un motor de procesamiento de intenciones y generación de respuestas, el cual se ha aplicado al diseño de la arquitectura de LaraBot, propuesto en el caso de estudio.

¿Cómo funcionan y cómo se integran con la arquitectura diseñada los servicios cognitivos o frameworks de código abierto que permiten la identificación de la intencionalidad en un mensaje?

- Tras un análisis comparativo de servicios cognitivos y frameworks de código abierto de NLP, se encontró cómo característica común el uso de modelos pre entrenados de propósito general para el reconocimiento de intencionalidad, los cuales sirven de base para entrenar un nuevo modelo con un corpus específico de intenciones y frases sinónimas relacionadas con la aplicación o sistema a implementar.
- Con base a los hallazgos anteriores se puede concluir que la elección de un sistema cognitivo u otro dependerá principalmente de los costes y que tan fácil se pueden integrar sus APIs

con servicios de desarrollos locales, dado que en general la mayoría de ellos ofrecen una buena precisión en el reconocimiento de intencionalidad.

- Estos servicios se integran con la arquitectura diseñada mediante el componente de detección de intenciones (#IntentDetector), el cual es un microservicio que recibe cómo entrada un mensaje y lo procesa mediante los servicios cognitivos internos o externos, entregando cómo resultado un identificador de intención basado en el corpus de intenciones preestablecido y un umbral de similitud mínimo tolerado.

¿Qué retos se deben resolver para lograr una adopción total de los chatbots en la industria?

- La inteligencia artificial y sus ramas de interés relacionadas con este proyecto, como el NLP y el Machine Learning, han alcanzado hitos muy relevantes en la última década que los han posicionado como las áreas de interés actuales. Sin embargo, estas áreas aún no son tan accesibles, ya que la velocidad con la que se puede realizar una implementación de algún algoritmo es muy limitada en comparación con la gran cantidad de contenido académico que surge cada año. Por un lado, esta lenta capacidad de implementar y mantener algoritmos (actualizados con su contraparte académica) ha generado que la inteligencia artificial y sus ramas de interés caigan en problemas que degradan la experiencia del usuario, pues los modelos genéricos fallan al brindar tratos personalizados; todo esto, a pesar de contar actualmente con una gran cantidad de servicios cognitivos emergentes con características interesantes. Por otro lado, el porcentaje de error sigue siendo alto y sus costos pueden ser elevados a medida que el sistema escala.
- Con base en la premisa anterior, se evidencia por qué gran parte de los chatbots implementados en el mercado siguen estando basados en una lógica de reglas y árboles de decisión y usando palabras clave y expresiones regulares simples como medios de identificación de intención, ya que es más barato y sencillo.
- Con base en las predicciones de Gartner y su modelo de tendencia sobre tecnologías emergentes (ver sección 2.3.1), se estima que aún faltan de 2 a 5 años de madurez en el desarrollo de chatbots para lograr una adopción productiva y accesible de las tecnologías relacionadas con la creación de experiencias conversacionales.
- Tras los resultados arrojados por la investigación y el diseño de la arquitectura, podemos concluir que el manejo de contextos es uno de los grandes retos que se presentan actualmente en el desarrollo de chatbots, dada la necesidad de ofrecer una asistencia personalizada a cada usuario y de retornar resultados basados en conversaciones previas que pueden modificar sustancialmente el significado de las intenciones; es decir, sin limitarse solo a una memoria a corto plazo, sino también utilizando una memoria de flujos pasados que recorra el historial de conversaciones.

¿Cómo se determina que la arquitectura diseñada cumple con los objetivos del proyecto?

- Cumpliendo con la metodología SEI-A, se siguió un proceso iterativo de diseño de componentes por etapas, en las cuales todas verificaban y garantizaban la correspondencia

de resultados con el conjunto de objetivos de negocio, requerimientos funcionales y no funcionales establecidos durante la fase inicial QAW-A en el capítulo 4.

- Adicionalmente durante el capítulo 6, con base a la metodología SEI-A se lleva a cabo un proceso de evaluación siguiendo el método ATAM-A, en el cual se crean un conjunto de matrices de correspondencia y trazabilidad que permiten verificar que los objetivos de negocio, requerimientos funcionales y no funcionales se articulan a lo largo del diseño de la arquitectura final.
- Finalmente, como un valor agregado se realizó la implementación de una prueba de concepto durante el capítulo 7, que permitió validar mediante experimentación el desarrollo de productos mínimos viables (MVP)[11] de algunos componentes del sistema y se presentó una propuesta de arquitectura prototipo con un diagrama de componentes y tecnologías recomendadas para su implementación.

¿Cómo permite la arquitectura diseñada ayudar a mejorar la implementación de interfaces conversacionales en la actualidad?

- El estado del arte presentado en este documento aporta un alto valor como base referencial para otros proyectos cuyos objetivos estén enmarcados en la especificación y diseño arquitectónico de un sistema que busque cumplir con altos estándares de calidad.
- La documentación y los resultados del diseño del caso de estudio sirven de base para la implementación de cualquier sistema de índole conversacional.
- La investigación y el estado del arte conglomerado en este proyecto permiten demostrar cómo potenciar las capacidades de un sistema a través de la interoperabilidad de una amplia gama de tecnologías, herramientas y conceptos de diferente índole (como lo pueden ser el uso de diferentes modelos de computación en la nube como SaaS, FaaS, IaaS y BaaS, además de conceptos como el Machine Learning, NLP, metodologías para el diseño de arquitecturas de software, patrones de diseño, servicios web, API Rest y entre otros).
- El diseño de arquitectura final de LaraBot parte de la experiencia de más de cinco años de nuestros arquitectos analizando e implementando experiencias conversacionales en diferentes plataformas y objetivos de negocio variados, lo cual aporta insights valiosos al estado del arte del desarrollo de chatbots.

[11] Producto Mínimo Viable (MVP): Una versión simple de un producto que permite llevar a cabo un conjunto de pruebas que permitan verificar su viabilidad.

9.2. Trabajos futuros

- A partir de una implementación madura del diseño del caso de estudio, es posible desarrollar un framework o librería genérica de código abierto que permita implementar chatbots de manera ágil.
- Presentar los resultados de este proyecto condensados en un artículo de investigación puede tener un gran valor como base referencial.
- Es posible extender la integración del chatbot presentado en el prototipo de LaraBot desde Slack, Google Chat y Microsoft Teams a más plataformas como Telegram, Whatsapp, Messenger, Instagram, entre otras, lo cual se ha especificado como característica clave en el diseño de la arquitectura dada su orientación hacia la extensibilidad e interoperabilidad.
- Extender la interacción de texto a audio es una tarea sencilla que se puede integrar en la implementación del sistema siguiendo el mismo modelo de arquitectura mediante la integración de servicios cognitivos de conversión de voz a texto y de texto a voz.
- En el estado del arte del NLP, específicamente en la sección 3.2.1.4, se hizo una breve recopilación sobre las diferentes formas en las que el NLP está integrado en las arquitecturas actuales. En esta sección se habló de cómo, dependiendo del objetivo de la arquitectura, se podía orientar el NLP en tres campos: el NLP para estructuración y formalización del lenguaje, el NLU para identificación de sentimientos e intenciones, y el NLG para la generación natural del idioma. El discurso de este proyecto estuvo enfocado en la problemática que hay alrededor de la poca capacidad de un chatbot para relacionar la intención del usuario con los recursos apropiados, lo cual es un área del NLU. Esto quiere decir que las soluciones obtenidas en esta arquitectura contemplan identificar la intención, pero no buscan generar respuestas personalizadas o adaptadas al usuario; de hecho, estas respuestas son generadas por un controlador de forma estática y muy restringida.

  Como fuente fundamental de trabajos futuros se plantea realizar una revisión de la arquitectura obtenida en este proyecto para la integración de un motor de generación de respuestas personalizadas basado en contexto e historial de conversación. El NLG es, en realidad, un área muy compleja que, aunque en su estado del arte ha tenido grandes avances sigue teniendo inconvenientes [98, 99], particularmente, se ha encontrado que grandes referentes y modelos del NLG, como GPT-3 tiene grandes resultados, pero no son muy buenos respondiendo preguntas [100], esto quiere decir que puede generar contenido idiomático a partir de una petición, pero cuando se le hace preguntas las respuestas generadas pueden sentirse raras y con muy poco sentido. Nuevamente se aclara que la naturaleza para generar contenido idiomático no hacía parte de los objetivos del proyecto, ya que este último buscaba generar respuestas más naturales a través de la identificación de intenciones.

  Estas problemáticas alrededor de la generación de respuestas naturales deberán ser abordadas en un proyecto futuro y reevaluar la arquitectura en términos de cómo esta implementación puede afectar la arquitectura de un sistema de chatbot.
- De igual forma que el NLG no era una prioridad en el proyecto, la creación de modelos propios e implementación de NLU nativo en nuestro sistema tampoco lo era, nos referimos a los microservicios de NLU en el módulo del **#IntentDetect**. El principal objetivo del

proyecto como se ha mencionado anteriormente era proporcionar una solución arquitectural a los problemas que los chatbots han tenido en su adopción. Es por ello por lo que, aunque en nuestra arquitectura se diseñó un módulo de microservicios para la identificación de intenciones de forma nativa en el sistema, este era un módulo secundario en comparación a los servicios cognitivos.

Sin embargo, un trabajo futuro que se considera interesante al ya tener una arquitectura extensible, mantenible y escalable como resultado de este proyecto es realizar una investigación para la implementación de técnicas de NLP/NLU libres a servicios de terceros, en este proyecto damos sugerencias de herramientas que pueden ser usadas para este propósito en el módulo de microservicios de NLP, un ejemplo es Spacy [69].

Aunque consideramos que el uso de servicios cognitivos es una solución que permite ahorrar recursos y desligarse del esfuerzo que puede significar la implementación de un modelo de NLU, lo cual se justifica en este proyecto, también entendemos las necesidades que hay alrededor de usar recursos propios para mejorar los procesos implementados en el sistema ya que tener este tipo de información da un valor de mercado importante, por lo cual tomamos este como un trabajo futuro retador e interesante.

10. CAPÍTULO XI: REFERENCIAS BIBLIOGRÁFICAS

[1] N. Albayrak, A. Özdemir and E. Zeydan, "An overview of artificial intelligence based chatbots and an example chatbot application" 2018 26th Signal Processing and Communications Applications Conference (SIU), Izmir, 2018, pp. 1-4. disponible en: https://ieeexplore.ieee.org/document/8404430.

[2]. A M Rahman, Abdullah Al Mamun and Alma Islam (2018). Programming challenges of chatbot: Current and future prospective. Dhaka, Bangladesh. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8288910.

[3]. Vagelis Hristidis (2019). Chatbot Technologies and Challenges. Laguna Hills, CA, USA. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8665692.

[4]. György Molnár & Zoltán Szüts (2018). The Role of Chatbots in Formal Education. Subotica, Serbia. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8524609.

[5]. R. Singh, M. Paste, N. Shinde, H. Patel and N. Mishra, "Chatbot using TensorFlow for small Businesses," 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, 2018, IEEE [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8472998.

[6]. Mohammad Nuruzzaman, Omar Khadeer Hussain (2018). A Survey on Chatbot Implementation in Customer Service Industry through Deep Neural Networks. Xi'an, China. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8592630.

[7] Paula Lauren and Paul Watta (2018). A Conversational User Interface for Stock Analysis. Los Angeles, CA, USA. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/9005635.

[8]. Artificial Solutions (2020). Chatbots: The Definitive Guide, THE RISE OF THE CONVERSATIONAL ASSISTANT. [En Línea]. Disponible desde: https://www.artificial-solutions.com/resources/white-papers/the-rise-of-the-conversational-assistant.

[9]. Florian Daniel, Maristella Matera, Vittorio Zaccaria and Alessandro Dell'Orto (2018). Toward Truly Personal Chatbots: On the Development of Custom Conversational Assistants. Gothenburg, Sweden. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8452787.

[10]. Amber Nigam, Prashik Sahare and Kushagra Pandya (2019). Intent Detection and Slots Prompt in a Closed-Domain Chatbot. Newport Beach, CA, USA. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8665635.

[11]. Kumar Shridhar, Ayushman Dash, Amit Sahu, Gustav Grund Pihlgren, Pedro Alonso, Vinaychandran Pondenkandath, György Kovács, Foteini Simistira and Marcus Liwicki (2019).

Subword Semantic Hashing for Intent Classification on Small Datasets. Budapest, Hungary. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8852420.

[12]. Bhriguraj Borah, Dhrubajyoti Pathak, Priyankoo Sarmah, Bidisha Som and Sukumar Nandi (2019). Survey of Text Based Chatbot in Perspective of Recent Technologies. Springer. [En Línea]. Disponible desde: https://link.springer.com/chapter/10.1007/978-981-13-8581-0_7

[13]. Uroš Arsenijevic and Marija Jovic (2020). Artificial Intelligence Marketing: Chatbots. Belgrade, Serbia. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/9007330.

[14]. Eko Handoyo, M. Arfan, Yosua Alvin Adi Soetrisno, Maman Somantri, Aghus Sofwan, Enda Wista Sinuraya (2018). Ticketing Chatbot Service using Serverless NLP Technology. Semarang, Indonesia. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8576921.

[15]. Pankaj R. Telang, Anup K. Kalia, Maja Vukovic, Rahul Pandita and Munindar P. Singh (2020). A Conceptual Framework for Engineering Chatbots. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8625898.

[16]. Mykhailo Lobur, Andriy Romanyuk and Mariana Romanyshyn (2011). Using NLTK for educational and scientific purposes. Polyana-Svalyava, Ukraine. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/5744524.

[17]. S. SeshathriAathithyan, M. V. Sriram, S. Prasanna and R. Venkatesan (2016). Affective — Hierarchical classification of text — An approach using NLP toolkit. Nagercoil, India. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/7530228.

[18]. Eko Handoyo, M. Arfan, Yosua Alvin Adi Soetrisno, Maman Somantri, Aghus Sofwan and Enda Wista Sinuraya (2018). Ticketing Chatbot Service using Serverless NLP Technology. Semarang, Indonesia. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8576921.

[19]. Mohit Sewak and Sachchidanand Singh (2018). Winning in the Era of Serverless Computing and Function as a Service. Pune, India. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8529465.

[20]. José Luis Vázquez-Poletti and Ignacio Martín Llorente (2018). Serverless Computing: From Planet Mars to the Cloud. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8625892.

[21]. Zaid Al-Ali, Sepideh Goodarzy, Ethan Hunter, Sangtae Ha, Richard Han and Eric Keller (2018). Making Serverless Computing More Serverless. San Francisco, CA, USA. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8457832.

[22]. Brian Manusama, Bern Elliot, Magnus Revang, Anthony Mullen (2019). Market Guide for Virtual Customer Assistants [En línea]. Disponible en https://www.gartner.com/en/documents/3947357/market-guide-for-virtual-customer-assistants.

[23]. Artificial Solutions (2020). [En Línea]. Disponible desde: https://www.artificial-solutions.com/chatbots.

[24]. Microsoft (2020). How bots work [En Línea]. Disponible desde: https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-basics.

[25]. 2 Megatrends Dominate the Gartner Hype Cycle for Artificial Intelligence, 2020. [En Línea]. Disponible desde: https://www.gartner.com/smarterwithgartner/2-megatrends-dominate-the-gartner-hype-cycle-for-artificial-intelligence-2020/.

[26]. Veton Këpuska and Gamal Bohouta (2018). Next-generation of virtual personal assistants (Microsoft Cortana, Apple Siri, Amazon Alexa and Google Home). Las Vegas, NV, USA. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8301638.

[27]. Jan-Gerrit Harms, Pavel Kucherbaev, Alessandro Bozzon and Geert-Jan Houben (2018). Approaches for Dialog Management in Conversational Agents. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8536470.

[28]. Aditya Ankur Yadav, Ishan Garg and Dr. Pratistha Mathur (2020). PACT - Programming Assistant ChaTbot. Jaipur, India. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8969070.

[29] ISO/IEC/IEEE Systems and software engineering -- Architecture description, in ISO/IEC/IEEE 42010:2011(E) desde: https://ieeexplore.ieee.org/document/6129467.

[30] K. E. Harper and A. Dagnino, Agile Software Architecture in Advanced Data Analytics, 2014 IEEE/IFIP Conference on Software Architecture, desde: https://ieeexplore.ieee.org/document/6827125

[31] ISO/IEC/IEEE International Standard - Software, systems and enterprise -- Architecture evaluation framework, in ISO/IEC/IEEE 42030:2019, desde: https://ieeexplore.ieee.org/document/8767001

[32] Characteristics of Good Architecture, Enterprise Architect User Guide v13.0, Sparx Systems, 2018, visitado ultima vez agosto 2020, url: https://sparxsystems.com/enterprise_architect_user_guide/13.0/guidebooks/ea_characteristics_of_good_architecture.html

[33] David Garlan and Mary Shaw. An Introduction to Software Architecture. SEI Technical Report CMU/SEI-94-TR-21.

[34] ISO/IEC/IEEE International Standard - Software, systems and enterprise -- Architecture processes, in ISO/IEC/IEEE 42020:2019(E) , vol., no., pp.1-126, 19 de Julio de 2019, desde: https://ieeexplore.ieee.org/document/8767004

[35] Software architecture in practice / Len Bass, Paul Clements, Rick Kazman. 3rd ed. p cm.-(SEI series in software engineering), septiembre de 2015.

[36] Stakeholder Theory: The State of the Art R. Edward Freeman, Jeffrey S. Harrison, Andrew C. Wicks, Bidhan L. Parmar, Simone de Colle, revisión del 2010.

[37] OMG. 2011. OMG Unified Modeling LanguageTM - Superstructure Version 2.4.1.

[38] "ISO/IEC/IEEE Systems and software engineering -- Architecture description," in ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000) , https://ieeexplore.ieee.org/document/6129467

[39] Medvidovic N, Taylor RN. A classification and comparison framework for software architecture description languages. IEEE Trans Softw Eng 2000;26:70–93, IEEE Press.

[40]. Eko Handoyo, M. Arfan, Yosua Alvin Adi Soetrisno, Maman Somantri, Aghus Sofwan, Enda Wista Sinuraya (2018). Ticketing Chatbot Service using Serverless NLP Technology. Semarang, Indonesia. IEEE. [En Línea]. Disponible desde: https://ieeexplore.ieee.org/document/8576921.

[41] Mengting Yan, Paul Christesen Castro, Perry Cheng, Vatche Ishakian (2016). Building a Chatbot with Serverless Computing. ACM. [En Línea]. Disponible desde: https://dl.acm.org/doi/10.1145/3007203.3007217.

[42] Gojko Adzic, Robert Chatley (2017). Serverless computing: economic and architectural impact. ACM. [En Línea]. Disponible desde: https://dl.acm.org/doi/abs/10.1145/3106237.3117767.

[43] Javascript. MDN web docs. Mozilla. [En Línea]. Disponible desde: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[44] Python. [En Línea]. Disponible desde: https://www.python.org/.

[45] SEI software architecture in practice. [En Línea]. Disponible desde: https://sites.google.com/site/softwarearchitectureinpractice/home.

[46] Node.js. [En Línea]. Disponible desde: https://nodejs.org.

[47] Mattsson, Michael & Grahn, Håkan & Mårtensson, Frans. (2006). Software Architecture Evaluation Methods for Performance, Maintainability, Testability, and Portability.

[48] Nuñez-Reyna and H. Cervantes, "Using Adapted Software Architecture Development Methods in a SOA Context," Mexican International Conference on Computer Science, Mexico City, 2010, [En línea] disponible desde: https://ieeexplore.ieee.org/document/5452556.

[49] History of the innovation at the SEI, Software Engineering institute, 2016, disponible desde: https://resources.sei.cmu.edu/asset_files/Book/2016_014_001_634914.pdf.

[50] Marcia Villalba. Curso de desarrollo de aplicaciones Serverless (2018). [En Línea]. Disponible desde: http://marciavillalba.com/sls-esp/.

[51] Flórez Galeano, Sergio Alexander; González Díaz, Jhonatan Esteban. Especificación del diseño de la arquitectura para prototipo de ajedrez que permite la interacción con personas que padecen discapacidad visual, 2014, [En línea] disponible desde: http://repositorio.utp.edu.co/dspace/handle/11059/5196.

[52] Prabal Mahanta, Anil Kumar Pole, Vittalraya Shenoy Adige and Rajkumar M. DevOps Culture and its impact on Cloud Delivery and Software Development (2016). IEEE. [En línea] desde: http://repositorio.utp.edu.co/dspace/handle/11059/5196.

[53] Architecture software, SEI approach for architecture software. [En línea] desde: https://www.sei.cmu.edu/our-work/projects/display.cfm?customel_datapageid_4050=21328

[54] Lic. José Ismael Nuñez Reyna. Adaptación de una Metodología de Desarrollo Arquitectónico al Contexto de Equipos de Desarrollo Pequeños (2009).Universidad Autónoma Metropolitana. Maestría en Ciencias y Tecnologías de la Información. [En línea] desde: http://bindani.izt.uam.mx:3000/catalog.html

[55] ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering (2018). [En línea] desde: https://ieeexplore.ieee.org/document/8559686.

[56] Prakash M Nadkarni, Lucila Ohno-Machado, Wendy W Chapman; Natural language processing: an introduction (2011). [En línea] disponible desde: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3168328/

[57] Ilya Pestov, A history of machine translation from the Cold War to deep learning, freecodecamp, 12 de marzo del 2018, [En línea] disponible desde: https://www.freecodecamp.org/news/a-history-of-machine-translation-from-the-cold-war-to-deep-learning-f1d335ce8b5/

[58] E. Cambria and B. White, "Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article]," in IEEE Computational Intelligence Magazine, vol. 9, no. 2, pp. 48-57, May 2014, [En línea] disponible desde: https://ieeexplore.ieee.org/document/6786458

[59] T. Young, D. Hazarika, S. Poria and E. Cambria, "Recent Trends in Deep Learning Based Natural Language Processing [Review Article]," in IEEE Computational Intelligence Magazine, vol. 13, no. 3, pp. 55-75, Aug. 2018, [En línea] disponible desde: https://ieeexplore.ieee.org/document/8416973.

[60] Markets and Markets. One stop solution for all Market Research & Consulting needs. [En línea] desde: https://www.marketsandmarkets.com/AboutUs-8.html

[61] R. Chang, C. Ziemkiewicz, T. M. Green and W. Ribarsky, "Defining Insight for Visual Analytics," in IEEE Computer Graphics and Applications, vol. 29, no. 2, pp. 14-17, March-April 2009, doi: 10.1109/MCG.2009.22. [En línea] disponible desde: https://ieeexplore.ieee.org/document/4797511.

[62] Top 12 Chatbots Trends and Statistics to Follow in 2020. REVE Chat. [En línea] desde: https://www.revechat.com/blog/chatbots-trends-stats/.

[63] Cheng yun Je, Aarhus University of Denmark, Genetic Algorithms, [En línea] desde: http://www.jade-cheng.com/au/coalhmm/optimization/

[64] Alejandra Puente Chandia, Ha Nui Baek, metodología para realizar predicción de fuga de clientes en una empresa de retail, [En línea] disponible desde: http://repositorio.uchile.cl/bitstream/handle/2250/103633/bastias_pl.pdf?sequence=4

[65] Vicente Gerardo Guzman Lucio. Comparativa de plataformas para Chatbots (2019). [En línea] desde: https://medium.com/mybots-latam/comparativa-de-plataformas-para-chatbots

[66] G. Abinaya, G. Ranjan and P. Aswin Karthik, "Continuous learning mechanism of NLU-ML models boosted by human feedback," 2019 International Conference on Computational Intelligence in Data Science (ICCIDS), Chennai, India, 2019, IEEE, [En línea] disponible desde: https://ieeexplore.ieee.org/document/8862102

[67] Vijay Polsani, Natural Language Processing (NLP) - Strategy for Enterprise Architecture, Julio 13 del 2020, Medium, [En línea] disponible desde: https://medium.com/@vijay.polsani/natural-language-processing-nlp-strategy-for-enterprise-architecture-9138b6af570b

[68] NLP Architecture by intel AI labs, 2019, [En línea] disponible desde: https://www.intel.com/content/www/us/en/artificial-intelligence/posts/introducing-nlp-architect-by-intel-ai-lab.html

[69] Spacy documentation, visitada en octubre del 2020, [En línea] disponible desde: https://spacy.io/usage/spacy-101

[70] NLTK, toolkit para el desarrollo y análisis de software con NLP, documentación, visitado en octubre del 2020, [En línea] disponible desde: https://www.nltk.org/book/ch08.html

[71] K. Kritikos and P. Skrzypek, "A Review of Serverless Frameworks," 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, 2018, pp. 161-168, doi: 10.1109/UCC-Companion.2018.00051. [En línea] disponible desde: https://ieeexplore.ieee.org/document/8605774.

[72] A. Palade, A. Kazmi and S. Clarke, "An Evaluation of Open Source Serverless Computing Frameworks Support at the Edge," 2019 IEEE World Congress on Services (SERVICES), Milan, Italy, 2019, pp. 206-211, doi: 10.1109/SERVICES.2019.00057. [En línea] disponible desde: https://ieeexplore.ieee.org/document/9049531.

[73] Brecht DeRooms, A Comparison of Serverless Function (FaaS) Providers (2020). [En línea] disponible desde: https://fauna.com/blog/comparison-faas-providers.

[74] Amazon Alexa. Overview of the Alexa Voice Service Device (AVS) SDK (2020). [En línea] disponible desde: https://developer.amazon.com/en-US/docs/alexa/avs-device-sdk

[75] Felix Bachmann Len Bass Gary Chastek, The Architecture Based Design Method. 2000

[76] Juan Ángel Pastor Franco, Evaluación y desarrollo incremental de una arquitectura software de referencia para sistemas de teleoperación utilizando métodos formales. 2002

[77] Andrea Delgado, Alberto Castro, Martín Germán, Evaluación de Arquitecturas de Software con ATAM (Architecture Tradeoff Analysis Method) un caso de estudio, 2007 http://www.bvs.hn/cu-2007/ponencias/CAL/CAL027.pdf

[78] Andrzej Zalewski, Szymon Kijas, Beyond ATAM: Early architecture evaluation method for large-scale distributed systems, 2012, [En línea] disponible desde: https://www.sciencedirect.com/science/article/abs/pii/S0164121212003032

[79] Nemías Saboya Ríos, Omar L Loaiza Jara, Danny Lévano Rodríguez, diseño de un modelo de arquitectura empresarial para publicaciones científicas basado en adm - togaf 9.0, 2018, [En línea] disponible desde: https://www.redalyc.org/jatsRepo/4676/467655911004/html/index.html

[80] Armando Cabrera S, Marco Abad E, Danilo Jaramillo H, Freddy Romero S, Definición y validación de la capa arquitectónica de aplicaciones a través de ADM-TOGAF y ADLs Definition and validation of the architectural application layer through ADM-TOGAF and ADLs, Agosto 2015, [En línea] disponible desde: https://www.researchgate.net/publication/282329821_Definition_and_validation_of_the_architectural_application_layer_through_ADM-TOGAF_and_ADLs

[81] Mohamed Taleb, Omar Cherkaoui, Pattern-Oriented Approach for Enterprise Architecture: TOGAF Framework, Diciembre 20 de 2011, [En línea] disponible desde: https://www.researchgate.net/publication/260317584_Pattern-Oriented_Approach_for_Enterprise_Architecture_TOGAF_Framework

[82] Robert L, Nord William G. Wood, Paul C. Clements, Integrating the Quality Attribute Workshop (QAW) and the Attribute-Driven Design (ADD) Method, julio de 2004, [En línea]: https://resources.sei.cmu.edu/asset_files/TechnicalNote/2004_004_001_14321.pdf

[83] John Bergey, Mario Barbacci, William Wood, Using Quality Attribute Workshops to Evaluate Architectural Design Approaches in a Major System Acquisition: A Case Study, julio 2000.

[84] Ashraf Anwar, A Review of RUP (Rational Unified Process), 2014, [En línea] disponible desde: https://www.cscjournals.org/library/manuscriptinfo.php?mc=IJSE-142

[85] Scott W. Amabler, A Manager's Introduction to The Rational Unified Process (RUP), Diciembre 2005, [En línea] disponible desde: http://www.ambysoft.com/downloads/managersIntroToRUP.pdf

[86] Roy Thomas Fielding, Architectural Styles and the Design of Network-based Software Architectures, 2000, [En línea] disponible desde: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[87] University of houston, The Rational Unified Process, [En línea] disponible desde: https://sceweb.uhcl.edu/helm/RationalUnifiedProcess/

[88] Software engineering institute(SEI), Attribute-Driven Design, 2018, [En línea] disponible desde: https://resources.sei.cmu.edu/asset_files/FactSheet/2018_010_001_513930.pdf

[89] Software engineering institute(SEI), The SEI Quality Attribute Workshop (QAW), [En línea]: https://resources.sei.cmu.edu/asset_files/FactSheet/2018_010_001_513488.pdf

[90] Paul Clements, Rick Kazman, Mark Klein, Evaluating Software Architectures: Methods and Case Studies, 2001, capítulo 2, [En línea] disponible desde: https://www.pearsonhighered.com/assets/samplechapter/0/2/0/1/020170482X.pdf

[91] Serverless framework documentation, octubre de 2020, [En línea] disponible desde: https://www.serverless.com/framework/docs/

[92] Google cloud,What are containers?, enero de 2020, [En línea] disponible desde: https://cloud.google.com/containers

[93] Docker documentation, What is a container?, enero de 2020, [En línea] disponible desde: https://www.docker.com/resources/what-container.

[94] Refactoring guru, patron de diseño Strategy, visitado en noviembre de 2020, [En línea] disponible desde: https://refactoring.guru/es/design-patterns/strategy

[95] Gartner, Information technology, glossary and concepts, Framework, API and SDK. [En línea] disponible desde: https://www.gartner.com/en/information-technology/glossary/

[96] Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures (2000). [En línea] disponible desde: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[97] MDN web docs. Hypertext Transfer Protocol (HTTP). [En línea] disponible desde: https://developer.mozilla.org/en-US/docs/Web/HTTP.

[98] Medium, Abhishek Sunnak, Evolution of the NLG, Marzo 15 de 2019, [En línea] desde: https://medium.com/sfu-cspmp/evolution-of-natural-language-generation-c5d7295d6517

[99] Medium, A Comprehensive Guide to Natural Language Generation, Julio 4 de 2019 [En línea] desde: https://medium.com/sciforce/a-comprehensive-guide-to-natural-language-generation-dd63a4b6e548

[100] Mindmatters, news, GPT-3 Is "Mindblowing" If You Don't Question It Too Closely, recuperado en Noviembre de 2020, [En línea], disponible desde: https://mindmatters.ai/2020/07/gpt-3-is-mindblowing-if-you-dont-question-it-too-closely/

[101] D. Jurafsky, (Speech and Language Processing), recuperado en Diciembre de 2020, [En línea], disponible desde: https://web.stanford.edu/~jurafsky/slp3/.

[102] Spacy. Linguistic Features. Word vectors and semantic similarity [En línea], disponible desde: https://spacy.io/usage/linguistic-features#vectors-similarity.