Informe Lab2 – Prolog

**Nombre:** Víctor Duarte Arce.

**Rut:** 21.467.246-4

Tabla de contenidos:

|  |
| --- |
| [Introducción.](#_Introducción:_1) |
| [Descripción del problema.](#_Descripción_del_problema:) |
| [Descripción del paradigma.](#_Descripción_del_paradigma) |
| [Análisis del problema.](#_Análisis_del_problema:) |
| [Diseño de la solución.](#_Diseño_de_la) |
| [Aspectos de Implementación.](#_Aspectos_de_la) |
| [Instrucciones de uso.](#_Instrucciones_de_uso:) |
| [Resultados y autoevaluación.](#_Resultados_y_autoevaluación:) |
| [Conclusiones del Trabajo.](#_Conclusiones_del_Trabajo:) |
| [Referencias](#_Referencias:). |
| [Anexos](#_Anexos). |

# Introducción:

Si bien explica a grandes rasgos el informe y que lo contiene, ¿dónde queda la motivación para el lector?

El problema consiste en desarrollar un sistema para la creación, despliegue y administración de “chatbots” estructurados, específicamente “chatbots” de Respuesta de Interacción a Texto (ITR). Estos “chatbots” responden a preguntas basadas en opciones específicas y no involucran procesamiento avanzado del lenguaje natural ni inteligencia artificial. Los estudiantes deben implementar un ambiente que permita:

1. Crear un entorno contenedor de “chatbots”.
2. Crear “chatbots” individuales.
3. Identificar “chatbots”.
4. Agregar preguntas y opciones a los “chatbots”.
5. Especificar enlaces para dirigir la interacción entre “chatbots”.
6. Interactuar con los “chatbots” a través de opciones y palabras clave.
7. Ofrecer una síntesis de las interacciones.

Los “chatbots” pueden estar interconectados, de modo que las opciones de un “chatbot” puedan dirigir a un usuario a otro “chatbot”. Los estudiantes deben cumplir con requisitos mínimos obligatorios y realizar autoevaluaciones de su trabajo.

# Descripción del problema:

Descripción breve del problema (en sus palabras - máx. 1/2 página) (5%)

No existe

# Descripción del paradigma (Declarativo Lógico):

Se describe el paradigma, no el lenguaje a utilizar, son cosas distintas.

2) Descripción breve del paradigma y los conceptos de este que se ven aplicados en este proyecto (en sus palabras - máx. 1/2 página) (5%)

La base del paradigma declarativo lógico son las relaciones, las cuales se definen como proposiciones que pueden ser verdaderas o falsas.

“Un programa escrito en un lenguaje declarativo como Prolog, puede usarse como una especificación de un problema, en lugar de establecer pasos para llegar a la solución de un problema” (Flores V. 2020).

# Análisis del problema:

3) Análisis del problema respecto de los requisitos específicos que deben cubrir (máx. 1 página) (10%)

En esta sección no se habla de cómo abordar el problema, eso va en Diseño de la solución.

En esta sección se analizan los requerimientos solicitados.

**Objetivos del Sistema:**

**El sistema debe permitir a los usuarios:**

1. **Crear un ambiente contenedor de chatbots:** Los usuarios deben poder crear un entorno que albergue múltiples chatbots.
2. **Crear chatbots individuales:** Se deben poder crear chatbots específicos, cada uno con su conjunto de opciones y preguntas.
3. **Identificar chatbots:** Los usuarios deben poder identificar y distinguir entre los chatbots disponibles en el sistema.
4. **Agregar preguntas y opciones:** Para cada chatbot, los usuarios deben poder agregar preguntas y definir opciones para que el chatbot responda a las preguntas con las opciones proporcionadas.
5. **Especificar enlaces para la interacción:** Deben poder definir cómo los chatbots interactúan entre sí, permitiendo que las opciones de un chatbot dirijan a los usuarios a otro chatbot.
6. **Interactuar con los chatbots:** Los usuarios deben poder interactuar con los chatbots, ya sea seleccionando opciones específicas o utilizando palabras clave que coincidan con las opciones.
7. **Ofrecer una síntesis de las interacciones:** Al finalizar la interacción con un chatbot, el sistema debe proporcionar una síntesis de la conversación y la acción tomada.

**Características Clave:**

1. Los chatbots son estructurados y no implican procesamiento avanzado del lenguaje natural o inteligencia artificial.
2. Los chatbots pueden estar interconectados, permitiendo una conversación fluida y dirigida por opciones.
3. El sistema debe ser flexible y permitir la creatividad de los usuarios en la configuración de los chatbots y sus interacciones.

# Diseño de la solución:

4) Diseño de la solución (presentar su enfoque de solución, describir, diagramar, descomposición de problemas, algoritmos o técnicas empleados para problemas particulares, recursos empleados) (máx. 1 página. Puede incluir diagramas como anexos fuera del límite de las 5 páginas) (30%)

En esta sección debe hablar de sus TDAs, debe describirlos y justificar sus decisiones. Debe explicar cómo funciona su solución y qué estrategias usa.

**Enfoque de Solución:**

El enfoque de solución se basa en la creación de un sistema de chatbot en el lenguaje de programación Prolog. El sistema permite la creación de múltiples chatbots, flujos de conversación y opciones de respuesta, así como la interacción con usuarios. El diseño se centra en la estructura de datos eficiente para garantizar la escalabilidad y flexibilidad del sistema.

**Descomposición del Problema:**

La solución se descompone en los siguientes componentes clave:

* **TDA System:** Este TDA consiste en una lista compuesta de los siguientes TDAs:
  + **name (string):** Representa el nombre del sistema.
  + **InitialChatbotCodeLink (int):** Es la id del chatbot de inicial o de bienvenida al usuario.
  + **chatbots:** Es una lista de 0 o más chatbots.
  + **users:** Lista de 0 o más users.
    - **user (string):** Es un string con el nombre del usuario
  + **chatHistory:** Es una lista de 0 o más userHistory
    - **userHistory:** Es una lista de dos elementos compuesta del usuario y de su registro de conversaciones con el sistema.
  + **actCID (int):** Un entero que representa la id del chatbot con el que el usuario está conversando actualmente, sirve para moverse entre chatbots.
  + **actFId (int):** Un entero que representa la id del flow con el que el usuario está conversando actualmente.
* **TDA Chatbot:** Es una lista compuesta por los siguientes TDAs:
  + **chatbotID (int):** Representa el identificador del chatbot dentro del sistema.
  + **name (string):** Es el nombre del chatbot en cuestión.
  + **welcomeMessage (string):** Representa el mensaje de bienvenida del chatbot, lo utiliza el chatbot de bienvenida o inicial del sistema.
  + **startFlowId (int):** Es la id del flow con el que el chatbot inicia, se usa cuando el usuario habla por primera vez con el sistema.
  + **flows:** Lista de 0 o más flow.
* **TDA Flow:** Es una lista compuesta por los siguientes TDAs:
  + **id (int):** Es el identificador del flow en un chatbot.
  + **name-msg (string):** Es el nombre del flujo, puede ser una pregunta como “¿A qué lugares quiere viajar?”.
  + **options:** Lista de 0 o más opciones.
* **TDA Option:** Es una lista compuesta por los siguientes TDAs:
  + **code (int):** Es el identificador de la opción en un flow.
  + **message (string):** Es el mensaje que se mostrará en pantalla, un ejemplo puede ser “1) Chile”.
  + **ChatbotCodeLink (int):** Es la id del chatbot al cual se enviará si se elige la opción.
  + **InitialFlowCodeLink (int):** Es la id del flow al cual se enviará si se elige la opción.
  + **Keywords:** Es una lista de 0 o más palabras con la que se puede referenciar a la opción

En resumen, la descomposición del problema se basó en estructuras de listas (ver Figura x2), la cual cada elemento marcado en rojo implicó la existencia de un nuevo archivo, ya que a pesar de que estas puede que no sean la estructura más eficiente de todas, son las más fáciles e intuitivas de usar.

**Algoritmos y Técnicas:**

Para la solución, se usaron los siguientes algoritmos basados en listas:

* **Búsqueda secuencial:** se usó para verificar la existencia de elementos en una lista (Figura x).

Texto

Descripción generada automáticamente

Figura x: “option\_member\_list(O, L).” Elaboración propia.

* **Filtración de elementos repetidos:** este algoritmo usó la búsqueda secuencial para filtrar elementos repetidos de la siguiente manera (Figura x1).

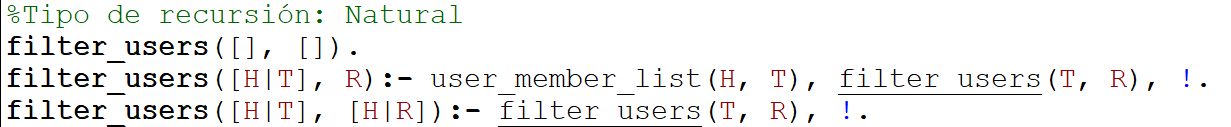


Figura x1: “filter\_users(UsersIn, UsersOut).” Elaboración propia.

También se utilizaron las siguientes técnicas:

* **Descomposición de problemas:** Para abordar cada problema planteado, se dividió en varios subproblemas. Por ejemplo, al enfrentarnos al desafío de crear un flujo que no contenga opciones repetidas, fue necesario resolver el subproblema de filtrar una lista de opciones para asegurar que ninguna se repitiera. A su vez, esta tarea implicó resolver otro subproblema: reconocer si una opción pertenece a una lista según su identificador (ver Figura x).
* **Abstracción de la solución:** Para implementar un ambiente generador de chatbots, se abordó el problema de una forma abstracta, pensando en distintos TDAs, como por ejemplo el TDA option, el cual tiene distintas operaciones que pueden ser utilizadas para la creación de la solución.

**Recursos Empleados:**

* **Lenguaje de Programación Prolog:** Se utiliza Prolog como el lenguaje principal para implementar el sistema de chatbot debido a su capacidad de manejar estructuras de datos y funciones de manera eficiente.
* **Estructuras de Datos**: Se emplean listas y funciones lambda para representar y gestionar las opciones, flujos, chatbots y usuarios.

# Aspectos de la implementación:

5) Aspectos de implementación (estructura del proyecto, bibliotecas empleadas, razones de su elección, compilador o interprete usado, etc.) (máx. 1/5 página) (10%)

Falta mencionar qué archivos tiene y cómo los conecta, también el porqué.

Para la implementación de esta solución, se ha utilizado el lenguaje de programación Racket. Racket es una elección apropiada para este proyecto debido a su capacidad para trabajar con estructuras de datos y funciones lambda de manera efectiva. No se han empleado bibliotecas externas, ya que Racket proporciona las herramientas necesarias para el desarrollo del sistema de chatbot.

El intérprete de Racket se utilizó para ejecutar el proyecto, y la elección de Racket se basó en su capacidad para trabajar con listas de manera eficiente, su sintaxis clara y legible, su versatilidad, su amplia comunidad y su portabilidad.

En resumen, la implementación se realizó en Racket debido a su idoneidad para manipular estructuras de datos y su facilidad de lectura, sin necesidad de bibliotecas externas.

# Instrucciones de uso

6) Instrucciones de uso (ejemplos, resultados esperados, posibles errores) (máx. 1/2 página. Complementar con ejemplos detallados en anexos que están fuera del límite de 5 páginas) (15%)

Dice qué hay que hacer, pero no el cómo. Es importante tener claridad en esta sección para que cualquiera pueda utilizar tu programa. También es posible acompañar con ejemplos, posibles fallos y otros.

**Paso 1: Ejecución del Programa**

Para ejecutar el programa, simplemente abra un entorno de Prolog y consulte el archivo TDASystem\_214672464\_DuarteArce.pl

**Paso 2: Creación del Entorno**

Para crear el entorno usted debe definir las options, los flows, los chatbots y el system que utilizará en este respectivo orden, ya que el system contiene muchos chatbots, que a su vez contienen flows, que a su vez contienen options.

Para definir las options, los flow, los chatbots y el system, debe consultar los predicados dejando una variable al final de cada uno para poder almacenarlos, luego separar por el conector “,”.

Ejemplo:

option(1, "1) Viajar", 2, 1, ["viajar", "turistear", "conocer"], OP1), option(2, "2) Estudiar", 2, 1, ["estudiar", "aprender", "perfeccionarme"], OP2), option(3, "3) Comer", 2, 1, ["comer", "aprender"], OP3), flow(1, "flujo1", [OP1, OP2], F10), flowAddOption(F10, OP3, F11), chatbot(0, "Inicial", "Bienvenido\n¿Qué te gustaría hacer?", 1, [F11], CB0), system("Chatbots Paradigmas", 0, [CB0], S0), systemAddUser(S0, "user1", S2), systemAddUser(S2, "user2", S3), systemAddUser(S3, "user3", S5), systemLogin(S5, "user1", S7), systemLogout(S7, S9), systemLogin(S9, "user2", S10).

**Paso 3: Interacción con el Sistema**

Para interactuar con el sistema, tiene los siguientes predicados:

* systemTalkRec(SystemIn, Message, SystemOut).
* systemSynthesis(SystemIn, User, String).
* systemSimulate(SystemIn, MaxInteractions, Seed, SystemOut).

Ejemplo:

systemTalkRec(S10, "hola", S11), systemTalkRec(S11, "1", S12), systemTalkRec(S12, "1", S13), systemTalkRec(S13, "Museo", S14), systemTalkRec(S14, "1", S15), systemTalkRec(S15, "3", S16), systemTalkRec(S16, "5", S17), systemSynthesis(S17, "user2", Str1), systemSimulate(S3, 5, 32131, S99), systemSynthesis(S99, "user", Str2), write(Str2).

Si quiere ver un ejemplo de uso vea el anexo.

# Resultados y autoevaluación

7) Resultados y autoevaluación: Resumen de resultados obtenidos. Esto puede abordarse listando todos los requerimientos del proyecto en una tabla indicando el grado de alcance. Indicar que tipos de pruebas se hicieron. Cuantas de las pruebas fueron exitosas, cuantas fracasaron, razones de fallos. Especificar qué funciones no se completaron y el por qué no se completaron. (máximo 1/5 página) (10%)

Es esencial mencionar cómo probó lo que afirma. También debe ser más claro con su autoevaluación (por ejemplo, referenciarla). También es importante explicar por qué no se logró lo que faltó.

|  |  |  |
| --- | --- | --- |
| **Predicado** | **Grado de alcance (0-1)** | **Descripción** |
| option | 1 |  |
| flow | 1 |  |
| flowAddOption | 1 |  |
| chatbot | 1 |  |
| chatbotAddFlow | 1 |  |
| system | 1 |  |
| systemAddChatbot | 1 |  |
| systemAddUser | 1 |  |
| systemLogin | 1 |  |
| systemLogout | 1 |  |
| systemTalkRec | 1 |  |
| systemSynthesis | 1 |  |
| systemSimulate | 1 |  |

# Conclusiones del Trabajo

Muy bien, pero falta conectar la conclusión a la problemática planteada.

Como recomendación, intente dejar más de lado lo personal y dele un enfoque más relacionado a la ingeniería.

**Alcances:**

El sistema de chatbot implementado en Racket permite la creación y gestión de múltiples chatbots, flujos de conversación y opciones de respuesta.

Los usuarios pueden interactuar con los chatbots a través de mensajes de texto, y el sistema responde según las opciones disponibles en cada flujo de conversación.

Las funcionalidades básicas, como la creación de chatbots, flujos y opciones, así como la interacción con los usuarios, se implementaron de manera exitosa.

**Limitaciones:**

La implementación actual es una versión básica del sistema de chatbot y no incluye características avanzadas como el procesamiento de lenguaje natural (NLP) o la gestión de contextos complejos.

La interfaz de usuario es limitada y se realiza a través de líneas de comandos en un entorno de desarrollo Racket, lo que puede ser poco amigable para usuarios no técnicos.

La falta de una interfaz gráfica de usuario (GUI) limita la accesibilidad y la usabilidad del sistema.

**Dificultades y Desafíos:**

El paradigma de programación funcional, aunque poderoso y elegante, puede ser menos intuitivo para aquellos familiarizados con enfoques más tradicionales como la programación orientada a objetos (POO).

La falta de experiencia previa en Racket y la ausencia de bibliotecas específicas para el desarrollo de chatbots en este lenguaje representaron un desafío en la implementación.

La gestión de flujos de conversación y opciones requiere un diseño cuidadoso para garantizar la coherencia y la eficiencia.

# Referencias

No tiene ninguna cita (debe hacerlas en formato APA).

¿Hizo todo el informe con una sola referencia? ¿Tampoco utilizó nada de lo visto en clases?

# Anexos

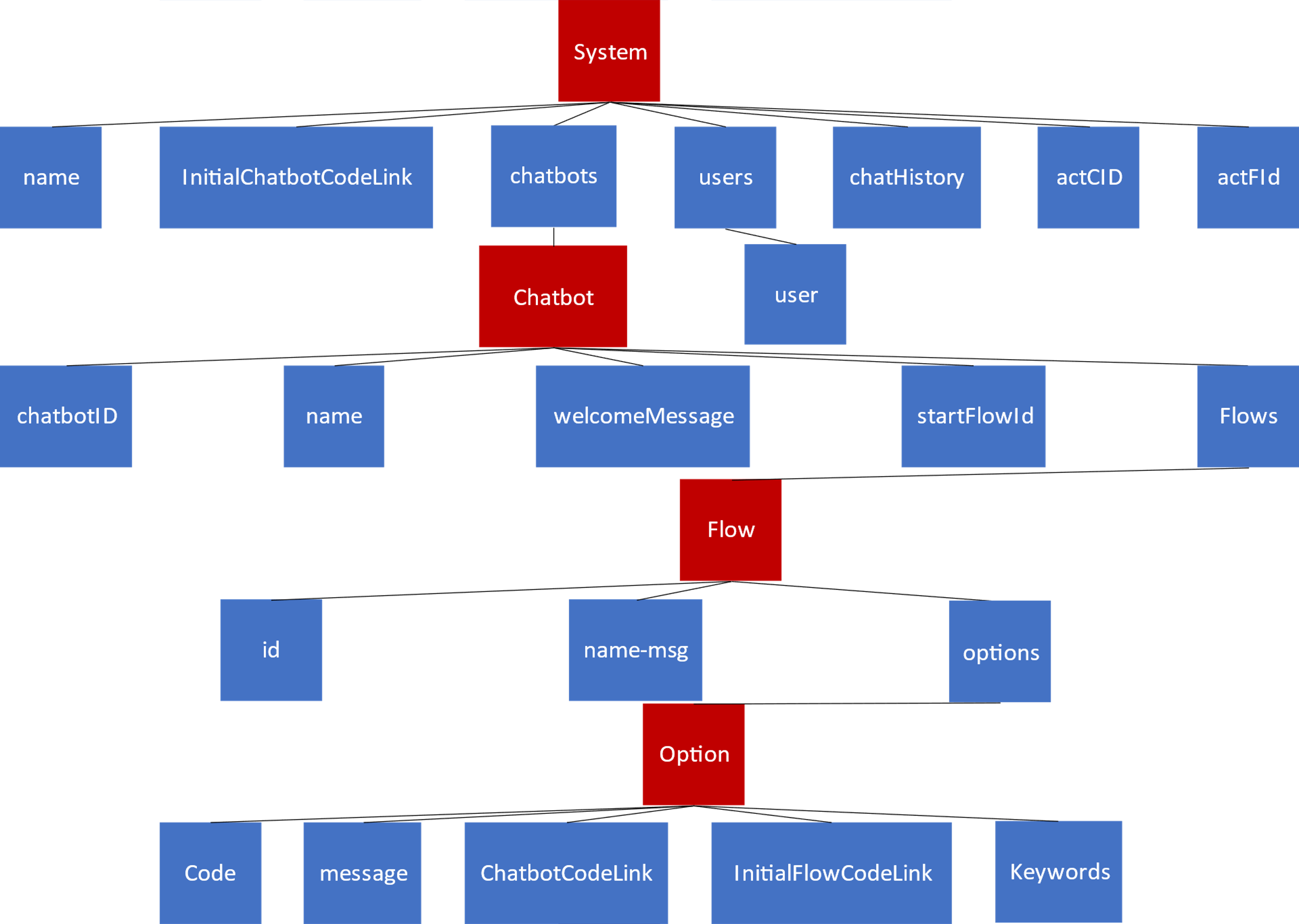


Figura x2: “Diagrama de estructura de system”. Elaboración propia.

Ejemplo de uso del sistema:

option(2, "2) Estudiar", 2, 1, ["estudiar", "aprender", "perfeccionarme"], OP2), option(1, "1) Viajar", 1, 1, ["viajar", "turistear", "conocer"], OP1), flow(1, "flujo1", [OP2], F10), flowAddOption(F10, OP1, F11), chatbot(0, "Inicial", "Bienvenido\n¿Qué te gustaría hacer?", 1, [F11], CB0), option(1, "1) New York, USA", 1, 2, ["USA", "Estados Unidos", "New York"], OP3), option(2, "2) París, Francia", 1, 1, ["Paris", "Eiffel"], OP4), option(3, "3) Torres del Paine, Chile", 1, 1, ["Chile", "Torres", "Paine", "Torres Paine", "Torres del Paine"], OP5), option(4, "4) Volver", 0, 1, ["Regresar", "Salir", "Volver"], OP6), option(1, "1) Central Park", 1, 2, ["Central", "Park", "Central Park"], OP7), option(2, "2) Museos", 1, 2, ["Museo"], OP8), option(3, "3) Ningún otro atractivo", 1, 3, ["Museo"], OP9), option(4, "4) Cambiar destino", 1, 1, ["Cambiar", "Volver", "Salir"], OP10), option(1, "1) Solo", 1, 3, ["Solo"], OP11), option(2, "2) En pareja", 1, 3, ["Pareja"], OP12), option(3, "3) En familia", 1, 3, ["Familia"], OP13), option(4, "4) Agregar más atractivos", 1, 2, ["Volver", "Atractivos"], OP14), option(5, "5) En realidad quiero otro destino", 1, 1, ["Cambiar destino"], OP15), flow(1, "Flujo 1 Chatbot1\n¿Dónde te Gustaría ir?", [OP3, OP4, OP5, OP6], F20), flow(2, "Flujo 2 Chatbot1\n¿Qué atractivos te gustaría visitar?", [OP7, OP8, OP9, OP10], F21), flow(3, "Flujo 3 Chatbot1\n¿Vas solo o acompañado?", [OP11, OP12, OP13, OP14, OP15], F22), chatbot(1, "Agencia Viajes", "Bienvenido\n¿Dónde quieres viajar?", 1, [F20, F21, F22], CB1), option(1, "1) Carrera Técnica", 2, 1, ["Técnica"], OP16), option(2, "2) Postgrado", 2, 1, ["Doctorado", "Magister", "Postgrado"], OP17), option(3, "3) Volver", 0, 1, ["Volver", "Salir", "Regresar"], OP18), flow(1, "Flujo 1 Chatbot2\n¿Qué te gustaría estudiar?", [OP16, OP17, OP18], F30), chatbot(2, "Orientador Académico", "Bienvenido\n¿Qué te gustaría estudiar?", 1, [F30], CB2), system("Chatbots Paradigmas", 0, [CB0], S0), systemAddChatbot(S0, CB1, S01), systemAddChatbot(S01, CB2, S02), systemAddUser(S02, "user1", S2), systemAddUser(S2, "user2", S3), systemAddUser(S3, "user3", S5), systemLogin(S5, "user1", S7), systemLogout(S7, S9), systemLogin(S9, "user2", S10), systemTalkRec(S10, "hola", S11), systemTalkRec(S11, "1", S12), systemTalkRec(S12, "1", S13), systemTalkRec(S13, "Museo", S14), systemTalkRec(S14, "1", S15), systemTalkRec(S15, "3", S16), systemTalkRec(S16, "5", S17), systemSynthesis(S17, "user2", Str1), systemSimulate(S3, 5, 32131, S99), systemSynthesis(S99, "user", Str2), write(Str2).