UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Informe de Laboratorio 1: Creación de un Simulador de Sistema de Chatbots Simplificado en Scheme

Integrantes: Agustín Vera
Profesor: Edmundo Leiva
Asignatura: Paradigmas de programación

Tabla de contenido

1.- Introducción

En el presente informe correspondiente al curso Paradigmas de Programación. Se expondrá una solución propuesta a un problema planteado. La solución se planteará desde la perspectiva del paradigma funcional, mediante el lenguaje de programación Scheme junto al compilador DR. Racket y será explicada en el siguiente informe. El informe posee la siguiente estructura: Introducción con sus subcategorías: descripción del problema, descripción del paradigma, objetivos del proyecto, después se tiene, desarrollo, con sus subcategorías correspondientes: análisis del problema, diseño de la solución, aspectos de implementación, ejemplos de uso, resultados y autoevaluación, finalizando se encuentra una conclusión al trabajo realizado, la bibliografía consultada y anexos.

1.1.- Descripción del problema

Se desea construir un sistema para la creación, despliegue y administración de chatbots simplificados, como por ejemplo, los chatbots de ITR (Respuesta de Interacción a Texto). Estos sistemas permiten a sus usuarios entrar a un sistema mediante un registro en concreto, para luego poder interactuar con el sistema de chatbots. Las interacciones que realizan los usuarios tienen una respuesta concreta en texto del sistema de chatbots. Los chatbots dentro del sistema pueden tener relaciones de flujo entre ellos para lograr una interacción más complementaria a lo que el usuario desee, además los usuarios pueden visualizar su historial de interacciones con el sistema.

1.2.- Descripción del paradigma

El paradigma funcional, es un paradigma en donde el enfoque en la resolución de problemas no toma como objetivo al ¿Cómo? resolver una problemática, más bien realiza su enfoque respondiendo al ¿Qué?

En este paradigma se utilizan las llamadas “funciones”, dichas funciones son las utilizadas para resolver las problemáticas con el paradigma. Una función se puede definir como un algoritmo que frente a una entrada en concreto siempre entrega la misma salida, siendo estos el dominio y recorrido de la función respectivamente. En este paradigma solo se trabaja con funciones, no se tiene asociado el concepto de variable utilizado en otros paradigmas.

Las herramientas que proporciona el paradigma funcional a la hora de resolver problemas son:

Funciones anónimas: Son funciones que se expresan sin nombre dado una entrada y no se almacenan en memoria.

Recursividad: Es una propiedad que poseen las funciones en donde se llaman a si mismas para resolver una parte de la problemática en concreto. Existen tres tipos de recursión: de cola, natural y arbórea.

Funciones de orden superior: Son aquellas funciones que pueden recibir a otras funciones como parámetro.

Currificación: Son funciones que reciben sus argumentos de manera secuencial, dejan un procedimiento a la espera de otros procedimientos que esperan.

1.3.- Objetivos

Como objetivos del proyecto se tiene aprender y entender el paradigma funcional, mediante el lenguaje Scheme en conjunto con el compilador Dr. Racket, además, la aplicación de estos conocimientos para dar solución a la problemática planteada como enunciado del laboratorio.

2.- Desarrollo

2.1.- Análisis de problema

Se tiene la entidad de sistema de chatbots, el cual almacenará los diferentes chatbots que el sistema necesite para su funcionamiento, los diferentes usuarios con sus respectivos historiales de interacción con el sistema de chatbots. Los chatbots en cuestión, pueden ser agregados al sistema, también pueden interactuar entre sí, esto lo realizan mediante una serie de flujos que estos poseen y permiten la interacción. Las funciones que debe aceptar el sistema para su funcionamiento son:

option, flow, flow-add-option, chatbot, chatbot-add-flow, system, system-add-chatbot, system-add-user, system-login, system-logout, system-talk-rec, system-talk-norec, system-synthesis, system-simulate.

También se solicita que cada TDA utilizado este en un archivo propio.

2.2.- Diseño de solución

Al diseñar la solución se utilizaron diferentes tipos de datos: datos nativos del lenguaje y se crearon seis TDA que serán utilizados para la abstracción y simplificación del problema, estos TDA son:

TDA System:

- **Representación:** Una lista que posee el nombre del sistema como un string, un número con el ID del chatbot inicial, la lista de chatbots, la lista de users agregados al sistema, la lista de chatHistorys de los usuarios, dos

números que representan el currentChabotID y el currentFlowID, finalmente se tiene un string con la fecha de creación del sistema.

- **Constructor:** Se entregan todos los elementos antes mencionados, si solo se está creando un system por primera vez, solo se introduce el nombre, el ID del chatbot inicial y los chatbots que se desen (pueden ser 0 o muchos)

- Funciones: Anexo 1

TDA Option:

- **Representación:** Una lista, donde que posee un número que representa el ID único de la option (code), un string con el mensaje de la opción, dos números con el ID del chatbot al cual apunta dicha opción y el ID del flujo inicial de dicho chabot, finalmente se posee una lista con keywords que representan el distintivo de la option.

- **Constructor:** Se entregan todos los elementos antes mencionados, pero si no se desea agregar keywords, se entregan todos los elementos menos la lista de keywords antes mencionada.

- Funciones: Anexo 2

TDA Flow:

- **Representación:** Una lista, la cual posee un número correspondiente al ID único del flujo, un string correspondiente al nombre del flujo con su respectivo mensaje sobre las opciones que posee, finalmente una lista de options, las cuales representan las diferentes opciones que tiene un usuario para escoger mientras interactúa.

- **Constructor:** Se entregan todos los elementos mencionados anteriormente, si solo se esta creando un flujo sin opciones, se ingresan todos los elementos mencionados anteriormente pero no se ingresan las opciones correspondientes al flujo

- Funciones: Anexo 3.

TDA Chatbot:

- **Representación:** Una lista, la cual posee un número que representa el ID único del chatbot, dos strings que representan el nombre del chatbot y el mensaje de bienvenida del chatbot, un número que representa al flujo inicial del chabot, finalmente se tienen todos los flujos que posee el chatbot en cuestión.

- **Constructor:** Se entregan todos los elementos mencionados anteriormente, si se crea un chatbot sin flujos, se ingresan todos los elementos mencionados anteriormente excepto los flujos del chatbot.

- Funciones: Anexo 4.

TDA User:

- **Representación:** Una lista, donde el primer elemento es un string que representa el nombre del usuario, el segundo elemento es un boolean que representa el estatus del usuario en el sistema, si se encuentra iniciado o si su sesión está cerrada.

- **Constructor:** Se entrega el nombre del usuario que se desea agregar al sistema.

- Funciones: Anexo 5.

TDA ChatHistory:

- **Representación:** Una lista, donde el primer elemento es un string que representa el nombre del usuario que interactúa con el sistema de chatbots, el segundo elemento es un string que representa el diálogo con fechas que posee el usuario con el sistema de chatbots.

- **Constructor:** Se entrega el nombre del usuario que interactuará con el sistema de chatbots.

- Funciones: Anexo 6.

Con la creación de las estructuras mencionadas anteriormente, se crea el sistema de chatbots. Cada estructura cumple un rol dentro del sistema, interactuando unas con otras para poder complementarse y generar el sistema de chatbots, para ver la relación entre las diferentes estructuras ver anexo 7.

Para agregar opciones a los flujos, flujos a los chatbots y chatbots al sistema, se utilizaron técnicas como: la recursión y la implementación declarativa, esto debido a que en ciertas funcionalidades se necesitó ver el ¿cómo? operaba la función por debajo y en otras no era necesario enfacarse en el cómo, si no, más bien en el ¿qué?.

2.3.- Aspectos de implementación

Para este laboratorio el compilador utilizado es Dr Racket, en versiones 6.11 o superiores. Se pueden utilizar todas tipo de funciones de Scheme y Racket.

Se trabaja en su mayoría con listas, pero se crean diferentes funciones o se utilizan distintos símbolos para poder acceder a estas. El programa creado es indiferente frente a mayúsculas y minúsculas, para el programa representan lo mismo, ósea el string “Museo” es igual a “MUSEO”.

El programa funciona sin validación de entradas de usuario, es decir, se asume que el usuario ingresará entradas válidas para el funcionamiento del programa.

Para llevar un mayor orden se distribuyen todos TDAs en archivos separados donde cada uno posee sus funciones correspondientes, para que los distintos archivos puedan tener acceso entre sí, se utiliza require / provide para importar y exportar funciones, respectivamente. También se tiene el archivo main que contiene todos los requerimientos funcionales implementados y el scriptDePruebas que es un archivo que posee los ejemplos de utilización de los requerimientos funcionales.

Los archivos mencionados anteriormente son: “main.rkt”, “TDASystem.rkt”, “TDAOption.rkt”, “TDAFlow.rkt”, “TDAChatbot.rkt”, “TDAUser.rkt”, “TDAChathistory.rkt” y el “scriptDePruebas.rkt” (con las identificaciones pedidas).

2.4.- Ejemplos de uso

Se debe verificar que se tengan todos los archivos mencionados anteriormente, el archivo main necesita a todos los archivos TDAs implementados.

El programa creado cuenta con un formato especifico para poder se usado de manera correcta, sin que este presente errores, para ello, los usuarios deben usar de manera correcta los requerimientos implementados. El programa implementado no distingue entre mayúsculas y minúsculas, el texto es el mismo, además, lo usuarios deben ingresar entradas correctas debido a que no se realizan validaciones de entradas de usuario.
Para ver un ejemplo de uso del programa, ver anexo 8.

2.5.- Resultados y autoevaluación

Los resultados obtenidos son los esperados para la mayoría de las funciones implementadas, logrando hacer un programa completamente funcional. En el cual se colocan diferentes entradas posibles, o utilizando funciones cuando no corresponde y el programa responde de manera correcta exceptuando las funciones que no fueron implementadas.

Se logran completar 14 requerimientos funcionales propuestos para el proyecto.

La autoevaluación radica entre valores de 0 (no implementado) hasta 1 (implementado y sin errores) aumentando en una escala de 0.25 y se puede observar en el anexo 9.

Se les asigna 1 a las funciones que trabajan de manera correcta, y se les asigna 0.5 a las funciones que trabajan de manera no completa (cumplen su función, sin embargo, no realizan todos los cambios que deberían o no cumplen todos los requisitos pedidos)

3.- Conclusión

Una vez terminado el proyecto de laboratorio, se puede concluir que se cumple con los objetivos principales de este, debido a que se logra entender de mejor manera el paradigma funcional, así como también se aprende a utilizar Scheme y Racket para dar solución a la problemática planteada como enunciado del proyecto.

Durante la realización del laboratorio una de las complicaciones más grandes fue que al estar trabajando con un nuevo paradigma, había que pensar de manera diferente, se pensaban distintas ideas para la realización de ciertas funciones, sin embargo, se complicaba debido a la forma de implementar dichas ideas, que en muchas ocasiones tuvieron que ser cambias en más de una ocasión, lo que ocasiono otro problema, el tiempo, ya que es un proyecto no menor que requiere de una compresión correcta de este.

4.- Bibliografía

1.- Gonzales, R. (2023). “Proyecto semestral de laboratorio”. Recuperado de: https://docs.google.com/document/d/1L-B2b3J71Baqa_IuZt6EmRwDlxC0qzWn9YJAm6FIiJk/edit?usp=sharing

2.- Flatt, M. y Bruce, R. (2023). “The Racket Guide”. Recuperado de: https://docs.racket-lang.org/guide/

3.- Gonzales, R. (2023). “3 – P. Funcional”. Paradigmas de Programación. Material de clases Online. Recuperado de: https://uvirtual.usach.cl/moodle/course/view.php?id=10036&section=11

5.- Anexos

1.- Funciones implementadas del TDA System (Los requerimientos funcionales se dejan dentro del archivo “main.rkt” como es pedido):

Tipo de función	**Nombre**	**Descripción**
Constructor	system	Crea un system por primera vez
Constructor	new-system	Crea un system
Pertenencia	logged-users?	Verifica si hay un usuario iniciado
Selector	get-system-name	Obtiene el nombre del sistema
Selector	get-system-initial-chatbot-code-link	Obtiene el ID del chatbot inicial
Selector	get-system-chatbots	Obtiene la lista de chatbots
Selector	get-system-users	Obtiene la lista de usuarios
Selector	get-system-chat-history	Obtiene la lista de chatHistorys
Selector	get-system-current-chatbotID	Obtiene el ID del chatbot actual
Selector	get-system-current-flowID	Obtiene el ID del flow actual
Selector	get-system-date	Obtiene la fecha de creación del sistema
Selector	get-initial-flow-id-by-initialChatbotCodeLink	Obtiene el flow ID inicial de un chatbot dado su ID
Selector	get-system-user-logged	Obtiene al usuario iniciado en el sistema
Modificador	add-unique-chatbots	Guarda solo los chatbots con un ID único
Modificador	add-chatbot-to-chatbots	Agrega un chatbot a una lista de chatbots
Modificador	add-chatHistory-to-system	Agrega un chatHistory a una lista de chatHistorys
Modificador	add-user-to-users	Agrega un usuario a una lista de usuarios
Modificador	logout-user	Cierra la sesión del usuario iniciado
Modificador	set-system-startFlowID	Actualiza el current flow ID del sistema ID del flow inicial del chatbot inicial
Modificador	change-system-current-chatbot-id	Cambia el current chatbot ID dada la opción elegida por un usuario
Modificador	change-system-current-flow-id	Cambia el current flow ID dada la opción elegida por un usuario
Otras	add-system-current-flow-id	Obtiene el current Flow ID en caso de tener un chatbot inicial
Otras	new-interaction-rec	Agrega una interaccion de un usuario con un chatbot al chatHistory de un usuario

Otras	new-interaction-norec	Agrega una interaccion de un usuario con un chatbot al chatHistory de un usuario
Otras	change-system-current-chatbot-id-norec	Cambia el current chatbotID dado la option-code o keyword (user-message) ingresado, obtiene el chatbotCodeLink de la option
Otras	change-system-current-flow-id-norec	Cambia el current flowID dado la option-code o keyword (user-message) ingresado, obtiene el InitialFlowCodeLink de la option

2.- Funciones del TDA Option:

Tipo de función	**Nombre**	**Descripción**
Constructor	option	Crea una option
Pertenencia	equal-option-code?	Verifica si una option tiene el mismo code que otra option
Pertenencia	option-exist?	Verifica si una option existe dentro de una lista de option
Pertenencia	message-is-valid?	Verifica si una mensaje es valido, si es una keyword o si es un ID existente
Pertenencia	option-exist-rec?	Verifica si existe una option con un code correspondiente
Pertenencia	keyword?	Verifica si una palabra es una keyword dentro de una option
Pertenencia	keyword-exist?	Verifica si una palabra se encuentra dentro de una lista de keywords
Selector	get-option-code	Obtiene el code de la option
Selector	get-option-message	Obtiene el mensaje de la option
Selector	get-option-ChatbotCodeLink	Obtiene el ID del chatbot al cual posee un link la option
Selector	get-option-InitialFlowCodeLink	Obtiene el ID del flow inicial al cual apunta la option
Selector	get-option-keywords	Obtiene las keywords de la option
Selector	get-option-by-message	Obtiene una option dentro de una lista de options dado un message
Selector	get-option-by-message-norec	Obtiene una option dentro de una lista de options dado un message
Otras	transformar-a-minusculas	Dada una lista de keywords las transforma a minusculas
Otras	options-to-string-rec	Crea un string con todas las options que tenga

3.- Funciones del TDA Flow:

Tipo de función	**Nombre**	**Descripción**
Constructor	flow	Crea un flow por primera vez
Constructor	new-flow	Crea un flow
Selectore	get-flow-id	Obtiene el ID del flow
Selector	get-flow-name-msg	Obtiene el nombre mensaje del flow
Selector	get-flow-options	Obtiene las options que posee el flow
Selector	get-flow-by-id-rec	Obtiene un flow dado su ID
Selector	get-flow-by-id	Obtiene un flow dado su ID
Pertenencia	flow-exist?	Verifica si un flow existe

Pertenencia	equal-flow-id?	Verifica si dos flows tienen el mismo ID
Modificadores	add-option-to-options	Agrega una option a una lista de options
Modificadores	add-unique-options	Obtiene las options la primera ocurrencia de cada option, discriminando por su code o ID

4.- Funciones del TDA Chatbot:

Tipo de función	**Nombre**	**Descripción**
Constructor	chatbot	Crea un chatbot por primera vez
Constructor	new-chatbot	Crea un chatbot
Pertenencias	equal-chatbot-id?	Verica si un chatbot es igual a otro comparando sus ID
Pertenencias	chatbot-exist?	Verifica si un chatbot nuevo existe dentro de una lista de chatbots
Selector	get-chatbot-id	Obtiene el ID del chatbot
Selector	get-chatbot-name	Obtiene el nombre del chatbot
Selector	get-chatbot-welcomeMsg	Obtiene el mensaje de bienvenida del chatbot
Selector	get-chatbot-startFlowID	Obtiene el ID del flow inicial del chatbot
Selector	get-chatbot-flows	Obtiene los flows que posea el chatbot
Selector	get-chatbot-by-id	Obtiene un chatbot dado su ID desde una lista de chatbots
Selector	get-chatbot-by-id-rec	Obtiene un chatbot dado su ID desde una lista de chatbots
Selector	get-chatbot-by-message	Obtiene un chatbot dado un message
Modificador	add-flow-to-flows	Agrega un flow a una lista de flows
Modificador	add-unique-flows	Agrega los flujos que sean unicos, no se repiten, en base a su ID (agrega una ocurrencia de cada flow)

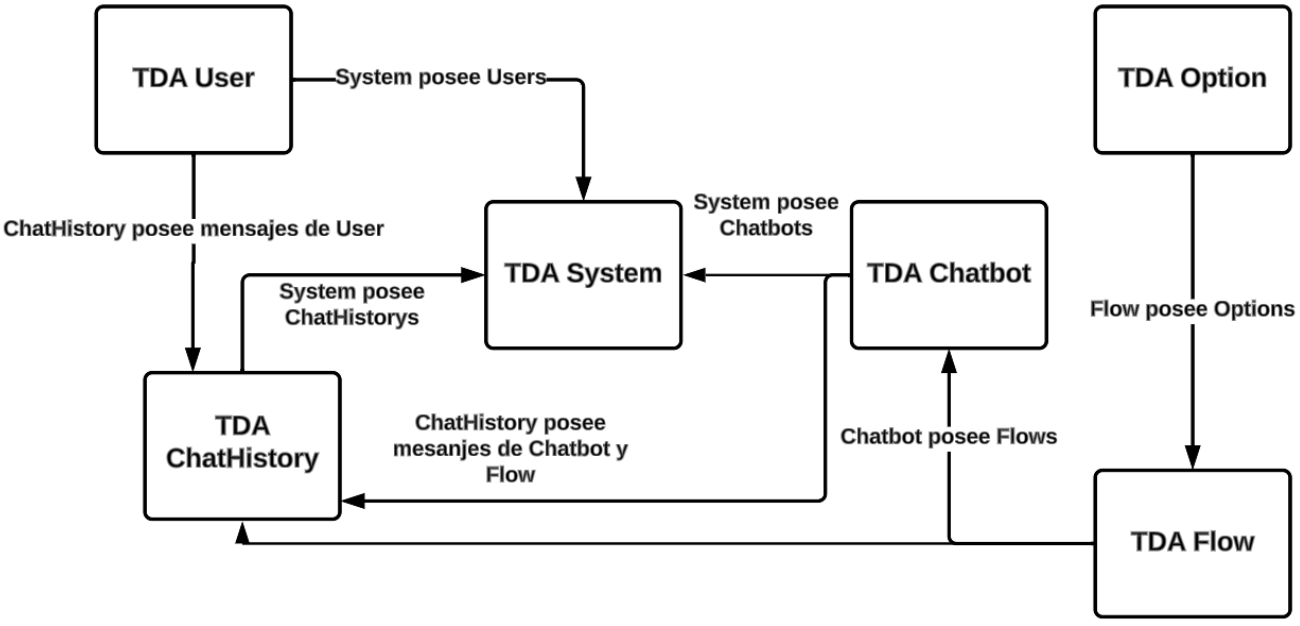
5.- Funciones del TDA User:

Tipo de función	**Nombre**	**Descripción**
Constructor	new-user	Crea un usuario
Pertenencia	user-exist?	Verifica si un usuario existe en una lista en base a su ID (su nombre)
Pertenencia	logged-user?	Verifica si existe un usuario iniciado
Selector	get-user-name	Obtiene el nombre del usuario
Selector	get-user-status	Obtiene el estado del usuario
Selector	get-logged-user	Obtiene al usuario iniciado
Modificador	log-in	Inicia la sesión de un usuario
Modificador	login-user	Inicia sesión de un usuario existente
Modificador	logout	Cierra la sesión de un usuario

6.- Funciones del TDA ChatHistory:

Tipo de función	Nombre	Descripción
Constructor	chatHistory	Crea un chatHistory por primera vez
Constructor	new-chatHistory	Crea un chatHistory
Selector	get-chatHistory-user	Obtiene el nombre del usuario del chatHistory
Selector	get-chatHistory	Obtiene el string que representa la interacción del chatHistory
Modificador	make-chat-message	Crea la nueva interaccion entre el user y un chatbot
Modificador	make-user-message	Crea el mensaje de entrada del usuario
Modificador	make-system-message	Crea el mensaje respuesta del chatbot
Modificador	update-chatHistory	Actuliza el chatHistory de un usuario, agregando una interacion nueva
Modificador	repeat-message-chatHistory	Actuliza el chatHistory de un usuario, agregando una interacion nueva, el chatbot repite la respuesta anterior debido a una entrada no valida
Modificador	update-chatHistory-user	Agrega una interacion realizada por el usuario y un chatbot
Modificador	update-chatHistory-norec	Actualiza el chatHistory del usuario que realizo la interacion
Modificador	repeat-message	Agrega una interacion realizada por el usuario y un chatbot
Modificador	repeat-chatHistory-norec	Actualiza el chatHistory del usuario que realizo la interacion, el chatbot repite el mensaje
Otras	make-date	Obtiene la hora actual
Otras	transform-message	Transforma un mensaje en formato string a numero

7.- Relación entre estructuras:



8.- Ejemplos de uso:

- (define op1 (option 1 "1) Viajar" 1 1 "viajar" "turistear" "conocer"))
- (define op2 (option 2 "2) Estudiar" 2 1 "estudiar" "aprender" "perfeccionarme"))
- (define f10 (flow 1 "Flujo Principal Chatbot 1\nBienvenido\n¿Qué te gustaría hacer?" op1 op2 op2 op2 op2 op1))
- (define f11 (flow-add-option f10 op1)) ;se intenta añadir opción duplicada
- (define cb0 (chatbot 0 "Inicial" "Bienvenido\n¿Qué te gustaría hacer?" 1 f10 f10 f10 f10))
- (define s0 (system "Chatbots Paradigmas" 0 cb0 cb0 cb0))
- (define s3 (system-add-user s2 "user2"))
- (define s11 (system-talk-rec s10 "hola"))
- (define s12 (system-talk-rec s11 "1"))
- (display (system-synthesis s12 "user2"))

Existen más ejemplos de uso en el archivo “Pruebas.rkt” donde se puede apreciar como hacer un correcto uso del programa.

9.- Autoevaluación requerimientos funcionales y no funcionales:

Autoevaluación	Puntaje
TDA	1
option	1
flow	1
flow-add-option	1
chatbot	1
chatbot-add-flow	1
system	1
system-add-chatbot	1
system-add-user	1
system-login	1
system-logout	1
system-talk-rec	1
system-talk-norec	1
system-synthesis	1
system-simulate	0

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Autoevaluación	Puntaje
Autoevaluación	1
Lenguaje	1
Versión	1
Standar	1
No variables	1
Documentación	1
Dom->Rec	1
Organización	1
Historial	1
Script de pruebas	1
Prerrequisitos	1