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

Informe Laboratorio 1
Paradigma Funcional
Sistema de creación, despliegue y administración de chatbots

Stephan Paul Ramirez

Fecha:

8 de octubre 2023

UdeSantiago de Chile

Índice de Contenidos

UdeSantiago de Chile

1. INTRODUCCIÓN

En el presente informe se documenta el proceso de realización del primer laboratorio de la asignatura paradigmas de programación. El problema consiste en implementar un ambiente contenedor de chatbots que permita a un usuario crear e interactuar con distintos chatbots, en donde cada uno contendrá a su vez distintas preguntas de distintos temas y distintas opciones asociadas a otros chatbots, así de esta forma se pueden enlazar las opciones con otras preguntas y así simular una conversación con un usuario. Las operaciones que se abarcan con respecto a crear un chatbot son; identificar un chatbot, añadir preguntas al mismo, añadir opciones a las preguntas y especificar enlaces para el flujo de interacción dentro del chatbot hacia otros chatbots. Luego con respecto a las interacciones del chatbot se tiene que mediante una serie de opciones enumeradas el usuario pueda escoger el número de alguna de estas para seguir la conversación o responder con textos considerando match exacto con palabras clave. También se considera como operación la posibilidad de mostrar una síntesis de la conversación, esto se puede lograr guardando en un historial cada interacción con algún chatbot de cada usuario por separado, para que cuando llegue el momento se muestre.

El paradigma funcional es el paradigma con el cual se aborda el proyecto, este consiste en basar la programación en el uso de funciones, es decir de una manera declarativa, en otras palabras, es centrarse en el que y no en el cómo, por otra parte, este paradigma usa los fundamentos del cálculo lambda, el cual consiste en una notación para definir o expresar funciones de manera prefija. (Fernández, 2022)

En el contexto del proyecto este paradigma se emplea usando el lenguaje Racket, el cual es una especie de ”Scheme moderno”, que está fuertemente basado en el paradigma funcional, su origen está asociado al lenguaje Lisp que como su nombre insinúa es un lenguaje en donde comúnmente se manipulan listas, luego como el problema consiste en almacenar cosas y enlistarlas, se aplicaran los conceptos de listas para abordar el problema, siendo estos enlistar elementos, seleccionar elementos enlistados, agregar elementos, etc.

UdeSantiago de Chile

2. ANÁLISIS DEL PROBLEMA

Identificando los principales componentes de lo que debería ser la implementación de la solución se puede determinar que usando como estructura base las listas, se deberá implementar un sistema “global” que sea una lista de listas que contenga todos los chatbots del sistema, en este contexto se deberán usar funciones que construyan estas estructuras enlistándolas, y teniendo en cuenta que el problema especifica que cada estructura deberá tener un id único para su conjunto, eso quiere decir que dentro de un flujo por ejemplo, todas las opciones deberán tener un id único, esto implica reservar solo la primera instancia que se tenga con un id y las demás descartarlas, pero eso es el problema de manera general, debido a lo compleja que queda la estructura debido a tantos componentes, se debe empezar por algún subproblema, en este caso lo más básico, una opción.

Por otro lado, como se tienen distintas estructuras con distintas operaciones, se deberá implementar una manera de dejar programado cada una de estas por separado para cada una de ellas, en este sentido se deben crear distintos tipos de datos abstractos en donde cada uno tendrá su representación, constructor, selectores, modificadores y otras funciones asociadas.

También como se debe interactuar con un usuario, se debe implementar una manera de que un usuario se registre en el sistema, inicie sesión, cierre sesión y que contenga su historial en el mismo.

UdeSantiago de Chile

3. DISEÑO DE LA SOLUCIÓN

Como el primer requerimiento especifica la creación de los TDA para cada estructura, simplemente se siguió la estructura vista en clases para especificar cada TDA, esto implica indicar documentando en archivos separados para cada tipo de dato en forma de comentario la representación del TDA y el nombre de sus operaciones, luego más adelante en la implementación se agregan estas operaciones como los selectores que son sinónimos de funciones de listas. Luego para la función option se optó por simplemente listar cada uno de los argumentos del dominio, después para verificar unicidad de cada opción en un flujo, se declaró otra función que usa recursividad de cola, luego para flow también simplemente se enlistan los parámetros pero procesando los option en dicha función, después para agregar una opción a un flujo, se opta por usar member del id a añadir y map con el selector del id de las opciones agregadas al flujo, para obtener la implementación declarativa. De manera similar se sigue trabajando con chatbot ya que simplemente enlista parámetros y procesa los flujos con otra función que usa recursión de cola y como añadir un flujo requería recursividad también se usa la misma. Para system, de nuevo se enlistan parámetros y se procesan datos con una función recursiva de cola pero ya que la implementación de este puede tener más libertad, se optó por agregar 2 elementos más aparte de los argumentos recibidos, siendo estos la lista con el usuario que inicio sesión y la lista de los historiales de los usuarios, en donde los historiales corresponden al TDA chatHistory que se implementó con una lista de 2 elementos siendo un TDA usuario que es un string realmente y otro que es el registro de su historial que es un string formateado para usar display, y por otro lado el otro elemento agregado es la interacción actual que corresponde al último flujo enlazado por el mensaje del usuario, siendo una lista del id del chatbot que contiene al flujo, el id del flujo y el mensaje que lo enlazo. (ver imagen 1)

Continuando con el aspecto de la implementación, están las interacciones con el usuario, empezando por system-add-user, la cual corresponda a la función que registra usuarios en el sistema, para el diseño de esta simplemente se almaceno un chatHistory con el usuario recibido con un historial vacío en la lista de los historiales. Luego para system-login y system-logout, se optó por tener un apartado que contiene el usuario que inicio sesión, en este contexto si el usuario se encuentra dentro de la lista de historiales, se agrega su nombre a ese apartado, y cuando cierra sesión, se quita. Y por último para la función system-talk-rec, la implementación consiste en buscar el mensaje del usuario usando las “coordenadas” de la interacción actual y este es reemplazado con el chatbotcodelink e initialflowcodelink de la opción con la cual hizo match el mensaje del usuario y si es que es el primer mensaje hace lo mismo, pero con los parámetros iniciales del system y el chatbot inicial.

UdeSantiago de Chile

4. ASPECTOS DE LA IMPLEMENTACIÓN

Con respecto a la estructura del proyecto, primero se abordaron los requerimientos funcionales, empezando por los TDA, se documento en forma de comentario las operaciones de cada uno y luego se empezó a trabajar las funciones de manera ordenada, es decir de 1 en 1 cada día y realizando los commit correspondientes por cada avance en GitHub, sin embargo, se separaron los archivos de los TDA luego de tener al menos los constructores de cada uno. Después de terminar la parte funcional, se trabajo en el script de pruebas y la autoevaluación para finalmente hacer el informe del proyecto.

5. INSTRUCCIONES DE USO

Para el uso adecuado del sistema, se pueden ejecutar archivos similares al que se muestra como script de prueba, o también se pueden ejecutar las mismas instrucciones desde la consola, esto implica definir funciones constantes que sirvan como nombre para cada estado de alguna parte del sistema e ir renombrándolas con cada modificación usando las funciones anteriormente mencionadas. Cabe recalcar que debido a que la única forma de la implementación de reiniciar una conversación es cerrando sesión e iniciar con otra, si es que se inicia sesión con uno antiguo y había una conversación antes, este debe empezar la conversación de 0. (ver imagen 2)

6. RESULTADOS Y AUTOEVALUACIÓN

TDAs	option	flow	flow-add-option	chatbot
1	1	1	1	1
chatbot-add-flow	system	system-add-chatbot	system-add-user	system-login
1	1	1	1	1
system-logout	system-talk-rec	system-talk-norec	system-synthesis	system-simulate
1	0.75	0	0	0

Con respecto al 0.75 de system-talk-rec, esto es debido a lo mencionado anteriormente de hablar con un usuario que ya tenía un historial, este debe empezar la conversación de 0 (esto no implica que el historial se borre, se conserva pero no se puede continuar una conversación de ahí). El nulo trabajo de las ultimas 3 funciones es debido a que se prefirió usar el tiempo que quedaba de la entrega para realizar el informe, y ya que no se realizó system-talk-norec, no se pudo implementar las ultimas 2 debido a sus prerrequisitos.

UdeSantiago de Chile

7. CONCLUSIÓN

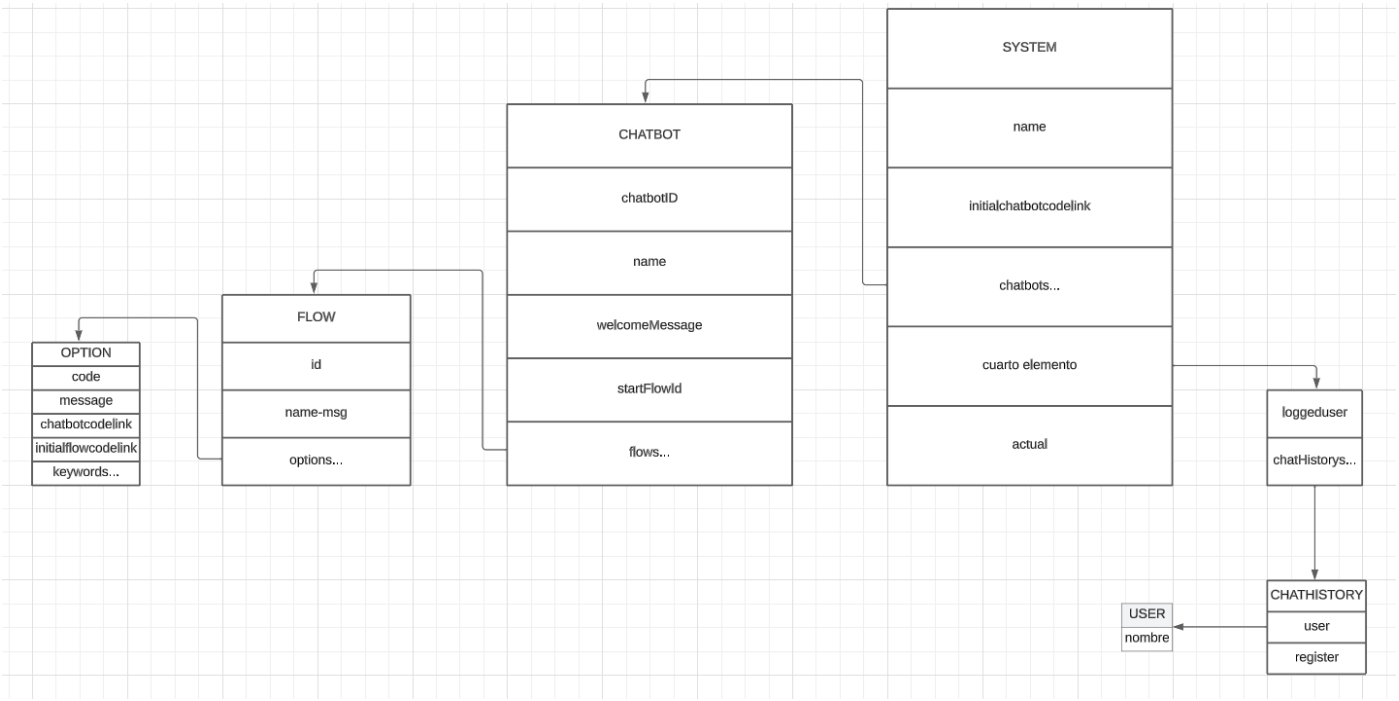
Para finalizar se puede decir que el resultado obtenido cumple con casi todas las funciones fundamentales de lo requerido, exceptuando la síntesis, ya que el sistema permite crear chatbots con la función chatbot, permite añadir preguntas con chatbot-add-flow, permite añadir opciones con flow-add-option, los enlaces quedan en la misma estructura de una opción y se puede interactuar con system-talk-rec usando las palabras clave y el numero de la opción. Algunas de las limitaciones del programa son el hecho de que para interactuar con un chatbot requiere definir funciones constantes que identifiquen al sistema en un estado luego de usar system-talk-rec, lo cual dificulta las interacciones con el programa junto con que no se tiene añadida una función que vaya mostrando por pantalla la respuesta inmediatamente, por otro lado cabe recalcar algo mencionado anteriormente, la cual es otra limitación, y es que el sistema no sigue la conversación en donde un usuario la dejo, es por esto que un usuario debe empezar la conversación de 0 para evitar errores. Un problema utilizando el paradigma funcional es que debido a que no se pueden utilizar variables, la única forma de conservar datos es utilizando funciones recursivas o el uso de let, lo cual a la larga puede afectar en la legibilidad del código al tener que componer una gran cantidad de funciones, y hablando de composición como el paradigma se basa en usar funciones y componerlas, a veces se tienen que declarar muchas funciones antes de crear una principal lo cual des agiliza la programación.

8. BIBLIOGRAFÍA

Fernández, P. (2022). *Qué es la programación funcional y sus características.* OpenWebinars. https://openwebinars.net/blog/que-es-la-programacion-funcional-y-sus-caracteristicas/#:~:text=La%20programaci%C3%B3n%20funcional%20(PF)%20es,modelar%20las%20soluciones%20de%20software.

9. ANEXOS

-Imagen 1



-Imagen 2

```
;Mis ejemplos para cada Requerimiento funcional

;ejemplos option
(define option11 (option 1 "1) Adoptar" 1 0 "adoptar" )) ; se crean options con distintos links
(define option12 (option 2 "2) Estudiar" 2 0 "estudiar"))
(define option1 (option 1 "1) perro" 1 1 "perro" "can" "perrito")) ;se crean con mas de una keyword
(define option2 (option 2 "2) gato" 1 2 "gato" "felino" "gatito" "michi"))
(define option4 (option 1 "1) calculo" 0 0 "calculo")) ;se crean options que mandan al inicio de la conversacion
(define option5 (option 2 "2) algebra" 0 0 "algebra"))
(define option6 (option 3 "3) fisica" 0 0 "fisica"))
(define option7 (option 1 "1) grande" 0 0 "grande"))
(define option8 (option 2 "2) chico" 0 0 "chico"))
(define option9 (option 1 "1) corto" 0 0 "corto" ))
(define option10 (option 2 "2) largo" 0 0 "largo"))

;ejemplos flow
(define flow4 (flow 0 "¿Que te gustaria hacer?")) ;se crea flow sin options
(define flow0 (flow 0 "¿Que te gustaria adoptar?" option1 option2)) ;flow con 2 options
(define flow2 (flow 0 "¿Que ramo quieres estudiar?" option4 option4 option4 option5 option6)) ;se deja la primera instancia si hay id repetido de options en un flow
(define flow3 (flow 1 "¿Que tamaño?" option7))
(define flow5 (flow 2 "¿Que tan largo el pelo?" option9 option10))
;ejemplos flow-add-option
(define flow6 (flow-add-option flow3 option8)) ;se agregan options nuevos
(define flow7 (flow-add-option flow4 option11))
(define flow8 (flow-add-option flow7 option12))
(define flow9 (flow-add-option flow8 option12)) ;no se agrega nada por id repetido
;ejemplos chatbot
(define chatbot0 (chatbot 0 "inicial" "Bienvenido" 0)) ;chatbot sin flows
(define chatbot1 (chatbot 1 "Adoptar" "¿que quieres adoptar?" 0 flow0 flow6)) ;chatbot con 2 flows
(define chatbot2 (chatbot 2 "Estudiar" "¿que quieres estudiar?" 0 flow2 flow2 flow2)) ;deja la primera instancia si hay id repetido del flow en el chatbot
;ejemplos chatbot-add-flow
(define chatbot3 (chatbot-add-flow chatbot0 flow8)) ;se añade un flow
(define chatbot4 (chatbot-add-flow chatbot1 flow5))
(define chatbot5 (chatbot-add-flow chatbot1 flow5)) ;no hace nada porque esta repetido el id del flow
;ejemplos system
(define system1 (system "chatbots ejemplo" 0)) ;system sin chatbots
(define system2 (system "chatbots ejemplo" 0 chatbot3))
(define system3 (system "chatbots ejemplo" 0 chatbot3 chatbot3 chatbot3)) ;se deja la primera instancia si hay chatbots con id repetido en un system
```