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

INFORME PARADIGMAS DE PROGRAMACIÓN
PROGRAMACIÓN LÓGICA

Autor: Nicolás Venegas

Asignatura: Paradigmas de Programación

25 De Enero Del 2022

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

Tabla de Contenidos

1. Introducción

Para este informe se tiene como proyecto la elaboración de un editor de texto colaborativo que siga el estilo de Google Docs, esto mediante el lenguaje Prolog siguiendo el paradigma de programación lógico, guiándose por relaciones y hechos, además será explicado el problema, viendo su definición y conceptos necesarios a la hora de abordarlo, seguido por conceptos necesarios para poder entender este laboratorio, también veremos nuestro análisis y propuesta de solución, explicaremos como esta pasó a ser implementada y veremos algunos ejemplos de uso dejando al final una lista de ejemplos para que cualquier persona pueda ocupar el programa final.

2. Descripción del problema

Para este laboratorio se tiene la falta de un editor de texto colaborativo al estilo de Google Docs por lo que se solicitó la creación del mismo usando una tecnología en específico, que es este caso es Prolog, siguiendo una cantidad mínima de requerimientos(que serán mencionados en análisis del problema) para que la herramienta sea funcional, todo guiándonos en base a los hechos y reglas(que será explicada su definición más adelante)

3. Descripción breve del paradigma y los conceptos del mismo que se ven aplicados en este proyecto

En este desarrollo se ocupó el paradigma lógico el cual está basado en los hechos y relaciones formando entre relaciones entre variables y datos. Lo que nos permite centrarnos en la lógica de predicados dejando los temas de lenguaje al compilador encargado(en este caso Swi-Prolog)

En la construcción del programa se van a diferenciar 3 términos regularmente:

- Recursión: es el proceso donde una función se llama a sí misma con un objetivo, esta puede ser de cola, natural u arbórea dependiendo de cómo se implemente.

- Hecho: sentencias que verifican la relación entre 2 o más términos

- Regla: es una forma de representar conocimiento y define las relaciones entre 2 o más términos

Análisis del problema respecto de los requisitos específicos que deben cubrir

Como se mencionó anteriormente se nos pide programar un editor de texto colaborativo del estilo Google Docs pero este tiene una lista de funcionalidades mínimas para que se considere como exitoso, siendo estas:

- Creación de la plataforma
- Registro de usuarios
- Logeo de usuarios
- Creación de documentos
- Compartir documentos entre usuario
- Agregar contenido a un documentos
- Restaurar Versiones de un documento
- Mostrar información del usuario junto a la plataforma creada

Como se puede ver en esta lista de requerimientos ya se puede ir deslumbrando lo que se necesitará para cada estructura, por ejemplo ya sabemos que una plataforma requerirá una lista con usuarios y que estos necesitan un nombre y contraseña para poder loguearse.

También es posible visualizar la necesidad de las siguientes estructuras representativas que contengan distintas características::

Estructura/TDA	Componentes
Usuarios	Nombre Contraseña
Versión	Id Fecha Contenido
Acceso	Usuario Tipo Acceso
Documento	Nombre Autor Fecha de creación Lista de permisos Lista de usuarios compartidos Contenido
Plataforma	Nombre plataforma Fecha de creación Lista de usuarios registrados Lista de documentos usuario Activo

Por último cada estructura debe tener sus propias funcionalidades internas, que al actuar en conjunto a la de las demás estructuras es posible realizar las tareas pedidas anteriormente.

4. Diseño de la solución

En esta sección de diseño de solución se especificará qué funciones fueron necesarias para los TDA/estructuras con el fin de crear las funcionalidades mínimas.

Antes de todo fue la preparación y programación de los TDA de las estructuras dichas en el análisis de solución por lo que a continuación se mencionarán todos los TDA y lo que representan a nivel de proyecto

A partir de eso se creó un constructor para los TDA, como estamos en programación lógica cada constructor puede servir de selector y editor a la vez por lo que no fue necesario en la mayoría de TDAs, pero sí hubo una excepción en los TDA de paradigmadocs y Documentos más que nada por comodidad a la hora de usarlos.

- ➔ TDA Paradigmadocs: Representa a la plataforma y editor de texto del proyecto, este contiene un nombre, fecha de creación, lista de usuarios, lista de documentos y un usuario activo, la idea es que todas los requerimientos sean llamados mediante este TDA pero realizado en conjunto de los demás.

- ➔ TDA Usuario: Como dice el nombre representa a cada usuario de la plataforma(paradigmadocs) y posee un nombre, contraseña y una fecha de registro.

- ➔ TDA documento: Representa a un documento, donde el usuario podrá guardar información y compartirla con los demás usuarios, este TDA posee un id único, un nombre, el nombre del usuario autor, fecha de creación, contenido, una lista de permisos, lista de usuarios compartidos y una lista de versiones del mismo documento.

- ➔ TDA Versión: Representa una versión previa del contenido de un documento, por lo que posee el contenido del documento en ese instante, la fecha de guardado y un id único.

- **Registro de usuarios**
  - Para este predicado solo fue necesario un predicado de apoyo que verifique si un usuario ya está en la lista de usuarios, posteriormente se agrega a la lista si este no es el caso y se retorna(no es retorno como tal pero es una buena forma de

enterderlo) el nuevo paradigma en la última variable del predicado.

- **Logeo de usuarios**
    - Para este predicado se creó un predicado de apoyo para ver si el usuario ingresado(con la contraseña) coinciden simultáneamente, luego si el usuario puede loguearse se agrega el mismo a la sección de usuario activo o en caso contrario todo tira false.

- **Creación de documentos**

    - Para esta sección además de la creación del TDA documento fue necesario un predicado de apoyo que ingresara un documento en un paradigmadocs, finalmente se comprueba si existe algún usuario logeado, y de ser ese el caso simplemente extrae los datos, crea el documento y lo inserta en el paradigmadocs ingresado.

- **Compartir documentos entre usuarios**
    - Para este predicado se crearon 3 más de apoyo, uno que inserte una lista de usuarios compartidos a un documento, otro que inserte una lista de accesos a un documento y finalmente otro predicado que permita reemplazar un documento por otro en una lista de los mismos, este último predicado se usará mucho ya que “devuelve” el paradigmadocs entero, después simplemente se comprueba que exista un usuario logueado y de ser ese el caso se extraen los datos, se insertar los usuarios y permisos en el documento y se reemplaza el documento antiguo con el nuevo.

- **Agregar contenido a un documentos**
    - Para agregar contenidos un documento se creó un predicado de apoyo que hiciera exactamente eso, luego se comprueba que exista un usuario logueado, luego se extraen los datos, se modifica el documento agregando el contenido y finalmente se reemplaza con el original en la lista del paradigmadocs.

- **Restaurar Versiones de un documento**
    - Esta fue muy simple ya que se reutilizaron la mayoría de funciones de la funcionalidad anterior, cambiando que en vez de agregar contenido al final del documento este se setea con el contenido guardado en la versión buscada por id (funcion creada en TDA Documento) y el contenido previo fue guardado en una nueva versión.

5. Aspectos de implementación

Para la implementación de este proyecto se usó el intérprete SWI-Prolog en su versión 8.4.1 para 64 bits.

De igual manera no fue necesaria ninguna biblioteca adicional además de la que proporciona el mismo Prolog, ya que todas los otros predicados y estructuras necesarias fueron escritas en distintos segmentos del codigo dependiendo de la necesidad.

6. Instrucciones de uso

A continuación se detalla cómo hacer uso de los predicados implementados, indicando sus variables de entrada y salida, lo que debería suceder en caso de una ejecución correcta e incorrecta.
Antes de indicar los predicados tomar en cuenta que se tomarán todos los parámetros como entradas exceptuando el último que será llamado de salida
EJ: paradigmaDocs(nombre,fecha,Sout) → entrada: nombre, fecha
salida: Sout

- paradigmaDocs: Predicado que en las dos variables de entrada indican un nombre para la plataforma y la fecha de creación respectivamente, la variable de salida es la plataforma con algunos parámetros adicionales inicializados vacíos, en caso de ingresar mal los datos en programa arroja un false.

- paradigmaDocsRegister: Predicado que se le entrega un documento a editar, la fecha de registro del usuario, el nombre y contraseña del usuario y en la variable de salida entrega la plataforma con el usuario agregado si este no existía previamente, en el caso que existiera toda la consulta entrega false.

- paradigmaDocsLogin: Predicado que logea sesión en la plataforma, se le entrega la plataforma, el nombre y contraseña del usuario y devuelve la plataforma con el usuario logeado, la consulta entrega false en caso que no coincidan el nombre y contraseña o que ya hubiera otro usuario logueado.

- paradigmaDocsCreate: Predicado que crea un documento, se le ingresa la plataforma, la fecha de creación del documento, el nombre y el contenido del mismo, si los datos son correctos la variable de salida es la plataforma con el documento agregado

- paradigmaDocsShare: Si se le entrega una plataforma, id de un documento, lista con permisos, lista de usuarios, la variable de salida es la plataforma con los usuarios agregados al documento con id ingresado teniendo los permisos que estaban en la lista, en caso de que los permisos no sean “R”, “W”, “C” la consulta entregará false.

- paradigmaDocsAdd: Si se le entrega una plataforma un id de un documento, la fecha actual, un contenido, la variable de salida es la plataforma donde al texto con el id entregado se le agrega el contenido indicado, en caso de tener datos erróneos la consulta entregará false.

- paradigmaDocsRestoreVersion: Si se le entrega una plataforma, un id de documento y un id de versión la variable de salida es la plataforma donde el documento con el id entregado fue restaurado a la versión con id entregada, en caso de entregar datos falsos la consulta retorna false.

En el único anexo existente se puede encontrar algunos ejemplos del uso del programa.

7. Resultados y autoevaluación

Durante la creación del programa y desarrollo del laboratorio surgieron bastantes errores respecto a las funciones mínimas requeridas pero todos estos fueron solucionados dando de esta manera una versión estable y correcta del proyecto, por lo que la autoevaluación me concedo un aprobado ya que todos los requerimientos fueron cumplidos exceptuando por la muestra de información.

En cuanto a los resultados esperados de la funciones se puede decir que estas funcionan buena manera exceptuando los casos donde se introduzcan datos erróneos o inexistentes, arrojando un false(también esperado), esto ocurre debido a que este lenguaje y el laboratorio en general está orientado a usuarios que ya saben usar el lenguaje y tienen un plan de lo que quieren hacer, por lo que a pesar de que es posible ocuparse de esos errores no nos corresponde hacerlo (en otras palabras no es nuestra responsabilidad).
En la sección de anexos es posible encontrar los ejemplos de todas las funciones siguiendo un camino entendible.

8. Conclusiones

Después de observar todo el proyecto se puede concluir que se cumplieron los objetivos iniciales, pero teniendo una serie de dificultades, como por ejemplo el lenguaje y el paradigma lógico, ya que este difiere mucho de lo que he probado anteriormente, ya que este se centra en la lógica de los predicados sin darle mucho énfasis, siendo algo completamente nuevo y teniendo en cuenta el tiempo igual fue complicado entenderlo pero una vez que se entendió el laboratorio fue realizado mucho más rápido que el anterior de Scheme esto gracias a que se le pudo sacar mayor ventaja al lenguaje.

Luego de realizar este proyecto es posible ver la programación con un nuevo punto de vista, a pesar de que hubo complicaciones al aplicar este punto de vista también es posible decir que ahora la versatilidad y proponer nuevas ideas o formas de abordar un problema han aumentado.

Si se compara con el proyecto de programación funcional debo decir que a mi parecer el de Scheme fue más sencillo pero más que nada por que el salto de la programación “común” es menor, pero que de igual manera me gustó este laboratorio por la vuelta de tuerca que me hizo dar.

9. Referencias

Merino, M. (2020, 9 agosto). El lenguaje Prolog: un ejemplo del paradigma de programación lógica. Genbeta. Recuperado 25 de enero de 2022, de https://www.genbeta.com/desarrollo/lenguaje-prolog-ejemplo-paradigma-programacion-logica

10. Anexos

- Ejemplos

// se crea paradigmadocs y se registra un usuario, si se toma el ultimo register tira false

paradigmaDocs("Word", [27, 12, 2021], Word),

paradigmaDocsRegister(Word, [03,05,2020], "nico", "1234", Word1),

paradigmaDocsRegister(Word, [03,05,2020], "nico", "1234234234", Word1).

// el usuario se logea en Word2

paradigmaDocs("Word", [27, 12, 2021], Word),

paradigmaDocsRegister(Word, [03,05,2020], "nico", "1234", Word1),

paradigmaDocsLogin(Word1, "nico", "1234", Word2).

// se crea un documento en Word2 y se guarda en Word3

paradigmaDocs("Word", [27, 12, 2021], Word),

paradigmaDocsRegister(Word, [03,05,2020], "nico", "1234", Word1),

paradigmaDocsLogin(Word1, "nico", "1234", Word2),

paradigmaDocsCreate(Word2, [4,4,2021], "Primer Documento", "Contenido 1", Word3).

// se comparte el documento de word3 con usuarios diversos y se les dan distintos permisos, todo se guarda en word5

paradigmaDocs("Word", [27, 12, 2021], Word),

paradigmaDocsRegister(Word, [03,05,2020], "nico", "1234", Word1),

paradigmaDocsLogin(Word1, "nico", "1234", Word2),

paradigmaDocsCreate(Word2, [4,4,2021], "Primer Documento", "Contenido 1", Word3),

paradigmaDocsLogin(Word3, "nico", "1234", Word4),

paradigmaDocsShare(Word4, 0, ["R","C","W"], ["u1","u2"], Word5).

// se le agrega contenido el documento con id 0 en el Word5 y se guarda en Word7

paradigmaDocs("Word", [27, 12, 2021], Word),

paradigmaDocsRegister(Word, [03,05,2020], "nico", "1234", Word1),

paradigmaDocsLogin(Word1, "nico", "1234", Word2),

paradigmaDocsCreate(Word2, [4,4,2021], "Primer Documento", "Contenido 1", Word3),

paradigmaDocsLogin(Word3, "nico", "1234", Word4),

paradigmaDocsShare(Word4, 0, ["R","C","W"], ["u1","u2"], Word5),

paradigmaDocsLogin(Word5, "nico", "1234", Word6),

paradigmaDocsAdd(Word6, 0, [1,1,1]," Extension 1", Word7).

// se le restaura la version 0 del documento con id 0 en Word7 y se guarda en Word9

paradigmaDocs("Word", [27, 12, 2021], Word),

paradigmaDocsRegister(Word, [03,05,2020], "nico", "1234", Word1),

paradigmaDocsLogin(Word1, "nico", "1234", Word2),

paradigmaDocsCreate(Word2, [4,4,2021], "Primer Documento", "Contenido 1", Word3),

paradigmaDocsLogin(Word3, "nico", "1234", Word4),

paradigmaDocsShare(Word4, 0, ["R","C","W"], ["u1","u2"], Word5),

paradigmaDocsLogin(Word5, "nico", "1234", Word6),

paradigmaDocsAdd(Word6, 0, [1,1,1]," Extension 1", Word7),

paradigmaDocsLogin(Word7, "nico", "1234", Word8),

paradigmaDocsRestoreVersion(Word8, 0, 0, Word9).