Informe Laboratorio 2: Paradigmas de programación

Rodrigo Mardones

29-06-2020

Indice

Introducción

Los paradigmas de programación son marcos de trabajo para, desde cierta perspectiva abordada según el problema a solucionar, trabajar con herramientas ya establecidas según las normas que lo rigen en el cotexto dado.

Es preciso señalar que indiferente de los lenguajes nuevos, modernos y sus herramnientas propias asociadas, siempre se regirán por al menos uno o más paradigmas de programación. Pues son estos quienes nos permiten analizar de buena manera la mejor respuesta a un problema y su contexto especifico.

Objectivo General

Para este el caso de este laboratorio, se plantea la construcción de una pieza de software pequeña que permita emular un controlador de versiones tipo GIT, dentro de los parametros del paradigma de programacion lógica y sus visitudes.

Objectivos Específicos

Para abordar el problema se plantean lo siguientes objectivos especificos

- Análisis del problema
- Aspectos de implementación
- construccion de implementacion
- Conclusiones

Este informe solo recopila el estudio y construccion del proyecto de laboratorio descrito, por lo que no se abordarán otros paradigmas que pueden mencionarse como referencia para la construcción del mismo en las páginas contiguas.

Descripción Del Problema

La necesidad de mantener y producir software escalable, ordenado y mantenible durante el tiempo es una cuestión importante dentro de la industria. Poder llevar todos los cambios asociados a un proyecto, trabajado por un grupo importante de personas y todo lo que conlleva el manejo de errores es parte de lo que resuelven los software denominados “controladores de versiones”.

Para el problema en cuestión se requiere de una pieza de software que permita realizar las siguientes acciones dentro de un directorio de trabajo, independiente del trabajo realizado dentro de este:

Los problemas más comunes que se desprenden de esta problematica son los siguientes:

- Cambios no supervisados
- errores en cambios no supervisados
- manejo de errores en versiones funcionales del proyecto
- manejo de nuevas funcionalidades en versiones futuras del proyecto
- registro de cambios realizados, quién los hizo , cuándo los hizo y qué hizo
- estaciones de trabajo independientes para no cruzar cambios de distintos colaboradores
- versionado de proyecto para su despligue

Es por lo anterior mencionado que nacen los “controladores de versiones”, software encargado de solucionar todos los estados del problema anterior mencionado, ayudando al al monitoreo, control y desarrollo del proyecto en cuestión más eficiente.

Descripción Del Paradigma

El Paradigma de programación lógico es una paradigma de programacion que, basa su estructura de desarrollo en "hechos" o verdades de caracter booleano. Es decir, toda declaracion dentro de un programa inmerso en este paradigma es de caracter declarativo y logico, y atribuido a un valor booleano como tal. Por lo que cada declaración puede estar construida en base a verdades, en base a su propias declaraciones o depender de hechos ya declarados anterior a este. Una serie de verdades, hechos y reglas ya declaras dentro de nuestro programa se conoce como "base de datos" o "base de conocimientos". ya una vez declara esta base de conocimientos podemos realizar consultas de caracter lógicos que también se describen como hechos en nuestro programa. El paradigma de programación lógico trabaja bajo ciertos mecanismos básicos a la hora de realizar consultas logicas en nuestra base de conocimientos, estos mecanismos son los siguientes:

Mecanismos básicos

- Unificación: Mecanismo recursivo utilizado para resolver una meta o consulta a la base de conocimientos de un programa. intentando igualar terminos en esta consulta para encontrar el resultado verdadero.
- Backtraking automático: Mecanismo recursivo para resolver también consultas a la base de conocimientos de un programa, al no encontrar un resultado optimo se da un paso atrás para intentar con otras posibles soluciones a la consulta
- Estructuras de datos basadas en árboles: El uso de arboles o busquedas de espacios de estado para la respuesta es las consultas sobre una base de conocimientos es escencial como mecanismo básico dentro del paradigma y utilizado por "prolog" en este caso.

Estructura de programa

Para manejar orden dentro de un programa lógico se requiere de una estructura organizacional de código, pues los hechos, verdades, o reglas poseen caracteríasticas sintácticas similares que pueden confundir al desarrollador, por lo anterior dicho, las partes importantes dentro de un programa lógico son:

- Documentacion:
  - Dominios : descricion breve del tipo de dato de las variables utilizadas dentro del programa
  - Predicados : lista de todos los predicados de nuestra base de conocimiento.
  - Metas: consultas directas sobre la base de conocimientos previamente escrita, pueden ser primarias(consultas por usuario) o secundarias(consultas por nuestro programa).
- Clausulas:
  - hechos : verdades abslutas dentro de nuestra base de conocimientos.
  - Reglas: objetos o consultas construidos en base a nuestra base de conocimientos.

Análisis del problema

Para el problema en cuestion se han dispuesto distintas zonas de trabajo las cuales constituyen un flujo de trabajo ordenado y unidireccional que a su vez son hechos que conformarán nuestra base de conocimientos, estos son:

- workspace : Espacio de trabajo donde se guardaran los archivos que queremos agregar como cambios a nuestro flujo de trabajo principal
- Index : Espacio de trabajo donde se preparan los archivos que están listos para ser enviados al nuestra rama de trabajo y que cuentan como cambio oficial dentro de nuestro flujo de trabajo mencionado.
- LocalRepository: Espacio de trabajo donde se encuentran nuestros cambios locales, no compartidos con el equipo de trabajo en su totalidad.
- RemoteRepository: Espacio de trabajo donde se guardan nuestros cambios globales, estos son compartidos en su totalidad por el equipo de trabajo que utiliza el espacio en cuestión.

Se definen dos unidades básicas dentro de nuestro programa, las cuales son:

- Archivo: documento que contiene el trabajo realizado dentro de un proyecto el cual se guardará dentro de nuestro flujo de trabajo.
- Commit: Unidad que tiene como proposito guardar todos los archivos y sus cambios asociados a los mismos. además de metadatos asociados a la persona creó el mismo con el fin de llevar el registro de los cambios dentro del proyecto.

Otro concepto importante dentro del problema en sí es el uso de “ramas” espacios de trabajo independiente unos de otros y que pueden surgir a partir de ellos mismos. los cuales pueden tener commits distintos entre sí. Para el caso de este laboratorio solo haremos uso de una sola rama la cual será la definida al momento de crearse dentro del programa. Estos conceptos se verán reflejados dentro de hechos y reglas de nuestro código que se presentarán a continuación, dentro de los aspectos de implementación del programa.

Aspectos de la implementación

Con respecto a los aspectos de la implementación de este laboratorio se definieron los siguientes:

NOMBRE	TIPO	DESCRIPCION
REPO	list	repositorio en cuestion
NOM	string	nombre del repositorio
AU	string	nombre del autor del repositorio
CRAT	string	fecha de creacion del repositorio
INDEX	list	lista de cambios(Index)
LOCAL	list	repositorio local(LocalRepository)
REM	list	repositorio remoto(remoteRepository)

La estructura de un "REPO" a modo de ejemplo está formada por de la siguiente manera:

- repository:
  - NOM "master" : nombre de la rama o proyecto.
  - AU "Rodrigo Mardones" : nombre de la persona que inicia el repo.
  - CRAT "07/10/1994" : fecha de inicio del proyecto.
  - INDEX [H | T] : lista con los archivos del flujo de trabajo.
  - LOCAL [H | T] : rama de trabajo local.
  - REM [H | T] : rama de trabajo remoto.

A su vez la estructura del index puede tener los siguientes estados:

- Index [INDEX] : solo con un archivo
- Index [INDEX | T] : con más de un archivo

Para el caso de LocalRepository y Remote Repository, las estructuras cumplen un patron similar:

- LOCAL [COMM]: para un solo commit local
- REM [COMM] : para un solo commit remoto
- LOCAL [COMM | T] : para más de un commit local
- REM [COMM | T] : para más de un commit remoto

Y las representaciones de archivo y commit son las siguientes:

- FILE "hola.py" : solo representado por un string, sin texto contenido.
- COMM [MSJ, FILE] : conteniendo un mensaje de tipo string y un archivo.

Para el caso del workingDirectory, su representación es meramente virtual y no fue contemplada al inicio de este proyecto pues solo se ocupa para guardar archivos

Dentro del programa se definieron los siguientes hechos:

- indexZone(INDEX) : crea una indexZone para su posterior añadido a al repository.
- localRepo(LOCAL) : crea un localRepository para su posterior añadido al repo.
- remoteRepo(REM) : crea un remoteRepository para su posterior añadido al repo.
- commit(mensaje,INDEX,COMM): recibe un mensaje y el index para crear un commit y agregarlo al localRepo.
- repository(NOM,AU,CRAT,repoOut): recibe como argumentos nombre del repo, autor, y fecha y retorna un repositorio armado por estructura.
- gitInit(NombreRepo,Autor,RepoOut) : crea Repositorio a partir de base establecida, entregando nombre de repo y autor antes.
- gitAdd(RepoInput, Archivos, RepoOut): agrega archivos al Index para su posterior guardado.
- gitCommit(RepoInput, Mensaje, RepoOutput) : crea un commit con un mensaje y los suma a la rama trabajada.
- gitPush(RepoIn,RepoOut): envia los cambios del localRepository al RemoteRepository.
- git2String(RepoInput, RepoAsString) : muestra los elementos ordenados en un string.

Para correr el proyecto de manera correcta es necesario correr el archivo git_18975534_Mardones.pl con un interprete de SWI-PROLOG >= 8.X.X. el proyecto ya viene con algunos ejemplos escritos al final del documento. para realizar consultas basta con tomar alguna de las consulta descritas en metas y realizarlas tal cual se especifica o tomar como ejemplo las ya hechas.

Conclusiones

Sin duda el paradigma de programación lógica nos aporta una nueva mirada al momento de resolver problemas de distinto índole, no solo en la forma de construcción de los problemas sino también en el entendimiento de estos con las distintas caracteristicas asociadas al uso de este paradigma, como lo son el backtracking automático, la unificación y las construccion de estructuras basadas en arboles.

El paradigma de programación lógica nos permite especificar una base de conocimientos ya establecida, por lo que la solución de problemas se construye en base a lo que ya sabemos del problema persé, haciendo que este se vuelva mucho más entendible al trabajar con algo preestablecido.

Al compararlo con el paradigma visto en el laboratorio anterior, resulta mucho más sencillo trabajar en este paradigma, pues declarar hechos como casos base y en base a caracteristicas de caracter booleano, se hace mucho más semejante al algebra booleana.

Referencias

Para el desarrollo de este laboratorio se han consultado las siguientes fuentes bibliograficas:

- Castel de Haro, M., & Llorens Largo, F. (2005). Practicas de logica (1.a ed., Vol. 1). Universidad de Alicante.
- SWI-Prolog documentation. (s. f.). Documentacion Oficial de Prolog. https://www.swi-prolog.org/pldoc/index.html
- Clocksin, W. F. (2003). Programming in Prolog: Using The Iso Standard (5th ed.). Springer.