Universidad Central “Marta Abreu” de las Villas

Facultad Matemática Física y Computación

Licenciatura en Ciencias de la Computación

Trabajo de Diploma

Un método para elaborar simulaciones del lenguaje Prolog

Autores: Welkis Yasel Pérez Carrillo.

Andrés Eduardo Gutiérrez Rodríguez.

Tutor: Dr. Mateo Gerónimo Lezcano Brito.

Santa Clara

2006

“Año de la Revolución Energética en Cuba”

Hago constar que el presente trabajo fue realizado en la Universidad Central Marta Abreu de Las Villas como parte de la culminación de los estudios de la especialidad de Ciencias de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

_____ _____

Firma del autor Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del tutor Firma del jefe del Seminario

Pensamiento

"…el corazón del sabio discierne el tiempo y el juicio."

Ec 8:5.

Dedicatoria

A mi Dios y Salvador Jesucristo.
A mi querida esposa y a mis padres.

Welkis.

A mis padres.
A mi familia.
A Maire.

Andrés.

Agradecimientos

A Mateo Lezcano, por toda la dedicación y ayuda que nos ha brindado.
Y a todas las personas que de una forma u otra, han contribuido con la realización de este trabajo.

Resumen

RESUMEN

El trabajo que se presenta, comienza haciendo un estudio de diferentes posibilidades que se pueden usar para hacer simulaciones del lenguaje Prolog. A partir de ese análisis y reconociendo las insuficiencias de las formas estudiadas, se define un método nuevo de trabajo para llevar a cabo esa labor. Se toma en cuenta que las simulaciones que se deben presentar tienen el objetivo de mostrarse a través de la WEB y por eso resulta imprescindible cumplir con un conjunto de requisitos que permitan, el acceso rápido, homogéneo y la no necesidad de descargar las animaciones.

Abstract

ABSTRACT

The project that is presented begins making a study of different possibilities that they can be used to make simulations of the language Prolog. Starting from that analysis and recognizing the inadequacies in the studied ways, it defines a new method of work to carry out that work. It takes into account that the simulations that should be presented have the objective of being shown through the WEB and for that reason it is indispensable to fulfill a group of requirements that they allow the quick, homogeneous access and the non necessity of discharging the animations.

ÍNDICE

ÍNDICE

Introducción

INTRODUCCIÓN

La Informática Educativa ha mostrado ser un apoyo efectivo para auxiliar al profesor en el proceso de enseñanza-aprendizaje, en ese entorno el Grupo de Informática Educativa (IE) de la UCLV ha logrado avances significativos.

Entre los logros del Grupo de IE, se destacan algunas herramientas para la enseñanza de la Programación Lógica.

La asignatura Programación Lógica, forma parte del plan de estudio de la carrera Ciencia de la Computación, el paradigma se enseña cuando ya los estudiantes han adquirido habilidades de programación en lenguajes que siguen, en general, la línea imperativa y al enfrentarse a la nueva forma de pensar, el estudiante trata de extrapolar las ideas anteriores, lo que trae por resultado que los programas se hagan de forma ineficiente y no exploten las verdaderas potencialidades del lenguaje (Lezcano, 1998).

El paradigma de la Programación Lógica se debe estudiar antes que otros debido a que se acerca más a la forma de pensar de los humanos, sin embargo, en la actualidad eso no es así y no existe perspectiva que indique que cambiará.

Para lograr una mayor eficiencia e integración de las herramientas desarrolladas por el Grupo de IE, sus miembros han dirigido esfuerzos a la integración de algunas de sus herramientas dentro de mapas conceptuales. En ese sentido se ha trabajado en varios aspectos, entre ellos: complejidad de algoritmos y estructuras de datos, que se enmarcan en las carreras de perfil Informático y se ha excursionado también en la enseñanza de la Botánica.

En ese contexto se enmarca el presente trabajo, el cual tiene la finalidad de establecer una forma de trabajo que permita realizar simulaciones de algunos de los mecanismos fundamentales del lenguaje Prolog para incorporarlos, en un futuro, a un sistema de mapas conceptuales que ya ha dado sus primeros pasos y se soporta sobre la herramienta cmap tools (http://cmap.ihmc.us).

Para realizar el sistema es necesario estudiar los mecanismos internos fundamentales de la Programación Lógica, de forma que se puedan programar simuladores que ayuden a visualizar esos mecanismos con el objetivo central de que sean comprendidos a través de técnicas de enseñanza por descubrimientos.

Tomando en cuenta los aspectos mencionados, la tesis que se presenta se propone como:

Objetivo general:

- Establecer una forma de trabajo para desarrollar simuladores de mecanismos propios del paradigma de la Programación Lógica.

Objetivos específicos:

- Estudiar posibles variantes para simular los mecanismos internos fundamentales de la Programación Lógica.
- Programar simuladores usando las técnicas presentadas en el trabajo.
- Usar técnicas de enseñanza por descubrimientos para que la visualización sirva de método de enseñanza.

En la presente investigación se ofrecen las técnicas y la forma de trabajo para realizar las herramientas de simulación comentadas, las ideas descritas se materializan en un pequeño sistema que simula algunos de los mecanismos del lenguaje Prolog.

Capítulo I

CAPÍTULO I. LA INFORMÁTICA EDUCATIVA Y LA PROGRAMACIÓN LÓGICA

I.1 Introducción

La propuesta de utilizar computadoras en la enseñanza surge prácticamente con las computadoras mismas, si bien ella no se comenzó a destacar hasta la aparición de las microcomputadoras de bajo costo y se ha visto recientemente fortalecida con el desarrollo de los multimedios y las redes de computadoras. En línea con este propósito se han acuñado los términos hispanos Informática Educativa (IE), Enseñanza Asistida por Computadoras (EAC), Enseñanza Asistida por Ordenador (EAO), Software Educativos (SE), Programas Instructivos (PI) y otros más (Lezcano, 1998).

El uso de las técnicas de EAC se ha extendido notablemente en los últimos años. Nuestro país, que siempre se ha caracterizado por sus esfuerzos en el campo de la enseñanza, no podía ser menos y hoy en día puede mostrar una amplia gama de medios de enseñanza basados o soportados en computadoras. En ese entorno se ubica el presente trabajo.

La enseñanza de cualquier asignatura necesita de métodos auxiliares para lograr que sus conceptos sean bien comprendidos y asimilados, esa aseveración se hace mas real en la enseñanza de la Programación Lógica debido a los aspectos señalados anteriormente y basado, además, en el hecho de que la mayoría de las dificultades de los estudiantes en la utilización del lenguaje provienen de diseñar la solución al problema que enfrentan en términos de estructuras propias de paradigmas basados en la programación imperativa y los procesos particulares del Prolog como la unificación, el backtracking, y en general los mecanismos propios del lenguaje no se usan de la forma adecuada. La IE es una vía eficaz para lograr ese objetivo.

En este capítulo se analiza el desarrollo de la IE en forma general y la manera particular en que puede usarse para ayudar a enseñar aspectos propios de la Programación Lógica.

I.2 Informática Educativa. Enseñanza por descubrimiento

La enseñanza del actual siglo ha estado matizada por el uso de los medios técnicos auxiliares, dentro de los cuales la computadora ha desempeñado una función preponderante por las ventajas que incorporó, tanto para la explicación de los conceptos como para su apropiación (Almeida et al., 1997). En nuestro país el sistema educacional también ha tenido cambios y transformaciones a raíz del desarrollo de las nuevas tecnologías, como es la inserción del uso de las computadoras en las escuelas, donde la EAC viene a jugar su papel.

La historia de la IE como disciplina, se remonta a épocas tan temprana como la década del 50, cuando se desarrollaron sistemas mecánicos y electromecánicos que permitían la presentación de programas lineales, basados en el principio de respuesta activa (Chambers, 1983). En ese entonces quedó claro que la enseñanza se debía personalizar y surgió la idea de permitir la ramificación del proceso de enseñanza de acuerdo a la validez de las respuestas de los estudiantes, objetivo que fue posible alcanzar cuando surgió la programación ramificada.

Desde principios de la década del 60, las computadoras sirvieron como base de los sistemas de enseñanza automatizados, existiendo la convicción de que la EAC debía proporcionar nuevos e importantes cambios a la enseñanza en un futuro no muy lejano.

A finales de los años 60 y principios de los 70 surgieron los *sistemas generativos*, asociados a una nueva filosofía educativa que manifiesta: "los alumnos aprenden mejor enfrentándose a los problemas de dificultad adecuada, que atendiendo a explicaciones sistemáticas"; es decir, adaptando la enseñanza a sus necesidades.

Durante parte de los años 70 y hasta comienzos de los 80 se produce un estancamiento, principalmente debido a la falta de madurez del desarrollo tecnológico: los reducidos rendimientos y prestaciones de las computadoras en comparación con sus costos y también debido al escaso desarrollo conceptual y metodológico (Ruiz, 1996).

Un importante momento para los sistemas de EAC en particular fue el surgimiento del microprocesador, que rompió con las grandes limitaciones de enormes computadoras a costos muy elevados para los sistemas de educación.

En los últimos años se han diseñado e implementado ambientes de aprendizaje poderosos, que se compenetran con las características de los procesos de aprendizaje en forma efectiva y que involucran una nueva concepción del aprendizaje (Lezcano, 1998).
La IE no es una disciplina puramente técnica; psicólogos, educadores y sociólogos discuten con relación a su valor pedagógico real. Hoy en día los avances en la psicología, combinados con la explosión tecnológica, han dado lugar a una serie de sistemas educativos que permiten resolver algunas limitaciones de la educación tradicional (Cabrera, 1995).

I.2.1 Lo que el programa educativo debe tener

Como resultado de las experiencias pedagógicas realizadas en un trabajo de investigación de la Universidad Pedagógica "José Martí" de Camagüey (Ulloa, 2005) se puede afirmar que los factores que hacen posible un mejor aprendizaje mediante el uso de software educativo son:

- La ejercitación: Cuanto más se ejercita, más se aprende.
- Interés: Los alumnos deben darse cuenta efectiva de la conveniencia de mejorar los resultados del aprendizaje.
- Efecto: Se aprende mejor aquello que provoca efectos placenteros.
- Ritmo: El aprendizaje es más efectivo cuando se realizan ejercitaciones breves, separadas por cortos intervalos.
- Desarrollo psicomotriz: El educando solo aprende cuando está maduro en sus capacidades básicas.
- Progresión: Se aprende mejor cuando la intensidad y la complejidad de las experiencias suceden en un orden progresivo superando las dificultades.

Según la profesora Madelen Piña de la Universidad de Carabobo (Piña, 2004) que imparte la asignatura Módulo de Materiales Educativos Computarizados, para considerar que un MEC está diseñando correctamente, se debe tener en cuenta que garantice lo siguiente:

- Facilitar la motivación.
- Recordar el aprendizaje anterior.
- Proporcionar nuevos estímulos.
- Activar la respuesta de los alumnos.
- Proporcionar información.
- Estimular la práctica.
- Establecer una secuencia de aprendizaje.
- Propiciar recursos.
- Generar efectos visuales y auditivos.
- Ser cómodamente interactivos.
- Poder procesar símbolos y ser modificables.

I.2.2 Tipos de Software Educativo

El grupo de informática educativa de la UCLV considera que una posible clasificación de software educativo puede ser:

a. Función educativa:
    - Sistema Tutorial.
    - Sistema de ejercitación y práctica (entrenador).
    - Simulaciones.
    - Juegos didácticos.
b. Por su forma de presentación: (sin pasar por alto el desarrollo de las NTIC que permiten la existencia de este tipo de sistema)
    - Multimedia.
    - Hipermedia.
    - Sitios web.
c. Otra clasificación atendiendo al uso o no de técnicas de Inteligencia Artificial:
    - Sistemas convencionales.
    - Sistemas inteligentes. Estos sistemas a su vez pueden ser del tipo a) o b).

Todas estas clasificaciones no son excluyentes y pueden aparecer combinadas en un mismo software educativo.

I.2.3 Enseñanza por descubrimiento

El impacto de las computadoras en la educación ha sido ampliamente estudiado durante los últimos años y no todos los resultados han sido satisfactorios, lo que indica claramente la necesidad de continuar realizando investigaciones que arrojen más luz sobre este importante medio de enseñanza. Una gran ventaja, que se ajusta mucho a este trabajo, es que mediante la EAC los estudiantes tienen la posibilidad de observar la simulación del comportamiento de los sistemas que están estudiando.

Este proyecto vincula la IE con la Enseñanza por descubrimiento para que se ajuste más a nuestros objetivos y tenga un campo bastante amplio de aplicación.

En la Enseñanza por descubrimiento el aprendiz es el actor principal. El instructor no expone los contenidos de un modo acabado; su actividad se dirige a darles a conocer una meta que ha de ser alcanzada y además de servir como mediador y guía para que los individuos sean los que recorran el camino y alcancen los objetivos propuestos. En otras palabras, el aprendizaje por descubrimiento es cuando el instructor le presenta todas las herramientas necesarias al individuo para que éste descubra por sí mismo lo que se desea enseñar y él debe aprender.

Constituye un aprendizaje útil, pues cuando se lleva a cabo de modo idóneo, asegura un conocimiento significativo y fomenta hábitos de investigación y rigor en los individuos.

Jerome Bruner (citado en el artículo electrónico “Aprendizaje por descubrimientos”) atribuye una gran importancia a la actividad directa de los individuos sobre la realidad.

I.3 Informática Educativa en la UCLV

El Grupo de Informática Educativa de la UCLV ya tiene una trayectoria de muchos años, en los cuales se ha realizado una gran variedad de trabajos en diferentes campos.

- De la Facultad de Matemática Física y Computación se destacan varios proyectos:

  - La Máquina de Inferencia para Sistemas de Enseñanza MISE (García et al., 2006). Es una herramienta para elaborar sistemas de enseñanza. Este trabajo obtuvo premio en el 09 Forum de Ciencia y Técnica.
  - Un Software educativo sobre manifestaciones neurológicas en lesiones del tronco encefálico (Santana, 1998).
  - Una herramienta RAR para software educativo. RIMI para Windows (Rodríguez, 1996).
  - El Sistema para la Enseñanza de Sistemas Expertos (SESE). (Lezcano, 1998).
  - El SE Teaching (Lezcano, 1998) para apoyar la enseñanza de la Programación Lógica, el cual constituye una pieza clave en la realización de este trabajo.
- Otras facultades también han realizado trabajos relevantes, destacándose la Facultad de Eléctrica:
  - Una estrategia para la enseñanza de la Electrónica de la Facultad de Eléctrica (Chaljub, 2006), que también obtuvo premio en el 09 Forum de Ciencia y Técnica.
  - Herramientas para la educación y el entretenimiento. Proyecto MLUDI GRAPHS (Pérez, 1995).

Actualmente están en ejecución diferentes proyectos del Grupo de Informática Educativa de la Facultad de Matemática, Física y Computación (GIE-MFC) y nuestro trabajo forma parte de uno de ellos.

I.4 Enseñanza del lenguaje Prolog

Prolog es el más extendido de los lenguajes de programación lógica, se basa en el cálculo de predicados de primer orden y se utiliza ampliamente en investigaciones de Inteligencia Artificial (Kowalski, 1986), lo que justifica su inclusión en los programas de estudios de la carrera Ciencia de la Computación.

Enseñar el lenguaje, es un gran reto para los profesores debido a que el paradigma de programación difiere, en gran medida, de los paradigmas anteriormente aprendidos por

los estudiantes, los cuales, muchas veces, pretenden extrapolar las ideas de los paradigmas conocidos.

Estudios comparativos entre los paradigmas imperativos y declarativos, han revelado que las dificultades de los estudiantes en la utilización de Prolog proviene de diseñar la solución al problema en términos de estructuras propias del paradigma procedural, como ***if***, ***while***, ***repeat***, etc., y no encontrar al momento de la implementación, primitivas Prolog para traducir dichas estructuras (Peri y Godoy, 1996).

La fase de diseño dentro del paradigma lógico, a diferencia del imperativo, consiste en la descripción del conocimiento involucrado en el problema con el fin de obtener una representación simbólica del mismo a través de un conjunto ordenado de relaciones (aserciones y reglas de inferencias). Dentro de esta representación se incluye el conocimiento sobre el problema y también la estrategia de resolución. De esta forma el programador puede abstraerse de las características de control del lenguaje en favor de la definición lógica de las relaciones.

El obtener los resultados de salida esperados consiste, entonces, en una o más consultas a la base de datos en donde se encuentra traducida la representación en cláusulas lógicas, para de esa manera encontrar instancias individuales de las relaciones allí descritas.

Todo esto ocasiona grandes dificultades a los estudiantes que se enfrentan por primera vez con el Prolog, pues la mayoría de ellos enfoca el diseño de la solución como un plan y luego falla al intentar traducirlo (Peri y Godoy, 1996).

Aquí no terminan los problemas del lenguaje. Un punto muy importante es que los estudiantes carecen de un modelo concreto de cómo actúa la computadora en el cálculo de la solución, no siendo así con los algoritmos procedurales en los que se comprende con claridad que el programa es ejecutado línea por línea hasta alcanzar el final del programa.

En esta tesis se presenta una forma o método de trabajo que permite realizar simulaciones de los mecanismos internos del lenguaje, la materialización de este método se aprecia a través de un conjunto reducido de applets de Java, los cuales permiten, entre otras cosas, explicar de forma visual cómo se alcanzó un objetivo, cómo se instancian las variables, etc. Del mismo modo se puede explicar por qué una conclusión particular no puede ser alcanzada.

El mecanismo descrito anteriormente ayuda a que los alumnos comprendan la forma en que actúa el lenguaje y se percaten de que muchas de las cosas que pretenden hacer imperativamente las hacen los mecanismos propios del lenguaje y lo único que hay que hacer es cambiar la forma de pensar y declarar correctamente las cláusulas que describen el problema a resolver para que esos mecanismos se encarguen de resolverlo.
Diversos son los planes de estudios y los programas realizados para impartir la asignatura Programación Lógica en todo el mundo. Los objetivos esenciales de estos programas y planes de estudio se pueden resumir en:

- Justificar la aparición de lenguajes declarativos y de los modelos computacionales que representan, así como las líneas de evolución fundamentales.
- Indicar las características representativas de la programación lógica y de la programación funcional, comparando ambas con el modelo imperativo tradicional.
- Adquirir la capacidad de diseñar e implementar programas en un lenguaje representativo de cada paradigma.
- Valorar las ventajas e inconvenientes de cada paradigma y comparar con otros modelos computacionales.
- Fomentar la capacidad crítica de los alumnos respecto a los lenguajes de programación.
- Comprensión de la relación de esta materia con otras disciplinas, particularmente en las áreas de aplicación más cercanas como la Inteligencia Artificial.
- Identificar los problemas de la utilización de características no declarativas en lenguajes declarativos.

En la UCLV se tratan de cumplir todos esos objetivos, aunque algunos no de forma tan directa, motivando al estudiante con clases teóricas y prácticas que faciliten su comprensión. Incluso, las estrategias han cambiado, unas veces dándole más importancia a la parte teórica, otras a la práctica y, por último, buscando un equilibrio entre ambas.
La parte práctica de la asignatura ha sido pensada como soporte y apoyo a los conceptos y técnicas descritas y trabajadas en la parte de teoría, facilitando con ello la comprensión

de conceptos abstractos (Barwise, 1996). Durante este tiempo de laboratorio los estudiantes utilizan el lenguaje Prolog y otras herramientas informáticas.

I.5 Informática Educativa para enseñar Prolog

Después de una exhaustiva búsqueda por Internet no se encontraron SE para enseñar Prolog que se ajusten al presente trabajo y que se puedan poner como ejemplos o que se puedan usar directamente para resolver algunos de los problemas planteados. Sólo el sistema Teaching ha guiado la confección del proyecto.

I.6 Resumen

Este capítulo enfatiza en el desarrollo de la IE como disciplina y de cómo se ha convertido, en nuestros días, en una pieza clave de todos los sistemas de enseñanza del mundo y en particular de nuestro país, que tantos logros ha obtenido en la educación de sus habitantes.

La UCLV ha realizado trabajos importantes en el campo de la IE, con el fin de utilizar los avances de las nuevas tecnologías en el proceso docente-educativo. Una de las técnicas empleadas es la enseñanza por descubrimientos.

En el capítulo siguiente se presenta un método de trabajo general para hacer simulaciones del lenguaje Prolog que permitirán su empleo en la enseñanza de tópicos fundamentales de este lenguaje.

Capítulo II

CAPÍTULO II. MÉTODO DE TRABAJO

II.1 Introducción

En el capítulo I, se justificó la existencia de mecanismos auxiliares para la enseñanza en general y para la enseñanza del Prolog en particular. Quedó claro que una vía adecuada para comprender los mecanismos internos del lenguaje era hacer simulaciones que mostraran la forma de funcionamiento de un programa en ejecución.
En este capítulo se hace un análisis de diferentes variantes que se pueden seguir para programar las simulaciones y se establece una forma general para hacerlas.

II.2 Técnicas analizadas. Análisis crítico

Lo primero que debe hacerse cuando se acomete una tarea como la planteada en el epígrafe anterior es realizar una búsqueda de las técnicas disponibles para ver si alguna de ellas resuelve el problema planteado, si ese no es el caso será necesario hacer una valoración crítica de los aspectos positivos y negativos de cada una de las técnicas analizadas para, a partir de ahí, presentar un método de trabajo nuevo, en el resto del epígrafe se fundamenta esta forma de trabajo.

II.2.1 Proyecto anterior

Como parte del plan de estudio de la asignatura Programación Lógica, en el curso 2004-2005, los autores realizamos dos proyectos, que fueron nuestros primeros pasos en el tema que hoy presentamos.
En esa ocasión se desarrollaron dos herramientas en Delphi para sustituir dos de las tareas realizadas por el software Teaching, que estaba implementado en Arity Prolog con algunos elementos de Ensamblador.
Uno de los aspectos principales que trata el proyecto Teaching es el backtracking (mecanismo poderoso de Prolog que permite la búsqueda de soluciones alternativas), mostrando a través de varios ejemplos la traza que deja un programa Prolog después de ejecutar una instancia determinada.

El backtracking es un regreso atrás que se produce ante un fallo. El objetivo del regreso es buscar soluciones alternativas, para lo cual se desvalorizan todas las variables que hayan sido acotadas en el paso previo y se regresa al objetivo anterior para buscar otro camino de solución (una solución alternativa), fallando sólo cuando no existan más soluciones. Este mecanismo produce una búsqueda exhaustiva en el árbol de soluciones, lo que garantiza que se exploren todas las alternativas de solución de un problema dado (Lezcano et al., 2000).

Por su complejidad, el Teaching muestra de una manera diferenciada la señalización de la traza cuando ocurre backtracking y cuando no ocurre. El propósito es que el alumno preste más atención a lo que está pasando y a los nuevos estados de las variables.

Los dos problemas escogidos para realizarle una interfaz más acorde con las necesidades actuales, que las que se podían brindar con el Arity, fueron el problema de las derivadas y el problema de los autómatas.

El problema de las derivadas calcula la derivada de una función que se entra como dato. El problema de los autómatas implementa un autómata de pila no determinístico. La programación de los predicados que resuelven estos problemas es relativamente sencilla, ya que el lenguaje se presta, en forma natural, para tareas de este tipo.

En los proyectos que se realizaron el año pasado se utilizó el Amzi Prolog para crear los predicados Prolog que dieran solución a los problemas descritos, y con una técnica ingeniosa, utilizando las ventajas del lenguaje y del propio backtracking, se crearon desde el Amzi archivos textos con toda la información correspondiente a las cláusulas que fueron llamadas, después de cada ejecución, y a las instanciaciones de las variables, señalando, por supuesto, cuándo ocurrió backtracking.

De esta manera se puede invocar al Prolog con cualquier instancia del problema escogido, es decir, las consultas pueden variar una y otra vez.

Desde la interfaz en Delphi se ordena ejecutar al programa Prolog correspondiente. Esto se logra a través de la unit **amzi.pas** y de la dll **amzi.dll**, que son las que enlazan con Amzi Prolog. Se crea, en el Delphi, un objeto de tipo TLSEngine, y éste es el encargado de cargar el archivo donde está la regla consult y después que esté conformada la pregunta, llamar al Prolog y recibir los resultados (true o false) para luego convertirlos a cadena.

El archivo texto creado por el Amzi Prolog se analiza luego desde el Delphi para mostrar, en la ventana principal, las trazas correspondientes a cada paso y las acotaciones de las variables.

Para poder ver la simulación de un problema no sólo es necesario el archivo ejecutable que genera Delphi, hace falta tener en una misma carpeta la dll amzi.dll, los archivos de prolog compilados y espacio suficiente para guardar los archivos textos creados.

No es bueno insertar estos programas dentro de un mapa conceptual ya que dificulta la navegación por la red. Hay que descargar los archivos necesarios y no resulta cómodo para el estudiante, que puede perder el interés por las simulaciones o temer por la seguridad de su máquina.

Ante la dificultad mencionada anteriormente y tomando en cuenta que la nueva aplicación iba a trabajar a través de la WEB, fue necesario buscar vías alternativas para resolver este problema y poder brindar al alumno una forma fácil y ágil de acceso a las simulaciones.

II.2.2 Técnicas analizadas

Los mapas conceptuales ofrecen grandes ventajas como herramientas para la navegación en Internet, aprovechando verdaderamente la flexibilidad que ofrecen las nuevas tecnologías. El problema de la navegación, común en los sistemas de multimedia y persistente en la navegación de la WWW, se resuelve fácil con estos mapas. Su menú de iconos corresponden a diversos medios (texto, imágenes, vídeos, otros mapas conceptuales, simulaciones, etc.) y éstos facilitan el proceso de navegación, ya que puede accederse a ellos desde cualquier nodo de la red.

Los mapas conceptuales son muy útiles en la educación a distancia y, junto con sus ligas y recursos, son también accesibles como páginas de HTML. Por eso, los sistemas insertados en sus iconos deben ser fáciles de descargar y resulta más provechoso si esta operación no es necesaria.

Los programas escritos en Java pueden ser ejecutados en cualquier plataforma computacional, el lenguaje está muy orientado al trabajo en red, soportando protocolos como TCP/IP, UDP, HTTP y FTP.

Además de los aspectos mencionados anteriormente, Java ofrece los Applets, los cuales son programas diseñados para ser ejecutados en una página web y son muy seguros, pues en general no permiten realizar cualquier acción que pudiera dañar la máquina o violar la intimidad del que visita la página web, a lo anterior se une el hecho de que no necesitan ser descargados.

Queda claro entonces que los applets de Java se constituyeron en nuestro mejor candidato al reunir todas las condiciones que se necesitan para ubicar las simulaciones dentro de los nodos de los mapas conceptuales.

La construcción de estos applets puede ser sencilla o compleja, dependiendo del problema a resolver. En Internet se encuentran herramientas que ayudan a confeccionarlos y un sistema llamado Ejs (Easy Java Simulations) es muy bueno para hacer applets y para crear simulaciones en particular (Esquembre, 2005), la figura 2.1 muestra una vista del mismo:

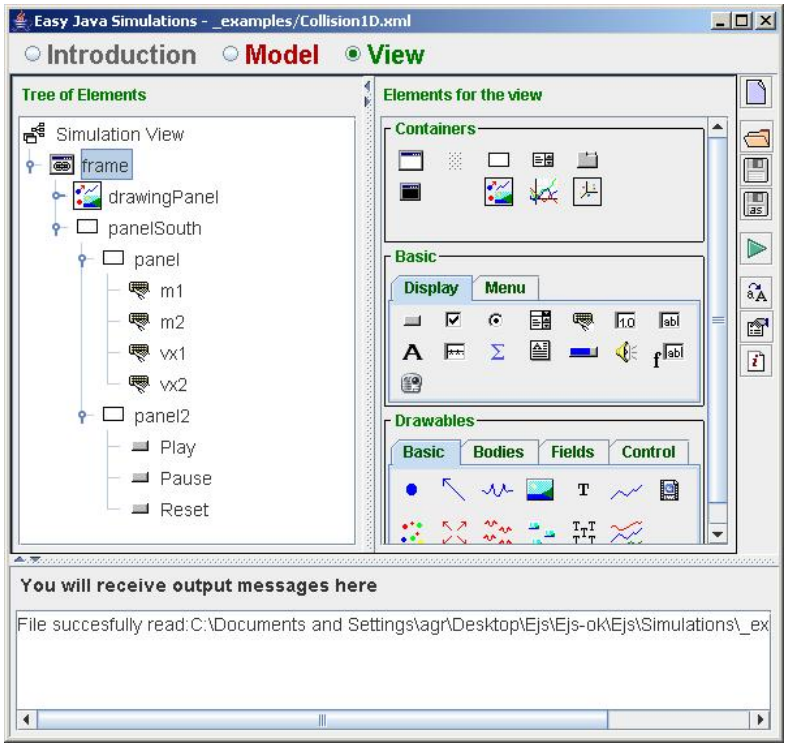

Figura 2.1. Ejs

Ejs, es un entorno gráfico de programación orientado al desarrollo de simulaciones científicas y técnicas. Permite desarrollar simulaciones Java a personas no expertas en informática y desconocedoras de éste lenguaje. Ejs genera Applets de Java, la ventaja de este lenguaje reside en que los applets que genera pueden ser incluidos en los archivos html con las ventajas que ello implica en cuanto a ser aplicaciones multiplataforma y utilizados en Internet. Posee un Modelo: conjunto de variables y ecuaciones; una Vista o interfase gráfica: donde se crean los elementos que componen la Vista de la aplicación utilizando básicamente el ratón; y un Control: acciones que el usuario quiere realizar.

Pero a pesar de que se pueden construir applets de cualquier tipo con esta herramienta, ella está principalmente diseñada para resolver problemas de Física, y crear simulaciones de otro tipo es particularmente complejo ya que definir variables y ecuaciones que controlen el movimiento de las imágenes dificulta el trabajo.

La Programación Lógica, hoy en día, está relacionada con Java y se encuentran diferentes aplicaciones de Programación Lógica en Java pero no para crear simulaciones. Sin embargo, pueden ser de gran ayuda como el WProlog (Barrera, 1999), para tomar algunas ideas. WProlog (figura 2.2) es un intérprete implementado en Java.

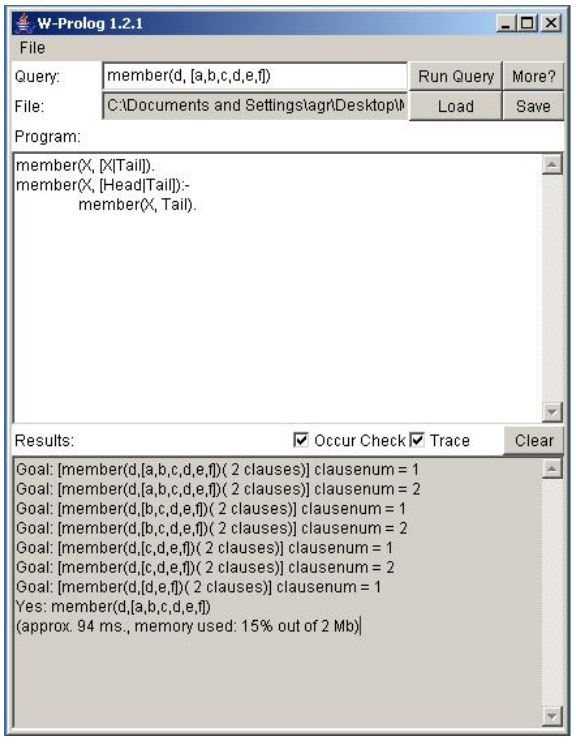

Figura 2.2. WProlog

Existen otras aplicaciones de Java más Programación Lógica como por ejemplo:

- Basado en JNI como: JASPER (Sictus), JPL (Swi) JIPL (K-Prolog)
- Basado en Socket como: Interprolo (XSB) y Prolog IV (Barrera, 1999).

Ninguna de estas herramientas ayuda a conformar las simulaciones necesarias para alcanzar los objetivos propuestos pues sólo funcionan como intérpretes del lenguaje Prolog. El Ejs tiene características específicas que interfieren en la creación del tipo de animaciones que se pretenden realizar.
Crear un sistema en otro lenguaje de programación distinto de Java no brindaría los requerimientos necesarios para navegar por la red.
Para confeccionar las simulaciones hay que hacer los applets en el propio lenguaje Java, esto permite, a parte de todas las ventajas del lenguaje mencionadas anteriormente, una mayor generalización de los problemas, para reutilizar el código necesario en cada uno de los ejemplos construidos. De esta forma se podría lograr que los ejemplos dieran solución a cualquier instancia, aunque por ahora basta con que las consultas a los ejemplos sean fijas para permitir que las simulaciones se muestren más rápido a través de la red.

II.3 Método de trabajo

El punto de partida para realizar el proyecto es escoger los problemas que se van a simular. Hay que decidir si las instancias o consultas a estos ejemplos son fijas o si pueden variar; esto es muy importante. Como el principal objetivo es enseñar, los primeros ejemplos tienen que ser sencillos y es bueno que las animaciones expliquen con la mayor claridad posible la ejecución de los mismos.
Si se construye una aplicación, o un applet, de propósito general, que actúe quizás hasta de intérprete, las animaciones tendrían que ser también generales y su programación sería muy compleja. Resultaría muy útil un sistema de este tipo, por sus múltiples aplicaciones. Por ahora es suficiente con ejemplos no muy complejos, que despierten el interés de los estudiantes que se inician en la asignatura.

Se decidió que las consultas a los ejemplos no cambiarán, es más fácil para que los applets naveguen por la red y para la comprensión de los estudiantes.

Los ejemplos desarrollados pretenden mostrar la solución a problemas sencillos que utilizan mecanismos generales del lenguaje como son la unificación y el trabajo con listas.

Podemos decir que dos términos machean o unifican cuando ellos son idénticos o cuando las variables en ambos términos pueden ser instanciadas por objetos en cuyo caso, después de la sustitución de las variables por esos objetos, los términos pasan a ser idénticos (Bratko, 1986). Existen varias reglas de unificación:

1. Dos cadenas se unifican cuando son exactamente iguales.
2. Dos números cualesquiera unifican cuando son exactamente iguales.
3. Para que dos listas se unifiquen tienen que responder al mismo patrón, o tener la misma cantidad de elementos y en cualquier caso todos los elementos de la lista también deben unificarse.
4. Una variable libre se unifica con cualquier otro elemento sintáctico del lenguaje y como resultado la variable queda instanciada o acotada con ese elemento.
5. Dos términos se unifican si tienen el mismo nombre y la misma cantidad de argumentos y cada uno de ellos también se unifica.

Una lista es una estructura de datos simple; es una secuencia de cualquier cantidad de elementos y puede ser vacía o no. Si es vacía se escribe como un átomo de Prolog, [ ]; si no, consta de dos partes: el primer elemento es la cabeza de la lista (*head*) y lo demás es el resto de la lista, conocido como *tail* (Bratko, 1986).

El primer problema presentado implementa la relación ***miembro***, que se define como:

“un elemento dado es miembro de una lista si está en su cabeza o está en el resto de la lista” (Bratko, 1986). Se muestran dos ejemplos, cuando el elemento dado es miembro de una lista dada y otro cuando no lo es.

El segundo problema también es una de las operaciones más comunes con listas, la ***concatenación***. Consiste en dado dos listas, devolver una tercera lista que sería la concatenación de las dos anteriores (Bratko, 1986).

El problema de la ***intersección*** de dos listas es el tercero que se resuelve, viene siendo como la intersección de dos conjuntos, donde dadas dos listas se devuelve otra que es la intersección de ambas; para resolverlo, desde el punto de vista del Prolog, se utiliza la implementación de la relación miembro. De esta manera los alumnos pueden analizar como el Prolog ejecuta dos predicados, donde uno llama al otro.

Los predicados se implementan en el Amzi Prolog. Tomando como referencia los proyectos que se explican en el epígrafe II.2.1, se pueden crear archivos textos desde Prolog para seguir las trazas después de realizadas las consultas pertinentes. Pero esa vía no fue la utilizada finalmente.

Para seguir las trazas se utiliza el WProlog. Esta aplicación interpreta el código de Prolog y genera un árbol con las cláusulas que fueron accedidas y las que fallaron después de una consulta determinada. Los nuevos problemas que se agreguen necesitan ser corridos en el WProlog (o en cualquier otro intérprete de Prolog) para conocer que línea del programa fue accedida en cada momento y el estado de las variables; sin esa información es imposible hacer las herramientas simuladas pues es utilizada en los applets (o en el frame) para mostrar todo el proceso de ejecución de un programa Prolog. Para crear las animaciones, aunque basta con ésta última vía, se puede utilizar de guía el archivo texto creado desde el amzi, para estar más seguros de cada traza y de la acotación de cada variable.

En el Capítulo I se mencionaron diversos factores que hacen posible un mejor aprendizaje con los SE: efectos placenteros, ejercitaciones breves, orden progresivo para superar las dificultades, entre otras. Los dos últimos factores justifican los ejemplos escogidos. El primero plantea una interrogante acerca de cómo realizar las animaciones. Quizás con un mayor rigor de investigación se puedan encontrar mejores soluciones para crear animaciones que sirvan a cualquier problema; para lograrlo, ellas necesitan ser implementadas desde el propio lenguaje Java y hay que analizar si solamente mostrarían el árbol de ejecución del Prolog o si es posible hacerlas más vistosas y útiles según el problema.

Como los primeros ejemplos son simples y es de vital importancia que despierten el interés de los estudiantes, además de mostrarles de una forma clara y natural todo el proceso de ejecución, las animaciones se realizaron en Flash, creándose imágenes

animadas que se cargan desde los applets y que son únicas del problema que simulan y de la consulta que cada ejemplo resuelve.

II.3.1 Método de trabajo generalizado

Cuando se quiera simular un nuevo problema se pueden seguir una serie de pasos que se describen a continuación:

1. Escoger e implementar el problema que se quiere simular después de un minucioso estudio acerca de en qué consiste, si es útil para el aprendizaje del estudiante, cuáles son los mecanismos de Prolog que utiliza y cuáles necesitan ser mostrados con más énfasis, si su implementación aporta elementos al desarrollo del alumno y su nivel de complejidad. Es necesario también crear una consulta adecuada.
2. Definir cómo se construirán las animaciones. Pueden ser creadas en Java, en Flash o en cualquier otro editor de imágenes. Es imprescindible conocer qué hay que mostrar en cada estado de la traza: una representación de las estructuras del problema o una modificación en las variables o el estado de la pila. Si es necesario animar éstas imágenes, Flash es una herramienta poderosa y sencilla de entender, cualquier movimiento puede ser creado: un elemento volando o entrando a la pila, o dos listas uniéndose, o insertar nodos en un árbol, entre otras múltiples aplicaciones; las animaciones creadas pueden ser exportadas como imágenes animadas o como vídeos.
3. Dentro de la carpeta principal del proyecto hay una carpeta images con las imágenes que él utiliza. Si las animaciones se crearon como imágenes animadas, que es el método utilizado en estas herramientas, tienen que guardarse allí. La clase JPaintScrollPane es el panel donde se muestran las imágenes; ella tiene un método fillvector que llena un arreglo con las imágenes animadas. Sólo hay que cambiar el tamaño de ese arreglo de acuerdo con la cantidad de animaciones que se utilicen.

4. La clase TrazaFrame3 crea un objeto Prolog, cuya clase se define dentro del paquete del mismo nombre y contiene el programa escrito en Prolog y el resultado de la traza del WProlog; también contiene métodos para su creación y manipulación. En el método getProlog() de la clase TrazaFrame3, la cadena text debe ser llenada con el programa Prolog y la cadena goalStr, con el resultado que devuelve el WProlog después de ejecutarse el predicado con la consulta determinada. Esta consulta se le pasa como parámetro al constructor del objeto Prolog, así como el nombre del ejemplo. La clase PrologInformer es la interfase entre el objeto Prolog y la clase visual, ella se encarga de mostrar lo que este objeto contiene.

Siguiendo estos pasos se puede agregar cualquier nuevo ejemplo, o más bien crearlos aparte, en otros applets. Basta con copiar el mismo código y sólo hacer los cambios que aquí se señalan. Si no se crean las animaciones como imágenes hay que tener mucho cuidado, porque entonces ya este método no serviría.

II.4 Diagramas y vista del sistema

Diagrama de Casos de Uso:

Figura 2.3. Diagrama de casos de uso

El sistema presenta un solo caso de uso: Analizar ejemplo. Este caso de uso tiene como objetivo que el actor Estudiante, que es el único que interviene, analice el ejemplo escogido de los cuatro que se dan como opciones en el menú. El actor puede iniciar la

ejecución del ejemplo e ir paso a paso analizándolo, ver el árbol de ejecución y la instanciación de las variables y terminar cuando llegue al final o cuando desee.
La figura 2.4 presenta un diagrama con cada una de estas actividades.

Diagrama de actividad:

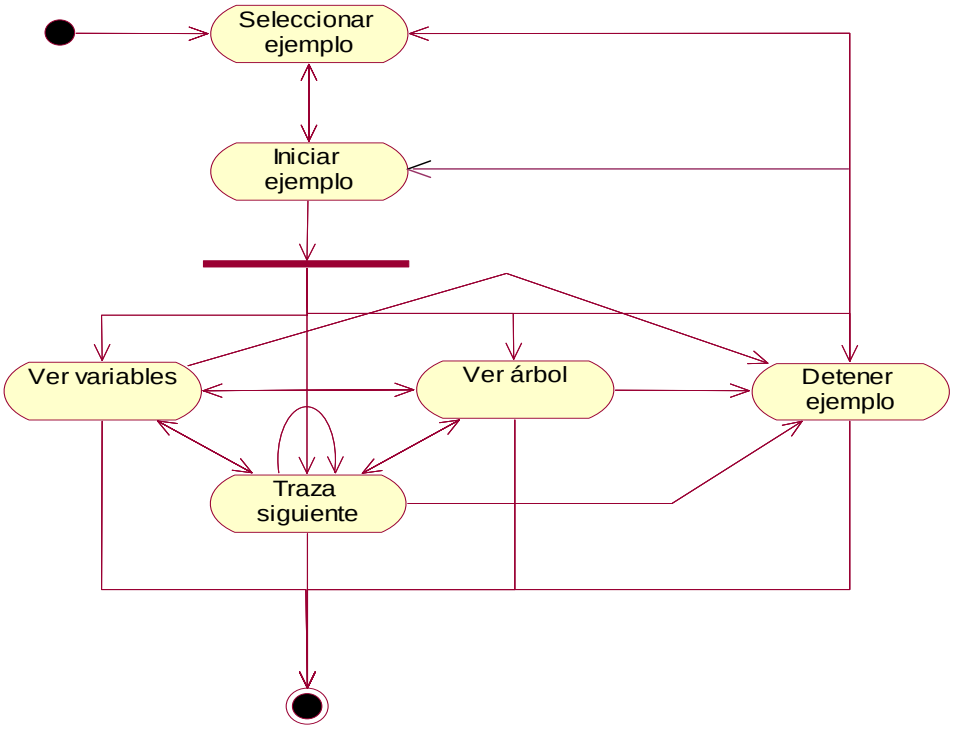

Figura 2.4. Diagrama de actividad

El Diagrama de clases de este sistema (figura 2.5) especifica la relación de asociación, que es la única que existe, entre cada una de las clases más importantes utilizadas en la implementación del sistema.

Diagrama de Clases:

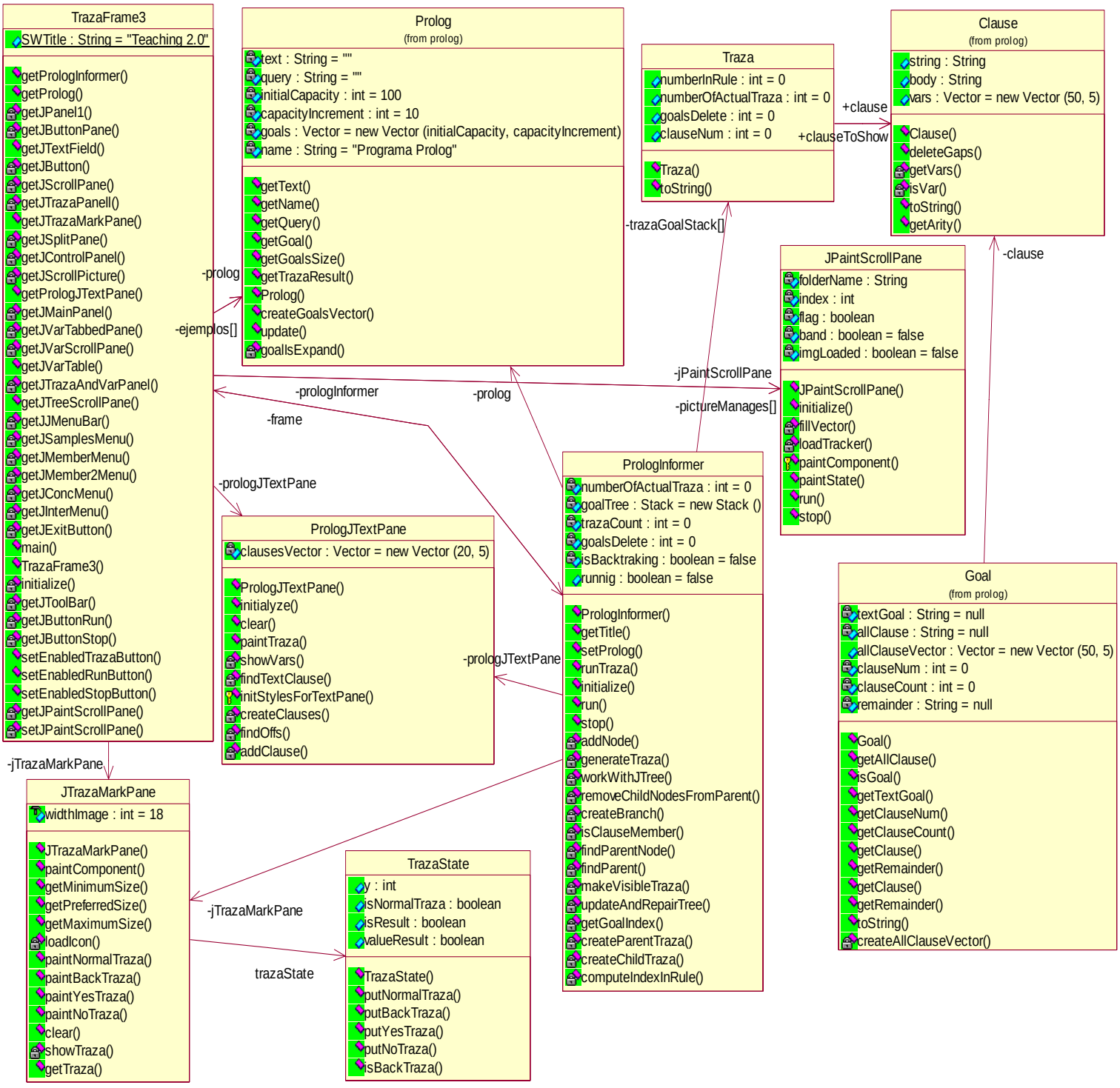

Figura 2.5. Diagrama de clases

Vista del sistema:

La figura 2.6 corresponde a la aplicación con los cuatro ejemplos integrados

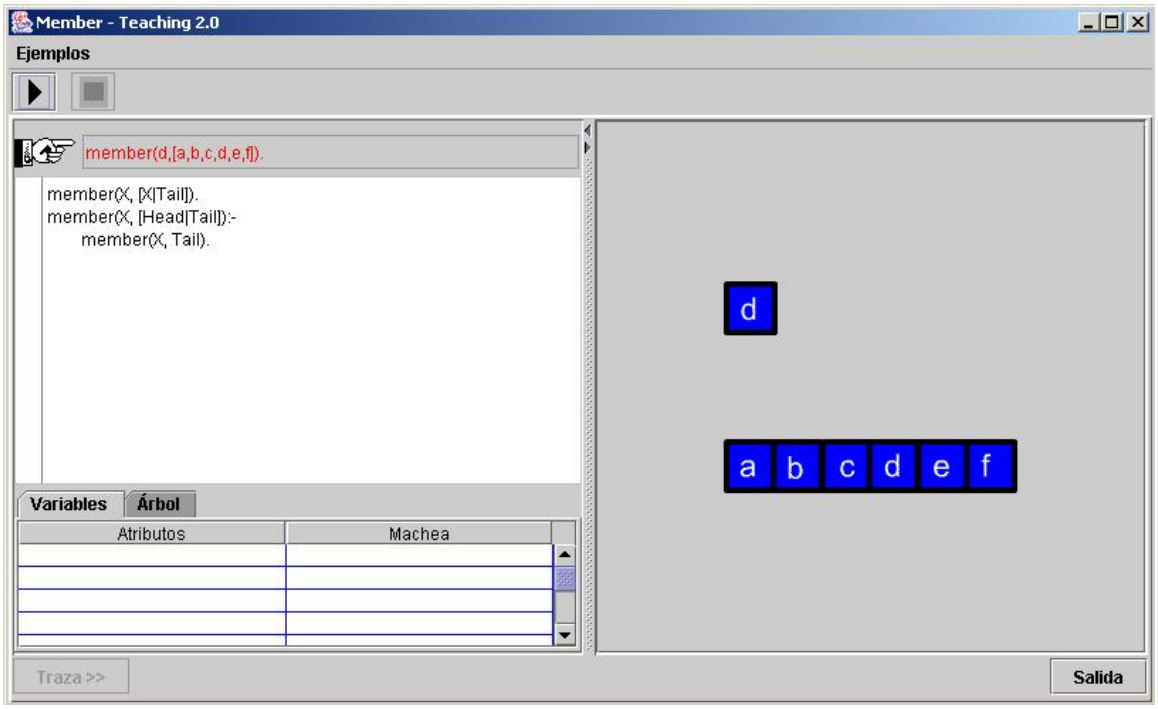

Figura 2.6. Vista del sistema

II.5 Resumen

En el capítulo se realizó un estudio de las técnicas que se pueden utilizar para construir una herramienta que cumpla con los objetivos expuestos y se describe con detalles la técnica empleada finalmente. Se expone un método general para simular nuevos ejemplos en el futuro y se analiza el diseño del sistema mostrando diagramas de casos de uso, de actividades y de clases.

El capítulo siguiente muestra la forma de trabajo del programador.

Capítulo III

CAPÍTULO III. GUÍA DEL PROGRAMADOR

III.1 Introducción

En este capítulo se brinda una información mas detallada de las clases (y sus métodos) principales que conforman el sistema.
También se presenta el proceso realizado para confeccionar la aplicación, con el objetivo de que otros programadores se familiaricen con el proyecto y puedan realizar labores de mantenimiento y actualización.
Los simuladores que se implementan, son la representación visual y animada de la traza generada por el "**WProlog**".

III.2 Los ejemplos implementados en Prolog

Para implementar los ejemplos se podría utilizar cualquiera de las implementaciones actuales que existen del lenguaje Prolog, se ha escogido el **Amzi Prolog** (Amzi! Inc, 2006) debido a que se usa en la carrera Ciencia de la Computación y, aunque no es un software libre, algunas de las versiones (no las últimas) se pueden bajar de Internet sin necesidad de pagar.
El Amzi, permite crear un archivo ***.pro*** con los predicados que se desean simular para probarlos a través de consultas, la forma de hacer lo anterior es la siguiente:

- Entrar a ***Listener*** y dar clic en ***Start*** para luego consultar el archivo creado. Después se puede escribir la consulta al programa y analizar la repuesta (figura 3.1).

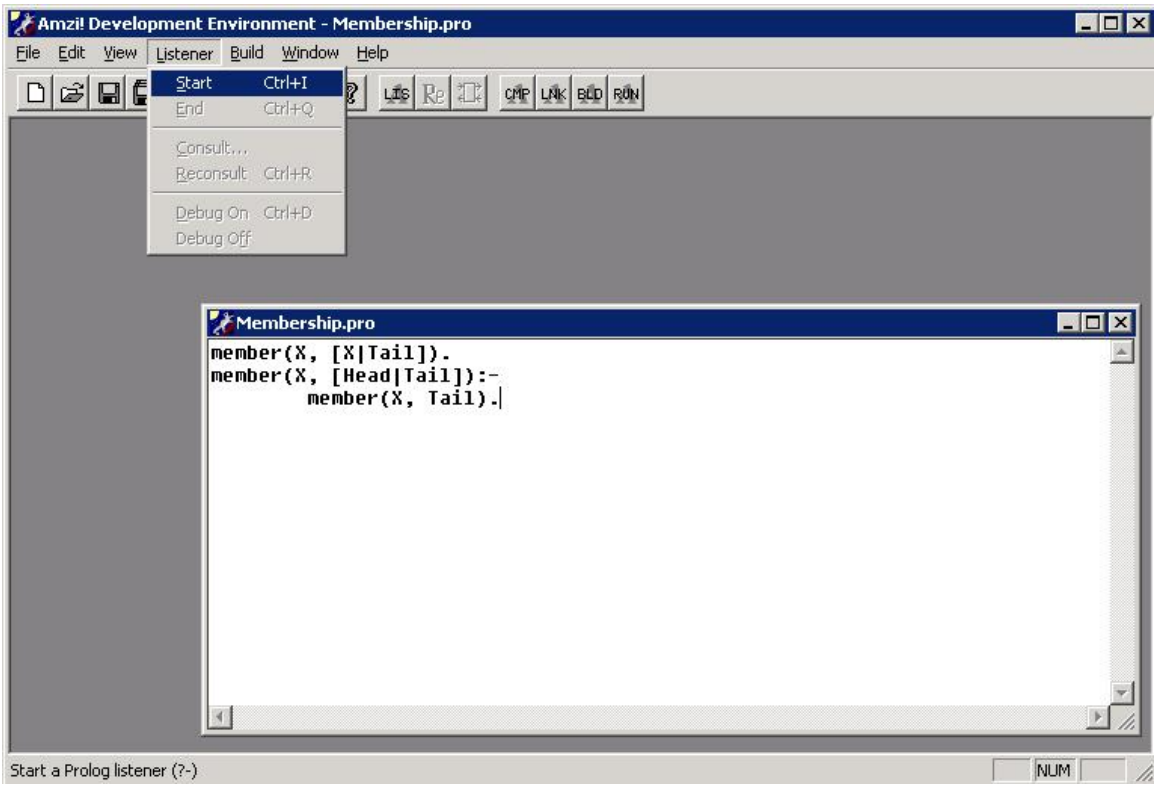

Figura 3.1. Ambiente Amzi

III.3 Las animaciones

Las animaciones que presenta el sistema se crean con **Flash** (Adobe, 2006) y son **.gif** animados o simplemente imágenes **.gif**.

Flash tiene una Ayuda muy poderosa, instructiva y fácil de comprender. Las lecciones muestran a través de ejemplos como crear aplicaciones para animar imágenes o para cualquier otra tarea de **Flash** resultando de gran utilidad para los principiantes y para los usuarios avanzados.

El ejemplo que resuelve el problema del miembro es sencillo y con él se mostrará cómo se crearon las imágenes y las animaciones para su simulación.

La primera imagen del ejemplo no es animada, ella muestra el elemento y la lista que conforman la consulta u objetivo. Para pintarlos se utiliza la paleta que tiene las herramientas de dibujo (figura 3.2)

Figura 3.2. Paleta de dibujo

Un elemento se dibuja con un rectángulo (***Rectangle Tool(R)*** en la paleta de dibujo) que forma parte de una lista y con el texto (***Text Tool(T)***) de su contenido en el centro.

Figura 3.3. Un elemento

Una lista es la unión de todos los rectángulos que son sus elementos.

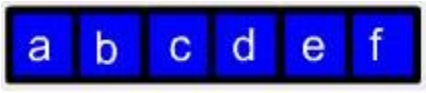

Figura 3.4. Una lista

Para hacer estos dibujos hay que crear un nuevo **layer** (capa) y realizar las modificaciones en su **frame** (marco) inicial, que es el único del **layer** y además es un **keyframe**.

La segunda imagen compara dos elementos (el elemento que se quiere conocer si es o no miembro de la lista y el elemento que es la cabeza de la lista en ese estado). Para ello es necesario insertar un texto entre ambos con el símbolo de la comparación y otro aparte con el resultado de la misma.

Figura 3.5. Comparación de dos términos

La tercera sí es una animación que muestra al elemento que "falló" en la comparación. Visualmente se muestra "volando", para que la lista se quede con los restantes elementos hasta la condición de parada.

Aquí hay que crear varios **layers**. Un **layer** es como una hoja de acetato transparente con una parte de la imagen, puestas una "encima" de la otra para mostrar la imagen final (téngase en cuenta que estas partes de imágenes también pueden ser animadas).

Los **frames** y **keyframes** de cada **layer** se muestran en la línea de tiempo (**Timeline**).

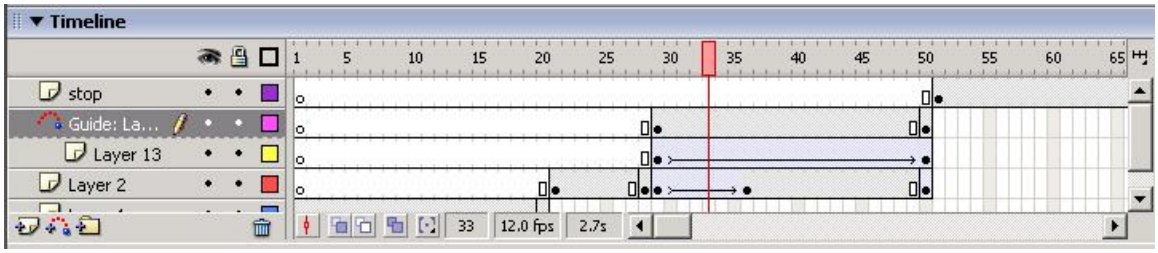

Figura 3.6. Línea de tiempo

Para clarificar la idea se hace referencia a la figura 3.6, aunque se puede hablar de forma general.

El primer **layer** es para mostrar durante un pequeño tiempo la imagen inicial del movimiento.

El segundo anima el corrimiento de la lista al perder el primer elemento. El **keyframe** en el **frame** 29 tiene la lista en su posición inicial pero sin el primer elemento y en el 36 la posición final después del corrimiento. Entre esos dos **frames** se crea una animación (**tweened animation**) representada por una flecha continua en el **Timeline**.

Para hacer una animación hay que seleccionar **Insert > Create Motion Tween**.

El tercer **layer** se encarga del elemento "que vuela". Para mover este elemento se utilizó una trayectoria o guía (**motion guide**), consiguiendo que todos los movimientos de los elementos fueran iguales y no tan simples entre un punto y otro. Esto se logra seleccionando el **layer** donde está el elemento y eligiendo **Insert > Motion Guide**. En el **Timeline** se selecciona un **frame** (el adecuado según el instante de tiempo) del **Guide:layer** y se inserta un **keyframe**: **Insert > Keyframe**. Usando el lápiz (**Pencil Tool (P)**) se dibuja una curva que representa el camino (esa línea o curva no debe ser visible). Posteriormente se selecciona un **frame** del **layer** del elemento y después de hacer clic en **Arrow tool** se selecciona el modificador **Snap** (); el **Snap** alinea los objetos unos con otros. Seguidamente se mueve el elemento al inicio de la curva y se selecciona **Insert > Create Motion Tween**, se escoge el **frame** final del movimiento y se coloca el elemento al final de la curva. Una flecha continua indica que no hubo error. En la figura 3.7 se muestra el resultado de este procedimiento en un momento determinado.

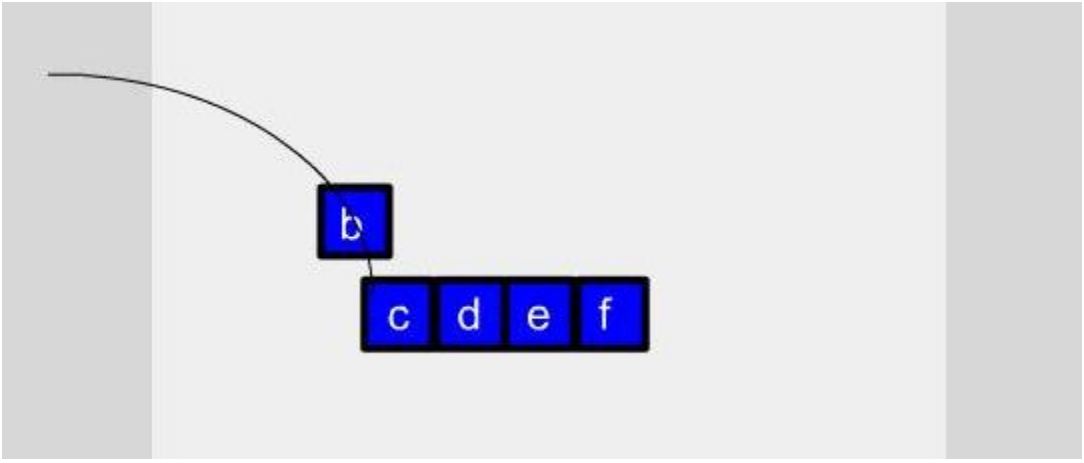

Figura 3.7. Elemento "volando"

El cuarto **layer** es para demorar la última imagen un tiempo, ya que los **.gif** animados nunca detienen su movimiento y se muestran cíclicamente.
Este procedimiento se emplea para todas las imágenes que quedan del ejemplo y un método muy parecido es seguido para cada una de las restantes animaciones.

III.4 Las clases del sistema

Existen tres grupos de clases: Las clases que representan la traza del "WProlog"; las clases visuales y las clases que manejan ambas.
Las clases que representan la traza del "WProlog" están contenidas en el paquete "prolog".
La traza está compuesta por una lista de "goal" y un resultado final "Yes" o "No". A continuación se muestra la traza del predicado miembro:

Goal: [member(d,[a,b,c,d,e,f])(2 clauses)] clausenum = 1
Goal: [member(d,[a,b,c,d,e,f])(2 clauses)] clausenum = 2
Goal: [member(d,[b,c,d,e,f])(2 clauses)] clausenum = 1
Goal: [member(d,[b,c,d,e,f])(2 clauses)] clausenum = 2
Goal: [member(d,[c,d,e,f])(2 clauses)] clausenum = 1
Goal: [member(d,[c,d,e,f])(2 clauses)] clausenum = 2
Goal: [member(d,[d,e,f])(2 clauses)] clausenum = 1
Yes: member(d,[a,b,c,d,e,f])

Un **goal** está formado por "cláusulas" e ilustra por donde va la traza. La primera cláusula del goal es la cláusula que está tratando de "machear". Al final nos dice cuantas cláusulas tienen este "arity" y el número que ocupa la cláusula actual.
El resultado en el caso de **"Yes"** está compuesto por "Yes" y el resultado de la **"query".** En el caso de **"No"** solo contiene "No".

III.4.1 Las Clases que interpretan la traza del WProlog

La Clase Prolog

Esta clase es la que contiene la "query", el resultado, y la lista de goals, como también el nombre que se va a mostrar en la barra de título cuando se esté ejecutando el ejemplo.

El Constructor de la misma es:

public Prolog(String name, String text, String query, String goalText).

Al constructor se le pasan todos estos parámetros, el parámetro **goalText** es un **String** donde los "goals" están separados por los cambios de líneas. El vector **goals** se crea con la función **createGoalsVector(goalText)**. También en esta función se obtiene el resultado.

La Función **getGoal(int index)** obtiene el objeto de tipo **Goal** que ocupa el índice **index**. Si el **index** está fuera de rango entonces devuelve **null**.

En la Función **getTrazaResult()** se devuelve el resultado de la pregunta "query".

La Clase Goal

Esta clase se construye a partir de un **String** que comienza con "Goal:". Tiene cláusulas que son representadas por la clase **Clause**, todas están almacenadas en un vector y en un **String**. Otro atributo es la cláusula que identifica al **Goal**, que es la que utiliza para tratar de unificar. Además, contiene la cantidad de cláusulas de este tipo escritas en el programa Prolog y el lugar que ocupa.

La Clase Clause

Al constructor de esta clase se le pasa como parámetro la cláusula en forma de **String**. Con la función **deleteGaps(String str)** se "borran" todos los espacios en blancos.

El atributo **string** es el parámetro con el cual se construyó la cláusula pero sin espacios en blancos. En el vector **vars** se encuentran todas las variables de esta cláusula.

Tiene una función que devuelve el "arity" de la cláusula.

La Clase TrazaResult

Esta es una clase abstracta que almacena el resultado de la traza. Al construir un objeto se le pasa el resultado en forma de **String,** este valor se almacena en El atributo text.

La Función abstracta **getValue()** devuelve si es afirmativo o no el resultado.

La Clase YesResult

Esta clase hereda de **TrazaResult** y es cuando el resultado es afirmativo.

La Función **getClauseResult()** devuelve el resultado de “query”.

La Clase NoResult

Esta clase hereda de TrazaResult.

La Clase Exception IsNoGoalStringException

Esta excepción es lanzada en el constructor de la clase **Goal**, y se lanza si el **String** que se le pasa no es un **Goal**.

III.4.2 Las clases visuales

La Clase TrazaFrame3

Esta clase es la que contiene todos los componentes visuales, es donde se crean y se inicializan.

La Clase JTrazaMarkPane

JTrazaMarkPane hereda de **JPanel**. Esta clase es la que dibuja la “manito” (☞) por donde va la traza a través de las funciones **paintNormalTraza(int y)**, **paintBackTraza(int y)**. El resultado se dibuja con **paintYesTraza()**, **paintNoTraza().** Para limpiar el panel se utiliza la función **clear()**.

La Clase PrologJTextPane

PrologJTextPane hereda de **JTextPane**. Esta clase es la que muestra el texto escrito en Prolog que se escribe con la función **initialyze(String t);** esta función busca todas las cláusulas contenidas en el texto.

La clase muestra las variables de la tabla y limpia los campos donde quedaron residuos. También devuelve las coordenadas donde se van a dibujar las trazas en el panel **JTrazaMarkPane.**

La Clase ClauseOnTextPane

Es creada en la clase **PrologJTextPane** y son las cláusulas contenidas dentro del texto. Contiene las coordenadas de la cláusula que ella representa y su comienzo y final. Esta clase está compuesta por cláusulas simples. Dentro de las cláusulas simples está la cláusula principal y las cláusulas secundarias. La cláusula principal es la que está antes de ":-" si es una regla o toda la cláusula si es un hecho. Las cláusulas secundarias son las que están después de ":-".

La Clase SimpleClause

De la siguiente cláusula:

```
inter([X|L1],L2,[X|L3]):-
    member(X,L2),
    inter(L1,L2,L3).
```

Las cláusulas simples son inter([X|L1],L2,[X|L3]), member(X,L2) y inter(L1,L2,L3).

Al constructor se le pasa la cláusula, donde comienza en el **PrologJTextPane**, donde termina y el objeto **JTextComponent** donde se encuentra.

La Clase Traza

Se usa para guardar el estado por donde va la traza, apunta a la cláusula en el texto, tiene los valores que se van a comparar y podemos ver si es o no backtracking.

Tiene un atributo **clause** y es la cláusula a buscar dentro del **PrologJTextPane**.

Por ejemplo, en la cláusula:

```
inter([X|L1],L2,[X|L3]):-
    member(X,L2),
    inter(L1,L2,L3).
```

El atributo **clause** tiene el "arity" inter/3 de (inter([X|L1],L2,[X|L3])).

El atributo **numberInRule** es el número que ocupa en la regla o en un hecho la cláusula que se quiere mostrar. Si la cláusula es un hecho este atributo es igual a cero.

Ejemplo:

```
inter([X|L1],L2,[X|L3])      numberInRule == 0
member(X,L2)                 numberInRule == 1
```

inter(L1,L2,L3) numberInRule == 2

El atributo **numberOfActualTraza** nos dice que goal fue quién generó esta traza.

El atributo **goalsDelete** es un entero que dice cuantos "goals" fueron borrados de la pila **goalTree**.

Si goalsDelete = = 0 quiere decir que es una nueva cláusula que es hija.

Si goalsDelete == 1 quiere decir que esta cláusula (la que lleva la traza) es la continuación de otra que se analizó y falló.

Si goalsDelete = = 1 quiere decir que hay backtraking.

El atributo **clauseNum** es un entero que dice el lugar que ocupa la cláusula contenida en El atributo **clause** con respecto a todas las cláusulas escritas en el **PrologJTextPane**.

El atributo **clauseToShow** solo es válido cuando **numberInRule > 0**, y es la cláusula que contiene las variables que se muestran en la tabla.

III.4.3 Las Clases Manejadoras

La Clase JPaintScrollPane

Esta clase hereda de **JPanel**. Es la encargada de mostrar las imágenes animadas en el panel de la derecha de la aplicación.

Tiene un arreglo **v** donde se van a cargar las imágenes. Al constructor se le pasa una cadena con la carpeta donde están y un entero con la cantidad de imágenes.

La función **initialize()** crea un objeto de tipo **MediaTracker** (Sun Site Tutorial) para cargar todas las imágenes en memoria y que no se demoren mostrándose. Obtiene la dimensión del panel y llama a los métodos **fillVector()** y **loadTracker()**.

La función **fillVector()** es la encargada de guardar las imágenes en el arreglo y de adicionarlas al objeto **tracker**.

En **loadTracker()** se espera porque cada imagen haya sido cargada en memoria.

La función **paintComponent(Graphics g)** chequea el estado del **tracker** y dibuja la imagen en el centro del panel. En **paintState()** se manda a pintar la imagen correspondiente.

En la función pública **stop()** se utiliza el método **flush()** de la clase **Image** para eliminar de la memoria el estado de cada imagen, logrando que las animaciones sean mostradas desde el principio cada vez.

La Clase PrologInformer

Esta clase es la que maneja y muestra la información de la traza en el **PrologJTextPane** (y este la tabla), la marca por donde va la traza y el árbol. Tiene cuatro funciones principales que se llaman en la clase **TrazaFrame3** las cuales son **initialize()**, que inicializa el objeto Prolog en los componentes visuales; **run()**, cuando comienza la ejecución; **stop()**, para la ejecución y **runTraza()**, que va mostrando paso a paso la traza.

El atributo **goalTree** es una pila que contiene todos los goal que hasta ese momento se están analizando, pueden ser: padres, ramas o hijos. O sea, aquellos goal que no tienen repercusión para el nodo actual no están aquí.

Ejemplo:

Si llegó a este goal(numberOfActualTraza ==2)

1. Goal: [member(d,[a,b,c,d,e,f])(2 clauses)] clausenum = 1
2. Goal: [member(d,[a,b,c,d,e,f])(2 clauses)] clausenum = 2

Se borra el goal (2).

El atributo **trazaGoalStack** es una pila que almacena las trazas que están por analizarse o se están analizando. En el se guardan las trazas que generan un goal. Esto se hace porque el goal solo dice la traza donde él termina y en ocasiones hay que encontrar de donde procedió.

Ejemplo:

Goal: [inter([a,e,b],[a,b,c,d],_0)(3 clauses)] clausenum = 2

Goal: [member(a,[a,b,c,d]), inter([e,b],[a,b,c,d],_5)(2 clauses)] clausenum = 1

En esta porción del vector de goals del objeto Prolog, la cláusula member/2 en (2) procede de la cláusula inter/3 en (1) y por tanto genera dos trazas:

private Traza[] trazaGoalStack = new Traza[2].

El atributo **trazaCount** es un entero que tiene tres valores.

trazaCount == 0 No hay traza que mostrar, generar nuevas trazas.

trazaCount == 1 Mostrar la traza trazaGoalStack[trazaCount--].

trazaCount == 2	Mostrar la traza trazaGoalStack[trazaCount--]. Esta traza es de donde procede la siguiente.

La Función **addNode** adiciona un nuevo nodo y lo muestra inmediatamente:
private void addNode(parentNode, childNode).

III.5 Resumen

En este último capítulo se ha realizado una guía para el programador con una breve descripción de todas las clases y métodos empleados en el sistema y de cómo construir las animaciones en Flash. El objetivo principal es poder realizar labores de mantenimiento y actualización.

Conclusiones

CONCLUSIONES

En el presente trabajo se ha presentado un método de trabajo que permite desarrollar simuladores de mecanismos propios del paradigma de la Programación Lógica con el fin de usar la enseñanza por descubrimientos como una vía eficaz para comprender los mecanismos internos del lenguaje.

Con ese fin se analizaron diferentes variantes, teniendo siempre en mente que las simulaciones debían mostrarse a través de la WEB.

Una vez establecido el método de trabajo se realizó un pequeño sistema con cuatro ejemplos que también pueden verse de manera independiente para ser insertados en los nodos de un mapa conceptual.

Recomendaciones

RECOMENDACIONES

Algunas de nuestras insatisfacciones se muestran en este apartado, ellas deben tomarse en cuenta para continuar y perfeccionar el trabajo presentado.

- Hacer un estudio de la productividad que se logra con el método presentado.
- Insertar los ejemplos presentados en el Mapa Conceptual para la enseñanza del Prolog que está realizando la Master Lydia Ríos de la Universidad de Sancti Spirítus y validar su verdadera efectividad en relación a la enseñanza.

Referencias Bibliográficas

REFERENCIAS BIBLIOGRÁFICAS

1. (2005) *The Java Tutorial* disponible en: http://java.sun.com/docs/books/tutorial/
2. (2006) "Aprendizaje por descubrimiento" en *El Rincón del Vago,* disponible en: http://html.rincondelvago.com/aprendizaje-por-descubrimiento.html
3. ADOBE, (2006) "Flash Professional 8" en *Macromedia – Flash,* disponible en: http://www.adobe.com/products/flash/flashpro/
4. ALMEIDA, S., FEBLES, J. P. y O. BOLAÑOS, (1997) *Evolución de la enseñanza asistida por computadoras*. Matanzas.
5. AMZI! INC, (2006) "Amzi! Prolog + Logia Server" en *Amzi!,* disponible en: http://www.amzi.com
6. BARRERA, M. I., (1999) *Informática Avanzada. Aplicación de Programación Lógica en Java*. Valdivia.
7. BARWISE, J., (1996) *Tarski's World e Hyperproof*. Stanford, California. CSLI Publications.
8. BRATKO, I., (1986) *Prolog Programing for Artificial Intelligence*. London. Addison-Wesley.
9. CABRERA, A., (1995) *Informática educativa: La revolución construccionista, Informática y Automática*. Vol. 28-1, pp. 24-31.
10. CHALJUB, J. A., (2006) La Estrategia para conducir la enseñanza de la Electrónica en la carrera de Ingeniería en Telecomunicaciones y Electrónica de la Facultad de Eléctrica, disponible en: http://bives.mes.edu.cu/cgi_bin/forum.exe?rec_id=022803&database=forum&search_type=link&table=mona&back_path=/forum/mona&lang=sp&format_name=sfmon
11. CHAMBERS, J. A. et al., (1983) *Computer-Assisted Instruction. Its Use in the classroom*. Ed. Prentice - Hall.
12. ESQUEMBRE, F., (2005) *Easy Java Simulations. The Manual. Version 3.4*, disponible en: http://www.um.es/fem/Ejs/Ejs_es/index.html

13. GARCÍA, Z., IGLESIA, M. y D. RONDA, (2006) Máquina de Inferencia para Sistemas de Enseñanza, disponible en: http://bives.mes.edu.cu/cgi_bin/forum.exe?rec_id=022803&database=forum&search_type=link&table=mona&back_path=/forum/mona&lang=sp&format_name=sfmon
14. KOWALSKI, R., (1986) *Lógica, Programación e Inteligencia Artificial*. Edición de Díaz de Santos.
15. LEZCANO, M., (1998) *Ambientes de aprendizaje por descubrimiento para la disciplina Inteligencia Artificial*. Santa Clara. Universidad Central "Marta Abreu" de Las Villas.
16. LEZCANO, M. et al., (2000) *Prolog y los sistemas expertos*. México. Universidad de Guadalajara. Centro Universitario de la Ciénaga.
17. PÉREZ, M. A., (1995) *Herramientas de software para la educación y el entretenimiento. Proyecto MLUDI GRAPHS*. Trabajo de Diploma. Santa Clara. Universidad Central "Marta Abreu" de Las Villas.
18. PERI, J. A. y D. L. GODOY, (1996) *Una experiencia del uso de Prolog en la resolución de problemas*. Argentina.
19. PIÑA, M., (2004) *Enseñanza asistida por computadora*. Universidad de Carabobo.
20. RODRÍGUEZ, A. E., (1996) *RIMI para Windows, herramienta RAR para software educativo*. Tesis de Maestría. Santa Clara. Universidad Central "Marta Abreu" de Las Villas.
21. RUIZ, F. et al., (1996) "Nuevas herramientas tecnológicas para la realización de cursos por computador". *Revista de Enseñanza y Tecnología*, No. 5, pp. 21-31.
22. SANTANA, A., (1998) *Software educativo sobre manifestaciones neurológicas en lesiones del tronco encefálico*. Tesis de Maestría. Santa Clara. Universidad Central "Marta Abreu" de Las Villas.
23. ULLOA, L., (2005) *Recursos didácticos para estimular mejores aprendizajes*. Universidad Pedagógica "José Martí" de Camagüey.