Centro de Investigación y estudios avanzados del Instituto Politécnico Nacional

Informe Final de Residencia Profesional

Realizado en el Laboratorio de ciencias computacionales con el proyecto Diseño e implementación de herramientas para la creación y edición de base de conocimientos para el módulo Editor Virtual el Proyecto GeDA-3D *(Generic Distributed Architecture for 3D applications).*

Ciclo:
2009ª

Periodo:
02 de Febrero de 2009 al 26 de Junio de 2009

Elaborado por
David Alejandro Gálvez Constantino
No Control: 03270318

Asesor o Responsable del proyecto
Dr. Félix Ramos Corchado

Auxiliar del proyecto:
M.C. Jaime Alberto Zaragoza Rios

Zapopan, Jalisco a 28 de Junio de 2009

Instituto Tecnológico de Tuxtla Gutiérrez.

Informe Final de Residencia Profesional

Realizado en el Laboratorio de ciencias computacionales con el proyecto Diseño e implementación de herramientas para la creación y edición de base de conocimientos para el módulo Editor Virtual el Proyecto GeDA-3D *(Generic Distributed Architecture for 3D applications).*

Ciclo:
2009ª

Periodo:
02 de Febrero de 2009 al 26 de Junio de 2009

Elaborado por
David Alejandro Gálvez Constantino
No Control: 03270318

Asesor Local del proyecto
Ing. Rigoberto Jiménez Jonapá.

Revisor del proyecto:
Ing. Raúl Moreno Rincón.

Tuxtla Gutiérrez, Chiapas a 2 de Julio de 2009

Índice

Capitulo 1

1.1 Introducción

En esta residencia se realizo una interfaz grafica para el uso del proyecto GeDA-3D *(Generic Distributed Architecture for 3D applications).* Para ello veremos el uso de la plataforma de programación en Java, así también como el uso de otologías.

Se dará explicación de cómo funciona el motor de GeDA-3D, y las aplicaciones que se encuentran para este proyecto que se encuentra en crecimiento y es muy prometedor en cuanto al uso de nuevas tecnologías y la forma en que se desarrollaran las computadoras con el ser humano.

No solo dando un paso gigantesco para esa barrera técnica que se tiene entre el lenguaje humano-maquina, si no también el desarrollo de nuevas tecnologías.

El proyecto GeDA-3D es una investigación que se desarrollo de los requisitos y las aplicaciones de un middleware. Un middleware es un software que conecta varios componentes para que entre ellos intercambien datos. Para que de ella se pudiera desarrollar una aplicación distribuida en 3D. Más que nada enfocadas a crear mundos virtuales donde se representen cosas descritas e interactúen entre ellas, llamando cosas, a las criaturas, objetos o entes que se encuentren en el mundo virtual.

En el mercado existen muchos "middleware" que realizan esta misma operación, pero es difícil poner en practica una aplicación distribuida con una interfaz en 3D, por ello el caso de desarrollar una interfaz para una interacción mas amigable con el usuario.

Veremos la evolución que se da en la Internet con las web semánticas, y el uso de ontologías, que dan un nuevo paso para la búsqueda de datos.

La interfaz grafica que se realizo para poder enlazar estas dos ramas, ontologías y el GeDA-3D, para un mayor aprovechamiento y comprensión para las nuevas generaciones que lo usen. Se manejo todo através de un lenguaje de programación multiplataforma JAVA, que es lo que esta revolucionando hoy en dia la programación ya que no depende de un Sistema Operativo en especifico, ya que con solo tener en el CPU la maquina virtual de Java, esta funciona sin mayores requerimientos.

Capitulo 2

2.1 Planteamiento del problema

Los programas para representación de ambientes virtuales, requieren de mucha especialización en el tema para desarrollar ambientes e interacciones naturales entre los objetos y entes dentro del mundo virtual.

Por lo mismo crear bases de datos, o en nuestro caso que usamos ontologías, resulta un poco complejo, ya que se deben tener valores múltiples para alimentar nuestro programa de manera amplia. El uso de la antología y el middleware al mismo tiempo puede ser un poco complejo o confuso por las ventanas y relaciones que esta se va generando. Es aquí donde entra GeDA-3D, intentando facilitar este proceso.

Se creara una interfaz de fácil acceso para la ontología donde se mostraran las clases que ya se tienen así como sus superclases y sus subclases, con los datos que contienen y a que grupo pertenecen, así como también la dirección que tienen. Todo esto en una interfaz diferente a la del editor de otologías y poder así utilizar fácilmente el GeDA-3Dajo

2.2 Justificación

Esta interfaz ayudara al usuario, a saber con que recursos cuenta la ontología accediendo a ella de manera fácil y rápida mostrando, las clases, los objetos e individuos contenidas en ella. Así también le dará la posibilidad, de agregar o eliminar datos de la ontología así como de modificarlos sin tener que entrar a un editor de ontologías (Protégé).

2.3 Objetivos

Los objetivos que se tienen para este proyecto son muchos, así como es la creación de ambientes virtuales con la descripción humana, también se pretenden hacer simuladores, ya sea de juegos o de programas, así también como dar hincapié, en el desarrollo de nuevas tecnologías para la enseñanza.

Este proyecto abre grandes puertas para muchas propuestas, ya que el entendimiento de la maquina podría crecer y darnos nuevas expectativas a las que tenemos ahora, ya que para nosotros es solo una herramienta de trabajo.

2.3.1 Objetivos Generales

Se debe crear una interfaz grafica para la ontología. Esta debe mostrar las clases, los objetos e individuos que engloba la ontología. Para ello se debe aplicar los conocimientos en programación orientada a objetos, en este caso bajo la plataforma de Java.

2.3.2 Objetivos Específicos

Crear la interfaz grafica, para visualizar los elementos en la ontología, en ella se deben presentar:

- La clase a la que pertenecen
- El objeto al que pertenecen
- El nombre que se les dio dentro del editor

Con ello se tendrá una visión mas amplia de los recursos que se podrán utilizar en el proyecto del GeDA-3D

3.3 Delimitación del problema

Se podría expander el programa, agregando o quitando clases, objetos o individuos, desde la misma interfaz, la posibilidad de presentar y editar los valores que cada uno tiene, así como también una imagen previa de cada individuo y las características en 3D que tendrá antes de realizar la escena.

Pero para ello se requiere más tiempo para analizar mas a fondo el API del editor de ontologías y conocimientos mas avanzados en cuanto a modelado en 3D. Por ello solo se pretende por el momento el presentar las clases, objetos e individuos contenidos en la ontología, así como la dirección específica que tienen dentro de la misma

Capitulo 3

3.1 ¿Qué es Java?

Como primera instancia daremos una breve explicación de la plataforma que se uso para iniciar el proyecto en este caso Java.

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Las aplicaciones Java están típicamente compiladas en un *bytecode*, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el *bytecode* es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del *bytecode* por un procesador Java también es posible.

La implementación original y de referencia del compilador, la máquina virtual y las bibliotecas de clases de Java fueron desarrollados por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre.

Entre noviembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java todavía no es software libre).

Como bien sabemos Java es un lenguaje en si orientado a objetos esto se refiere a un método de programación y diseño del lenguaje. Aunque hay muchas interpretaciones para OO, una primera idea es diseñar el software de forma que los distintos tipos de datos que usen estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el "comportamiento" (el código) y el "estado" (datos). El principio es separar aquello que cambia de las cosas que permanecen inalterables. Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema software. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software. Un objeto genérico "cliente", por ejemplo, debería en teoría tener el mismo conjunto de comportamiento en diferentes proyectos, sobre todo cuando estos coinciden en cierta medida, algo que suele suceder en las grandes organizaciones. En este sentido, los objetos podrían verse como piezas reutilizables que pueden emplearse en múltiples proyectos distintos, posibilitando así a la industria del software a construir proyectos de

envergadura empleando componentes ya existentes y de comprobada calidad; conduciendo esto finalmente a una reducción drástica del tiempo de desarrollo. Podemos usar como ejemplo de objeto el aluminio. Una vez definidos datos (peso, maleabilidad, etc.), y su "comportamiento" (soldar dos piezas, etc.), el objeto "aluminio" puede ser reutilizado en el campo de la construcción, del automóvil, de la aviación, etc.

La reutilización del software ha experimentado resultados dispares, encontrando dos dificultades principales: el diseño de objetos realmente genéricos es pobremente comprendido, y falta una metodología para la amplia comunicación de oportunidades de reutilización. Algunas comunidades de "código abierto" (open source) quieren ayudar en este problema dando medios a los desarrolladores para diseminar la información sobre el uso y versatilidad de objetos reutilizables y bibliotecas de objetos.

Otra de las características principales de java es su independencia de plataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware. Este es el significado de ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, '"write once, run everywhere"'.

Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como "bytecode" (específicamente Java bytecode)—instrucciones máquina simplificadas específicas de la plataforma Java. Esta pieza está "a medio camino" entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (JVM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Además, se suministran bibliotecas adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o threads, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (Just In Time).

Hay implementaciones del compilador de Java que convierten el código fuente directamente en código objeto nativo, como GCJ. Esto elimina la etapa intermedia donde se genera el bytecode, pero la salida de este tipo de compiladores sólo puede ejecutarse en un tipo de arquitectura.

La licencia sobre Java de Sun insiste que todas las implementaciones sean "compatibles". Esto dio lugar a una disputa legal entre Microsoft y Sun, cuando éste último alegó que la implementación de Microsoft no daba soporte a las interfaces RMI y JNI además de haber añadido características "dependientes" de su plataforma. Sun demandó a Microsoft y ganó por daños y perjuicios (unos 20 millones de dólares) así como una orden judicial forzando la acatación de la licencia de Sun. Como respuesta, Microsoft no ofrece Java con su versión de sistema operativo, y en recientes versiones de Windows, su navegador Internet Explorer no admite la ejecución de applets sin un conector (o plugin) aparte. Sin embargo, Sun y otras fuentes ofrecen versiones gratuitas para distintas versiones de Windows.

Las primeras implementaciones del lenguaje usaban una máquina virtual interpretada para conseguir la portabilidad. Sin embargo, el resultado eran programas que se ejecutaban comparativamente más lentos que aquellos escritos en C o C++. Esto hizo que Java se ganase una reputación de lento en rendimiento. Las implementaciones recientes de la JVM

dan lugar a programas que se ejecutan considerablemente más rápido que las versiones antiguas, empleando diversas técnicas, aunque sigue siendo mucho más lento que otros lenguajes.

La primera de estas técnicas es simplemente compilar directamente en código nativo como hacen los compiladores tradicionales, eliminando la etapa del bytecode. Esto da lugar a un gran rendimiento en la ejecución, pero tapa el camino a la portabilidad. Otra técnica, conocida como compilación JIT (Just In Time, o "'compilación al vuelo'"), convierte el bytecode a código nativo cuando se ejecuta la aplicación. Otras máquinas virtuales más sofisticadas usan una "'recompilación dinámica'" en la que la VM es capaz de analizar el comportamiento del programa en ejecución y recompila y optimiza las partes críticas. La recompilación dinámica puede lograr mayor grado de optimización que la compilación tradicional (o estática), ya que puede basar su trabajo en el conocimiento que de primera mano tiene sobre el entorno de ejecución y el conjunto de clases cargadas en memoria. La compilación JIT y la recompilación dinámica permiten a los programas Java aprovechar la velocidad de ejecución del código nativo sin por ello perder la ventaja de la portabilidad en ambos.

La portabilidad es técnicamente difícil de lograr, y el éxito de Java en ese campo ha sido dispar. Aunque es de hecho posible escribir programas para la plataforma Java que actúen de forma correcta en múltiples plataformas de distinta arquitectura, el gran número de estas con pequeños errores o inconsistencias llevan a que a veces se parodie el eslogan de Sun, "Write once, run anywhere" como "Write once, debug everywhere" (o "Escríbelo una vez, ejecútalo en cualquier parte" por "Escríbelo una vez, depúralo en todas partes")

El concepto de independencia de la plataforma de Java cuenta, sin embargo, con un gran éxito en las aplicaciones en el entorno del servidor, como los Servicios Web, los Servlets, los Java Beans, así como en sistemas empotrados basados en OSGi, usando entornos Java empotrados.

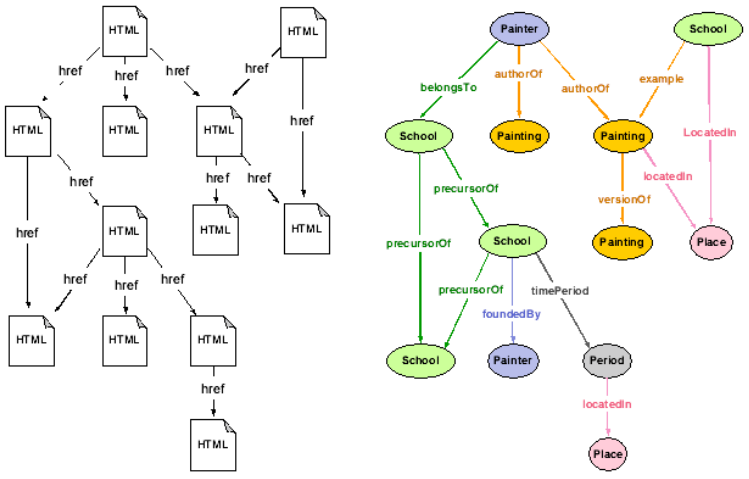
3.2 ¿Qué es una ontología?

En este proyecto se trabajara con ontologías para ello tendremos que ver primero que es y como funcionan. En primer lugar, daremos un pequeño resumen de lo que trata la ontología, para ello hablaremos de sus inicios que son las web semánticas, que son las nuevas aplicaciones que se le están dando a los buscadores web.

Por ejemplo, supongamos que buscamos Granada en Google. El buscador busca fielmente la cadena granada (sin tener en cuenta mayúsculas y minúsculas), y te devuelve páginas web sobre la Universidad de nuestros pecados, una compañía televisiva inglesa, la ciudad de Granada, la diputación de Granada. Si lo que buscabas era *"granada es una fruta"*, hay un poco de más suerte, se encuentra un artículo que la menciona, y nos aclara que se trata de una planta de la familia de las mirtáceas. Ahora bien ya sabemos que "*granada es una fruta del tipo de las mirtáceas*". Si queremos volver a buscar mirtáceas, a ver si hay algún otro fruto similar, pero que manche menos, tenemos que volver a introducir la palabra en el buscador, y así sucesivamente. Somos nosotros los que tenemos que reducir la búsqueda a algo manejable por un navegador, y ***comprender*** lo que leeremos para seguir encontrando lo que nos interesa.

La web semántica trata de hacer eso mucho más simple: trasladar parte de la comprensión de los datos en Internet al propio ordenador. Que el ordenador sea capaz de distinguir "*granada es una fruta*" de "*granada es una universidad*" de "*granada es una cosa que explotd*", y sea capaz de darte, en cualquier caso, lo más adecuado a la búsqueda. Aunque hablar de búsquedas de información, en realidad, es un poco restrictivo, porque la web semántica trata de recursos (que puede ser un documento o un servicio, o incluso un producto, tal como un libro).

La web semántica[1] (Berners-Lee 2001) propone superar las limitaciones de la web actual mediante la introducción de descripciones explícitas del significado, la estructura interna y la estructura global de los contenidos y servicios disponibles en la WWW(World Wive Web) . Frente a la semántica implícita, el crecimiento caótico de recursos, y la ausencia de una organización clara de la web actual, la web semántica aboga por clasificar, dotar de estructura y anotar los recursos con semántica explícita procesable por máquinas. La figura 2 ilustra esta propuesta. Actualmente la web se asemeja a un grafo formado por nodos del mismo tipo, y arcos (hiperenlaces) igualmente indiferenciados.



a) Web actual b) Web semántica

[1] http://www.semanticweb.org/, http://www.ontology.org/

Por ejemplo, que los pájaros son aves, que las aves son animales, que los animales son seres vivos, y todo eso que enseñan en la escuela. O que una universidad está dividida en departamentos, que los departamentos tienen un director; en fin, afirmaciones generales que se pueden aplicar a muchos dominios. A este grupo de afirmaciones se les llama ***ontologías***.

.

Pero claro, de lo que se trata es que todo ese proceso sea automático. Según se va escribiendo un texto, o a la hora de publicarlo, automáticamente se le añaden los metadatos pertinentes, y se le enganchan también las ontologías pertinentes. Ahora la pregunta es:¿Por qué querría alguien hacer eso?.

Pues en el caso de que se esté vendiendo algo, es esencial que se describa lo que se está vendiendo, para que motores automáticos de búsqueda lo encuentren, si es posible de forma relacionada.

3.1 Descripción del GeDA-3D.

Ahora bien ya se tiene un esquema mas amplio de lo que es una antología, o algo mas claro con lo que se esta trabajando en este proyecto, ahora bien mencionaremos, un poco mas en sí, de lo que se trata el proyecto de GeDA-3D *(Generic Distributed Architecture for 3D applications).*

Esta investigación es el desarrollo de los requisitos y la aplicación de un middleware, útil para desarrollar una aplicación distribuida que sirve de interfaz en 3D. Más que de la distribución virtual de aplicaciones, recrear un entorno virtual o mundo virtual. Para nosotros, un mundo virtual se entiende, como el conjunto de programas necesario para representar un mundo con todos los objetos, criaturas, avatares e interacciones etc. Por ejemplo, puede ser un mundo, todas las cosas necesarias en una planta de fabricación, es decir, los operadores de las máquinas, el material que vaya a transformarse, etc. y el tiempo. La motivación de nuestro trabajo es que hoy en día existen varios middleware útil para desarrollar aplicaciones distribuidas, algunas de ellas el apoyo interfaz gráfica muy desarrollada, pero incluso con estos "middleware" es muy difícil de poner en práctica una aplicación distribuida con una interfaz 3D.

El enfoque consistirá en ofrecer a los usuarios finales un conjunto de herramientas para desarrollar fácilmente las aplicaciones que objetivo. En esta investigación se propone un middleware orientado a agente útil para manejar la evolución de la interacción de tiro virtual de escenas de animación basada en el comportamiento. El middleware es la propuesta central de la serie de instrumentos que queremos ofrecer. Llamamos nuestro sistema GEDA-3D de Arquitectura Distribuida genéricos para aplicaciones 3D.

El núcleo de GeDA-3D es el principal responsable de:

a) El envío de agentes de la siguiente serie de objetivos de sus personajes tienen que cumplir y el local actual estado del mundo

b) La gestión de la interacción de las entidades virtuales de acuerdo con las leyes naturales del mundo en el que se basa.

c) Enviar un conjunto de comandos de nivel inferior a una herramienta de 3D, para hacer la escena.

Cada agente envía de vuelta a la base una secuencia de acciones para fomentar el cumplimiento de la menta actual.

El funcionamiento en forma general del GeDA-3D, es el siguiente:

1.- El guionista proporciona una descripción de la escena mediante limitaciones geométricas. Dicha descripción incluye:

a) La creación de entidades virtuales

b) La asignación de comportamientos de los personajes

c) La disposición de las entidades del mundo

d) Descripción de los objetivos que se cumplirán por los personajes.

Esta escena virtual se cumplirá y se traducirá en una descripción geométrica, solo si la descripción está libre de errores y es válida de acuerdo con una serie de normas.

2.- El mundo (contexto) envía al núcleo un conjunto de leyes naturales que normaran la interacción entre las entidades virtuales, e incluye descripciones de las acciones y entidades admisibles.

3.- La comunidad de agentes se encarga de representar a cada entidad descrita por el guionista.

El núcleo de GeDA-3D gestiona todos los contactos necesarios en los 3 pasos que se mencionan anteriormente. Pero su trabajo también incluye la gestión de la evolución de lo que pase en la escena, para ello, tiene a su cargo todas las entidades acerca de las interacciones entre medio ambiente, y comunicar la evolución del medio ambiente al hacer que los componente que muestran la evolución de la escena. El núcleo y el componente de renderizado se comunican por medio de un lenguaje de etiquetas como el XML con el fin de mostrar continuamente la evolución del medio ambiente.

Una aplicación interactiva permite al usuario tomar parte directamente de los acontecimientos que genera, simula o ejecuta. Así, el usuario es el responsable directo de los resultados del sistema.

Esta es una pequeña introducción del ambiente donde se estará trabajando y así poderles dar una visión mas amplia del trabajo

Capitulo 4

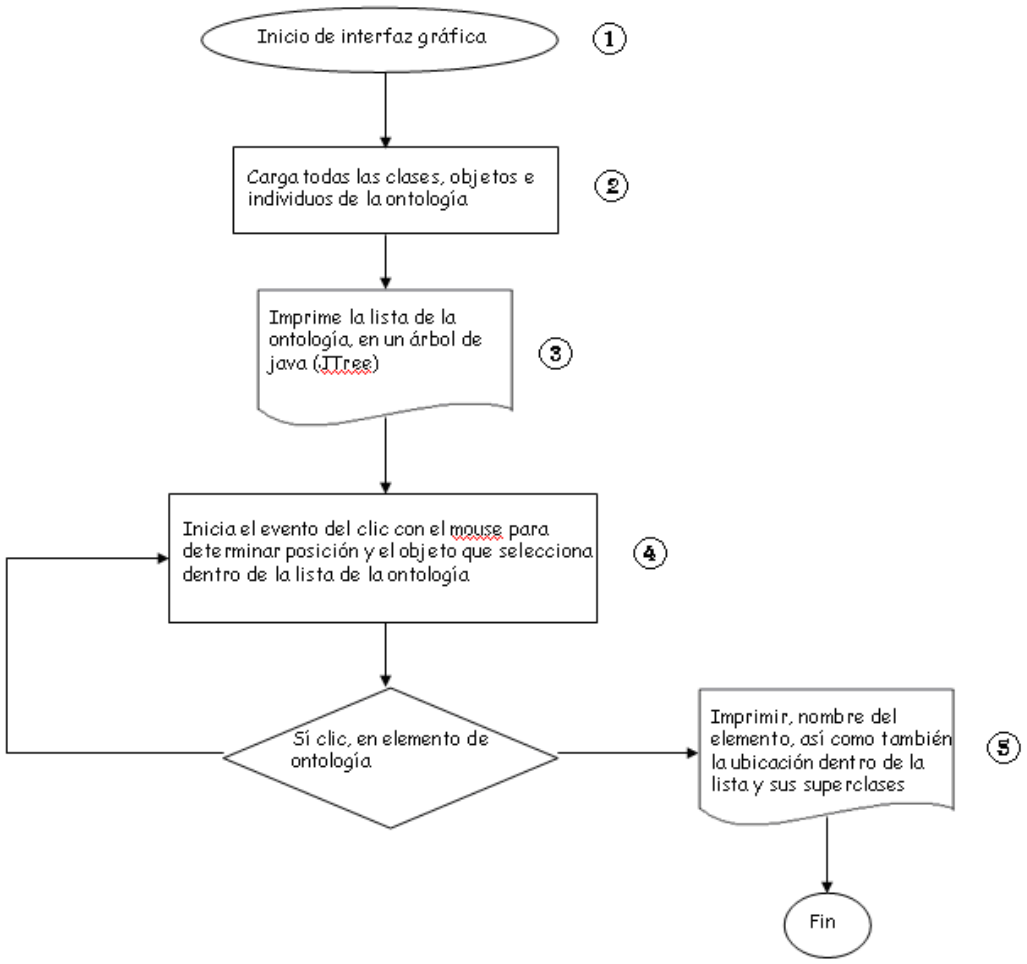
4.1 Desarrollo

Durante esta residencia, nos topamos con grandes problemas, con el desarrollo de la interfaz gráfica, ya que deberíamos tener un amplio conocimiento en el lenguaje java, así como también tener conocimientos en el idioma Ingles, para entender el manual de la API de la ontología.

Como primer problema nos topamos que al usar Eclipse como programa para este lenguaje, con su editor de gráfico de java Jigglo, nos presentaba la creación de elementos como botones, ventanas, cuadros de texto, etc. De manera muy fácil, pero la manipulación de estos no lo era tanto ya que como incluía el código de programación de manera automática no era posible manipular del todo este objeto, ya que presentaba grandes restricciones.

Por lo tanto tuve que empezar a programar la interfaz grafica desde el lenguaje puro de Java, ya que con ello la manipulación de estos objetos resultaba mas fácil para las acciones que quería realizar

A continuación presento el diagrama de flujo que debería requerir la interfaz una vez hecha, para las acciones a realizar.



- **1.-** Se inicia la interfaz grafica, cargando todas las clases de la paquetería de java, y crea la mascara de la interfaz gráfica y lo muestra en pantalla.

- **2.-** Se carga la ontología, desde el programa y se extraen superclases, clases y objetos de la ontología.

- **3.-** Los datos obtenidos de la carga de la ontología se presentan en forma visual al usuario por medio de la interfaz, en forma de un árbol.

- **4.-** El Mouse selecciona uno de los elementos del árbol y este rastrea el movimiento y se comprueba que se haya dado clic sobre algún elemento de la ontología.

- **5.-** Una vez obtenida la información de la posición del Mouse y los eventos clic que se realizaron con ellos, se guarda la dirección o posición del elemento checado dentro de la ontología y se presenta la dirección al usuario.

Capitulo 5

5.1 Código fuente

Lo que se obtuvo durante la estancia es una venta, donde nos muestra el enlace que se realiza con la ontología así como también las clases, objetos e individuos que contiene. A continuación dejo el código que se realizo para obtener esto:

```
//TODO: Paso 1 dentro del diagrama

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.File;
import java.net.MalformedURLException;
import java.net.URI;
import java.net.URL;
import java.util.Collection;
import java.util.Iterator;

import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.JTree;
import javax.swing.WindowConstants;
import javax.swing.SwingUtilities;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeModel;
import javax.swing.tree.TreePath;

import sun.java2d.pipe.OutlineTextRenderer;

import edu.stanford.smi.protegex.owl.ProtegeOWL;
import edu.stanford.smi.protegex.owl.jena.JenaOWLModel;
import edu.stanford.smi.protegex.owl.model.OWLIndividual;
import edu.stanford.smi.protegex.owl.model.OWLNamedClass;
import edu.stanford.smi.protegex.owl.model.RDFProperty;
import edu.stanford.smi.protegex.owl.model.RDFResource;
import edu.stanford.smi.protegex.owl.model.RDFSClass;
import edu.stanford.smi.protegex.owl.model.RDFSLiteral;
```

```
public class Ven_1 extends javax.swing.JFrame {
	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	JTextField jtf;
	private JTree jTree1;
	String prueba="algun valor";
	String temp_1;
	int contador=0;
	private static JenaOWLModel owlModel=null;
	public static void main(String[] args) {

//TODO: Paso 2 dentro del diagrama
//TODO:Cargado de la ontología

		File file =new File("C:/Baco/ontology.rdf-xml.owl");
		URI _turi=null;
		URL _url=null;
		try{
			_turi=file.toURI();
			_url=_turi.toURL();
			}
		catch(MalformedURLException _ei){
			System.out.println(_ei.getMessage());
			}
		String _uri=_url.toString();
		try{
			 owlModel=ProtegeOWL.createJenaOWLModelFromURI(_uri);
			}
		catch(Exception _e1){
			System.out.println("Error: "+ _e1.getMessage());
			System.out.println("No se encontro la ontologia");
		}

//TODO: Fin del paso 2

//TODO: Paso 1 Creación y presentación de los elementos
//TODO:Creación de ventana y sus componentes


		SwingUtilities.invokeLater(new Runnable() {
			public void run() {
				Ven_1 inst = new Ven_1();
				inst.setLocationRelativeTo(null);
				inst.setVisible(true);
			}
		});
	}
```

```
	public Ven_1() {
		super();
		initGUI();

	}

	private void initGUI() {
		try {
			BorderLayout thisLayout = new BorderLayout();

	setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
			getContentPane().setLayout(thisLayout);


			{
				jTree1 = new JTree();
				JScrollPane scrollpane1=new JScrollPane(jTree1);
				getContentPane().add(scrollpane1, BorderLayout.WEST);
				scrollpane1.setPreferredSize(new
java.awt.Dimension(150,320));


			}
			pack();
			setSize(600, 350);
		} catch (Exception e) {
			e.printStackTrace();
		}
//TODO: Paso 3 dentro del diagrama
//TODO: Creación del arbol y estructura base
		DefaultMutableTreeNode RaizOnt=new
		DefaultMutableTreeNode("Ontologia");
		DefaultTreeModel ArbolOnt = new DefaultTreeModel(RaizOnt);
		jTree1.setModel(ArbolOnt);


//TODO: Extraccion de las clases

Collection classes = owlModel.getUserDefinedOWLNamedClasses();

				DefaultMutableTreeNode Actions=new
DefaultMutableTreeNode("Actions");
				RaizOnt.add(Actions);

				DefaultMutableTreeNode Enviroment=new
DefaultMutableTreeNode("Enviroments");
			RaizOnt.add(Enviroment);
```

```
                        DefaultMutableTreeNode Keywords=new
DefaultMutableTreeNode("Keywords");
                        RaizOnt.add(Keywords);

                        DefaultMutableTreeNode Objects=new
DefaultMutableTreeNode("Objects");
                        RaizOnt.add(Objects);

                        DefaultMutableTreeNode Actor=new
DefaultMutableTreeNode("Actor");
                        RaizOnt.add(Actor);

// TODO:Población del árbol-------------------------

for (Iterator it_clas=classes.iterator(); it_clas.hasNext();){
        OWLNamedClass cls =(OWLNamedClass)it_clas.next();
        Collection instances =cls.getInstances(false);
        System.out.println("Clase "+cls.getBrowserText() + "("+instances.size()+")");
        Collection supcls=cls.getNamedSuperclasses(false);

        DefaultMutableTreeNode clase = new
DefaultMutableTreeNode(cls.getBrowserText());

        for (Iterator it_sr = supcls.iterator(); it_sr.hasNext();){
        OWLNamedClass superClass = (OWLNamedClass) it_sr.next();
System.out.println("Super Class: " + superClass.getBrowserText());

        for(Iterator it_ins=instances.iterator(); it_ins.hasNext();){
        OWLIndividual individual=(OWLIndividual)it_ins.next();
System.out.println(" - " + individual.getBrowserText());

        DefaultMutableTreeNode nuevo=new
DefaultMutableTreeNode(individual.getBrowserText());
        clase.add(nuevo);
                                                }

        if(superClass.getBrowserText().equals("Actor"))
                        {
                        Actor.add(clase);
                }
                else
                        {
        if(superClass.getBrowserText().equals("Objects"))
                        {
                                Objects.add(clase);
                        }
                                else
                                {
        if(superClass.getBrowserText().equals("Actions"))
                                {
```

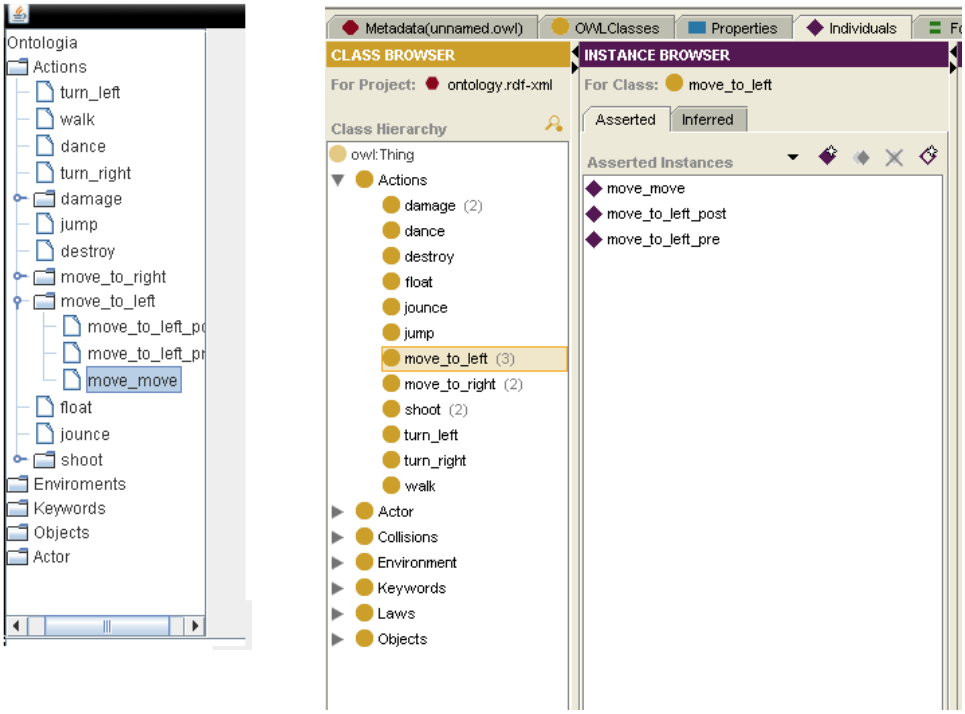
```
                                    Actions.add(clase);
                                    }
                                                        else
                                                        {
        if(superClass.getBrowserText().equals("Environment"))
                                                        {
                                                Enviroment.add(clase);
                                                }
                                                        else
                                                        {
                                                                Keywords.add(clase);
                                                        }
                                                        }
                                    }
                            }
System.out.println("Super "+cls.getBrowserText()+"("+supcls.size()+")" );

                                                }
                                    }


        jtf = new JTextField("",20);
        jtf.setPreferredSize(new java.awt.Dimension(500, 290));




//TODO: Paso 4 dentro del diagrama
//TODO: evento mouse---------------------------------
        jTree1.addMouseListener(new MouseListener(){public void
mouseEntered(java.awt.event.MouseEvent e){return;}
        public void mouseClicked(java.awt.event.MouseEvent e){return;}
        public void mouseExited(java.awt.event.MouseEvent e){return;}
        public void mouseReleased(java.awt.event.MouseEvent e){return;}
        public void mousePressed(java.awt.event.MouseEvent e){return;}});

        jTree1.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent me) {
              doMouseClicked(me);
            }
           });
         }

          private Object integer(int contador2) {
// TODO Auto-generated method stub
                return null;
        }

        void doMouseClicked(MouseEvent me) {
```

```
            TreePath tp = jTree1.getPathForLocation(me.getX(), me.getY());
            if (tp != null){

         //TODO: impresion del titulo clic
                  //String _subject = tp.toString();
                  String _pru=jTree1.getLastSelectedPathComponent().toString();
                  jtf.setText(_pru);

 }

//TODO: Paso 5 dentro del diagrama
// TODO: Impresion de la direccion del mouse----------
        {
                  jtf = new JTextField();
                  getContentPane().add(jtf, BorderLayout.SOUTH);
        }

}
//TODO:Fin del programa
```

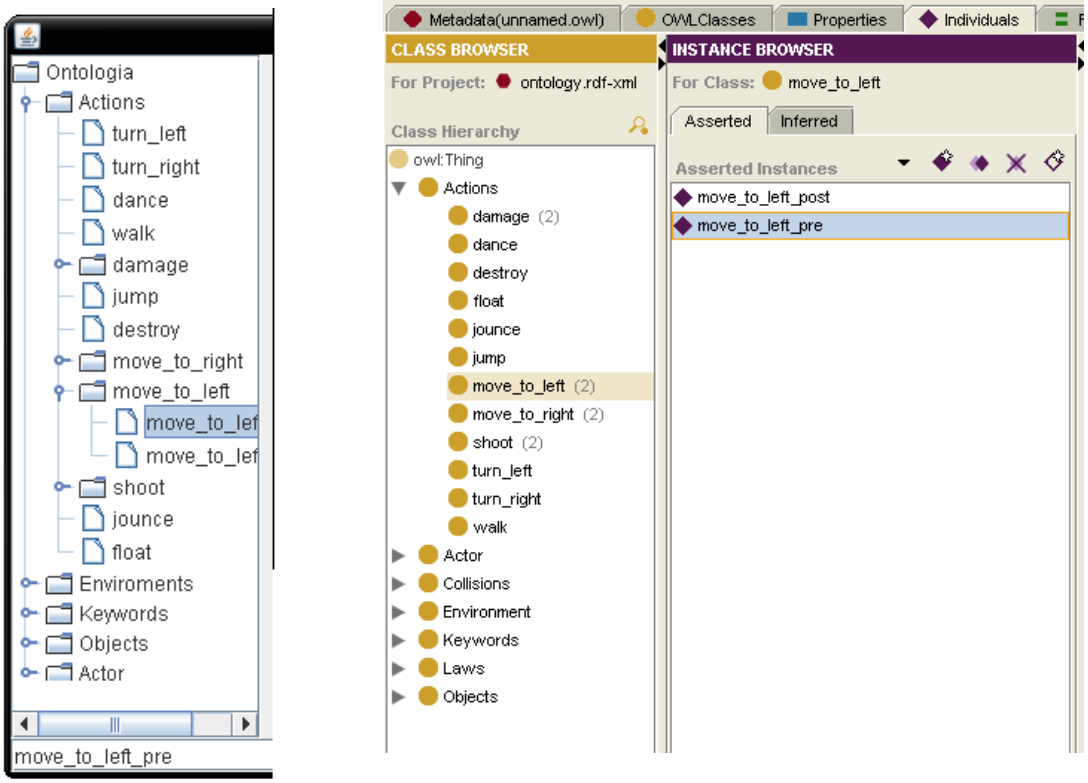
NOTA: Se anexaran imágenes del programa funcionando para mejor comprensión.

5.2 Resultados obtenidos.

Se obtuvo la ventana y la presentación grafica de los elementos en la ontología, por medio de la ventana. Aun no se consigue la obtención de las propiedades de las clases para presentar sus valores en dicha ventana



Aquí se puede ver en primer lugar la ventada de la interfaz mostrando los datos de la segunda ventana en forma de árbol captando todo los cambios que se realizan en protégé.



Capitulo 6

6.1. Análisis personal de la residencia profesional

A mi punto de vista es un proyecto que tiene mucho futuro en el campo de la computación, la comprensión de una computadora no como un objeto, si visto a futuro como un individuo, es un pequeño paso, para la interacción total entre el ser humano y maquina sin lenguajes complicados.

Dejo un buen sabor de boca este proyecto, ya que aprendí a realizar cosas que no tendía a entender, como en si aprender un poco de Java, aunque se tiene planeado seguir con el proyecto para conseguir los demás objetivos y tener una ventana que sea de ayuda, en el proyecto global del GeDA-3D

6.2 Conclusiones

Obtuve unos resultados un tanto satisfactorios ya que el proyecto requiere de mucha elaboración y trabajo para pulirlo mas. Pero en si es un resultado que puede abrir opciones en dicho futuro como el primer paso a realizar

6.3 Recomendaciones

Para futuros compañeros que sigan trabajando en este programa o realizar mas anexos a este, se requiere tener un conocimiento previo de lo que es la programación orientada a objetos. Para poder realizarlo de manera eficiente y rápida.

6.4 Trabajos a Futuro

Ya con una base planteada de lo que será la ventana grafica del proyecto, se pueden ir anexando mas elementos o funciones a esta como la creación de nuevas clases o individuos, así como también una previsualización grafica en 3D de los elementos, para que sea mas fácil, la elección de elementos que gobernara el ambiente.

Bibliografía.

- http://protege.stanford.edu/plugins/owl/api/guide.html
- http://www.eclipse.org/downloads/
- http://todojava.awardspace.com/manuales-java.html?nombre=eclipse/eclipse.pdf
- http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parte14/cap14-13.html
- http://www.chuidiang.com/
- http://java.sun.com/docs/books.html
- http://www.apl.jhu.edu
- http://www.magusoft.net
- http://eclipsetutorial.forge.os4os.org/in1

Referencias.

Jaime Alberto Zaragoza Rios. Representation and exploitation of knowledge for thedescription phase in declarative modeling of virtual environments. Master's thesis, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Unidad Guadalajara, Guadalajara, México, 2006.

P. Hugo, F. Z úniga, and F. Ramos, "A platform to design and run dynamic virtual environment," International Conference on Cyber Worlds, Tokyo Japan, pp. 18–20, November 2004.

H. R. Orozco Aguirre, J. A. Zaragoza Rios and D. Thalmann, Animation of Autonomous Avatars over the GeDA-3D Agent Architecture. Fourth National Week of Electronic Engineering (SENIE 2008). Panamerican University, Aguascalientes, Mexico, October 1-3, 2008.

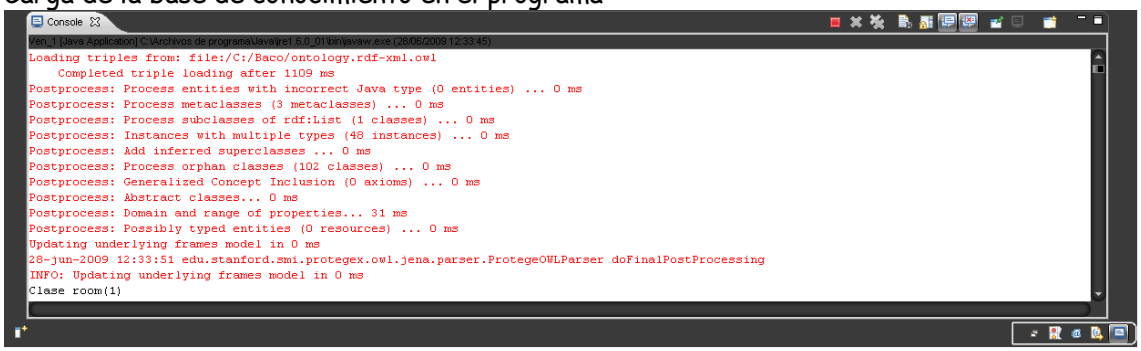
J. A. Zaragoza Rios, H. R. Orozco and V. Gaildrat, Modelado Declarativo de Ambientes Virtuales Basado en Explotación del Conocimiento. Cuarta Semana Nacional de Ingeniería Electrónica (SENIE 2008). Universidad Panamericana, Aguascalientes, México, 1-3 Octubre 2008.

Conference at the VRLab.
Jaime Zaragoza, Félix Ramos, Véronique Gaildrat Modeling of Virtual Environments Through Declarative Modeling Assisted by Knowledge, Workshop on Semantic User Descriptions and their influence on 3D graphics and VR. VRLab, EPFL, Lausanne, Switzerland, November 13, 2008.

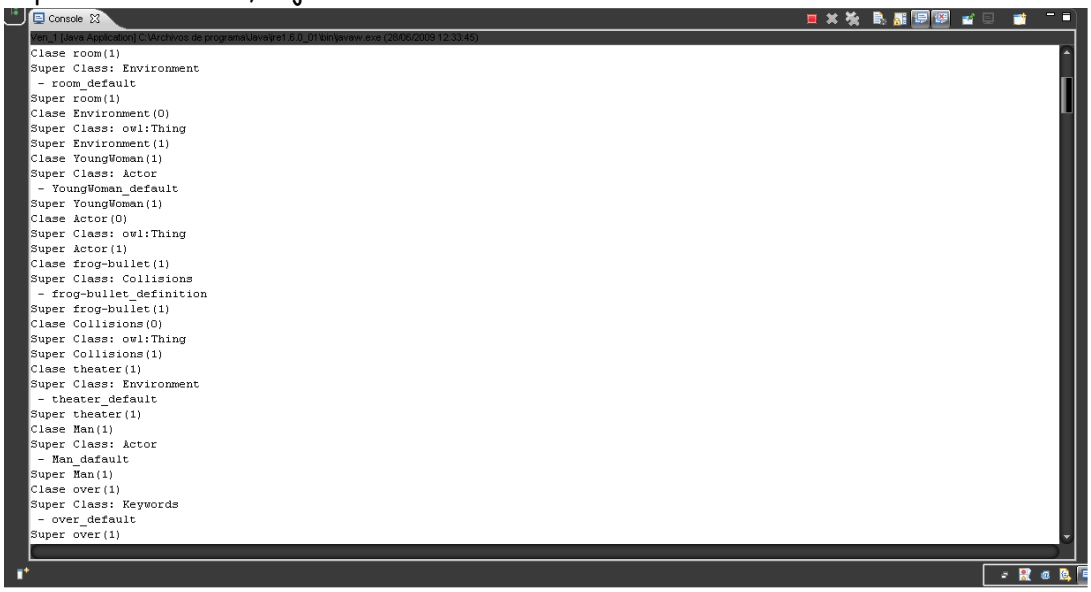
Book Chapters.
H. R. Orozco, F. Ramos, J. Zaragoza and D. Thalmann, Avatars Animation Using Reinforcement Learning in 3D Distributed Dynamic Virtual Environments. Frontiers in Artificial Intelligence and Applications (Advances in Technological Applications of Logical and Intelligent Systems). IOS Press, Washington, DC, Volume 186, 67-84, 2009.

Anexos

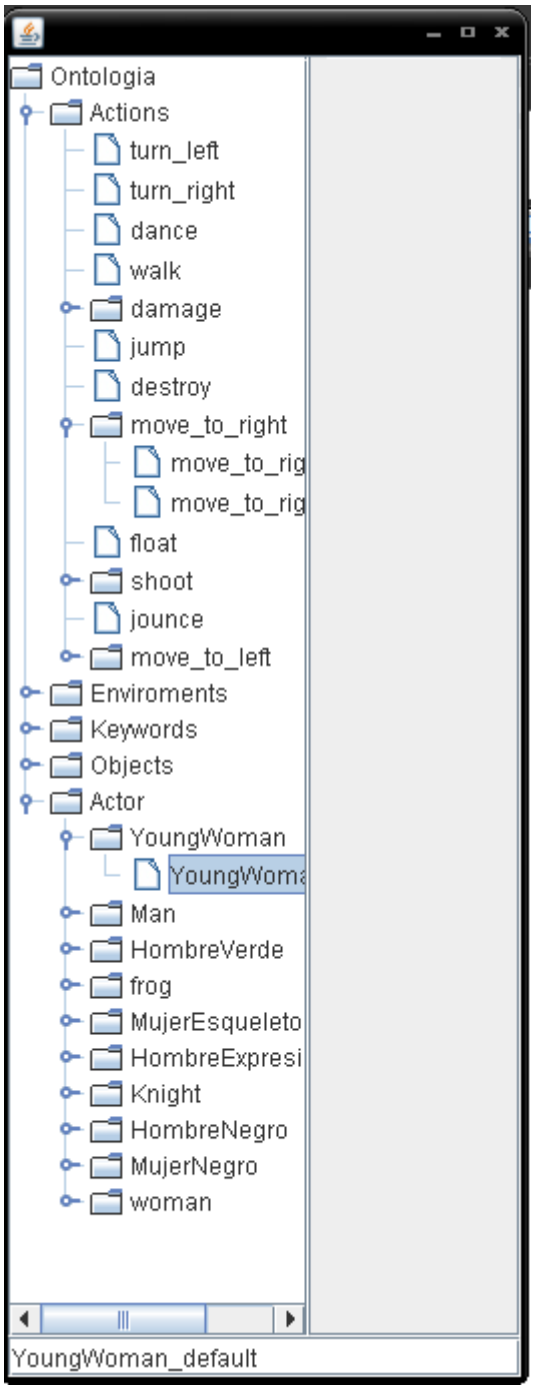
Carga de la base de conocimiento en el programa



Impresión de Clases, objetos e individuos en Consola



Presentación del árbol Con sus Clases, objetos e individuos



Presentación del editor de la Base de Conocimiento (Ontología) Protégé