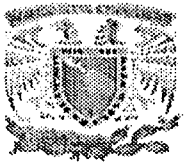

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE ESTUDIOS DE POSGRADO

TESIS

"AMBIENTES VIRTUALES PARA SIMULAR ROBOTS MÓVILES Y AGENTES"

PRESENTADA POR:

CARLOS DELGADO MATA

PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA
(ELÉCTRICA)

DIRIGIDA POR: DR. JESÚS SAVAGE CARMONA

México, D. F. 2003

Ambientes Virtuales para Simular Robots Móbiles y Agentes

Ing. Carlos Delgado Mata

5 de julio de 1999

2

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1 Antecedentes

En la actualidad la Realidad Virtual es aplicada en diversas áreas, como en la medicina, el aprendizaje, capacitación, entretenimiento. A medida que la tecnología se desarrolla, los costos se reducen y el equipo se vuelve accesible a más gente.

Los sistemas de Realidad Virtual nos ayudan a reducir riesgos, por ejemplo es mas fácil capacitar a una persona para manejará hierro fundido, con un sistema RV, en donde una equivocación no tendrá mayor consecuencia que volver a comenzar el programa de capacitación; así se evita que sufra un accidente desagradable durante la capacitación y reducir el riesgo de que se lastime durante el cumplimiento de sus labores posteriormente.

1.2 Problema a resolver

Se pretende simular robots y agentes en ambientes virtuales, inicialmente en una casa con varios cuartos; para lo cual se requiere de una herramienta capaz de crear ambientes virtuales.

También se busca que el sistema simule un robot que se encuentra actualmente en el Laboratorio de la Interfaces Inteligentes de la Facultad de Ingeniería , para lo cual se definió un protocolo de comunicación con los objetos movibles (agentes y robots) en el ambiente virtual, similar al que se

utiliza con el robot real.

El sistema debe ser de bajo costo, por lo que se sugiere que se utilice VRML y por consiguiente Java para el manejo del Robot Virtual.

1.3 Justificación y objetivos

1.3.1 Justificación

El proyecto presenta un reto interesante pues se pretende realizar una utilería para formar ambientes virtuales a partir de una descripción sencilla, como se utiliza en el Sistema Experto.

También es interesante señalar que utilizar una alternativa de bajo costo, es lo indicado debido a la situación económica del país; se pueden desarrollar proyectos similares con un presupuesto reducido, sin sacrificar la calidad en la simulación.

La comunicación entre procesos en diferentes lenguajes y posiblemente en diferentes plataformas es de actualidad, debido a la demanda de sistemas distribuidos.

1.3.2 Objetivos

- Ofrecer un sistema para crear archivos VRML 2.0 a partir de instrucciones sencillas.
- Lograr comunicación entre procesos en C++ y Java.
- Crear un protocolo de comunicación entre procesos de C++ y Java.
- Proveer de un ambiente virtual donde se pruebe los movimientos del Robot Virtual.
- Simular agentes en un ambiente virtual.

Capítulo 2

Ambientes Virtuales

2.1 Graficación por Computadora

Fue hasta principios de los 80s cuando las gráficas por computadora se volvieron accesibles, gracias a computadoras como Apple Macintosh y las IBM y compatibles popularizaron la utilización de graficas *bitmap*. Una vez que éstas estuvieron al alcance del público, existió un boom de aplicaciones gráficas de bajo costo. Los GUIs (Interfaces Gráficas de Usuario) permitieron a millones de nuevos usuarios controlar aplicaciones simples de bajo costo de una manera mas simple.

En la actualidad las graficas por computadora son en gran medida interactivas: El usuario controla el contenido, estructura y apariencia de los objetos y las imagenes desplegadas, donde se utilizan dispositivos como el teclado, el mouse, monitores sensibles al tacto, por mencionar algunos. Para complementar ver Foley [8]

2.2 Graficación en 3D

Las técnicas de graficación por computadora han permitido que se guarden objetos tridimensionales como descipciones geométricas, que se pueden convertir a una imagen especificando color, posición, escala y orientación en el espacio, y desde que posición se desea observar.

Algunos elementos para la creación de gráficas tridimensionales. Para

una explicación ver Ferraro [7] y Kerlow [12]

- Escena.
- Camaras.
- Polígonos.
- Iluminación.
- Materiales.
- Texturas.
- Bitmaps.
- Render.
- Objetos.

2.3 Modelado Geométrico

Los objetos tridimensionales se pueden representar por medio de polígonos, que es lo mas común. En la actualidad un chip para el manejo de gráficas es capaz de procesar millones de polígonos por segundo, es frecuente que se traten de reprentar los objetos por medio de triangulos, el polígono mas pequeño pues no importara si movemos un vértice de este, el polígono siempre será plano, cosa que no necesariamente ocurre en los polígonos de mas de tres vértices.

Los objetos que poseen superficies curvas son representadas por una serie de polígonos, los cuales pueden ser varidos en tamaño y por consiguiente en número para obtener una aproximación mas cercana a la realidad.

Para una excelente explicación revisar a Watt [23].

2.4 Ambientes Virtuales

Los simuladores de vuelo se puede considerar como un antecedente importante en los Ambientes Virtuales.

"... la vida de un simulador de vuelo puede extenderse por 20 años, por lo que un simulador de vuelo construido en 1997 puede seguir en operación para el año 2017. Si los simuladores existirán en 50 años es irrelevante. Un simulador, ya sea de avión, tanque o barco se construye sobre un modelo específico. Si los aviones, barcos y tanques del futuro son de piloto automático, la demanda para los simuladores de hoy desaparecerá junto con pilotos y capitanes.

Los ambientes virtuales también se utilizan en otras aplicaciones, como medicina, modelado molecular, ingeniería, arquitectura, visualización científica, entretenimiento y televisión. Estas son aplicaciones reales y serán un punto importante en el futuro.

Los ambientes virtuales se seguirán utilizando en todos los tipos de actividades de entrenamiento; ya sea para cirujanos, soldados, oficiales de policía, bomberos, niños o astrónomos. Los cirujanos serán capaces de practicar en tejidos y cuerpos virtuales, y desarrollarán procedimientos sin derramar una sola gota de sangre o causar dolor.

El entrenamiento solamente es una aplicación de los ambientes virtuales. Pero entre las aplicaciones de ingeniería que se desarrollaran en el futuro están los ambientes para la simulación física. los que serán importantes en todo sentido para el diseño y la educación.

Y como se puede apreciar en las investigaciones que se realizan en Reino Unido, la Universidad de Utah y la de Carolina del Norte, principalmente; podemos esperar un futuro con cooperación en ambientes virtuales distribuidos, y en algunos problemas que se requiera el empleo de Robots para Telepresencia." Vince [21]

2.5 Interactividad en Ambientes Virtuales : VRML 2.0

VRML de sus siglas en ingles tiene como significado: *Lenguaje de Modelado de Realidad Virtual.*

VRML se puede dividir en dos partes:

1. *Descripción geométrica* sirve para describir los objetos y sus atributos, por ejemplo, posición y orientación en el espacio, mapa de texturas, color, etc.

2. *Comportamiento.* Existen dos tipos de comportamiento, uno sencillo, en donde sólo se ligan los nodos para obtener comportamientos sencillos. El otro es por medio del nodo script programar comportamientos mas complejos en algún lenguaje de programación, hasta la fecha el que se ha utilizado es Java pues tiene la bondad de ser independiente de plataforma.

Para una itroducción completa a VRML y su combinación con Java recomiendo a *Lea* [13].

2.6 Redes de Computadoras

Las redes de computadoras han sido utilizadas en prácticamente todos los niveles, desde la conexión de una pequeña red local con unas cuantas computadoras hasta lo que se conoce como internet que es la conexión de millones de ellas.

Las redes de computadoras han sido de gran utilidad pues se comparten los recursos caros como son impresoras laser a color, plotters, estaciones de trabajo de altísima capacidad gráfica; pero al mismo tiempo se pueden realizar operaciones locales en la computadora personal que no requiera de un equipo muy sofisticado por lo que se obtiene lo mejor de los dos mundos.

La red de computadoras también se ha utilizado para mandar y recibir mensajes, se ahorra papelería y tiempo.

2.7 Internet

Internet es la red donde se conectan todas las redes, y mediante la cual es posible comunicarse con colegas, amigos y familiares que se encuentren en lugares distantes sin la necesidad de esperar dias a que llegue una carta o sin pagar grandes cuentas de teléfono.

Gracias a Internet es posible obtener información sobre algún tema deseado. Organizar equipos de trabajo con miembros que se encuentren en diferentes ciudades o paises inclusive.

2.8 Sockets

Una manera eficiente de comunicar procesos es por medio de sockets, para la comunicación por sockets es necesario definir por que puerto nos queremos comunicar. Los puertos son como una especia de línea telefónica. En una computadora estos puertos son disponibles en cualquier computadora basada en red TCP/IP.

Existen dos tipos de sockets:

- TCP *Transmission Control Protocol*, es un servicio orientado a conexión, es análogo al teléfono.
- UDP *Unreliable Datagram Protocol*, es un servicio sin conexión, es análogo a un buzón de correo.

Para complementar ver [20] para una explicación mas extensa sobre la comunicación entre procesos utilizando sockets desde y para programas desarrollados en Java.

Capítulo 3

Agentes Inteligentes

3.1 Agente

La pregunta ¿ Qué es un agente ? es un tanto complicada; es difícil dar una respuesta que sea aceptada universalmente.

Para algunos investigadores –principalmente aquellos que trabajan en Inteligencia Artificial– el término se conceptualiza utilizando conceptos que generalmente se le atribuyen a los humanos.

Según Wooldridge [25] el término de agente utilizado en un sistema computacional tiene las siguientes propiedades: *autonomía, habilidad social, reactividad y proactividad.*

El concepto de Agente se ha vuelto de suma importancia en tanto Inteligencia Artificial, como en las ciencias de la computación. Noción de Agentes, descrita por Wooldridge [25]., Un agente tiene las siguientes propiedades:

Autonomía : Los agentes operan sin la intervención directa de humanos u otros y tienen algún tipo de control sobre sus acciones y de su estado interno.

Capacidad Social : Los agentes interactúan con otros agentes (y probablemente humanos) mediante algún tipo de lenguaje de comunicación entre agentes.

Reactividad : Los agentes perciben su ambiente, (el cual puede ser su mundo físico, un usuario vía una interfaz de usuario gráfica, una colec-

ción de otros agentes, Internet, o tal vez la combinación de estos), y responden de manera rápida a los cambios que ocurren en él.

Proactividad : Los agentes no actúan solamente en respuesta a su ambiente, son capaces de exhibir un comportamiento dirigido a metas, tomando la iniciativa.

Con respecto al desarrollo de los agentes: cito a Russell [16] "El problema del diseño de agentes depende de las percepciones y acciones de las que dispone el agente, de las metas que debe satisfacer y de la naturaleza del ambiente mismo. Existen diversos diseños de agente, que van desde los agentes reflejo hasta los que son capaces de una total deliberación, agentes basados en el conocimiento".

Un agente es algo capaz de percibir y actuar ver figura 3.1. Por lo tanto se requiere de respresentar el conocimiento para poder razonar en base a él. Una herramienta utilizada realizar la representación del conocimiento es la plantilla de papel temático.

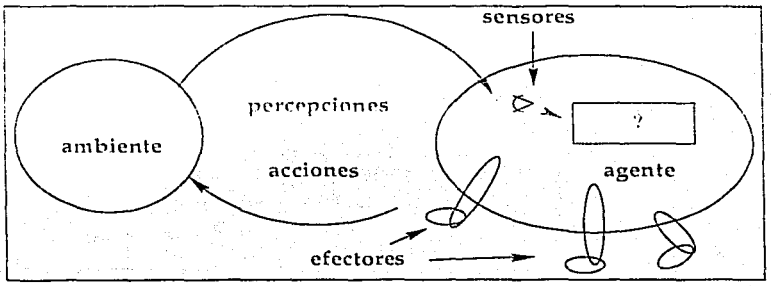

Figura 3.1: Los agentes interactúan a través de sensores y efectores

Daniel Ballin [1] ve la topología de los agentes como Agentes Cooperativos, que aprenden y Agentes Inteligentes en donde todos son Autónomos y Proactivos. Ver figura 3.2

3.1.1 Papel Temático

Como afirma Winston [24] "Mucho de lo que sucede en el mundo implica acciones y objetos que sufren cambios. Por lo tanto es natural que muchas

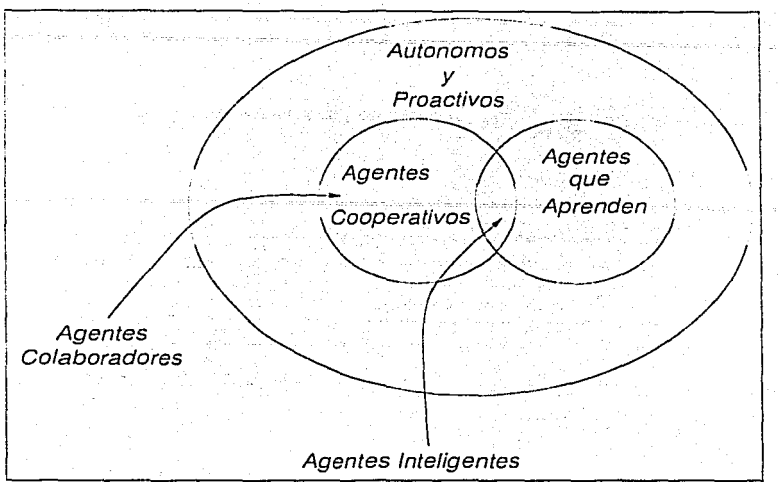

Figura 3.2: Topología de agentes según Ballin

oraciones del lenguaje humano sean descripciones que especifican acciones, identifiquen al objeto que experimenta un cambio e indiquen qué otros objetos intervienen implicados en tal cambio"

En términos lingüísticos los verbos son acciones y las frases sustantivas son objetos que participan en la acción.

Ejemplo: Carlos patea un balón con el tachón.

- Carlos es el agente.
- Balón es el objeto temático
- Tachón es el instrumento.

El número de plantillas de papeles temáticos varía dependiendo de la aplicación. Los valores de descriptor proporcionan respuesta a diferentes preguntas sobre lo que sucedió. En la figura 3.3 se representa un papel temático lleno, en donde se responden preguntas sobre acciones que realiza un agente y que beneficia a otro con un objeto utilizando un instrumento.

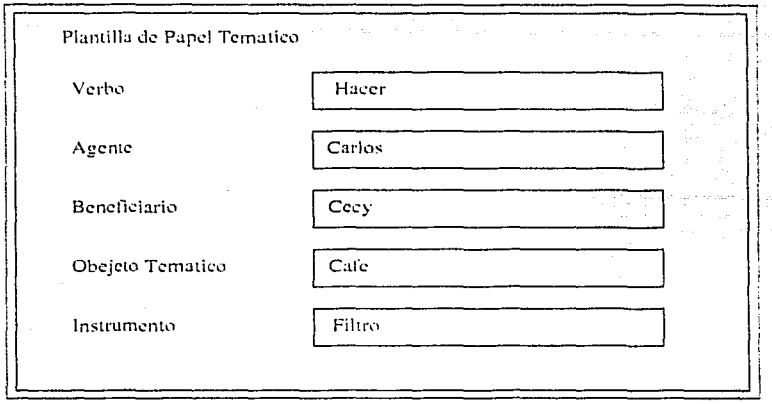

Figura 3.3: Papel temático con beneficiario

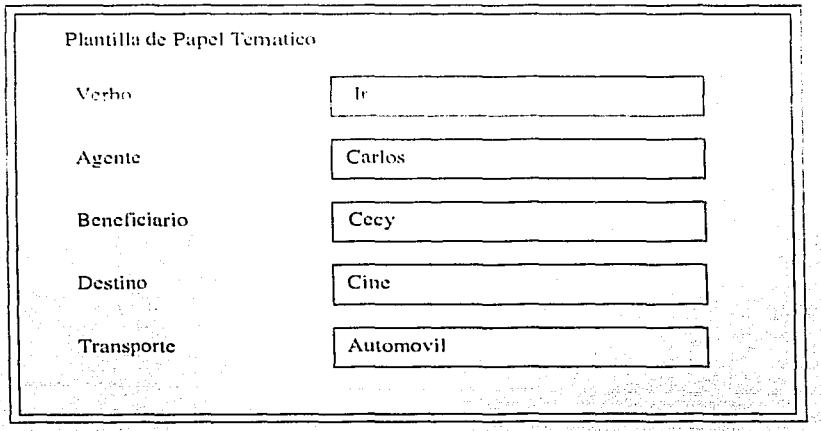

Figura 3.4: Papel temático para transporte

Con una oración como la siguiente *Carlos fue al cine con Cecy, en automóvil.* Como se muestra en la figura 3.4 vemos que tenemos 4 frases

sustantivas, cada una de las cuales se ajusta a un papel en particular.

¿ Quién fue ? agente Carlos ¿ Con quién fue ? coagente Cecy ¿ Adónde fue ? destino al cine ¿ En qué se fue ? trasporte automóvil

Winston afirma que "Los papeles temáticos corresponden de manera aproximada a algunas preguntas simples sobre las acciones.

Los valores de ranura se deben conseguir mediante un programa que entienda lenguaje, antes de que pueda ser utilizado para analizar las preguntas.

3.1.2 Clasificación de Agentes

Los agentes se pueden clasificar útilmente de acuerdo a sus propiedades. Cada agente debe satisfacer las condiciones de reactividad, autonomía, orientada a meta, y continuidad temporal. Al aadir otras propiedades produce otras clases, por ejemplo agentes que aprenden y agentes móviles.

Existen otros esquemas de clasificación. Por ejemplo se pueden clasificar agentes de software de acuerdo a las tareas que realizan. por ejemplo agentes recolectores de información o agentes de filtrado de correos electrónicos.

La taxonomía de agentes de software de Brustoloni [3] comienza un una clasificación de agentes de regulación, agentes de planeación o agentes adaptables. Un agente de regulación que se llama para la regulación de temperatura, mediante untermostato, reacciona a cada entrada sensorial que recibe, y siempre sabe que hacer. No planea ni aprende. Los agentes de planeación son los solucionadores de problemas, basados en casos o basados en métodos de investigación de operaciones o agentes que utilizan algoritmos aleatorios. Los agentes adaptables además de planear, aprenden.

Franklin [9] propone una taxonomía que toma forma de árbol similar a la utilizada en el modelo biológico. Por ejemplo los humanos se clasifican:

reino - animal

phylum - vertebrado

clase - mamífero

orden - primate

familia - pongidae

subfamilia - hominidae

género - homo

especie - sapiens

Para clásificar los agentes autónomos, Franklin fra en el nivel de reino, pone agentes biologicos, agentes roboticos y agentes computacionales, ver figura 3.5

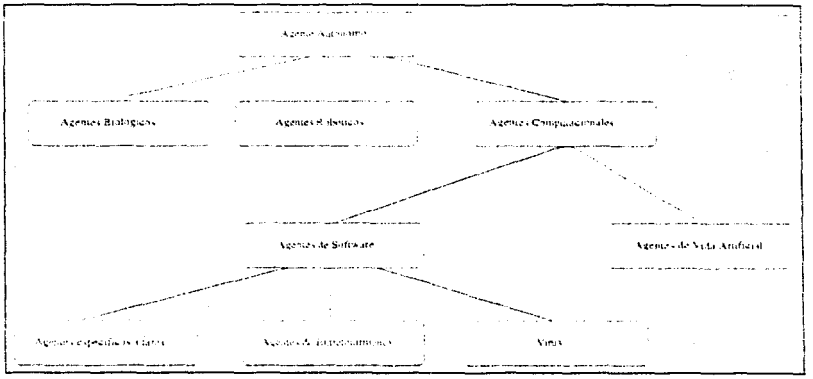

Figura 3.5: Taxonomía de Agentes utilizando modelo biológico

3.2 Inteligencia Artificial

Una definición aceptada es que IA es el arte de crear máquinas con capacidad de realizar funciones que realizadas por personas requieren de inteligencia. Según Russell [16], " el estudio de cómo lograr que las computadoras realicen tareas que, por el momento, los humanos hacen mejor." Las definiciones anteriores parten de la base que las computadoras actúen como lo harían los humanos. Las definiciones de Inteligencia Artificial se pueden agrupar en cuatro categorías.

1. Sistemas que piensan como humanos.

2. Sistemas que actúan como humanos.
3. Sistemas que piensan racionalmente.
4. Sistemas que actúan racionalmente.

Se considera que un sistema es racional si hace lo correcto. Entre los enfoques considerados anteriormente, el enfoque centrado en el comportamiento humano constituye una ciencia empírica. El enfoque racionalista combina las matemáticas y la ingeniería.

3.3 Agentes Inteligentes

Según Russell [16] Un agente racional es aquel que hace lo correcto. Obviamente, esto es preferible a que haga algo incorrecto, pero ¿ Que significa? Como un primer intento de aproximación, se afirmará que los correcto es aquello que permite al agente obtener el mejor desempeo." Existen diferentes maneras de medir el desempeño de un agente, una noción generalmente aceptada es que realice eficazmente lo que se le pide.

Para definir Agente Inteligente ideal me apoyo en Russell. " En todos los casos de posibles secuencias de percepciones, un agente racional deberá emprender todas aquellas acciones que favorezcan obtener el máximo de su medida rendimiento, basándose en evidencias aportadas por la secuencia de percepciones y en todo conocimiento incorporado en tal agente". Una parte importante de la racionalidad es emprender acciones con el fin de obtener información útil.

Ya que el comportamiento depende de un agente depende de la secuencia de percepciones en un momento dado, generalmente se realiza una tabla de acciones que éste emprende como respuesta a cualquier secuencia de percepciones posible. Según Russell "La lista se denomina como mapeo de secuencia de percepciones para acciones", si mediante los mapeos se caracterizan a los agentes, para los agentes ideales se caracterizan por mapeos ideales, esto es especificar que acción emprenderá un agente en respuesta a una secuencia determinada de percepciones.

Para proceder al diseo de un programa de agentes es necesario, es necesario contar con una idea bastante precisa de las posibles percepciones y acciones que intervendrán, que metas o medidas de desempeño llevará a cabo el agente, así como el ambiente en el que el agente operará. En la

figura 3.6 se muestran los elementos básicos que se consideran en la elección de los tipos de agente.

Tipo de Agente	Percepciones	Acciones	Metas	Ambiente
Sistema para diagnósticos médicos	Síntomas, evidencias y respuestas del paciente	Preguntas, pruebas, tratamientos	Paciente saludable, reducción al mínimo de costos	Paciente, hospital
Sistema para el análisis	Pixels de intensidad y colores diversos	Imprimir una clasificación de escena	Clasificación correcta	Imágenes enviados desde un satélite en órbita
Robot clasificador de partes	Pixels de intensidad variable	Recoger partes y clasificarlas poniéndolas en botes	Poner las partes en el bote que les corresponda	Banda transportadora sobre la que se encuentran las partes
Controlador de refinería	Lecturas de temperatura y presión	Abrir y cerrar válvulas, ajustar la temperatura	Lograr pureza, rendimiento y seguridad máximos	Refinería
Asesor interactivo de inglés	Palabras escritas a máquina	Ejercicios impresos, sugerencias y correcciones	Que el estudiante obtenga la máxima calificación en una prueba	Grupo de estudiantes

Figura 3.6: Ejemplo de diversos tipos de agentes

3.3.1 Agentes Inteligentes en Ambientes Virtuales

Agentes en Juegos de Video

En los juegos de video los oponentes tienen cierta inteligencia, entre los algoritmos mas comunes que se utilizan en los video juegos se encuentran los de persecución y evasión de contrincantes.

Juegos que fueron sumamente populares como PACMAN, son un caso claro de los algoritmos de persecución y de evasión, cuando PACMAN estaba en modo normal, los fantasmas lo persiguen, pero cuando se toma la pastilla especial, pasan a modo de evasión.

Los algoritmos utilizados en los video juegos, sirven para dar una sensación de vida en las creaturas y por lo tanto simular realidad en nuestros juegos.

Otro concepto que se utiliza en los video juegos es que los objetos que se encuentran en él, ya sean naves espaciales o creaturas de ultratumba es el programarles cierto patrón que simule un comportamiento inteligente.

En la actualidad los objetos, *agentes*, dentro de un juego de computadora, pueden obtener comportamientos complejos en grupo, como es el caso de los juegos de deportes, como el futbol americano y el futbol soccer, donde se tiene persecución individual y colectiva, relevos, etc.

3.3.2 Agentes en Internet

Con el *boom* que se ha vivido recientemente en Internet, se ha vuelto común la utilización de agentes distribuídos. un ejemplo es el de una pecera virtual en donde en diferentes nodos se tiene programado el comportamiento de cada pez, y donde en cada nodo se puede visualizar, reaccionar e interactuar con los otros peces que se encuentran en otras computadoras.

La utilización de agentes en Internet se volverá sumamente popular debido a que ya es mucha la atención que se ha puesto en la creación de una comunidad virtual, existen ejemplos concretos como el Community Place Bureau de Sony. ver [13]

Gracias a VRML y Java es posible la creación de objetos tridimensionales que simulen un comportamiento "inteligente".

Otros ejemplos de agentes en Internet se encuentran:

Agente Lifestyle finder *http://lifestyle.cstar.ac.com/lyfestyle.* En el cual se recomienda documentos a los usuarios, basados en su forma de vida.

WARREN *http://www.cs.cmu.edu/ softagents/warren.* Son agentes inteligentes para la administración de un portafolio financiero.

NETBOT Jango *http://www.jango.com.* Asistente de compras capaz de buscar miles de tiendas on-line.

Open Sesame *http://www.opensesame.com.* Observa tus actividades y aprende que tareas uno repite una y otra vez y después realiza esas tareas repetitivas automaticamente.

3.4 Planificación

Según Russell [16] "Para una planificación práctica es necesario

1. Restringir el lenguaje utilizado para definir los problemas. Cuando se usa un lenguaje restrictivo, hay menos soluciones posibles que buscar.

2. Para encontrar una solución que emplee un algoritmo de propósito especial, denominado planificador, en vez de un demostrador de problemas de propósito general."

Los dos elementos van de la mano, ya que cuando se define un nuevo lenguaje para describrir los problemas se requiere de un nuevo algoritmo de planificación para procesar tal lenguaje.

El método que se utiliza comunmente, empleado por la mayoría de los planificadores se caracteriza por describir estados y operadores mediante un lenguaje restringido conocido como lenguaje STRIPS.

3.5 Sistemas Expertos

Los sistemas expertos son utilizados para apoyar en la toma de decisiones simulando a un experto: prolog es un lenguaje que ha sido utilizado frecuentemente para el desarrollo de sistemas expertos. Estos sistemas generalmente se componen de una base de datos de reglas, i.e. si pasa tal cosa entonces se efectúa alguna acción, etc.

La programación de sistema experto permite separar el dominio de conocimiento del resto del programa. Watson [22] comenta que en inteligencia artificial , los sistemas expertos se implementan generalmente como un conjunto de reglas if-then.

3.6 Robots

La construcción de un robot involucra diferentes campos de estudio como son la ingeniería eléctrica, programación e ingeniería mecánica.

Jones [11] define "robótica es acerca de construcción de sistemas. Actuadores locomotrices, manipuladores, sistemas de control, grupos de sensores, fuentes de poder eficientes, sistemas de software bien diseñados –todos estos subsistemas tienen que ser diseñados para que encajen juntos en un paquete apropiado capaz de llevar a cabo la tarea del robot".

Los robots han sido utilizados en la industria donde realizan labores repetitivas, que requieran de gran precisión y/o donde sea difícil o peligroso para el humano.

3.7 Robots móviles

Los robots móviles son aquellos que se desplazan en tiempo y espacio para el cumplimiento de sus tareas. La manera mas fácil de desplazar a un robot es por medio de llantas, aunque también existen robots caminadores.

Los Robots móviles han sido popularizados por películas como la Guerra de las Galaxias. o en libros como el de Yo Robot de Isaac Asimov. Pero la verdad es que todavía nos encontramos lejos de la creación de esa clase de Robots complejos.

Los robots móviles de tierra se pueden clasificar de acuerdo a su medio locomotriz: robots oruga, robot con patas y robot con llantas.

1. Según Segovia [19] "los robot oruga son capaces se desplazarse en un área difícil, o subir escaleras. Algunos proyectos se ha desarrollado con propósitos militares".

2. Los robots con patas poseen un gran potencial de desempeño en terreno disparejo, en comparación con otros tipos de vehículos. Por lo general este tipo de vehículos ofrecen el mayor número de grados de libertad que otros tipos de vehículos, ya que cada pata tiene por lo menos dos grados de libertad. Estos robots son capaces de librar obstaculos ya sean trincheras o terrenos que tenga que escalar, o moniobrar en espacios reducidos. Sin embargo la coordinación de movimientos de la unión de las patas es necesaria para producir la locomoción deseada del vehículo, Segovia [19].

3. Los robots con llantas son los mas usuales. Existen diferentes tipos de robots con llantas, de acuerdo con la posición y número de llantas con motor. Independientemente del tipo, número mínimo de motores que este debe contener no puede ser menor de dos, lo que proporciona los grados de libertad para despazamiento en un plano. En Segovia [19] se mencionan los siguientes ejemplos.

   Triciclo La propulsión de un vehículo de este tipo se realiza a través de un motor aplicado a las llantas traseras, por medio de una

caja diferencial. El control direccional se realiza por medio de un segundo motor.

Vehículo de 4 ruedas Esta arquitectura es sin duda la mas utilizada. En esta configuración, existen dos llantas manejadas independientemente ligadas al mismo eje y dos llantas libres. La rotación del robot o cambio de dirección se define por la diferencia de desplazamientos de las ruedas.

Capítulo 4

Conceptualización

4.1 Objetivo del Proyecto

El objetivo del proyecto es simular agentes inteligentes y robots mobiles [5] en ambientes virtuales, en computadoras de bajo costo (como las computadoras PC compatibles).

4.2 Esquema General del Proyecto

Se detectaron tres programas clave para lograr el objetivo trazado en este proyecto.

- Un Sistema Experto para manejar un control "inteligente" sobre agentes y robots, el cual debe tener como objetivo principal procesar comandos en lenguaje natural, y dependiendo del comando mandar mensajes al ambiente virtual para mover los objetos correspondientes.
- Otro programa debe de crear un ambiente virtual, con los objetos que se van a mover dentro del mismo, a partir de un archivo de descripción del entorno que se quiere simular; se probó con una casa con dos cuartos, dos baños, una cocina, un garage, un comedor, un desayunador, una bodega y una cobacha. El archivo se lista en el apéndice B
- El otro programa es un Ambiente Virtual el cual debe de aceptar mensajes vía sockets y debe de modificar el ambiente para simular

movimiento de objetos en un espacio tridimensional.

Se partió de la base de un shell de sistema experto desarrollado en CLIPS [10], este shell acepta mensajes UDP, para los comandos en lenguaje natural y manda instrucciones para mover los agentes, o en su caso robots a un nuevo destino.

Se encontró que para simular ambientes virtuales con VRML; la parte en Java que se encarga de la comunicación tiene un mejor desempeño con mensajes TCP debido a que se requiere el cumplimiento de tiempos específicos para simular de forma correcta el moviento de los agentes y los robots. Debido a lo anterior se decidió crear un programa que fuera una interface de sockets, que recibiera mensajes en sockets UDP del sistema experto y mandara mensajes por sockets TCP al ambiente virtual.

El sistema experto recibe los comandos por sockets UDP, por lo que se decidio crear una página HTML con un applet en Java para mandar los comandos al sistema Experto.

En la figura 4.1 se muestran todos los programas que componen el proyecto y la relación entre ellos; con el funcionamiento correcto de todos ellos se pretende lograr el objetivo del proyecto.

En el siguiente capítulo se describe cada uno de los programas que componen el proyecto.

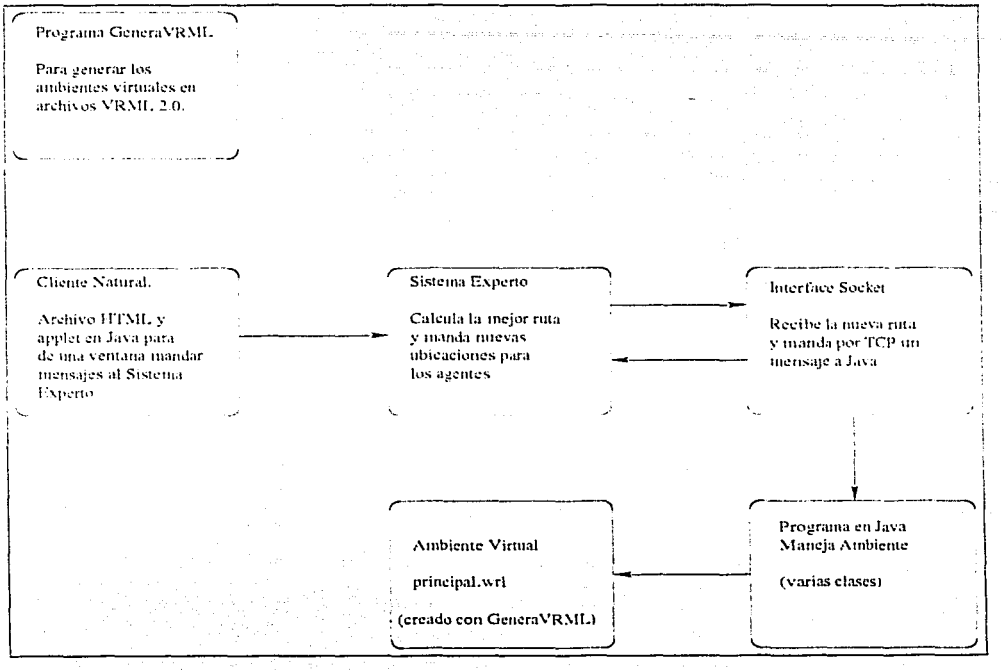

Figura 4.1: Programas involucrados en el proyecto

Capítulo 5

Componentes Proyecto

5.1 GeneraVRML

5.1.1 Análisis Orientado a Objetos del Sistema

El objetivo de este programa es el de crear archivos que tengan la estructura de un archivo VRML 2.0, a partir de sentencias sencillas donde se escriba el nombre del objeto, su posición y orientación en el espacio, así como la escala del objeto.

Este programa nació de la necesidad de crear ambientes virtuales a partir de sentencias simples, en donde se puedan reutilizar los modelos de objetos: silla, mesa, agente, robot, etc.

Para lograr esto se identificó que era útil emplear un método orientado a objetos, ya que en varios proyectos se ha comprobado la ventaja de utilizarlo, debido a la posibilidad de reutilizar componentes.

La metodología utilizada es la de Booch [2], ya que tiene bastante aceptación y es parte fundamental de la metodología UML que comprende a la de Booch, Jacobson y Rumbaugh.

El lenguaje de programación que se escogió para este programa es C++ [6] pues es Orientado a Objetos y de fácil portabilidad.

Diagrama de Clases

Las clases identificadas para este proyecto se muestran en la figura 5.1

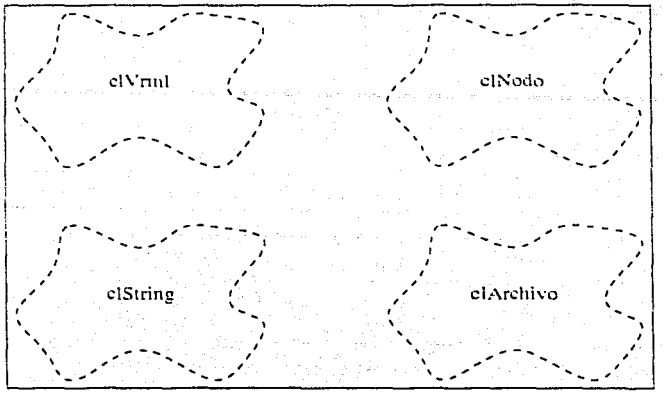

Figura 5.1: Clases para GeneraVRML

Las clases se relacionan como se muestran en la figura 5.2

5.1.2 Diseño Orientado a Objetos del Sistema

El funcionamiento del programa es el siguiente:

> El programa acepta dos archivos con la descripción del ambiente Virtual. El primero es una representación poligonal de los cuartos donde el robot móbil navega, ademas contiene los objetos y obstáculos en él. Es el mismo que utiliza con el sistema experto. El archivo completo se encuentra en apéndice B y El segundo archivo contiene los objetos que se utilizarán en el ambiente, con sus orientaciones, escalas y archivo VRML que se utiliza para representar cada objeto utilizado en la casa con agentes. Ver apéndice C. En el primer archivo tenemos sntencias y hechos para el sistema experto, donde se describen los objetos de la casa.

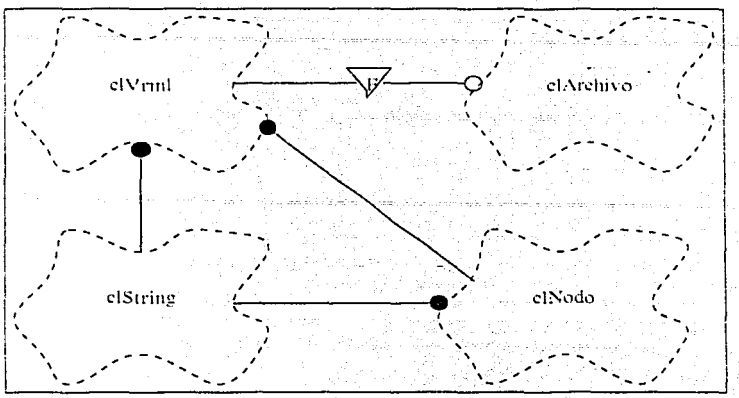

Figura 5.2: Diagrama de relaciones para GeneraVRML

En la figura 5.3 se muestra la entrada y salida que debe tener el programa, visto como una caja negra.

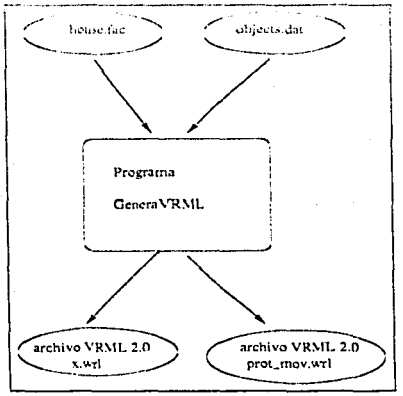

Figura 5.3: Programa Genera VRML

5.1.3 Programa en C ++

Los archivos que se utilizan en el proyecto GeneraVRML se encuentran enumerados en la tabla 5.1

Makefile
clarchivo.h
clarchivo.cpp
clnodo.h
clnodo.cpp
clstring.h
clstring.cpp
clvrml.h
clvrml.cpp
instruccion.h
const.h
iterator.h
list.h
principal.cpp

Tabla 5.1: Archivos utilizados por proyecto GeneraVRML

5.1.4 Prueba del Programa

Ejemplo de ejecución del Programa GeneraVRML ver tabla 5.2 :

En el siguiente ejemplo, se tiene una descripcin poligonal que representa al agente. En el archivo utilizado por el sistema experto, *apéndice B*, se tiene:

```
(mobile-agent  robot   living-room 16 12 0)
(static-object cama_hijo  childs-bedroom   15 3.2 0)
```

En el archivo donde tenemos la descripción geométrica de los objetos se tiene:

```
robot     robot_azul.wrl 0 0 1 0 2 2 2
```

```
[CadbluR@linux VRML]$ GeneraVRML
Inicio ...

Dar Nombre de archivo de Entrada : house.fac

Dar Nombre de archivo de Salida  : principal.wrl

Dar Nombre de archivo de Objetos : objetos.dat
```

Tabla 5.2: Ejemplo de corrida de GeneraVRML

```
cama_hijo     cama.wrl      0 0 1 3.1416  2.0 2.0 2.0
```

Utilizando esta información se genera una descripción VRML 2.0 como se muestra en la tabla 5.3.

El programa lee los archivos especificados, como se muestra en la tabla ?? y los guarda en una lista de objetos de la clase clString, para posteriormente procesarlos y convertirlos en una lista de objetos de la clase clNodo, que son listas de Strings donde guardan tokens, ejemplo El objeto clNodo tendria la siguiente lista:

```
mobile-agent
robot
living-room
16
12
 0
```

Tambin el archivo con las descripciones geométricas se generaría la siguiente objeto de la clase clNodo que a su vez estaría compuesta de lista de Strings, como se muestra:

```
robot
robot_azul.wrl
```

```
DEF AGENTE_robot  Obj_Movible {
    urlObjeto     ["robot_azul.wrl"]
    posicion      16.00 12.00  0.00
    orientacion    0.00  0.00  1.00 0.0000
    escala         2.00  2.00  2.00
}

DEF STATIC_cama_hijo Transform {
        translation  15.00  3.20  0.00
        rotation      0.00  0.00  1.00 3.1416
        scale         2.00  2.00  2.00
        children  [
           Inline { url "cama.wrl" }
        ] # Fin de children
} # Fin de Objeto Estatico
```

Tabla 5.3: Parte de un archivo VRML

```
0
0
1
0
2
2
2
```

Donde el primer token simboliza el nombre del agente, el segundo el archivo VRML que representa al agente los siguientes cuatro números representan la orientación y los últimos tres la escala en X en Y y en Z.

Estos datos se procesan y se convierten en objetos de clase clString los cuales se guardan en una lista de objetos clString y esta se utiliza para crear un archivo con formato VRML 2.0, con la clase clArchivo.

El resultado lo guardamos en el archivo *principal.wrl* ver apéndice D y en archivo *prot_mov.wrl*, ver apéndice E. Estos se utilizan en el browser de VRML 2.0 para generar el render del ambiente virtual.

FALTA

PAGINA

41

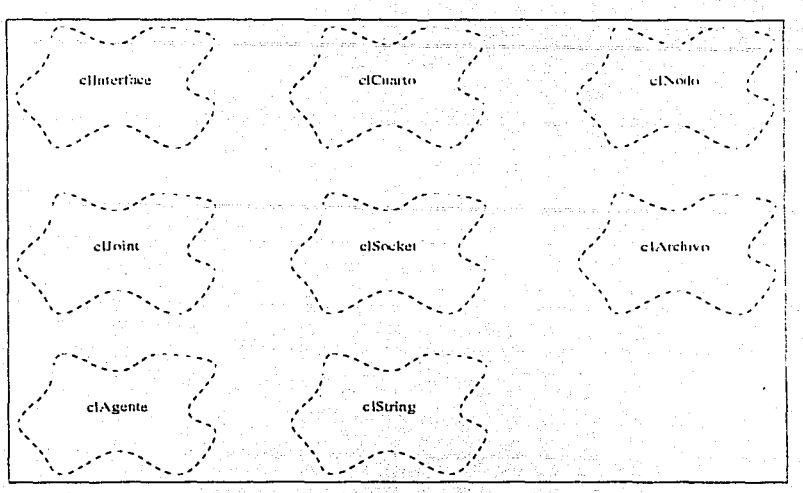

Figura 5.4: Clases para IntSocket

5.2.3 Programa en C ++

Los archivos que se utilizan en el programa IntSocket se encuentran enumerados en la tabla 5.4

5.2.4 Prueba del Programa

Ejemplo de ejecución del Programa GeneraVRML ver tabla 5.5, cuando se llama el programa se leen los archivos especificados y son procesados de igual manera que en el programa GeneraVRML para crear las listas de Nodos. Una con los datos de la descripción geométrica y otra con los datos utilizados por el Sistema Experto. Además se crean una lista de objetos de la clase clCuarto, donde se tiene una lista de objetos de la clase clJoint, para las puntos de unión con otros cuartos, y las coordenadas que describen los límites del cuarto y el punto que será el destino para los agentes que visiten el cuarto.

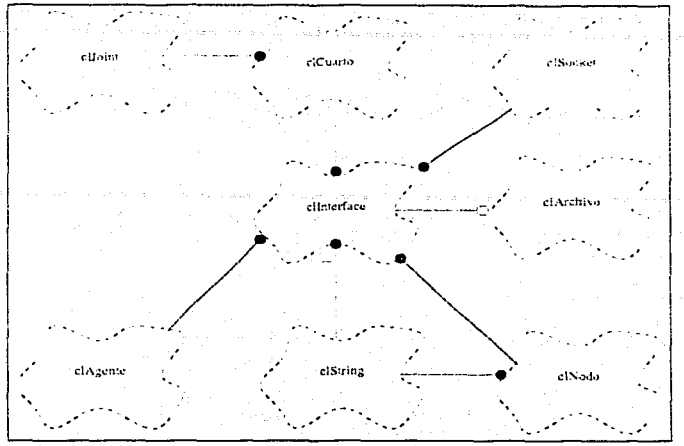

Figura 5.5: Clases para IntSocket

Se crea también una lista de de objetos de la clase clAgente, donde se tienen los datos de los agentes: ubicación, orientación, nombre y representación.

El objeto de la clase clSocket con el que se maneja la comunicación por medio de sockets TCP en la parte de linux.

Y en un procedimiento se escuchan mensajes UDP, y se procesan con los datos guardados en las listas de agentes y cuartos; en base al estado de los agentes y las descripciones de los cuartos se generan los mensajes TCP que se mandan al cliente Java.

En la figura 5.6 se muestra como es procesado un mensaje UDP recibido por el programa Interface Socket, y cual es el mensaje TCP que se manda.

Makefile.interface
clagente.cpp
clagente.h
clarchivo.cpp
clarchivo.h
clcuarto.cpp
clcuarto.h
clinterface.cpp
clinterface.h
cljoint.cpp
cljoint.h
clnodo.cpp
clnodo.h
clsocket.cpp
clsocket.h
clstring.cpp
clstring.h
clvrml.cpp
clvrml.h
const.h
gsNetPoll.h
instruccion.h
iterator.h
list.h
main_interface.cpp

Tabla 5.4: Archivos utilizados por programa IntSocket

5.3 ManejaVRML

5.3.1 Análisis Orientado a Objetos del Sistema

El objetivo de este programa es proporcionar una interfase para desplegar e interactuar con ambientes virtuales. A partir del archivo generado por el programa GeneraVRML, descrito en una sección anterior.

Para llevar a cabo los objetivos de este programa se encontraron las siguientes clases y relaciones.

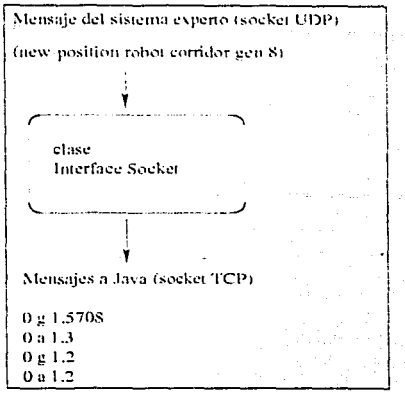

Figura 5.6: Programa Interface Socket

Diagrama de Clases

Las clases identificadas para la parte del cliente, esto es para el manejo de los sockets y del movimiento del robot, se muestran en la figura 5.7

Las relaciones de herencia se muestran en la figura 5.8; se detectaron las clases base de Java que se requerian para realizar las tareas requeridas y se heredaron para adecuarlas al requerimiento del sistema.

La clase principal es Cliente_Agentes, la cual contiene otras clases como

```
[CadbluR@linux CadbluR]$ SI.sh
Inicio ...

Dar Nombre de archivo de Entrada : house.fac

Dar Nombre de archivo de Objetos : objetos.dat
```

Tabla 5.5: Ejemplo de corrida de IntSocket

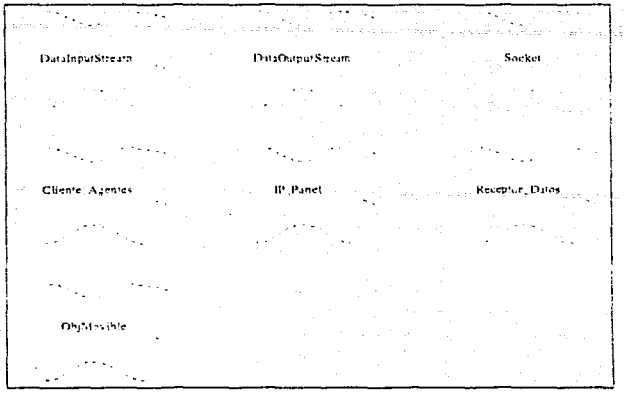

Figura 5.7: Clases para parte del cliente

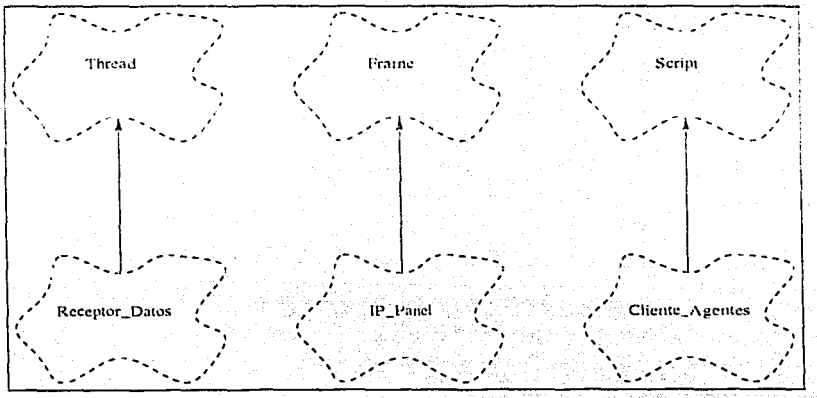

Figura 5.8: Diagrama de herencia para la parte Cliente

se muestra en la figura 5.9, y donde se realiza todo el proceso, esta clase es heredad de Script, la cual se utiliza para ligar Java con el comportamiento del ambiente virtual desplegado en el browser de VRML.

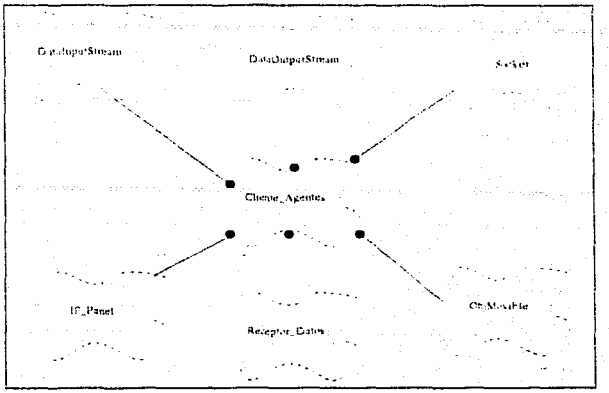

Figura 5.9: Diagrama de compuestos para la parte Cliente

5.3.2 Diseño Orientado a Objetos del Sistema

El programa debe de realizar el comportamiento que se muestra en la figura 5.10, para lograr esto se crearon objetos de las clases:

Cliente_Agentes :El objeto de la Clase principla donde se contienen las demás clases.

IP_Panel : El objeto de esta clase sirve para crear una ventana en donde se proporciona el host y el puerto para conectarse al servidor, que mandará las instucciones para actualizar la posición y orientación de los agentes.

DataInputStream Este objeto sirve para crear una via de entrada por medio del socket creado.:

DataOutputStream Este objeto sirve para crear una mandar datos al servidor, por medio del socket creado:

Socket Esta clase sirve para crear un socket, para trasmitir datos al servidor:

ReceptorDatos : El objeto de esta clase sirve para escuchar si llegan datos en el puerto donde el socket fue creado.

ObjMovible : Estos objetos son los agentes, donde se tiene información acerca del estado de cada uno de ellos en el ambiente virtual, como su posición orientación, también se describe a través de un método su movimiento.

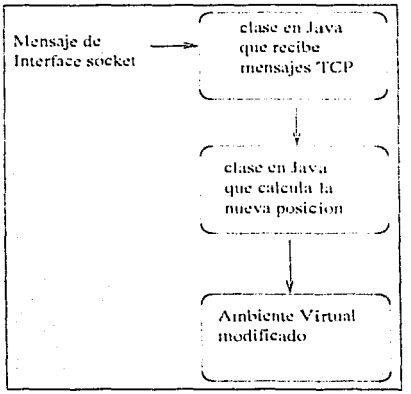

Figura 5.10: Clases para parte del cliente

Archivos VRML

Los archivos principales del ambiente virtual son *principal.wrl y prot_mov.wrl*, en principal.wrl se tiene la descripción del ambiente virtual y en prot_mov.wrl se tiene la abstracción del objeto movible; estos se listan en los apéndices D ??

5.3.3 Prueba del Programa

Para el fucionamiento correcto de este programa debe existir un servidor de sockets TCP levantado que este escuchando solicitudes de conexión en este caso se solicita una conexión a la computadora 200.200.77.2 (la computadora linux) por el puerto 4130, como se muestra en la figura 6.3 esta dirección IP fue creada en una máquina Linux, donde se encuentra un programa servidor de sockets, el program Interface de Socket.

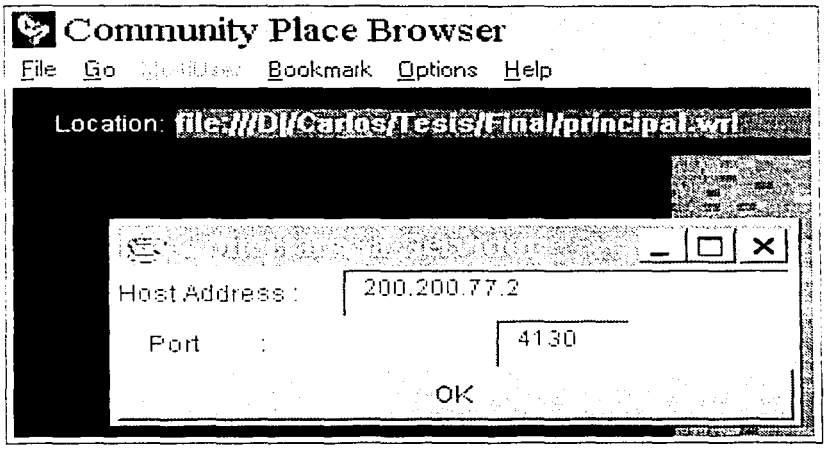

Figura 5.11: Conectarse al servidor

El objeto de Receptor_Datos escucha mensajes y cuando recibe uno se [illegible] y se actualiza el ambiente virtual, para simular el comportamiento de los agentes en un ambiente tri-dimensional.

5.4 Cliente Natural

5.4.1 Análisis Orientado a Objetos del Sistema

Este programa tiene como objetivo mandar comandos en lenguaje natural desde una página HTML, se utiliza un applet de Java para mandar mensajes UDP al sistema experto.

Diagrama de Clases

Las clases identificadas para la parte del cliente para Lenguaje Natural, esto es para el manejo de sockets UDP y para la ventana donde se mandará el comando al Sistema Experto se muestran en la figura 5.12

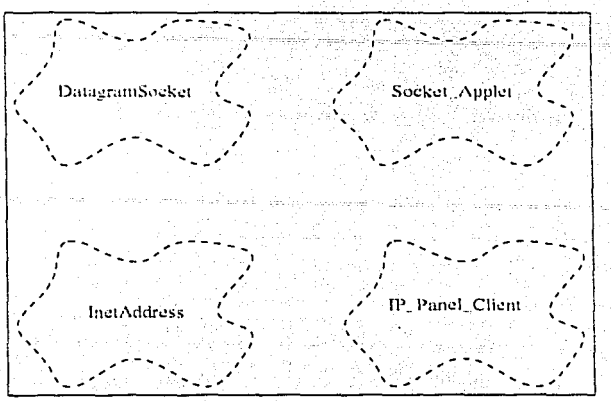

Figura 5.12: Clases para parte del cliente

La parte de herencia se muestra en la figura 5.13

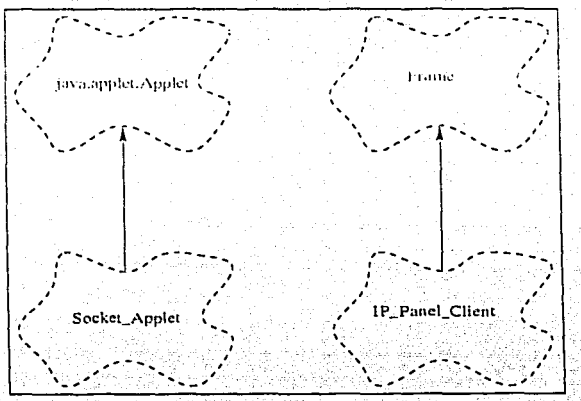

Figura 5.13: Diagrama de herencia para la parte Cliente

La clase principal es Socket_Applet, la cual contiene otras clases como se muestra en la figura 5.14

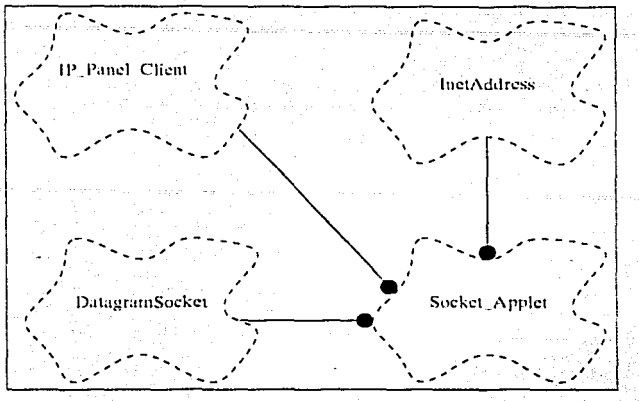

Figura 5.14: Diagrama de compuestos para la parte Cliente

5.4.2 Prueba del Programa

Para el fucionamiento correcto de este programa debe existir un servidor de sockets UDP levantado que este escuchando solicitudes de conexión en este caso se solicita una conexión a la computadora 200.200.77.2 (la computadora linux) por el puerto 2000, como se muestra en la figura 6.1 esta dirección IP fue creada en una máquina Linux, donde se encuentra el servidor de sockets UDP, el Sistema Experto desarrollado en CLIPS, el cual recibe ordenes en lenguaje natural. Este programa se crea una ventana donde se mandan las ordenes en lenguage natural, tal como se muestra en la figura 6.2.

5.5 Sistema Experto

Para el Sistema Experto se utilizó un shell desarrollado para el manejo de agentes inteligentes. para una explicación del funcionamiento del shell ver [5].

El sistema experto puede proveer entendimiento de alto y de bajo nivel de los eventos que ocurren en el ambiente virtual. El sistema experto interpreta los comandos, planea como ejecutarlos y manda los comandos al programa Interface de Socket, para que envíe los mensajes correspondientes

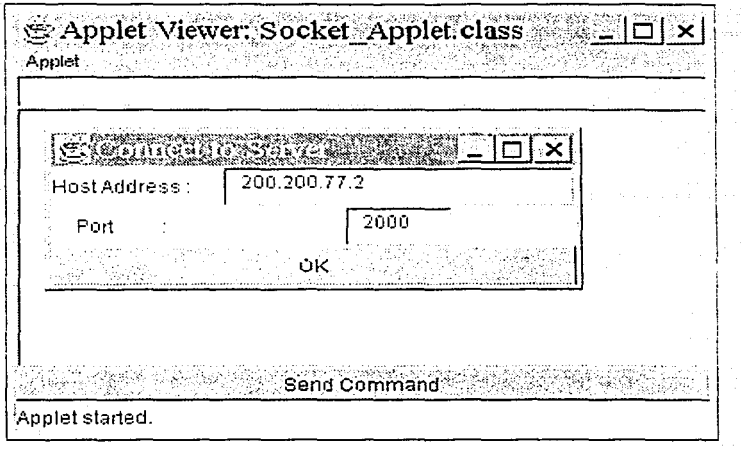

Figura 5.15: Conectarse al servidor

al ambiente virtual. El sistema experto realiza sus tareas a traves de un base de conocimiento que se encuentra en un conjunto de reglas de tipo *if-then*, tal como: si FACT-0 entonces HACER-ESTO, donde FACT-0 es el evento necesario para activar la regla y HACER-ESTO es la consecuencia.

Se utiliza *entedimiento semántico* en donde se emplean técnicas de lenguaje natural, lo que permite al sistema entender las características y responder a comandos individuales.

Entendimiento Semántico una vez que las características importantes se han extraído, se puede asignar significado a ellas utilizando técnicas de lenguaje natural. Lenguaje Natural es el que utilizamos para comunicarnos con otras personas, y se puede utilizar también para comunicarse como interface hombre-máquina. De esta manera la comunicación se realiza de forma natural. [18]

El proceso de entendimiento de lenguaje natural se puede descomponer en los siguientes pasos:

1. **Entrada de señal:** Ya sea texto proveniente del teclado o voz. Esta entrada se transforma en unidades básicas (palabras) para los sigu-

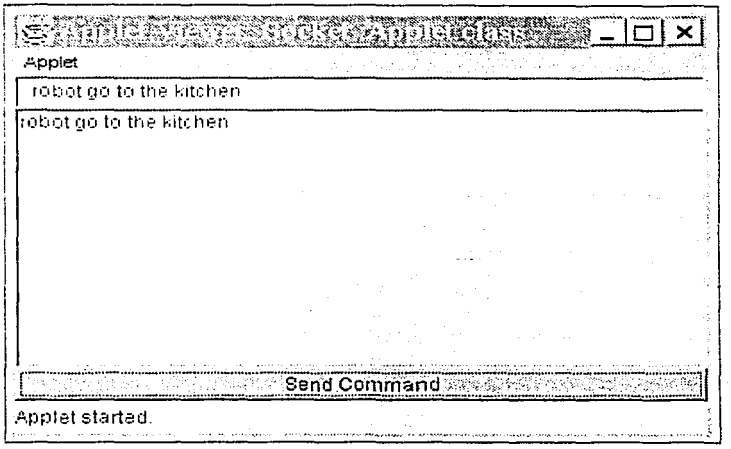

Figura 5.16: Mandar Ordenes Applet

ientes pasos.

2. **Análisis Sintáctico:** En este paso las palabras de entrada son probadas para ver si pueden ser agrupadas de acuerdo a reglas gramaticales; esto es, si pueden formar sentencias con algún significado.

3. **Análisis Semántico:** El significado de cada palabra y sentencia es asignado. Esta es una de las partes mas complicadas de los tres pasos, y se requiere se una gran base de datos de conocimiento, para algo mas que un dominio de problema extremadamente sencillo.

Para complementar revisar a Savage [18].

Este shell fue modificado para que se poder comunicarse con el Programa Interface de Socket, el cual se conecta con VRML para el manejo del Ambiente Virtual.

El shell está desarrollado en CLIPS, hecho por la NASA para crear sistemas expertos.

5.5.1 Prueba del Programa

Se recibe un mensaje (robot go to the garage) por el puerto 2000, entonces el sistema experto realizara las siguiente salida en pantalla. que es una representación de lo realizado por el mismo.

```
client-data robot go to the garage
 Data from the server: robot
 Data from the server: go
 Data from the server: to
 Data from the server: the
 last data from the server: garage
* (ptrans (actor robot)(obj robot)(to garage)(from living-room)(status 100) )

* I'm going to the corridor
 answer (new-position gen8 1 8.0 12.0 0)

* I'm now in the corridor

* I'm going to the garage
 answer (new-position gen10 1 5.0 5.0 0)

* I'm now in the garage
```

Es to es el sistema experto encuentra que el sujeto de la acción (actor) es el robot y que el objeto de la acción es el mismo y que realizará un desplazamiento del living-room al garage.

Para lo cual encuentra que primero debe ir al corridor y de corridor se irá al garage, así finalizando la tarea encomendada.

Los mensajes que manda al interface de Socket son:

```
new-position robot corridor gen8

new-position robot garage gen10
```

Los cuales son procesados por el programa Interfase de Socket, el cual responde con la posición nuevo y el número de solicitud que se está respondiendo. En este ejemplo gen8 para cuando se encuentre se actualice la posición del robot al corredor.

Capítulo 6

Resultados y Conclusiones

6.1 Resultados

Al concluir el proyecto se obtuvo una utilería para crear ambientes virtuales a partir de descripciones sencillas, con las cuales podemos simular agentes y robots.

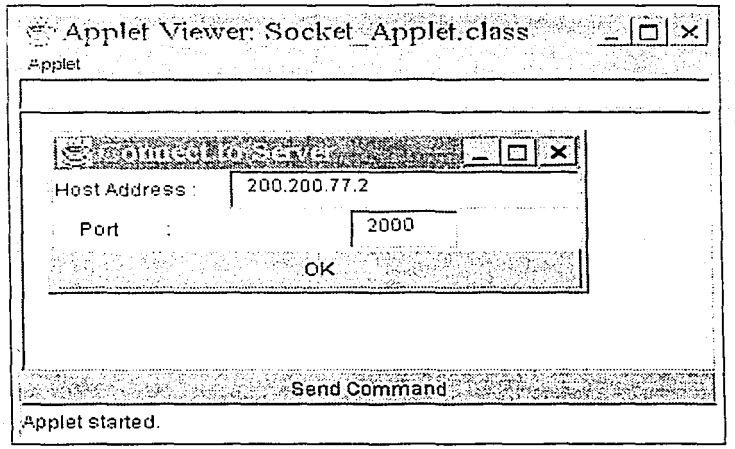

Figura 6.1: Poner Datos en Applet

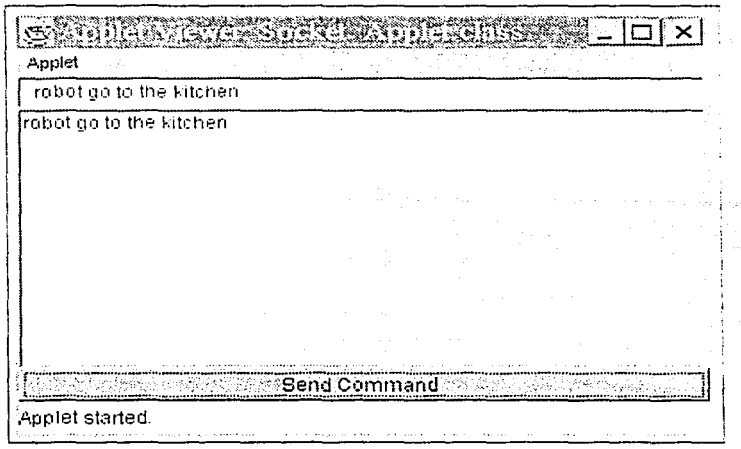

Figura 6.2: Mandar Ordenes Applet

Los ambientes virtuales se pueden observar desde diferentes puntos para evaluar el comportamiento del robot o los agentes que se deseen simular.

En la vista aerea ver figura 6.4 se puede observar el ambiente virtual que se genera a partir del archivo con la descripción del mismo.

Con la vista del robot ver figura 6.5 se observa lo que ve el robot, es de manera similar con los agentes.

Existen otras formas de ver el ambiente, como es el caso de lo que esta viendo el usuario del ambiente virtual, el cual se puede desplazar observando los detalles de la casa, como se muestra en la figura 6.6

6.2 Conclusiones

Para este proyecto se utilizaron dos computadoras PC compatibles, una con el sistema operativo Linux y otro con Windows 95. En Linux se ejecuta el sistema experto, que manda mensajes a través de sockets al programa Interface de socket que procesa los datos y despues manda mensajes al programa en Java para mover al robot.

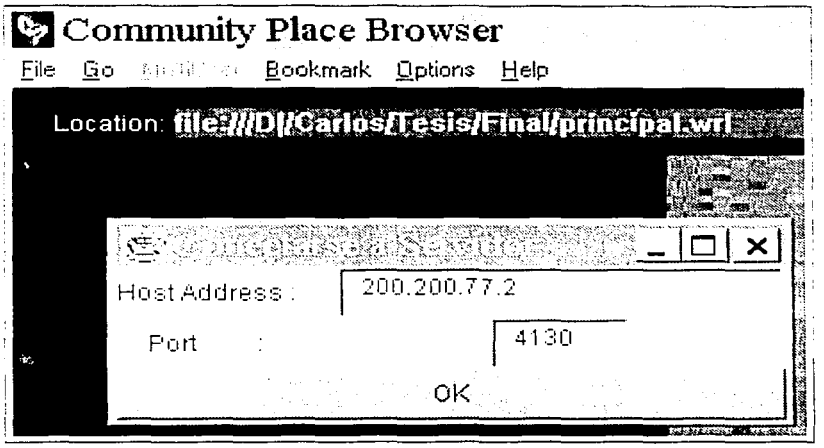

Figura 6.3: Conectarse a Servidor

El programa GeneraVRML crea archivos VRML 2.0. a partir de un archivo que describe como construir el ambiente virtual con cuartos, objetos y agentes con sus respectivas posiciones y orientaciones.

El visor de VRML se ejecuta en la máquina Windows 95, ver artículo del autor con Savage [5]. El sistema experto calcula la mejor secuencia de movimientos que los agentes necesitan desarrollar y son mandados via sockets al programa Interface de Sockets.

Los mismos mensajes que el programa Interface de Socket recibe, se utilizan para el robot real que se encuentra en el Laboratorio de Interfaces Inteligentes, por lo que se pueden probar los algoritmos con el robot virtual y luego ser utilizados en el real.

La utilización de este método muestra que podemos simular robots y agentes en una ambiente virtual utilizando computadoras de bajo costo con VRML, Java, C++ y un Sistema Experto. También muestra la ventaja de correr diferentes procesos en diferentes plataformas como son Linux y Windows 95.

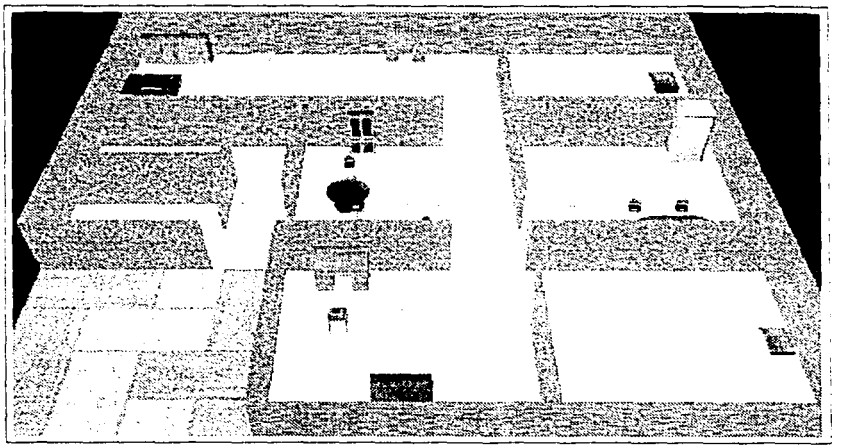

Figura 6.4: Vista Aerea

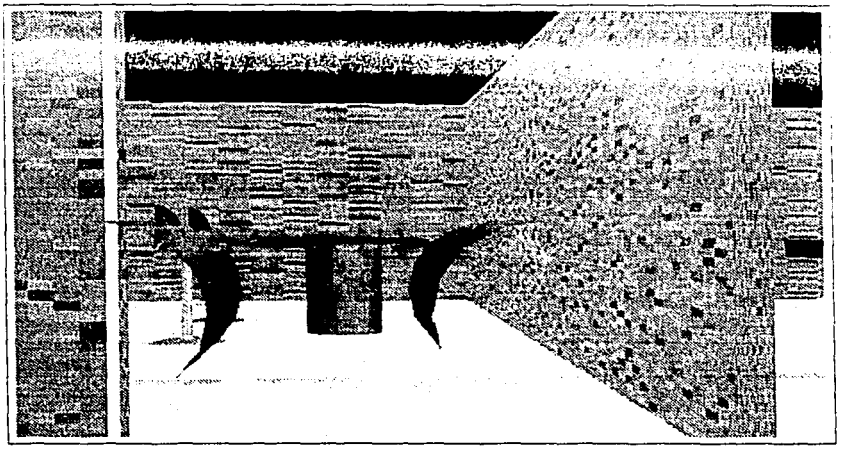

Figura 6.5: Vista del Robot

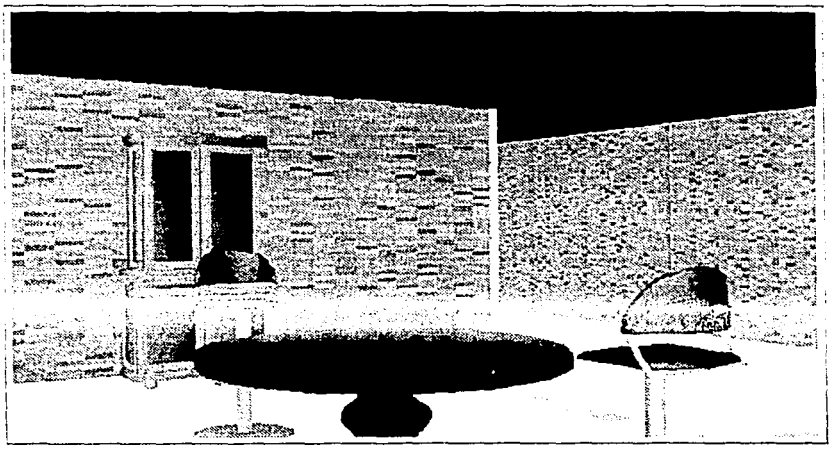

Figura 6.6: Vista del Usuario

Bibliografía

[1] BALLIN, Daniel: *Software Agents - A new generation of Software*, UK, http://shelob.iti.salford.ac.uk/ dan/presentations/agent-full/

[2] BOOCH, Grady: *Object Oriented Analysis and Design with Applications*, USA, Ed. Addison Wesley, 1994, 589 p.

[3] BRUSTOLONI, Jose C. Autonomous Agents:Characterization and Requirements, USA, Carnegie Mellon Technical Report CMU-CS-91-204, 1991

[4] COHN, Mike et. al : *Java Developer's Reference*, USA, Ed. Sams.net, 1996, 1258p.

[5] DELGADO, Carlos y SAVAGE, Jesús: *A Mobile Robot Simulator Using VRML, Java C++, and an Expert System*, Mexico, International Symposium on Robotics and Automation, 1998, 490 p.

[6] ELLIS, Margaret A. y STOUSTRUP, Bjarne: *Manual de Referencia C++ con Anotaciones*, USA, Ed. Addison Wesley, 1994, 541 p.

[7] FERRARO, Richard F.: *Learn 3D Graphics Programming on the PC*, USA, Ed. Addison Wesley, 1996, 1022 p.

[8] FOLEY, James et. al.: *Computer Graphics: Principles and Practice*, USA, Ed. Addison Wesley, 1992, 1174 p.

[9] FRANKLIN, Stan y GRAESSER, Art: *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*, USA, Proceedings of the Third International Workshop on Agent Theories, 1996

[10] GIARRATANO, Joseph: *CLIPS User's Guide*, USA, Software Technology Branch NASA, 1994, 162 p.

[11] JONES, Joseph y FLYNN, Anita: *Mobile Robots*, USA, Ed. A.K. Peters, 1993 . 349 p.

[12] KERLOW, Isaac Victor: *The Art of 3-D Computer Animation and Imaging*, USA. Ed. Van Nostrand Reinhold, 1996, 412 p.

[13] LEA, Rodger et. al.: *Java for 3D and VRML Worlds*, USA, Ed. New Riders Publishing. 1996, 399 p.

[14] MASTERS, Timothy: *Advanced Algorithms for Neural Networks*, USA, Ed. Wiley, 1995. 431 p.

[15] ROEHL, Bernie et al.: *Late Night VRML 2.0 with Java*. USA, ZD Press. 1997 ,710 p.

[16] RUSELL, Stuart y NORVIG, Peter: *Inteligencia Artificial: Un enfoque moderno*, Tr. Raúl Bautista Gutiérrez, México, Ed. Prentice Hall, 1996, 979 p.

[17] SAVAGE, Jesús, et al., *A virtual reality robot driven with voice commands*, Mexico, 4th Congress on Expert Systems. 1998

[18] SAVAGE, Jesús, et al., *Virbot: A Virtual Reality Robot Driven With Multimodal Commands*, UK, ECAI-98 Workshop. 1998

[19] SEGOVIA, Armando, et al., *Robots Móviles*, México, ISRA '98, 1998

[20] SRIDHARAN, Prashant: *Advanced Java Networking*, USA, Ed. Prentice Hall, 1997, 368 p.

[21] VINCE, John: *Virtual Reality Systems*, Great Britain, Ed. ACM Press, 1995, 388 p.

[22] WATSON, Mark, *AI Agents in Virtual Reality Worlds*, USA, Ed. Wiley, 1996, 309 p.

[23] WATT, Alan y WATT Mark, *Advanced Animation and Rendering Techniques*, Great Britain, Ed. Addison Wesley, 1992, 455 p.

[24] WINSTON, Patrick Henry, *Inteligencia Artificial*, USA, Ed. Addison Wesley, 1994, 805 p.

[25] WOOLDRIDGE, Michael y JENNINGS, Nicholas R., *Intelligent Agents: Theory and Practice*, United Kingdom, Knowledge Engineering Review, 1995, 62 p.

Apéndice A

Clases del Proyecto

Clase clArchivo

La clase clArchivo es una utiliría para el manejo de archivos; para leer un archivo de strings y guardarlo en una lista doblemente ligada de strings y para salvar en archivo, el contenido de una lista doblemente ligada.

La idea de creación de esta clase clArchivo es para poder reutilizarla en proyectos futuros.

La clase tiene los siguientes métodos:

Miembros Públicos

static int **leer_archivo** (char *, clLista_DE <clString> &)

static int **guardar_archivo** (char *,
clLista_DE <clString> &)

Clase clVrml

La clase clVrml tiene como objeto la conversión de un archivo simple, donde se describe el ambiente virtual, y otro archivo donde se encuentran la posición y orientación en el espacio, así como la escala deseada, de cada uno de los objetos que componen al ambiente virtual y se convierte a un archivo que tenga la estructura de un archivo VRML 2.0

Los miembros públicos, métodos que están disponibles para que los utilicen los clientes de esta clase son:

Miembros Públicos

friend class	**clArchivo**
	clVrml (void)
	~clVrml ()
void	**pedirNombres** (void)
void	**leerArchivo** (int iCual)
void	**escribirArchivo** (int iCual)
void	**convertir_aVRML** (void)
void	**llenar_Lista_Nodos** (void)
void	**llenar_Lista_Objetos** (void)
void	**crearProto** (void)

los miembros privados de la clase son:

Miembros Privados

clLista_DE <clString>	**oStringsEntrada**
clLista_DE <clString>	**oStringsSalida**
clLista_DE <clNodo>	**oListaNodos**
clLista_DE <clNodo>	**oListaObjetos**
char	**szNomArchEntrada** [MAX_STRING]
char	**szNomArchObjetos** [MAX_STRING]
char	**szNomArchSalida** [MAX_STRING]
void	**crearHeaderTodo** (void)
void	**crearFooterTodo** (void)

void	**crearLuces** (void)
void	**crearScript** (void)
void	**crearParedes** (void)
void	**crearHeaderArchivo** (void)
void	**crearAgentes** (void)
void	**crearObjetos** (void)
void	**crearPisos** (void)
int	**procesar_Linea** (clNodo &, clString &)

Clase clString

La clase clString tiene como objeto el de porporcionar una clase reutilizable, que provea a los clientes de la misma métodos para la manipulación de cadenas de caracteres.

Los métodos ofrecidos por esta clase son los siguientes.

Miembros Públicos

	clString ()
	clString (char *)
	~clString ()
	clString (clString &)
void	**actualizar** (char *)
void	**obtener** (char *)
int	**operator** == (const clString &)
void	**eliminarBlancos** ()
void	**mostrar** ()
void	**pegar** (char *)
size_t	**longitud** ()

La parte privada de esta clase sólo es el contenido de la misma; esto para encapsular el contenido.

Miembros Privados

char	szContenido [MAX_STRING]

Clase clNodo

La clase clNodo tiene como objeto propocionar una estructura en donde se guarden las propiedades de los objetos leídos, para luego procesarlos y crear el archivo.

Los miembros públicos de la clase.

Miembros Públicos

	clNodo (void)
	~clNodo ()
	clNodo (const clNodo& oCopia)
clNodo&	**operator=** (const clNodo& oCopia)
int	**operator ==** (const clNodo &)
void	**obtenerString** (int, char *)
void	meterString (char *)
void	**depurarTokens** ()
void	**mostrarTokens** ()
int	**darCantidad** (void)
void	**inicializar** (void)

La parte privada de la clase son las propiedades de los objetos que se quieren convertir en el archivo VRML, así como una variable (iCuantos) que es el número de elementos que se encuentran en la lista.

Miembros Privados

clLista_DE <clString>	**oListaTokens**
int	**iCuantos**

Clase clInterface

La clase clInterface, es la clase principal en el programa de Interface de Socket, en ésta se implementan los métodos públicos, que son los que se utilizan afuera de la clase. Entre estos se encuentran los constructores que se pueden llamar explícitamente en caso de poner parametros y de manera implícita, al crear un objeto; y los destructores que se llaman cuando ya no se necesita el objeto y se borra.

También esta el métodos para leer archivo, que se encuentran en la clase clArchivo. El método donde se lleva a cabo toda la acción es *principal*, y ahí se llaman varios métodos privados, los cuales no tiene caso hacerlos públicos pues sólo son necesarios para algunas funciones miembro de la clase.

Miembros Públicos

friend class	**clArchivo**
	clInterface (void)
	~clInterface ()
void	**pedirNombre** (void)
void	**leerArchivo** (int iCual)
void	**principal** (void)
void	**llenar_Lista_Nodos** (void)
void	**llenar_Lista_Objetos** (void)

En la parte privada se encuentran las listas de clases strings, nodos, cuartos y agentes; las cuales son necesarias para de un archivo de texto obtener las propiedades de los cuartos y de los agentes, y guardarlas en objetos de clases clCuarto y clAgente.

También se encuentran métodos necesarios sólo adentro de la clase clInterface.

Miembros Privados

clLista_DE <clString>

	oStringsEntrada
clLista_DE <clNodo>	**oListaNodos**
clLista_DE <clCuarto>	**oListaCuartos**
clLista_DE <clAgente>	**oListaAgentes**
clLista_DE <clNodo>	**oListaObjetos**
clSocket	**oServidor**
char	**szNomArchEntrada** [MAX_STRING]
char	**szNomArchObjetos** [MAX_STRING]
GSNPS*	**gHandle**
GSNPS*	**gHandle_W**
void	**crearCuartos** (void)
void	**crearAgentes** (void)
int	**procesar_Linea** (clNodo &, clString &)
void	**mandarMensaje** (char *)
void	**mandarUDP** (clString &)
void	**procesarMensaje** (char *)
void	**mandarAgentes** (void)
void	**mandarMensaje** (char *, int)
void	**esperar** (int)

Clase clAgente

Esta clase tiene como objetivo guardar el estado de cada uno de los agentes que se encuetran en el ambiente virtual, para lo cual se ponen a disposición los métodos públicos para accesar y modificar el estado.

Miembros Públicos

	clAgente (void)
	~clAgente ()
	clAgente (const clAgente& oCopia)
clAgente&	**operator=** (const clAgente& oCopia)
int	**operator ==** (const clAgente &)
void	**obtenerNombre** (char *)
void	**meterNombre** (char *)
void	**obtenerCuarto** (char *pCadena)
void	**meterCuarto** (char *pCadena)
void	**meterRef** (const char &)
char	**obtenerRef** (void)
void	**meterPosicion** (float &, float &, float &)
void	**obtenerPosicion** (float &, float &, float &)
void	**meterOrientacion** (float &, float &, float &, float &)
stMov	**mover** (float &, float &, float &)
void	**mostrar** (void)
void	**inicializar** (void)

La parte privada de la esta clase se conforma por el estado de los Agentes; esto es Nombre, posición y orientación en el espacio.

Miembros Privados

clString	**oNombre**
clString	**oCuarto**
float	**fPos_X**
float	**fPos_Y**
float	**fPos_Z**
float	**fOri_X**
float	**fOri_Y**

float	fOri_Z
float	fOri_T
char	cRepres

Clase clCuarto

La clase clCuarto tiene como objetivo guardar las características de los cuartos que conforman el ambiente virtual.

En la parte pública se encuentran los métodos que accesan y modifican las propiedades del objeto de la clase clCuarto.

Miembros Públicos

	clCuarto (void)
	~clCuarto ()
	clCuarto (const clCuarto&)
clCuarto&	**operator=** (const clCuarto&)
int	**operator ==** (const clCuarto &)
void	**obtenerNombre** (char *)
void	**meterNombre** (char *)
void	**meterLimites** (float &, float &, float &, float &)
void	**anadirJoint** (const clJoint &)
void	**meterDestino** (float &, float &, float &)
void	**obtenerDestino** (float &, float &, float &)
void	**mostrar** (void)
void	**inicializar** (void)
int	**obtenerJoint** (char *, float &, float &, float &, float &)

En la parte privada se encuentran las propiedades de los cuartos; los límites que el cuarto tiene, el punto destino de los agentes en el cuarto, el

nombre del cuarto y los puntos de unión que tiene el cuarto con los cuartos contiguos.

Miembros Privados

clString	**oNombre**
float	**fLim1_X**
float	**fLim1_Y**
float	**fLim2_X**
float	**fLim2_Y**
float	**fDest_X**
float	**fDest_Y**
float	**fDest_Z**
clLista_DE <clJoint>	**oListaJoints**

Clase clJoint

La clase clJoint sirve para guardar los puntos que unen dos cuartos. En la parte pública estan los métodos que accesan y modifican las propiedades de los objetos de esta clase. Además se tienen los métodos constructores y algunas operaciones sobrecargadas como comparación y asignación.

Miembros Públicos

	clJoint (void)
	~clJoint ()
	clJoint (const clJoint& oCopia)
clJoint&	**operator=** (const clJoint& oCopia)
int	**operator ==** (const clJoint &)
void	**obtenerDestino** (char *)
void	**meterDestino** (char *)
void	**meterCoords** (float &, float &, float &, float &)

void	**obtenerCoords** (float &, float &, float &, float &)
void	**mostrar** (void)
void	**inicializar** (void)

En la parte privada de la clase clJoint esta el nombre del cuarto destino y también los puntos que unen el cuarto que contiene el objeto con el cuarto destino guardado en esta clase.

Miembros Privados

clString	**oDestino**
float	**fJoi1_X**
float	**fJoi1_Y**
float	**fJoi2_X**
float	**fJoi2_Y**

Clase clSocket

La clase clSocket sirve para proveer una encapsulación a las funciones utilizados para crear una conexión de sockets TCP.

Miembros Públicos

	clSocket ()
	~clSocket ()
void	**cdmClose** ()
void	**cdmCrear_Socket** ()
void	**cdmListen** ()
void	**cdmSend** (char *, int)
void	**cdmReceive** (char *, int)
void	**cdmConnect** ()

void cdmPoner_Puerto (int)

void cdmPoner_Direccion (BYTE, BYTE, BYTE, BYTE)

Miembros Privados

int iHandle

struct sockaddr_in socket_address

int iHandle_inbound

struct sockaddr_in oDireccion_inbound

WORD iTam_inbound

unsigned char* vbyDireccion

int iPuerto

Clase Cliente_Agentes

La clase Cliente_Agentes es la principal de las clases utilizadas para la manipulación de los Agentes en el ambiente Virtual, pues aqui se tiene un vector con los agentes que se desean manipular en el ambiente, así como las otras clases que componen este programa.

public final static int COMIENZO

public final static int GIRAR

public final static int AVANZAR

public final static int GET_SENSORES

public final static int

METER

public final static int
LIGAR

public final static int
NUM_CHAR

public final static int
DIGITO_CERO

public final static byte
LIM_INF_CAR

public final static byte
LIM_SUP_CAR

private IP_Panel
oPanel

int iActual

private int
iPort

private String
sHost

private boolean
bCual

Socket oSocket_Cliente

Datainputstream
dis_Entrada

DataOutputStream
dos_Salida

Browser bMyBrowser

private static Vector
vAgente

private SFBool
oRobotView

private SFBool
oDefaultView

private boolean

```
            bBanderaVista
public Node
            obtenNodo (String sNombre)
public void
            initialize ()
private void
            girar (int iIndex, byte vbyMess [])
private void
            avanzar (int iIndex, byte vbyMess [])
private void
            obten_Sensores (int iIndex)
public void
            procesarMensaje (byte vbyMensaje[])
public void
            meterAgente (byte vbyMess [])
public void
            ligarTodos ()
public void
            putDatosConexion (String sHos, int iPuerto)
public void
            processEvent (Event evProcesar)
public void
            shutdown ( )
```

Clase IP_Panel

La clase IP_Panel sirve para crear una ventana en donde se acepten los valores del host, a donde se encuentra el servidor TCP que controlara el ambiente virtual, y también el puerto por donde se realizará la conexión.

TextField	**tfHost**
TextField	**tfPort**
Label	**laHost**
Label	**laPort**

```
Button      buOK
Cliente_Agentes
            ocVirbot
            IP_Panel (Cliente_Agentes cVirbot)
private void
            mandarDatos ()
public boolean
            action (Event un_evento, java.lang.Object arg)
```

Clase ObjMovible

La clase ObjMovible es donde se encuentran las propiedades de los agentes o el robot, según el caso, y donde se proporcionan métodos para mover los objetos cierta distancia o en su caso rotarlos. Existe un método para ligar este objeto a el nodo correspondiente en VRML 2.0 y así cuando se modifique el objeto de esta clase, se mueve el objeto representado en el ambiente por el nodo ligado.

```
private final static int
            NUM_INTP_ROTACION
private final static int
            DIV_INTP_ROTACION
private final static float
            GIRO_COMPLETO_RAD
String      sNombre
String      sNomView
float []    posicionInicial
float []    posicionFinal
Node        oNodoVRML
Date        dFecha
private SFVec3f
            oTrans
private MFVec3f
```

```
            oRuta
private SFTime
            oTiempoCom
private SFTime
            oIntervalo
private SFTime
            oTiempoComRot
private SFRotation
            oRot
private MFRotation
            oRutaGirar
private SFBool
            oAgenteView
private ConstSFTime
            oTocado
private ConstSFBool
            oDefaultView
private float
            fRotacionAnt
float []    vfCoord
public      ObjMovible ()
public void
            inicializar (String sNom)
public String
            obtenerView ()
public void
            actualizar_Posicion ()
public void
            actualizar_Orientacion ()
public void
            actualizarPosicion (float fX, float fY, float fZ)
public void
            moverPosicion (float fX, float fY, float fZ)
public void
```

```
            avanzarObjeto (float fDist)
public void
            ponerVista ()
public void
            girarObjeto (float fRad)
private double
            calcularTiempo ()
public void
            ligar (Cliente_Agentes oSender)
public void
            impresionNombre ()
```

Clase Receptor_Datos

Esta clase se dedica a escuchar si exisen mensajes que provengan del servidor; cuando existe algún mensaje llama al método que procesa mensajes en Cliente_Agentes.

```
public final static int
            COMIENZO
public final static int
            TAM_PAQUETE
byte        vbyBytesSocket []
DataInputStream
            dis_Entrada
DataOutputStream
            dos_Salida
Cliente_Agentes
            oCliente
            Receptor_Datos (DataInputStream dis_Input,
                            DataOutputStream
                            dos_Output, Cliente_Agentes
                            oClient)
public void
```

run ()

Clase IP_Panel_Client

La clase IP_Panel sirve para crear una ventana en donde se acepten los valores de el host a donde se encuentra el servidor TCP que controlara el ambiente virtual, y también el puerto por donde se realizará la conexión.

Miembros Privados

TextField **tfHost**
TextField **tfPort**
Label **laHost**
Label **laPort**
Button **buOK**
Socket_Applet
oApplet
IP_Panel_Client (Socket_Applet oSender)
private void
mandarDatos ()

Miembros Públicos

public void
principal ()

public boolean
action (Event un_evento, java.lang.Object arg)

Clase Socket_Applet

La clase Socket_Applet es un applet que se pone en un documento HTML, en el se tiene la clase IP_Panel_Client.

Miembros Privados

public final static int
TAM_MENSAJE

IP_Panel_Client
oPanel

TextField **tfMensaje**

Button **buAceptar**

List **liHistoria**

String **sHost**

int **iPort**

DatagramSocket
dsCliente

InetAddress
dirInternet

Miembros Públicos

public **Socket_Applet ()**

public void
start ()

public void
mandarMensaje ()

public boolean
action (Event ev, Object arg)

public void
destroy ()

public void
recibirHost (String sHo, int iPor)

Apéndice B

Archivo house.fac

```
;*********************************
;* Archivo: house.fac            *
;* Autor : Carlos Delgado Mata   *
;*         Noviembre 1998        *
;*                               *
;*********************************
(limit_area house 0. 0. 30.0 0. 30.0 30.0 0.0 30.0)
(dimensions house 30 30)
(polygon wall house wall_1 10.0 0.0 10.20 0.0 10.2 10.0 10.0 10.0)
(polygon wall house wall_2 20.0 0. 20.20 0.0 20.2 7.0 20.0 7.0)
(polygon wall house wall_3 10.0 10. 17.0 10.0 17.0 10.2 10.0 10.2)
(polygon wall house wall_4 20.0 10. 30.0 10.0 30.0 10.2 20.0 10.2)
(polygon wall house wall_5 0.0 10.0 7.0 10.0 7.0 10.2 0.0 10.2)
(polygon wall house wall_6 0.0 15.0 7.0 15.0 7.0 15.2 0.0 15.2)
(polygon wall house wall_7 0.0 20.0 17.0 20.0 17.0 20.2 0.0 20.2)
(polygon wall house wall_8 20.0 20.0 30.0 20.0 30.0 20.2 20.0 20.2)
(polygon wall house wall_9 10.0 13.0 10.2 13.0 10.2 20.0 10.0 20.0)
(polygon wall house wall_10 20.0 13.0 20.2 13.0 20.2 20.0 20.0 20.0)
(polygon wall house wall_11 20.0 20.0 20.2 20.0 20.2 27.0 20.0 27.0)
(polygon wall house wall_12 7.0 13.0 7.2 13.0 7.2 15.0 7.0 15.0)
(polygon wall house wall_13 7.0 15.2 7.2 15.2 7.2 17.0 7.0 17.0)
(limits living-room 10. 10. 20. 20.)
(walls living-room wall_3 wall_10 wall_7 wall_9)
(limits dining-room 20. 10. 30. 15.)
(walls dining-room wall_4 wall_11 wall_7)
```

```
(limits kitchen 20. 15. 30. 20.)
(walls kitchen wall_4 wall_11 wall_7)
(limits childs-bathroom 20.0 0.0 30.0 10.0)
(walls childs-bathroom wall_2 wall_4)
(limits childs-bedroom 10.0 0.0 20.0 10.0)
(walls childs-bedroom wall_1 wall_2 wall_3)
(limits garage 0.0 0.0 10.0 10.0)
(walls garage wall_1 wall_5)
(limits deck 0.0 10.0 8.0 15.0)
(walls deck wall_5 wall_6 wall_12)
(limits storage 0.0 15.0 8.0 20.0)
(walls storage wall_6 wall_7 wall_13)
(limits mothers-bedroom 0.0 20.0 20.0 30.0)
(walls mothers-bedroom wall_7 wall_11)
(limits mothers-bathroom 20.0 20.0 30.0 30.0)
(walls mothers-bathroom wall_11 wall_9)
(limits corridor 8.0 10.0 10.0 20.0)
(joint-points living-room dining-room 17.0 11.0 22.0 11.0)
(joint-points dining-room kitchen 22.0 14.0 22.0 16.)
(joint-points living-room childs-bedroom  18.2 11.0 18.2 9.0)
(joint-points living-room mothers-bedroom  18.2 19.0 18.2 21.0)
(joint-points mothers-bedroom mothers-bathroom 17.0 28.0 22.0 28.0)
(joint-points childs-bedroom childs-bathroom 17.0 8.0 22.0 8.0)
(joint-points living-room corridor 11.0 11.0 8.5 11.0)
(joint-points corridor deck 7.5 11.0 5.5 11.0)
(joint-points corridor storage 7.5 18.0 5.5 18.0)
(joint-points corridor garage 8.2 11.0 8.0 8.0)
(midpoint living-room 16 13 0)
(midpoint childs-bedroom 15 5 0)
(midpoint kitchen 25 17 0)
(midpoint friedge 27 12 0)
(midpoint mothers-bedroom 10 25 0)
(midpoint mothers-bathroom 25 25 0)
(midpoint childs-bathroom 25 5 0)
(midpoint deck 5 12 0)
(midpoint storage 5 17 0)
(midpoint dining-room 24 12 0)
(midpoint outside-door 8 9 0)
; Este es un comentario: arriba Cruz Azul
(midpoint garage 5 5 0)
```

```
(midpoint corridor 8 12 0)
; Aqui van paredes externas (CyC)
(polygon wall house Externa_1 0 10 0.2 10 0.2 30 0 30)
(polygon wall house Externa_2 0 29.8 0 30 30 30 30 29.8)
(polygon wall house Externa_3 29.8 0 30 0 30 30 29.8 30)
(polygon wall house Externa_4 10 0 10 0.2 30 0.2 30 0)
; Para los agentes
(mobile-agent  robot   living-room 16 12 0)
(mobile-agent  agent_1 dining-room 24 12 0)
(mobile-agent  agent_2 kitchen 22 16 0)
(mobile-agent  child childs-bedroom 15 6 0)
(mobile-agent  father mothers-bedroom 10 25 0)
(mobile-agent  mother living-room 12 15 0)
; para los agentes estaticos
(static-object mesa   living-room 12.5 15 0)
(static-object refrigerador kitchen 28.4 19.3 0)
(static-object vestidor     mothers-bedroom  2.4 29.1 0)
(static-object silla        living-room      13 13.5 0)
(static-object silla_2      living-room      12.5 17 0)
(static-object silla_3      dining-room      25 13.5 0)
(static-object silla_4      dining-room      27 13.5 0)
(static-object cama_hijo    childs-bedroom   15 3.2 0)
(static-object cama_mama    mothers-bedroom  3.2 25.0 0)
(static-object mesa_cocina  dinning-room     26.5  12 0)
(static-object mueble_nino  childs-bedroom   12.5 9.3 0)
(static-object silla_nino   childs-bedroom   12.5 6.5 0)
(static-object cabinete     living-room      13   19.4  0)
(static-object mesa_mama    mothers-bedroom  1    22   0)
(static-object mueble_mama  mothers-bedroom  15   29.4  0)
(static-object sillon_mama  mothers-bedroom  14   21   0)
(static-object lava_mama    mothers-bathroom 28.5 25 0.5)
(static-object lava_nino    childs-bathroom  29 5 0.5)
```

Apéndice C

Archivo objetos.dat

```
;*********************************************
;*                                           *
;*  Objetos.dat                              *
;*   Creado por Ing. Carlos Delgado Mata     *
;*   Noviembre 1998                          *
;*********************************************

; Estos son los agentes
robot     robot_azul.wrl 0 0 1 0 2 2 2
agent_1   robot_rosa.wrl 0 0 1 0 2 2 2
agent_2   agente_negro.wrl 0 0 1 0 5 5 5
child     ninio.wrl 0 0 1 0 1.5 1.5 1.5
father    papa.wrl 0 0 1 0 1.5 1.5 1.5
mother    mama.wrl 0 1 0 0 2 2 2
;Estos son los objetos estaticos
mesa      mesa.wrl 1 0 0 0 1.2 1.2 1.2
refrigerador refri.wrl -1 0 0 0 2 2 2
vestidor      vestidor.wrl -1 0 0 0 2.0 2.0 2.0
silla         silla.wrl     0 0 1 3.1416 1.2 1.2 1.2
silla_2       silla.wrl     0 0 0 0      1.2 1.2 1.2
silla_3       silla.wrl     0 0 0 0 1.2 1.2 1.2
silla_4       silla.wrl     0 0 0 0 1.2 1.2 1.2
cama_hijo     cama.wrl      0 0 1 3.1416  2.0 2.0 2.0
cama_mama     cama.wrl      0 0 1 1.5708 2.0 2.0 2.0
mesa_cocina   mesa_rect.wrl 0 0 0 0      1.5 1.5 1.5
```

```
mueble_nino   mueblecajones.wrl 0 0 0 0  1.5 1.5 1.5
silla_nino    sillaroja.wrl     0 0 1 3.1416 1.5 1.5 1.5
cabinete      cabinete.wrl      0 0 0 0      1.5 1.5 1.5
mesa_mama     mesapeque.wrl     0 0 0 0      1.5 1.5 1.5
mueble_mama   mueblecajones.wrl 0 0 0 0      1.5 1.5 1.5
sillon_mama   sillon.wrl        0 0 1 3.1416 1.5 1.5 1.5
lava_mama     lavabo.wrl        0 0 1 -1.5708 2.0 2.0 2.0
lava_nino     lavabo.wrl        0 0 1 -1.5708 1.5 1.5 1.5
```

Apéndice D

Archivo principal.wrl

```
#VRML V2.0 utf8
# Creado con utileria hecha por Ing. Carlos Delgado Mata
# GeneraVRML  version 0.87.7a
 NavigationInfo {
   avatarSize [0.25, 1.6, 0.25]
   headlight  TRUE
   speed 1.0
   type "EXAMINE FLY WALK"
    visibilityLimit 0.0 }

DEF VISTA Viewpoint {
        position      0  0.1  -0.5
     #    orientation  0  0  0 1.0
}


EXTERNPROTO  Obj_Movible [
   exposedField  MFString    urlObjeto
   exposedField  SFVec3f     posicion
   exposedField  SFRotation  orientacion
   exposedField  SFVec3f     escala
   eventIn       MFVec3f     ponerPosiciones
   eventIn       MFRotation  ponerOrientaciones
   eventIn       SFTime      empezarMover
   eventIn       SFTime      empezarGirar
```

```
    eventIn       SFTime      ponerTiempo
    eventOut      SFTime      tocado
    exposedField  SFBool      ponerVistaObj
] "./prot_mov.wrl"


DEF TODO Transform {
    translation   0 0 0
    rotation      -1 0 0 1.5708
    children      [

    DEF PAREDES Transform {
        children  [
         DEF wall_1 Transform {
            children  Shape {
               appearance DEF TEXTURA_PARED Appearance {
                  texture ImageTexture {
                       url "ladrillo.jpg"
                        repeatS TRUE
                       repeatT TRUE
                  }
               } # Fin de appearance
               geometry Box { size  0.20 10.00  3.50 }
            } # Fin de shape
            translation  10.10  5.00  1.75
         } # Fin de Pared
         DEF wall_2 Transform {
            children  Shape {
               appearance USE TEXTURA_PARED
               geometry Box { size  0.20  7.00  3.50 }
            } # Fin de shape
            translation  20.10  3.50  1.75
         } # Fin de Pared
         DEF wall_3 Transform {
            children  Shape {
               appearance USE TEXTURA_PARED
               geometry Box { size  7.00  0.20  3.50 }
            } # Fin de shape
            translation  13.50 10.10  1.75
         } # Fin de Pared
```

```
DEF wall_4 Transform {
    children  Shape {
       appearance USE TEXTURA_PARED
       geometry Box { size 10.00  0.20  3.50 }
     } # Fin de shape
     translation  25.00 10.10  1.75
} # Fin de Pared
DEF wall_5 Transform {
    children  Shape {
       appearance USE TEXTURA_PARED
       geometry Box { size  7.00  0.20  3.50 }
     } # Fin de shape
     translation   3.50 10.10  1.75
} # Fin de Pared
DEF wall_6 Transform {
    children  Shape {
       appearance USE TEXTURA_PARED
       geometry Box { size  7.00  0.20  3.50 }
     } # Fin de shape
     translation   3.50 15.10  1.75
} # Fin de Pared
DEF wall_7 Transform {
    children  Shape {
       appearance USE TEXTURA_PARED
       geometry Box { size 17.00  0.20  3.50 }
     } # Fin de shape
     translation   8.50 20.10  1.75
} # Fin de Pared
DEF wall_8 Transform {
    children  Shape {
       appearance USE TEXTURA_PARED
       geometry Box { size 10.00  0.20  3.50 }
     } # Fin de shape
     translation  25.00 20.10  1.75
} # Fin de Pared
DEF wall_9 Transform {
    children  Shape {
       appearance USE TEXTURA_PARED
       geometry Box { size  0.20  7.00  3.50 }
     } # Fin de shape
```

```
    translation  10.10 16.50  1.75
} # Fin de Pared
DEF wall_10 Transform {
   children  Shape {
      appearance USE TEXTURA_PARED
      geometry Box { size  0.20  7.00  3.50 }
    } # Fin de shape
    translation  20.10 16.50  1.75
} # Fin de Pared
DEF wall_11 Transform {
   children  Shape {
      appearance USE TEXTURA_PARED
      geometry Box { size  0.20  7.00  3.50 }
    } # Fin de shape
    translation  20.10 23.50  1.75
} # Fin de Pared
DEF wall_12 Transform {
   children  Shape {
      appearance USE TEXTURA_PARED
      geometry Box { size  0.20  2.00  3.50 }
    } # Fin de shape
    translation   7.10 14.00  1.75
} # Fin de Pared
DEF wall_13 Transform {
   children  Shape {
      appearance USE TEXTURA_PARED
      geometry Box { size  0.20  1.80  3.50 }
    } # Fin de shape
    translation   7.10 16.10  1.75
} # Fin de Pared
DEF Externa_1 Transform {
   children  Shape {
      appearance USE TEXTURA_PARED
      geometry Box { size  0.20 20.00  3.50 }
    } # Fin de shape
    translation   0.10 20.00  1.75
} # Fin de Pared
DEF Externa_2 Transform {
   children  Shape {
      appearance USE TEXTURA_PARED
```

```
            geometry Box { size 30.00  0.20  3.50 }
          } # Fin de shape
          translation  15.00 29.90  1.75
      } # Fin de Pared
      DEF Externa_3 Transform {
         children  Shape {
            appearance USE TEXTURA_PARED
            geometry Box { size  0.20 30.00  3.50 }
          } # Fin de shape
          translation  29.90 15.00  1.75
      } # Fin de Pared
      DEF Externa_4 Transform {
         children  Shape {
            appearance USE TEXTURA_PARED
            geometry Box { size 20.00  0.20  3.50 }
          } # Fin de shape
          translation  20.00  0.10  1.75
      } # Fin de Pared
    ]  # Fin de chidren paredes.
}   # Fin de transform paredes.
DEF PISOS Transform {
     children  [
      DEF Piso_house Transform {
         children [
            Shape {
            appearance DEF TEXTURA_PISO Appearance {
               texture ImageTexture {
                     url "piso.jpg"
                      repeatS TRUE
                     repeatT TRUE
               }
            } # Fin de appearance
              geometry Box { size 30.00 30.00  0.20 }
            } # Fin de shape
          ]
         translation  15.00 15.00 -0.10
      } # Fin de Piso
      DEF Piso_garage Transform {
         children [
            Shape {
```

```
                appearance DEF TEXTURA_PASTO Appearance {
                   texture ImageTexture {
                      url "pasto.jpg"
                       repeatS TRUE
                      repeatT TRUE
                   }
                  } # Fin de appearance
                  geometry Box { size 10.00 10.00  0.20 }
                } # Fin de shape
                DEF PISO_TOCADO TouchSensor {}
              ]
              translation   5.00  5.00  0.10
            } # Fin de Piso
          ]  # Fin de chidren pisos.
        }   # Fin de transform pisos.
        DEF OBJETOS_ESTATICOS Transform {
          children  [
           DEF STATIC_mesa Transform {
              translation  12.50 15.00  0.00
              rotation      1.00  0.00  0.00 0.0000
              scale         1.20  1.20  1.20
              children  [
                 Inline { url "mesa.wrl" }
              ] # Fin de children
            } # Fin de Objeto Estatico
           DEF STATIC_refrigerador Transform {
              translation  28.40 19.30  0.00
              rotation     -1.00  0.00  0.00 0.0000
              scale         2.00  2.00  2.00
              children  [
                 Inline { url "refri.wrl" }
              ] # Fin de children
            } # Fin de Objeto Estatico
           DEF STATIC_vestidor Transform {
              translation   2.40 29.10  0.00
              rotation     -1.00  0.00  0.00 0.0000
              scale         2.00  2.00  2.00
              children  [
                 Inline { url "vestidor.wrl" }
              ] # Fin de children
```

```
 } # Fin de Objeto Estatico
DEF STATIC_silla Transform {
   translation  13.00 13.50  0.00
   rotation      0.00  0.00  1.00 3.1416
   scale         1.20  1.20  1.20
   children  [
      Inline { url "silla.wrl" }
   ] # Fin de children
 } # Fin de Objeto Estatico
DEF STATIC_silla_2 Transform {
   translation  12.50 17.00  0.00
   rotation      0.00  0.00  0.00 0.0000
   scale         1.20  1.20  1.20
   children  [
      Inline { url "silla.wrl" }
   ] # Fin de children
 } # Fin de Objeto Estatico
DEF STATIC_silla_3 Transform {
   translation  25.00 13.50  0.00
   rotation      0.00  0.00  0.00 0.0000
   scale         1.20  1.20  1.20
   children  [
      Inline { url "silla.wrl" }
   ] # Fin de children
 } # Fin de Objeto Estatico
DEF STATIC_silla_4 Transform {
   translation  27.00 13.50  0.00
   rotation      0.00  0.00  0.00 0.0000
   scale         1.20  1.20  1.20
   children  [
      Inline { url "silla.wrl" }
   ] # Fin de children
 } # Fin de Objeto Estatico
DEF STATIC_cama_hijo Transform {
   translation  15.00  3.20  0.00
   rotation      0.00  0.00  1.00 3.1416
   scale         2.00  2.00  2.00
   children  [
      Inline { url "cama.wrl" }
   ] # Fin de children
```

```
 } # Fin de Objeto Estatico
DEF STATIC_cama_mama Transform {
   translation   3.20 25.00  0.00
   rotation      0.00  0.00  1.00 1.5708
   scale         2.00  2.00  2.00
   children  [
      Inline { url "cama.wrl" }
   ] # Fin de children
 } # Fin de Objeto Estatico
DEF STATIC_mesa_cocina Transform {
   translation  26.50 12.00  0.00
   rotation      0.00  0.00  0.00 0.0000
   scale         1.50  1.50  1.50
   children  [
      Inline { url "mesa_rect.wrl" }
   ] # Fin de children
 } # Fin de Objeto Estatico
DEF STATIC_mueble_nino Transform {
   translation  12.50  9.30  0.00
   rotation      0.00  0.00  0.00 0.0000
   scale         1.50  1.50  1.50
   children  [
      Inline { url "mueblecajones.wrl" }
   ] # Fin de children
 } # Fin de Objeto Estatico
DEF STATIC_silla_nino Transform {
   translation  12.50  6.50  0.00
   rotation      0.00  0.00  1.00 3.1416
   scale         1.50  1.50  1.50
   children  [
      Inline { url "sillaroja.wrl" }
   ] # Fin de children
 } # Fin de Objeto Estatico
DEF STATIC_cabinete Transform {
   translation  13.00 19.40  0.00
   rotation      0.00  0.00  0.00 0.0000
   scale         1.50  1.50  1.50
   children  [
      Inline { url "cabinete.wrl" }
   ] # Fin de children
```

```
 } # Fin de Objeto Estatico
DEF STATIC_mesa_mama Transform {
   translation   1.00 22.00  0.00
   rotation      0.00  0.00  0.00 0.0000
   scale         1.50  1.50  1.50
   children  [
      Inline { url "mesapeque.wrl" }
   ] # Fin de children
 } # Fin de Objeto Estatico
DEF STATIC_mueble_mama Transform {
   translation  15.00 29.40  0.00
   rotation      0.00  0.00  0.00 0.0000
   scale         1.50  1.50  1.50
   children  [
      Inline { url "mueblecajones:wrl" }
   ] # Fin de children
 } # Fin de Objeto Estatico
DEF STATIC_sillon_mama Transform {
   translation  14.00 21.00  0.00
   rotation      0.00  0.00  1.00 3.1416
   scale         1.50  1.50  1.50
   children  [
      Inline { url "sillon.wrl" }
   ] # Fin de children
 } # Fin de Objeto Estaticc
DEF STATIC_lava_mama Transform {
   translation  28.50 25.00  0.50
   rotation      0.00  0.00  1.00 -1.5708
   scale         2.00  2.00  2.00
   children  [
      Inline { url "lavabo.wrl" }
   ] # Fin de children
 } # Fin de Objeto Estatico
DEF STATIC_lava_nino Transform {
   translation  29.00  5.00  0.50
   rotation      0.00  0.00  1.00 -1.5708
   scale         1.50  1.50  1.50
   children  [
      Inline { url "lavabo.wrl" }
   ] # Fin de children
```

```
        } # Fin de Objeto Estatico
    ]  # Fin de chidren Objetos estaticos.
}   # Fin de transform Objetos estaticos.
DEF AGENTE_robot  Obj_Movible {
    urlObjeto     ["robot_azul.wrl"]
    posicion      16.00 12.00  0.00
    orientacion    0.00  0.00  1.00 0.0000
    escala         2.00  2.00  2.00
}
DEF AGENTE_agent_1  Obj_Movible {
    urlObjeto     ["robot_rosa.wrl"]
    posicion      24.00 12.00  0.00
    orientacion    0.00  0.00  1.00 0.0000
    escala         2.00  2.00  2.00
}
DEF AGENTE_agent_2  Obj_Movible {
    urlObjeto     ["agente_negro.wrl"]
    posicion      22.00 16.00  0.00
    orientacion    0.00  0.00  1.00 0.0000
    escala         5.00  5.00  5.00
}
DEF AGENTE_child  Obj_Movible {
    urlObjeto     ["ninio.wrl"]
    posicion      15.00  6.00  0.00
    orientacion    0.00  0.00  1.00 0.0000
    escala         1.50  1.50  1.50
}
DEF AGENTE_father  Obj_Movible {
    urlObjeto     ["papa.wrl"]
    posicion      10.00 25.00  0.00
    orientacion    0.00  0.00  1.00 0.0000
    escala         1.50  1.50  1.50
}
DEF AGENTE_mother  Obj_Movible {
    urlObjeto     ["mama.wrl"]
    posicion      12.00 15.00  0.00
    orientacion    0.00  1.00  0.00 0.0000
    escala         2.00  2.00  2.00
}
     DEF LUZ DirectionalLight {
```

```
            ambientIntensity 0.7
            color            1 1 1
            direction        0 0 -1
            intensity        0.5
            on               TRUE
        }

        DEF SPOT_ROBOT SpotLight {
            attenuation 0 4 3
            intensity   0.6
            location    0 4 -0.7
            direction   0 6 -1.0
            color       1 1 1
            cutOffAngle 0.785393
            radius      10
            on          TRUE
        }

        DEF LUZ_LAMPARA PointLight {
            location    -4.0 1 0.9
            on          TRUE
            intensity   0.7
        }
    ] #Fin de children TODO
}    # Fin de Transform TODO

DEF SCRIPT_MOVER Script {
    url "Cliente_Agentes.class"

    field   SFNode agente_robot USE AGENTE_robot
    eventIn SFTime agente_robot_View
    field   SFNode agente_agent_1 USE AGENTE_agent_1
    eventIn SFTime agente_agent_1_View
    field   SFNode agente_agent_2 USE AGENTE_agent_2
    eventIn SFTime agente_agent_2_View
    field   SFNode agente_child USE AGENTE_child
    eventIn SFTime agente_child_View
    field   SFNode agente_father USE AGENTE_father
    eventIn SFTime agente_father_View
    field   SFNode agente_mother USE AGENTE_mother
```

```
    eventIn SFTime agente_mother_View
    eventIn SFTime pisoTocado
    directOutput            TRUE

    eventOut SFBool       ponerDefaultView
}

# Aqui van los routes

ROUTE PISO_TOCADO.touchTime TO SCRIPT_MOVER.pisoTocado
ROUTE SCRIPT_MOVER.ponerDefaultView   TO VISTA.set_bind

# Routes para Viewpoints
ROUTE AGENTE_robot.tocado TO SCRIPT_MOVER.agente_robot_View
ROUTE AGENTE_agent_1.tocado TO SCRIPT_MOVER.agente_agent_1_View
ROUTE AGENTE_agent_2.tocado TO SCRIPT_MOVER.agente_agent_2_View
ROUTE AGENTE_child.tocado TO SCRIPT_MOVER.agente_child_View
ROUTE AGENTE_father.tocado TO SCRIPT_MOVER.agente_father_View
ROUTE AGENTE_mother.tocado TO SCRIPT_MOVER.agente_mother_View
```

Apéndice E

Archivo prot_mov.wrl

```
#VRML V2.0 utf8

# prot_mov.wrl
#
#   Archivo creado por:
#
#             Ing. Carlos Delgado Mata
#             Septiembre de 1998
#             Aguascalientes, Ags.
#
# Depurado Noviembre de 1998

PROTO OBJETO_MOVIBLE [
   exposedField  MFString    urlObjeto           [".\Robot.wrl"]
   exposedField  SFVec3f     posicion            0 0 0
   exposedField  SFRotation  orientacion         0 0 1 0
   exposedField  SFVec3f     escala              1 1 1
   eventIn       MFVec3f     ponerPosiciones
   eventIn       MFRotation  ponerOrientaciones
   eventIn       SFTime      ponerTiempo
   eventIn       SFTime      empezarMover
   eventIn       SFTime      empezarGirar
   eventOut      SFTime      tocado
   eventIn       SFBool      ponerVistaObj
]
```

```
{
   DEF OBJETO_MANIPULADO Transform {
        translation IS posicion
        rotation    IS orientacion
        scale       IS escala

        children [
            Inline {url IS urlObjeto}
            DEF OBJETO_TOCADO TouchSensor
                        { touchTime IS tocado }
            Transform {

              children [
                  rotation  1 0 0 1.5708
                  DEF OBJETO_VIEW   Viewpoint {
                   position    0.30 0.65 0.0
                    orientation 0 0 1 1.5708
                    set_bind    IS ponerVistaObj
                  }
               ]
            } #Fin de Transform OBJETO
    ] # Fin de children de el objeto movible.
   } # Fin de Transform

    DEF INTP_MOVIMIENTO PositionInterpolator {
       key [0 , 1]
       keyValue [0 0 0 , 1 0 0]
       keyValue IS ponerPosiciones
    }

    DEF INTP_ROTACION  OrientationInterpolator {
       key [0, 0.25, 0.5, 0.75, 1]
       keyValue IS ponerOrientaciones
    }

    DEF TIEMPO_MOV_OBJ TimeSensor {
      cycleInterval     3
      cycleInterval IS ponerTiempo
      loop              FALSE
```

```
  startTime   IS empezarMover
  #  stopTime 1

}
DEF TIEMPO_GIR_OBJ TimeSensor {
  cycleInterval  2
  loop           FALSE
  startTime IS empezarGirar

}

ROUTE TIEMPO_MOV_OBJ.fraction_changed
   TO INTP_MOVIMIENTO.set_fraction

ROUTE INTP_MOVIMIENTO.value_changed
   TO OBJETO_MANIPULADO.set_translation

ROUTE TIEMPO_GIR_OBJ.fraction_changed
   TO INTP_ROTACION.set_fraction

ROUTE INTP_ROTACION.value_changed
   TO OBJETO_MANIPULADO.set_rotation
```