xx (12578 2.1)

CINVESTAV

Centro de Investigación y de Estudios Avanzados del I.P.N.
Unidad Guadalajara

Lenguaje para Animación de Criaturas Virtuales

Tesis que presenta:

Alma Verónica Martínez González

para obtener el grado de:
Maestro en Ciencias

en la especialidad de:
Ingeniería eléctrica

Director de Tesis

Dr. Félix Francisco Ramos Corchado

Guadalajara, Jalisco, Agosto del 2005.

F.D. 125377 - 2001

Lenguaje para Animación de Criaturas Virtuales

Tesis de Maestria en Ciencias
Ingeniería eléctrica

Por:
Alma Verónica Martínez González
Ingeniero en Computación
Universidad de Guadalajara 1998-2002

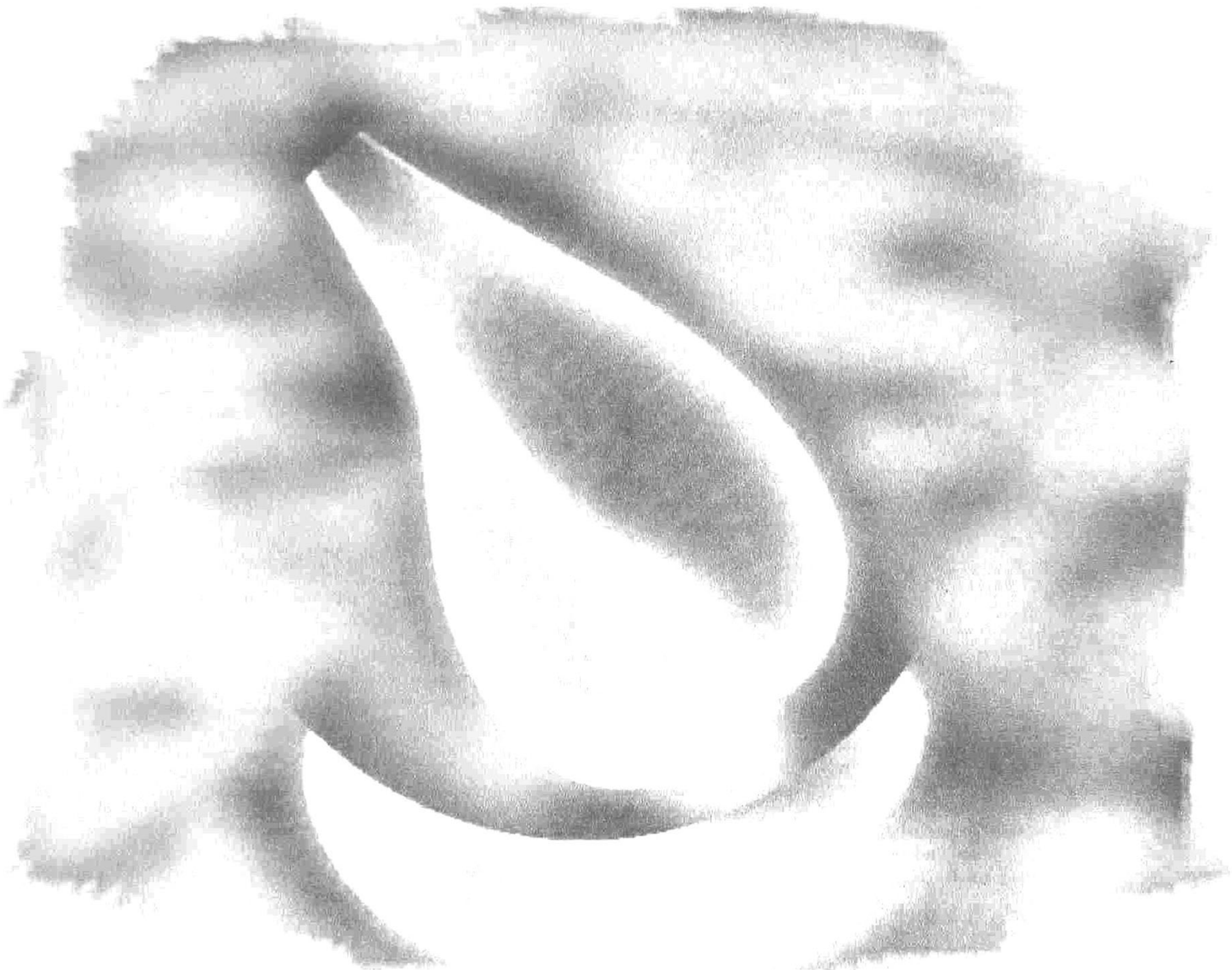
Becario de Conacyt, expediente no. 180844

Director de Tesis
Dr. Félix Francisco Ramos Corchado

CINVESTAV del IPN Unidad Guadalajara, Agosto del 2005.

ÍNDICE GENERAL

Capítulo 1 Introducción



Resumen

Se describirá el objetivo de esta tesis, la descripción a grandes rasgos del trabajo realizado para alcanzar dicho objetivo, así como la organización del presente trabajo.

1.1 Descripción del problema

Actualmente existe el proyecto nombrado GeDA-3D (Ambiente Genérico Virtual Distribuido) [1.1, 1.2, 1.3, 1.4], cuya arquitectura se muestra en la Fig. 1.1. Está arquitectura puede descomponerse en módulos por funcionalidad en Editor de Escena cuya funcionalidad es describir una escena en un lenguaje parecido al natural. El módulo de contexto cuyo objetivo es describir conceptos y leyes que rigen la escena que se describe. Un módulo de Control de Escena y Control de Agentes que conforman el núcleo de la arquitectura y es en realidad un midleware distribuido. Un conjunto de Agentes que pudiéramos llamar módulo de comportamiento y realización de una escena. Finalmente un módulo de rendering el cual es el encargado de efectuar la visualización de una escena en 3D.

La presente tesis se refiere al trabajo que se realizo para la realización de este último modulo de rendering.

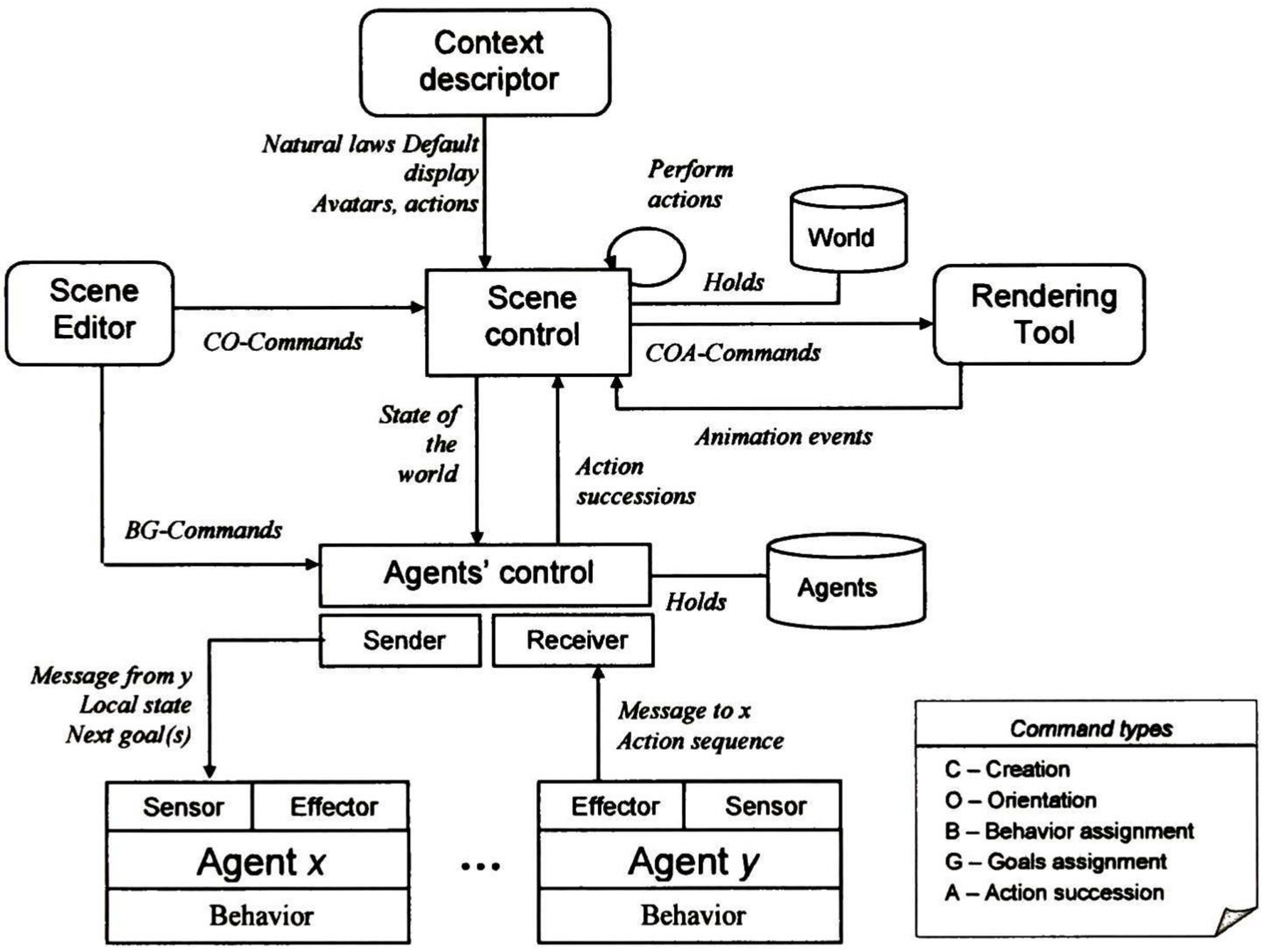


Fig. 1.1: Arquitectura de GeDA-3D

En un trabajo previo, este módulo está implementado en Java [1.5] y Java 3D [1.6]. La forma de manipular los avatares contenidos en un escenario es por medio de animaciones previamente diseñadas. Es decir, para realizar una animación de una persona corriendo, se diseñan los movimientos necesarios, los cuales son almacenados en imágenes fijas. De esta forma se muestra una secuencia de imágenes que da la apariencia que el avatar esta corriendo.

Sin embargo al manejar los avatares de esta forma no permite tener control de cada una de sus partes. Por lo cual la evolución del escenario de esta manera resulta con muy poco realismo. Por otro lado los objetos (avatares) que se manejan tienen una apariencia poco realista, pues son creados en un editor que quedo en desarrollo.

1.2 Objetivo de la tesis

El objetivo principal es realizar un estudio y concluir sobre la posibilidad de proveer a GeDA-3D de módulo que realice la tarea de rendering descrita previamente y que cumpla con los siguientes requerimientos:

- Permitir la visualización de un escenario 3D en tiempo real.
- Permitir la manipulación de cada parte de un avatar u objeto 3D para obtener animaciones realistas.
- Ser compatible con GeDA-3D.

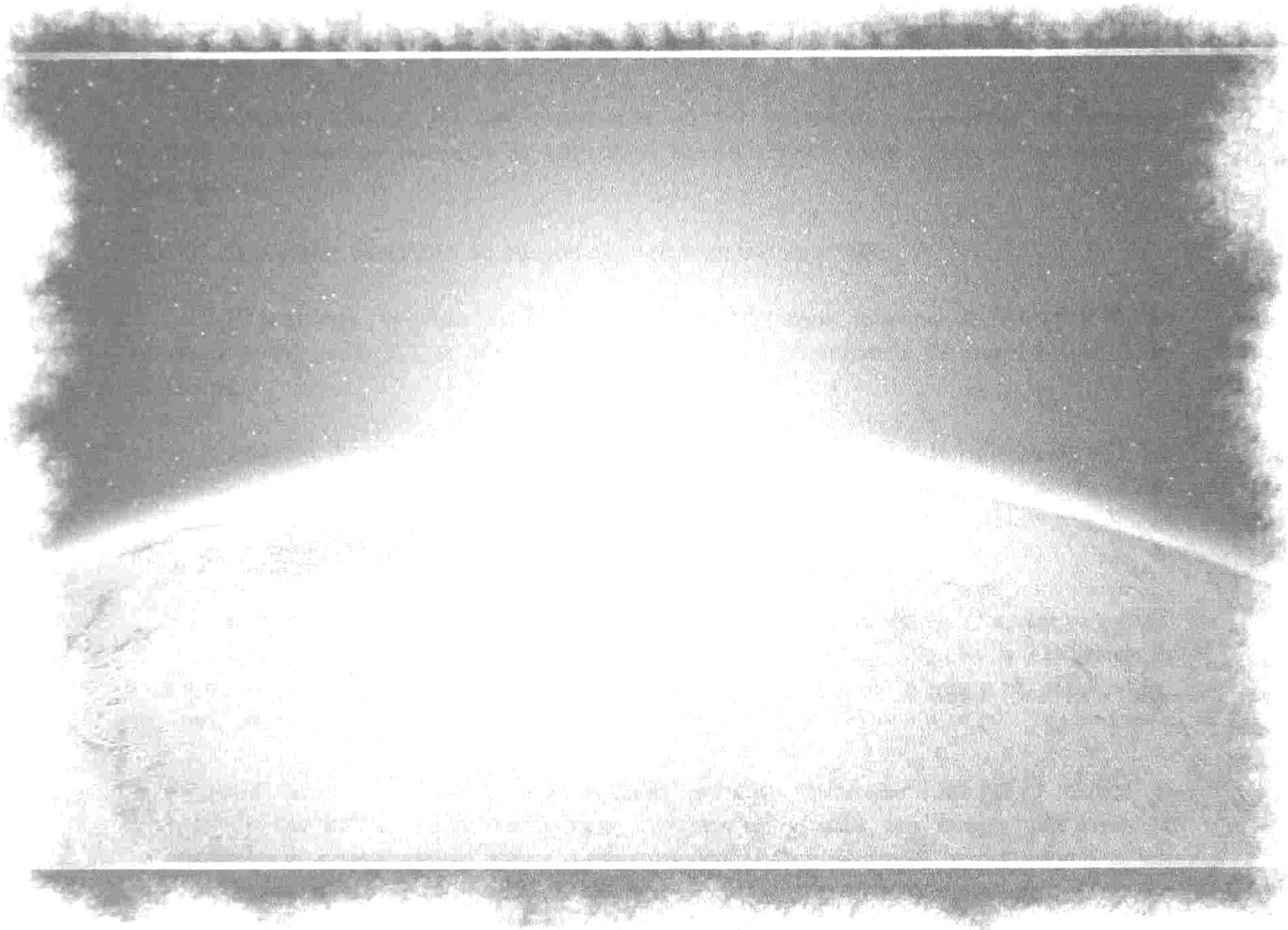
La manera a groso modo de abordar este problema es la siguiente y será detallada a lo largo del presente manuscrito es la siguiente:

Se realizó una revisión del estado del arte con el objetivo de conocer los trabajos relacionados y conocer los diferentes tipos de software que permiten el manejo y creación de objetos en 3D. De este estado del arte se concluyo con una clasificación del software disponible en: Editores 3D los cuales sirven para diseñar objetos y ambientes estáticos. Los Motores de render, los cuales son descritos en el capitulo 2. y finalmente los Motores de juegos o motores de 3D, en estos existe un grupo los cuales permitían cubrir los aspectos requeridos.

Sin embargo la herramienta no satisface los requerimientos con la plataforma, por lo que se diseño un lenguaje que permite la manipulación de un escenario 3D, el cual es independiente de un lenguaje de programación y de un motor de 3D. Este lenguaje nos permite tener una independencia tecnológica y una capacidad de adaptación incremental de la potencia del sistema propuesto.

La organización de este trabajo de tesis de maestría es la siguiente: El capitulo 1 hace una introducción de este trabajo. El capitulo dos está dedicado a presentar el estado del arte. El capitulo 3 presenta nuestro lenguaje para la manipulación de animaciones LIA·3D. El capitulo 4 presenta la arquitectura para visualización 3D que denominamos AVE·3D. El capitulo 5 presenta los resultados obtenidos de nuestra propuesta. Finalmente el capitulo 6 describe nuestras conclusiones y trabajos futuros.

Capítulo 2 Estado del Arte



Resumen

El objetivo de este capítulo es hacer una revisión de los sistemas comerciales y de investigación que existen y que pudieran satisfacer los requerimientos de GeDA-3D de mostrar en tiempo real la animación de un ambiente virtual.

2.1 Introducción

GeDA-3D es un proyecto que tiene como objetivo ofrecer usuarios inexpertos la posibilidad de generar escenarios por medio de una descripción tanto del escenario mismo como de los actores que participan, es decir un escenario esta constituido de actores autónomos y objetos. Una vez que el usuario termina su descripción tiene la posibilidad de efectuar la simulación de dicho escenario.

Para hacer una simulación realista, los objetos tienen que percibirse lo más realista posible, y los actores además de tener una apariencia real tienen que imprimir realismo en sus acciones. A continuación presentamos la revisión de trabajos y productos que posiblemente nos ayudarían a efectuar este trabajo. Pero antes haremos una revisión de definiciones que son necesarias.

Avatar: Objeto 3D que representa un ser animado, en general un humano virtual, sin embargo también se utiliza el término para otros seres vivos reales o fantasiosos.

Esqueleto: Conjunto de huesos que representan un avatar.

El software revisado puede clasificarse en tres tipos: editores de 3D, motores de render y motores de 3D. A continuación describimos brevemente las características de cada tipo.

Editores de 3D: Este software esta diseñado para el diseño de objetos de 3D. La mayoría de ellos nos permite crear animaciones, las cuales se pueden exportar a un formato de video (por lo general avi), dos de los software mas conocidos son Maya [2.1] y 3D Studio Max [2.2].

Motores de render: Este tipo de software se encarga de hacer el render es decir se encarga de producir visualmente la animación o imagen. Los editores de 3D incluyen cuando menos uno de ellos, este tipo de software genera un solo formato de salida (jpg, tga, avi, etc.).

Motores de 3D: Este tipo de software se caracteriza por realizar el render en tiempo de ejecución. Existen diversos motores de juegos los cuales permiten la importación de objetos o avatares de editores de 3D. Los avatares importados pueden ser utilizados dentro de un escenario. La mayoría de estos motores ejecutan una animación predefinida en un editor de 3D para mostrar una modificación en un avatar. Esto es, un motor de 3D permite hacer el cambio de animación en tiempo de ejecución.

Los dos primeros tipos de software antes mencionados, generan imágenes en los formatos mas comunes (bmp, jpg, tga, mpg), las cuales no se pueden manipular, por lo cual se busco un tipo de software que permitiera modificar las imágenes o representación de humanos virtuales (avatares) en tiempo de ejecución. Estas características no nos permiten hacer la manipulación de los avatares por lo que nuestro campo de búsqueda se redujo al tercer tipo de software para alcanzar el objetivo de esta tesis.

Dentro de los motores de 3D se busco como característica esencial, que el motor permitiera la manipulación de las partes de un avatar, esto es, permitiera el control de huesos del avatar en tiempo de ejecución.

2.2 Revisión de Software 3D

En las siguientes páginas se mostrara una lista de software que se estudio para mejorar la interfaz de GeDA-3D haciendo una breve descripción de sus características principales y revisando su utilidad para nuestro trabajo.

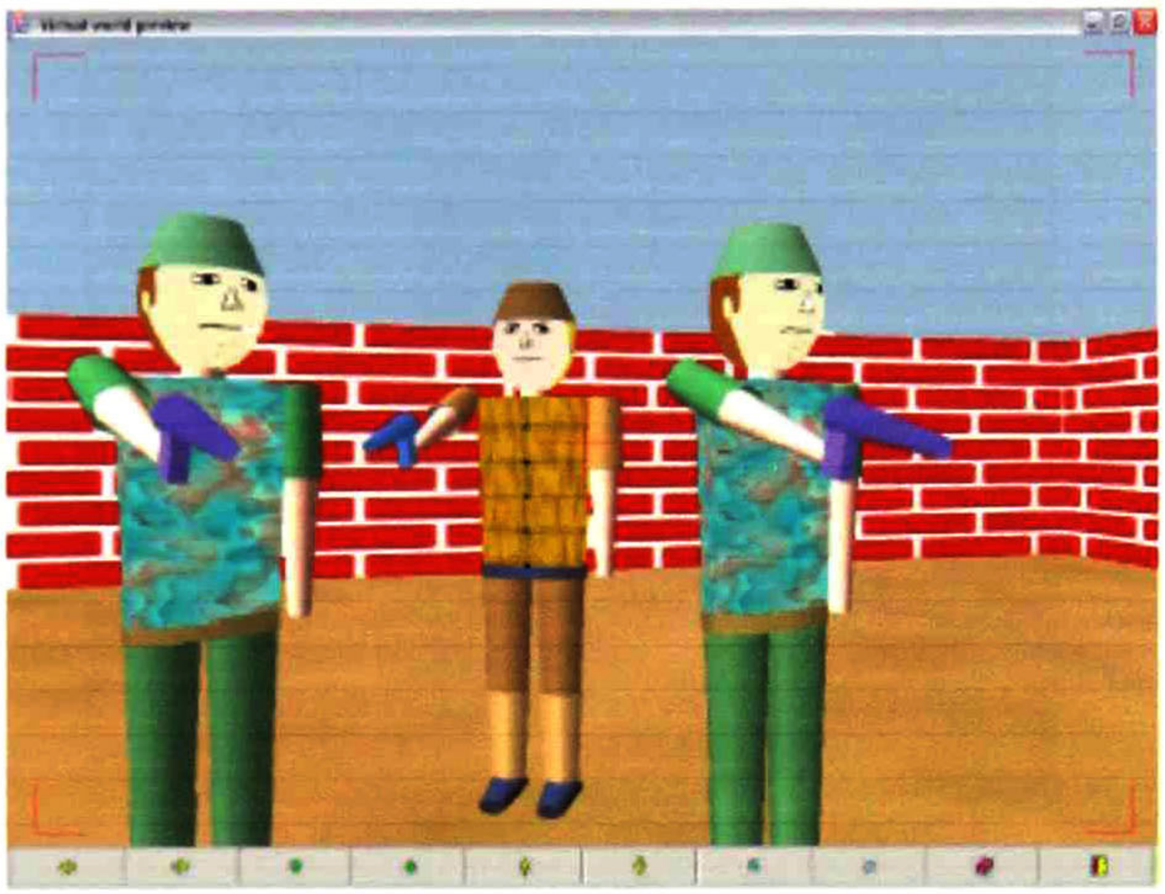


Fig. 2.1: Interfaz anterior de GeDA -3D

2.2.1 Editores de 3D

2.2.1.1 Art of Illusion

Art of Illusion [2.3] es un editor de 3D hecho completamente en Java, la primer versión fue hecha en 1999, por lo cual existen pocos plug-in y manuales para su manejo, la página oficial cuenta con un tutorial de su interfaz, la cual es una de las más sencillas de los software de 3D.

Entre sus características principales están: ofrece el manejo de luz, cámaras, texturas, etc., permite la incorporación de script hechos en Beanshell [2.4] el cual es un lenguaje muy similar a Java, con estos script puede ahorrarse varios pasos en la creación de un diseño o una animación, ya que permite guardar, abrir o editar un script para posteriormente cargarlo en el diseño en el que se desea aplicar.

Art of Illusion es un software el cual da como resultado una imagen prediseñada (en formato jpeg, gif, etc.), por lo cual no puede manejar colisiones, ni comportamiento de los avatares, tampoco maneja esqueletos estás características son necesarias en nuestro proyecto.

2.2.1.2 Game Maker

Es un editor de juegos en 2D. Sus principales características incluyen: una programación es sencilla, algo similar a Visual Basic. Su interfaz es sencilla, no requiere de tener conocimientos en editores de 3D. Actualmente se esta ampliando para soportar gráficos en 3D.

El soporte para las herramientas de 3D aun está en prueba, pero cuando esto se finalice será un editor bastante simple en su utilización. La ayuda es un poco difícil de conseguir, y existen pocos ejemplos dentro de Game Maker [2.5].

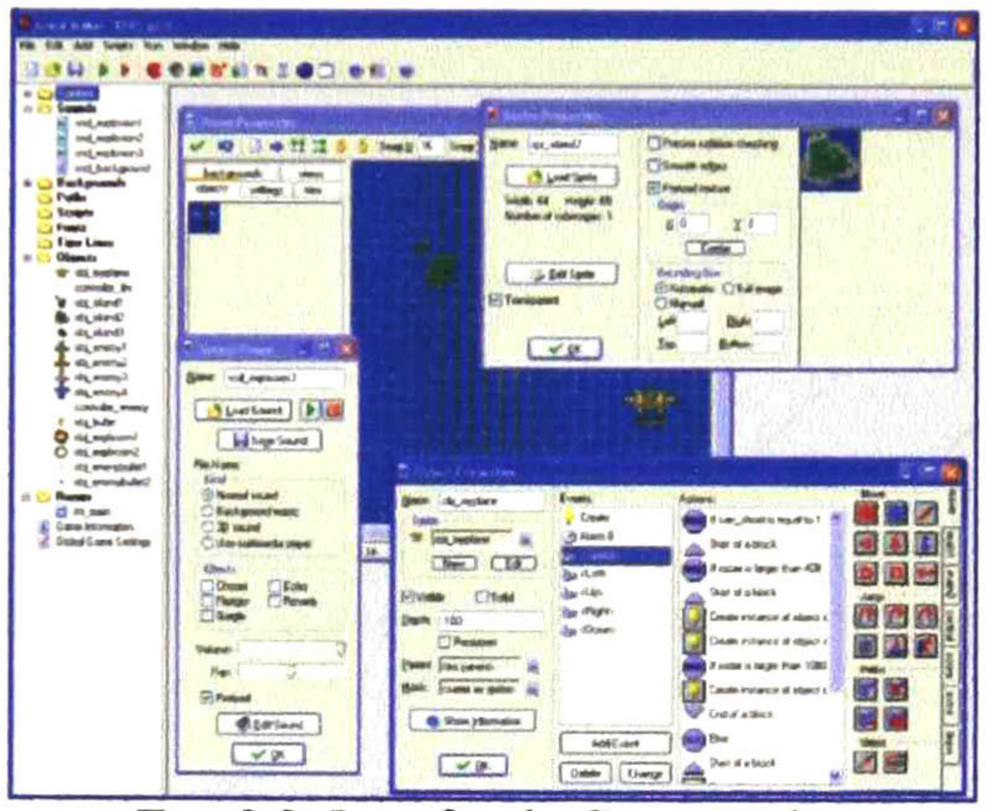


Fig. 2.2: Interfaz de Game Maker

Game Maker es una herramienta aunque de sencilla utilización, requiere de conocimientos de programación. Puede importar diseños de editores de 3D, se considera una herramienta para 2D. No hace manejo de esqueletos, de colisiones ni de luz.

2.2.1.3 GMax

GMax [2.6] es un editor 3D, permite texturas, modelado, animación, su interfaz es fácil de entender ya que es similar a la mayoría del software para diseño de 3D. Permite exportar al formato de 3D Studio Max.

Una de las principales características es que soporta la cinemática inversa, permite crear script para la exportación, su documentación es sencilla, contiene ejemplos los cuales muestran las principales ventajas de este editor.

Al ser un editor de 3D resulta no ser útil para lograr nuestro objetivo.

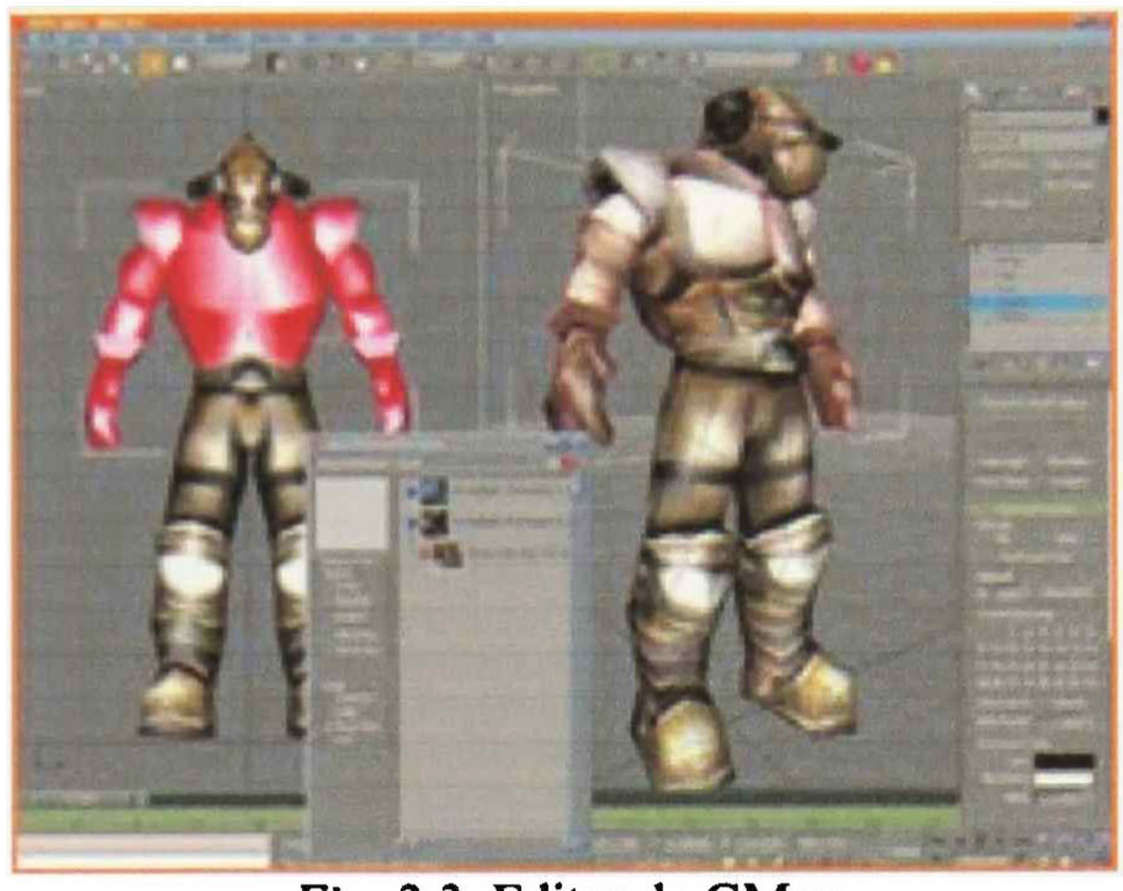


Fig. 2.3: Editor de GMax

2.2.1.4 Maya 6

Maya 6 [2.1] es un editor 3D que permite la manipulación de objetos en 2D y 3D, cuenta con un API/SDK para el desarrollo de plug-in para la posible incorporación con otros software, así como una herramienta de nombre MEL (Maya Embedded scripting Language) la cual nos permite incorporar script para la creación de animaciones.

Este es un software de los más utilizados para el diseño de imágenes 3D, ya que permite la manipulación de texturas, cámaras, luz y colisiones, entre otras cosas.

La curva de aprendizaje del Maya 6 es relativamente larga debido a que su interfaz grafica no es intuitiva y requiere de conocimientos previos o de cursos especiales para comenzar a utilizarlo.

En su herramienta para manipulación de las propiedades de los objetos, requiere de tener conocimientos de programación orientada a objetos, así como de amplio conocimiento de C++, ya que el MEL y API son muy similares a C++.

Puesto que es un editor, las imágenes o animaciones que generan tienen un formato predefinido (avi, jpg, etc).

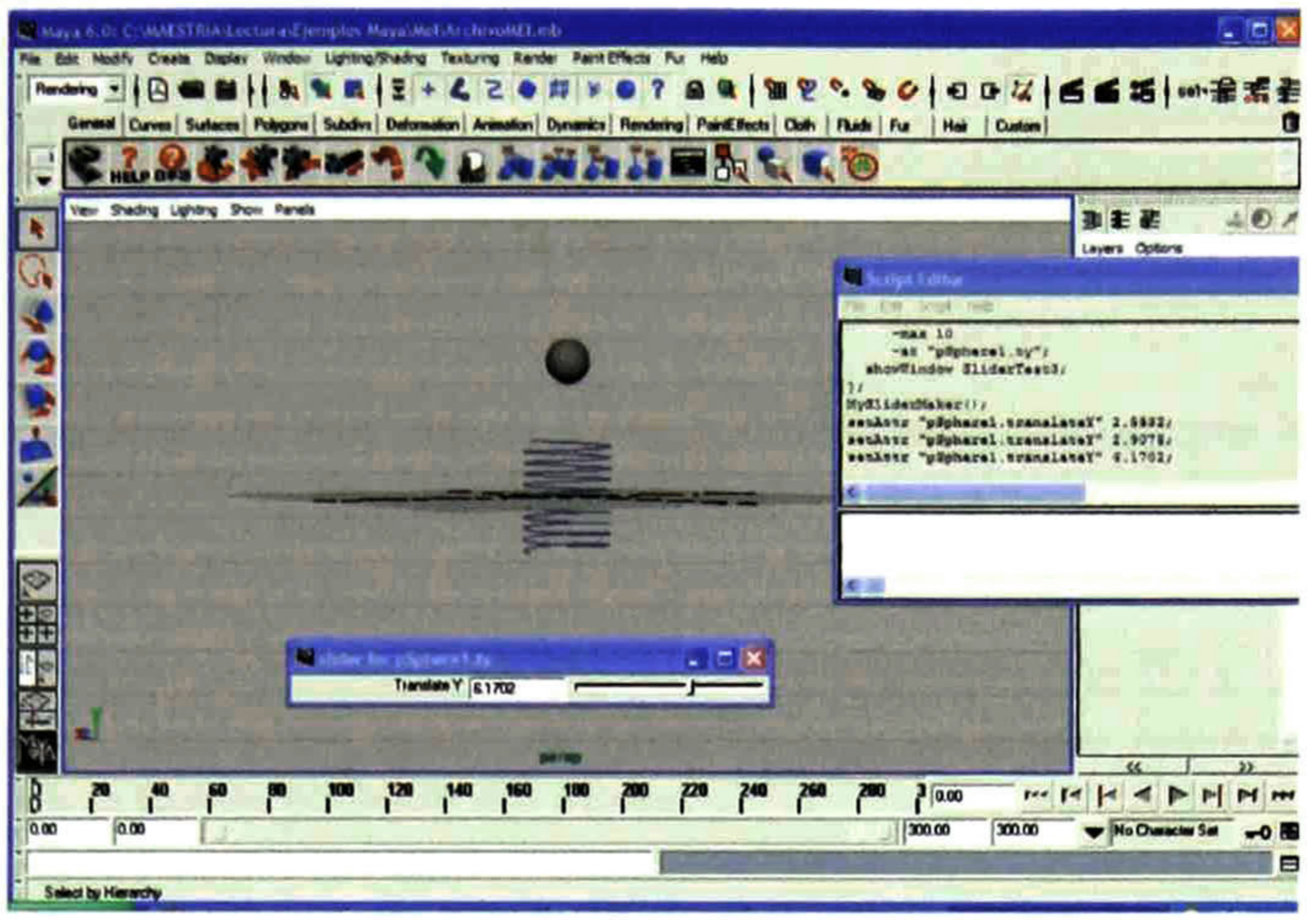


Fig. 2.4: Editor de Maya 6

2.2.1.5 MilkShape 3D

MilkShape 3D [2.7] Es un editor 3D bastante utilizado para el diseño de avatares utilizados en diversos juegos uno de ellos Unreal, también es utilizado como complemento de motores de 3D como Nebula [2.8]. Una de sus mayores ventajas es que provee varios formatos tanto para importar como para exportar.

MilkShape 3D soporta el manejo de cámaras, luces, el manejo de esqueletos, sus primitivas están basadas en cajas, esferas y cilindros entre otros.

Existe un archivo de ayuda que muestra los pasos a seguir para realizar diversos diseños, pero será un poco difícil de seguir si no se cuenta con conocimientos previos de modelado en 3D.

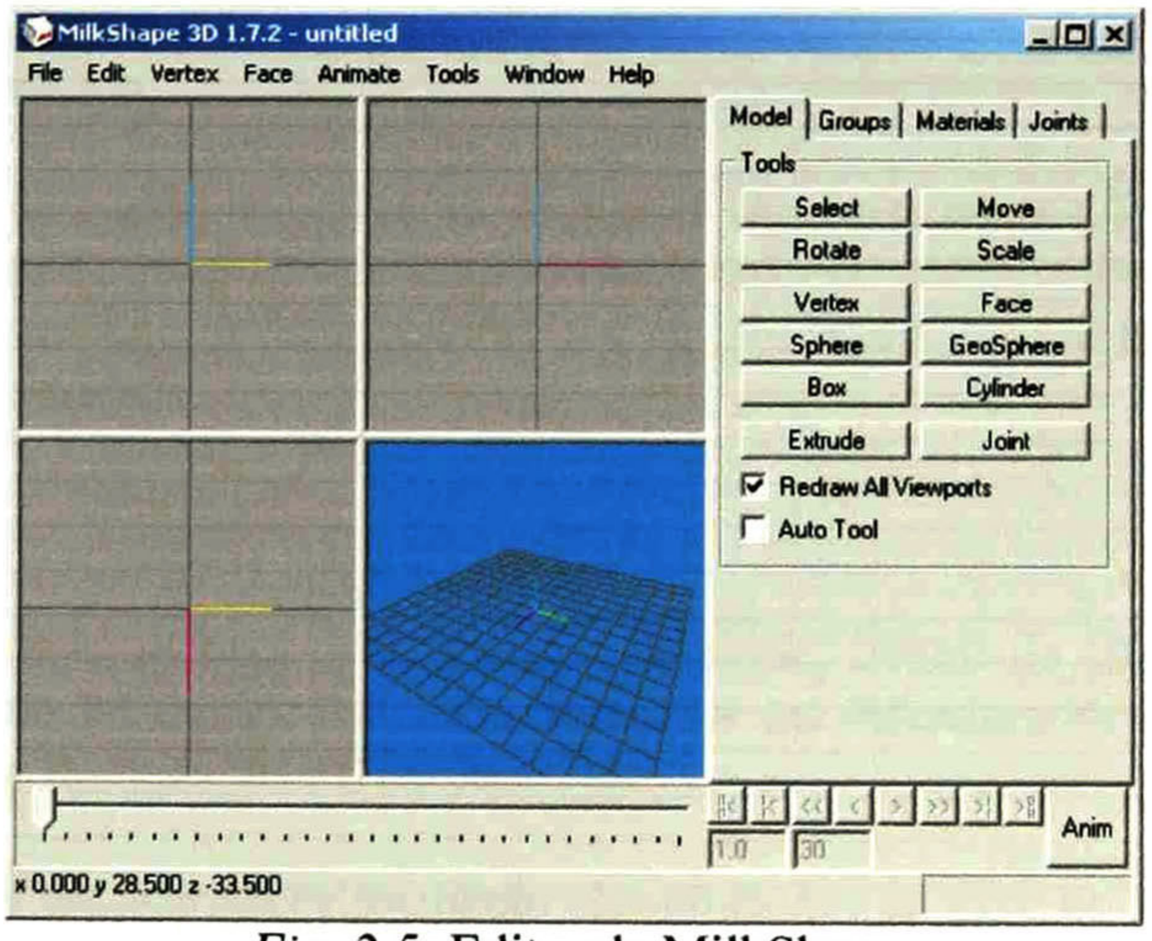


Fig. 2.5: Editor de MilkShape

2.2.1.6 Swift 3D

Swift 3D [2.9] es un editor 3D que permite la exportación de los avatares a Flash MX, la versión más reciente es 4, la cual tiene una interfaz fácil de comprender, similar a otros editores de 3D.

Swift 3D permite texturizar, aplicar luces, modelar, animaciones por frame (tipo flash), incluye script de JAVA, los cuales son utilizados para ahorrar trabajo en futuros diseños.

Al tiempo de exportar a Flash el objeto pierde su apariencia de 3D a un objeto en 2D, por lo que en una animación dentro de Flash se crea un nuevo Keyframe con un objeto que representa la nueva posición del avatar.

2.2.2 Motores de 3D

2.2.2.1 AgentFX™ 2

AgentFX™ 2 [2.10] es un conjunto de librerías (plataforma) 3D que ofrece características avanzadas para la generación de escenarios, tales como sombras en tiempo real, detección de colisiones también en tiempo real, manejo de texturas, animaciones de esqueletos, la carga de animaciones predefinidas, una combinación de una animación predefinida con una de esqueleto, lo cual permite tener mas control sobre los movimientos de los avatares. Además es multiplataforma ya que está desarrollado en Java.

AgentFX™ 2 contiene plug-in para importar los caracteres de los principales editores de 3D (3d Studio, Maya, Cinema 4D, Milkshape 3D, etc.). Su arquitectura permite el acceso directo a las funciones de OpenGL. El motor es rápido y fácil de utilizar por lo que se puede utilizar para una variedad amplia de usos como son juegos industriales y de simulación.

El motor esta en continua evolución y algunas versiones no son compatibles. Su uso requiere de un experto en el uso de las librerías y tener conocimientos de animación como son colisiones, manejo de iluminación, por lo que este software no es aplicable directamente a nuestro trabajo.

2.2.2.2 Blitz3D

Es un sistema que se basa en la sintaxis de Basic, dentro de Blitz3D[2.11] se manejan entidades, con las cuales podemos manipular las diferentes características de un ambiente 3D, como texturas, luces, cámaras, etc.

La creación de los objetos es mediante código lo que lo hace un poco difícil de visualizar en el momento de programación, la sintaxis del código es relativamente fácil de entender, las texturas se generan en base a imágenes prediseñadas.

Blitz3D permite tener una imagen prediseñada para poder asignarla a un objeto esto trae ventajas para las personas que no desean aprender a modelar en un software de diseño de 3D. Una sola imagen puede tener diferentes texturas, sin embargo es necesario conocer las coordenadas de donde inicia y terminan, para poder asignarlas al objeto. El número de comandos es pequeño en comparación al de otros lenguajes, lo que permite que sea mas fácil de recordar, con pocas líneas de código se puede crear un nuevo juego.

La creación de objetos y avatares por medio de código precisa de un experto en programación, Otra desventaja es que no le podrá dar un aspecto diferente a objetos del mismo tipo, por ejemplo si desea crear un libro de color verde y otro de color azul, deberá de tener dos imágenes para la textura de estos libros, lo cual hará el programa aun mas grande. Además el software no maneja esqueletos. Estas son las características por las que no resulta viable su uso en nuestro proyecto.

2.2.2.3 Cal3D

Cal3D [2.12] es un motor de 3D y esta dividido en una biblioteca de C++ y un exportador. La biblioteca 3D escrita en C++. El "exportador" permite la importación de objetos 3D de editores como 3D Studio.

El diseño del escenario se hace mediante la biblioteca de C++ la cual puede hacer el manejo de esqueletos, manejo de luz. Las animaciones pueden ser cargadas de modelos diseñados en uno de los editores (3D Studio Max, etc) por medio de su "exportador". Quizás una de las ventajas más importantes es que se pueden mezclar animaciones predefinidas con animaciones en tiempo real, tiene varios archivos de ejemplo, pero la ayuda es escasa.

Cal3D es una herramienta que requiere un experto en programación para su uso, además no tiene manejo de colisiones (aunque por el uso de tan bajo nivel se pudiera implementar a mano). Finalmente, debido que es muy nuevo el producto, la ubicación de objetos y avatares dentro de un escenario no es muy precisa.

2.2.2.4 Game Studio

Game Studio [2.13] es un set de herramientas: Un Editor de 3D, un Motor de Juegos de 3D, un Editor de script y un Editor de Scenarios.

El Editor 3D es útil para el diseño de avatares. El Editor de script para la programación. El Editor de escenarios para el diseño del ambiente del juego, y finalmente el Motor 3D para ejecutar el juego. El conjunto permite la manipulación de avatares en tiempo de ejecución además del manejo de partículas (gotas de lluvia, balas, etc.) para realizar diferentes efectos.

Permite el diseño de esqueletos, de esta forma se puede asignar un nombre cada una de las partes del esqueleto (huesos) para animarlos en tiempo de ejecución. Esto se realiza en el editor de 3d incluido, este soporta texturas, modelado, etc.

Dentro del editor de script, el lenguaje que se maneja es script de C, con los cuales podemos detectar colisiones, animar un avatar, mandar llamar una animación

predefinida. Permite identificar las palabras reservadas, depurar, ver la ayuda de la función que se esta utilizando solo con poner el cursor sobre el nombre de la función.

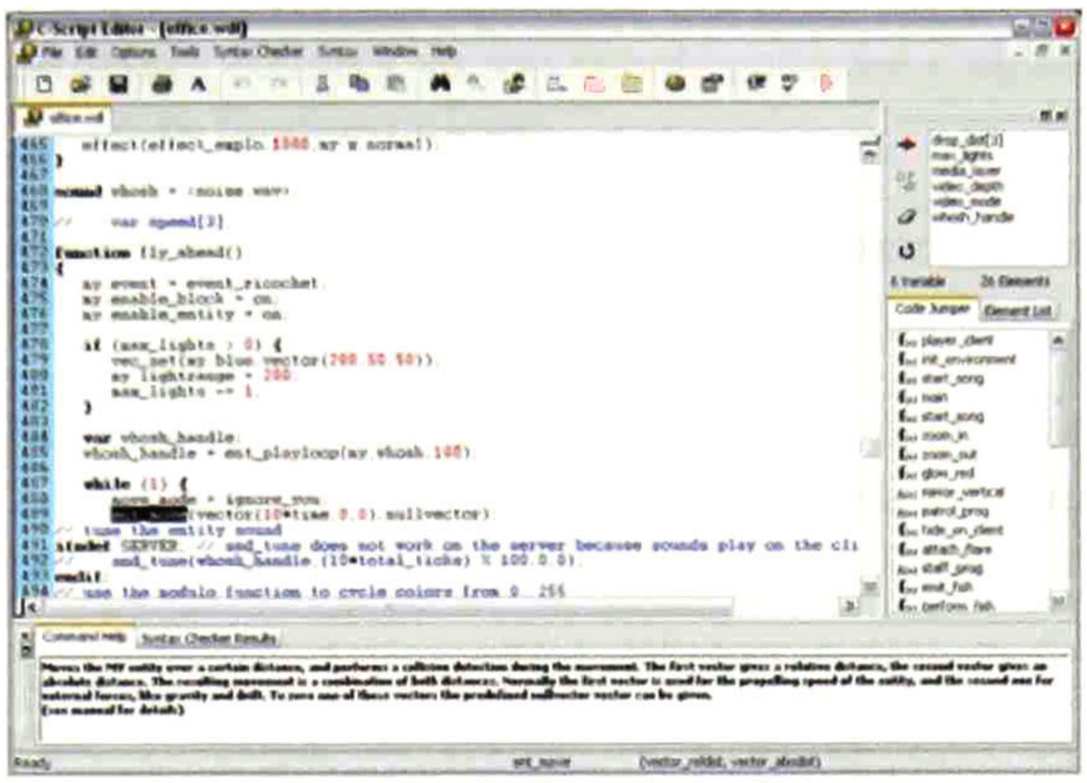


Fig. 2.6: Editor de script

El editor de juego permite el diseño del ambiente, edificios, terrenos, luces, incluir sonido, incluir los avatares a utilizar y los archivos que incluyen los script.

Este software también permite la creación de los archivos de código en Visual C++, por lo que se puede tener mayor manejo y comunicación en el juego.

La ayuda existente es abundante, cuanta con varios archivos de ayuda para los tres editores, el foro es bastante amplio por lo que la ayuda es accesible.

El software está en C lo cual limita a un cierto tipo de plataforma. Actualmente existe un demo de 30 días de este software, si uno desea continuar con este software necesita comprar la versión completa y su costo es considerable.

2.2.2.5 GameLet

GameLet [2.14] Es un motor de 3D hecho en Java, su estructura esta basada en agentes, lo cual lo hace diferente a los demás, ya que en lugar de hacer mención a objetos, avatar o caracteres hace más fácil el comprender su funcionamiento.

El manejo de colisiones y animaciones es por medio de programación, por lo que se debe tener conocimientos en estos temas.

Para su rápido aprendizaje es necesario tener conocimientos en programación orientada a objetos y Java. La ayuda no es muy abundante pero es sencilla, existen algunos ejemplos que muestran sus propiedades.

Las animaciones se generan por medio de un archivo de imagen, donde se muestran los movimientos que tendrá el agente (avatar), esta imagen es cargada en una matriz, por lo cual no se pueden manipular las animaciones el agente (avatar) en tiempo de ejecución.

2.2.2.6 GameSpace

GameSpace [2.15] cuenta con un editor de 3D muy completo, su interfaz es totalmente diferente a las demás editores, es sencillo de entender y no consta de muchos menús que nos confundan en su funcionamiento.

Puede crear animaciones directamente sobre el esqueleto del avatar, esta tarea puede ser apoyada por una barra de tiempo muy similar a la utilizada en Flash [2.16].

Existe la posibilidad de programar animaciones de avatares sencillos por medio del lenguaje Python [2.17], para esto se cuenta con una ventana sencilla en la cual podemos ver el código y ejecutarlo, mientras en el espacio de trabajo se ve lo obtenido del resultado del código.

Es posible obtener el editor de forma libre, su imitante es el numero de polígonos (650) que se pueden utilizar para el diseño de caracteres, no maneja esqueleto. La ayuda existente no es abundante por lo que se requiere tener conocimientos de programación y tecnicismos de 3d.

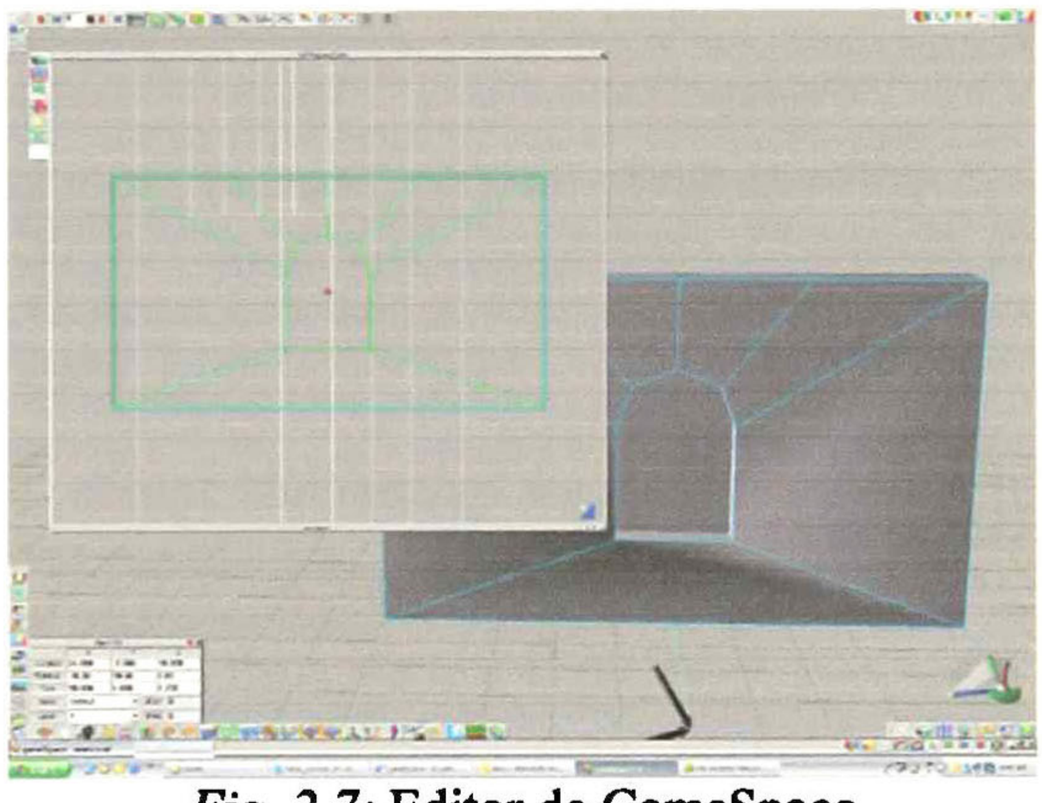


Fig. 2.7: Editor de GameSpace

2.2.2.7 Genesis3D

Genesis3D [2.18] es un motor de 3D de código abierto para del desarrollo del juego 3D, con una licencia diseñada para permitir el su uso en juego tanto comerciales como no comerciales.

La forma de hacer animaciones puede ser importada de diversos editores de 3D (3D Studio, Milksaphe, etc.), su programación también puede ser en varios lenguajes (Visual C++ 6 [2.19], Borland C++[2.20], Delphi[2.21], Visual Basic[2.22]), lo cual hace que más gente pueda utilizar este motor.

Contiene las características principales de un motor de 3D: manejo de colisiones, iluminación, importa de varios tipos de editores, debido a que es un código abierto esta en constante desarrollo.

Desarrollar un juego todavía requiere el conocimiento avanzado de programar, así como artistas expertos, modeladores, músicos, y a otros especialistas en el campo de juegos en 3D. La modelación de avatares no se basa en esqueletos.

2.2.2.8 Idx3d

Idx3d [2.23] es un motor de 3D creado en Java. Entre sus características están el manejo de texturas, cámaras, luces, vértices, etc.

Los objetos pueden ser importados de 3D Studio Max [2.2], por lo que las animaciones son predefinidas en un editor. La ayuda es poca, solo se encontró una explicación de los ejemplos existentes, por lo cual requiere de tiempo para lograr tener un juego completo. El motor no maneja esqueletos, por lo que las animaciones de avatares son prediseñadas.

2.2.2.9 Irrlicht

Irrlicht [2.24] un motor de 3D, el cual esta escrito en C++, Irrlicht es de código abierto, es ampliamente utilizado para el diseño de juegos en OpenGL [2.25] y Directx [2.26], cuenta con ayuda y foros de programación.

Soporta luces dinámicas, texturas, cámara, puede importar diseños de avatares de diversos software de 3D uno de ellos 3D Studio Max. El manejo de animaciones se basa en esqueletos. Otra característica es que soporta el movimiento de partículas y de sombras.

Al ser un motor de código libre esta en constante actualización, lo cual permite adaptar el motor a nuestras necesidades. El principal problema es que se requiere un conocimiento avanzado de programación C+ y OpenGL.

2.2.2.10 Jazz3D

Jazz3D [2.27] es un API de Java que funciona como motor de 3D. Jazz 3D permite renderizar objetos en 3D, permite poner como textura una imagen animada (.gif) lo cual puede ser atractivo para obtener mejores efectos en una aplicación.

Jazz3D soporta texturas, cámaras, luces, su codificación será más sencilla si se tiene conocimientos en Java. La ayuda existente no es muy amplia, pero contiene ejemplos con las características principales.

Permite la importación de objetos del editor 3D Studio, por lo que las animaciones son prediseñadas en este editor, no soporta las animaciones de esqueletos en tiempo de ejecución, por lo cual no se pude modificar un avatar en tiempo de ejecución.

2.2.2.11 jME

jME [2.28] (jMonkey Engine) es un motor en tiempo real hecho en Java, es código libre por lo cual esta en constante actualización, gracias a sus usuarios y su desarrollador.

Esta basado en OpenGL, tiene la ventaja de ser soportado en diversas plataformas, su aprendizaje es sencillo si se cuenta con conocimientos en Java y OpenGL.

Las animaciones prediseñadas son creadas en editores de 3D, uno de los más utilizados es Blender [2.29]. Por esta característica no se puede tener un control total de los avatares, lo cual es esencial para nuestros propósitos.

2.2.2.12 jPCT

jPCT [2.30] es un motor 3D hecho en Java, el cual soporta texturas, luces, colisiones, cámaras, se pueden importar de diferentes editores de 3D (3D Studio, Maya, etc.), esta basado en OpenGL.

Las animaciones mostradas son diseñadas previamente en un editor soportado por jPCT. Lo cual no permite la manipulación de los objetos que estén dentro del ambiente.

La ayuda es suficiente para comenzar un diseño simple de un juego, sus foros no son muchos y es difícil obtener una respuesta, por lo cual no es muy utilizado en diseños serios.

2.2.2.13 ODE

ODE [2.31] es una biblioteca del alto rendimiento para simulaciones en 3D. Esta hecha en C/C++. ODE es útil para simular vehículos, objetos en ambientes virtuales y a criaturas virtuales. Se utiliza actualmente en muchos juegos de 3D y herramientas de la simulación.

La licencia de ODE permite la utilización del código de fuente gratuitamente en sus productos comerciales. ODE le da más control sobre su producto.

Maneja eventos de colisiones, texturas, animaciones las cuales son diseñadas con anterioridad en un editor de 3D, permite el manejo de sombras y luces. Al tener que prediseñar las animaciones no permite tener el control absoluto de los avatares de un ambiente por lo cual no es de utilidad para nuestro trabajo.

2.2.2.14 OGRE

OGRE [2.32] es un motor de 3D para gráficos orientado a objetos, es un motor orientado a escenas, escrito en C++, diseñado para ser fácil e intuitivo para que los programadores de juegos que utilizan gráficos de 3D. La biblioteca de la clase abstrae todos los detalles de sistemas como Directx y OpenGL y proporciona una interfaz basada en objetos del mundo y otras clases intuitivas.

OGRE permite que su programación pueda ser realizada en Visual C++ para Windows y en GCC para Linux y MAC, ofrece diversas características como texturas, luces, partículas, entre otras. Pueden ser incorporados al juego avarates diseñados en editores de 3D (Maya, Milkshape, 3D Studio, etc.).

La ayuda existente esta diseñara para programadores con experiencia ya que su conjunto de instrucciones no son intuitivas, por este motivo se requiere de mas tiempo para lograr un aplicación que cumpla con nuestras necesidades.

2.2.2.15 Panda3D

Panda3D [2.33] es un motor de 3D de código abierto desarrollado por el estudio de Disney VR para su juego masivo multiplayer "Toontown". El interés primario de Disney en Panda3D es comercial. Una de los principales objetivos es que sea utilizado en escuelas de nivel universitario por lo cual es de fácil aprendizaje.

La programación en Panda3D es en Pitón ya que es un lenguaje completo. Al igual que los motores 3D soporta texturas, luces, cámaras, animaciones creadas en Maya 6 principalmente. Lo cual no permite la modificación de los avatares en tiempo de ejecución, característica principal que requiere GeDA-3D.

2.2.2.16 Power Render

Power Render [2.34] es un motor de 3D que ha estado en desarrollo desde 1994. El motor actual es Direct3D solamente y contiene las características mas recientes de DirectX. Se desarrollan aplicaciones con los lenguajes de Win32 C/C++, Borland Delphi, y NET.

Se exportan la geometría, la animación de keyframe, las texturas, los materiales, y los empalmes esto desde Maya 6. Apoya los dos métodos principales de animación: keyframe (cuerpos rígidos), morphing (interpolación de la cima).

Puede incorporar armas al juego mediante código y la colocación de la arma se puede ver de antemano en el redactor del carácter. Para crear una animación en tiempo de ejecución se utiliza la animación morphing la cual consiste en mover cada vértice de un avatar.

Para esto es necesario conocer detalladamente el avatar y saber como es que se tiene que modificar para lograr la animación. Para lograr una animación de esta forma se requiere demasiado tiempo y código, esto es por que no se cuenta con un soporte de esqueletos.

2.2.2.17 Reality Factory

Reality Factory [2.35] es un conjunto de herramientas que permite la creación de juegos de 3D que es mejorada continuamente. Al tener acceso completo al código de fuente, a la documentación extensa y a una comunidad activa, el programador tiene mejores posibilidades de encontrar la información que requiere, cuando la necesita.

Reality Factory contiene las principales características de un motor de 3D (texturas, animación, luces, cámaras), a demás de diferentes editores que permiten la modificación de las partes principales de un juego (caracteres, terrenos, objetos).

También permite la programación mediante script, los cuales facilitan el manejo de los eventos producidos durante el juego. Pero las animaciones utilizadas son realizadas previamente lo cual impide tener control sobre los avatares que se encuentren en el escenario virtual.

2.2.2.18 Torque

Torque [3.36] es un motor 3D muy poderoso, contiene las principales características de un motor de 3D. Es multiplataforma, se pueden hacer los juegos con características que soporten Directx u OpenGL, lo cual puede facilitar el diseño de un juego.

Torque contiene varios editores para cada una de las partes de un juego (terrenos, interfaz, script, etc.), el lenguaje de los script es C++, tiene soporte para crear un juego en red.

Es uno de los más completos del mercado, existen demos y una versión completa del motor, el diseño de un juego se facilita ya que existe diversas formas de obtener ayuda (libros, tutoriales en línea, foros, etc.).

Por medio de los script se pueden detectar las colisiones, la posición de los avatares dentro del escenario, manipular partículas con las cuales podemos obtener diversos efectos, crear efectos con las luces y sombras. Las animaciones son realizadas previamente en un editor de 3D, lo cual no es conveniente para nuestros propósitos.

2.2.2.19 UGS

Universe Generator System [2.37] es un motor de 3D que esta hecho para Linux, cuanta con varias características entre ellas el soporte de texturas, luces, etc.

Los avatares creados son un poco toscos, requieren de un mayor tiempo para su elaboración. Su ventaja es que es un código libre y multiplataforma.

La ayuda existente es muy poca ya que son pocos sus usuarios, esta basado en OpenGL, haciendo mas fácil su comprensión para programadores con conocimientos en OpenGL. De la documentación no se pueden determinar todas las características de este sistema.

2.2.2.20 UnrealScript

UnrealScript [2.38] es un sistema que consiste de un motor 3D, un conjunto de editores 3D y un lenguaje de programación de gran alcance. El sistema cubre las necesidades y los matices de la programación de juegos. Para el manejo de este lenguaje se debe tener conocimiento en programación orientada a objetos.

Tiene algunas similitudes con Java que cuenta con una maquina virtual, un recolector de basura entre otras características. Existen dos editores para este sistema, uno de ellos esta incluido en el juego llamado Unreal Tournament, con este editor se puede editar el comportamiento de los avatares y modificar el escenario. El otro editor es una versión educativa con cual puede generar un juego desde cero.

UnrealScript es un sistema que cuenta con diversos módulos que permiten la manipulación de avatares, el manejo de eventos (comportamientos), manejo de colisiones y sonido. Uno de los módulos permite incorporar inteligencia artificial a los avateres.

El sistema es bastante amplio, y el acceso a tutoriales y ejemplos es muy escaso, la curva de aprendizaje es larga, en el sitio oficial existen varios enlaces que muestran como ir construyendo un juego sencillo pero algunos de estos enlaces requieren contraseñas.

Una de sus ventajas es que existen plug-in para software de 3D más comerciales actualmente, estos nos permiten exportar los avatares para tener un mejor aspecto.

Para un nuevo usuario es difícil comprender su forma de estructurar un juego ya que se debe tener conocimiento de los temas manejados en el campo de 3D y en programación.

Las animaciones mostradas son prediseñadas en algún editor de 3D que permita la exportación del formato permitido por Unreal, lo cual no permite tener control en tiempo de ejecución sobre las partes del avatar.

2.2.3 Motor de Render

2.2.3.1 Yafray

Yafray [2.39] es un motor de render libre puede integrarse a Blender [2.32]. Genera imagen en formato .tga a partir del archivo de escena en formato XML. Soporta fotorrealismo, iluminación directa e indirecta.

El archivo XML suele ser de gran tamaño cuando se obtiene un mejor acabado, al observar la sintaxis del archivo se pueden deducir que es lo que se esta aplicando en general, pero no da una idea del resultado final.

Para obtener un buen diseño es recomendable utilizar Blender he incorporar el plug-in de Yafray, de esta forma podremos obtener el archivo XML, ya que es difícil manejar las luces, texturas, etc., por que se le tienen que poner las coordenadas de cada uno de ellos. El único formato que soporta hasta ahora es el tga, es uno de los motores más rápidos, la forma de generar la imagen es sencilla y rápida.

2.3 VHML

Existen varios lenguajes para especificar algunas acciones que podría realizar un avatar humano, estos lenguajes están basados en XML, cada uno de ellos trabajan en una área para la animación del avatar.

Los lenguajes existentes son los siguientes:

- EML Emulation Markup Languaje

- GML Gesture Markup Languaje
- SML Speech Markup Languaje
- FAML Facial Animation Markup Languaje
- BAML Body Animation Markup Languaje
- XHTML Extensible Hipertext Markup Languaje
- DMML Dialogue Manager Markup Languaje

Estos lenguajes están englobados en un solo lenguaje el cual es VHML [2.40] Virtual Human Markup Languaje, este lenguaje implementa una arquitectura para la integración de los lenguajes antes mencionados, haciendo su manejo sencillo.

Con esta arquitectura es fácil ver como es que cada uno de los lenguajes interactúa, en cuales lenguajes es necesaria una comunicación. La parte que se encarga de manejar el movimiento del cuerpo de un avatar de una persona es BAML [2.41], su trabajo consiste básicamente en mover cada parte del esqueleto del avatar, por lo cual para realizar una animación de caminar se le debe indicar que partes se deben modificar y como.

VHML solo es la forma en que se comunicaría el usuario con un motor de 3D, no contiene el motor que acepte estas cadenas en XML, por lo cual seria trabajo del usuario la creación de una aplicación que incluya el motor de su elección y la aceptación de las cadenas.

2.4 Conclusiones

El objetivo de este capitulo es hacer una revisión critica del estado del arte para tomar una decisión de que dirección tomar para resolver nuestro problema. Como conclusión de esta revisión podemos decir que: Existen varias herramientas que nos facilitan el diseño de un objeto en 3D, algunas de ellas son intuitivas en su interfaz, con estas podemos obtener un avatar sencillo en poco tiempo y sin grandes conocimientos de la terminología utilizada en este campo. Otros software de uso profesional permiten la creación de animaciones foto realistas, estos han sido utilizados en películas, como por ejemplo Maya fue seleccionado para crear parte de las animaciones en las películas Un día después de mañana [2.42] y Troya [2.43].

Este tipo de software requiere de una aplicación que transforme el diseño en un trabajo final (renderizar), este es el trabajo de los motores de 3D o motores de render, los cuales toman un archivo con ciertas características y generan un archivo final. Por lo general si es solo una imagen se crea en formato PNG, y si es una animación se genera con un formato avi.

Una vez creadas dichas animaciones no es posible modificarlas. Para lograr algún cambio se tiene que modificar el archivo y mandarlo nuevamente al motor de render, para generar de nuevo la animación. Este proceso puede durar días.

Algunos de los editores 3D tienen plug-in que nos permiten la exportación del avatar a formatos de archivos que pueden ser utilizados ya sea en otro editor o en algún juego. Por lo que se facilita la edición de un juego separando la tarea de diseño de avatares de su manejo que en general se realiza por un motor 3D.

Un motor de juegos es también llamado motor en tiempo real, ya que hace el trabajo de render en tiempo real, es decir, un motor de juegos nos permite manipular los caracteres en tiempo de ejecución, lo cual nos facilita el poder mostrar una secuencia de animaciones diferentes cada vez que se desee.

Los motores de juegos están basados en OpenGL o Ditectx, permiten programar los eventos que surgen durante su ejecución, una de las principales características que se buscaron fue la facilidad de manipular los avatares en tiempo de ejecución. Por esto se examinaron con más detalle los motores 3D que no llaman animaciones predefinidas en algún editor de 3D.

Esta característica es necesaria para dar un movimiento más real a los avatares ya que si, se manda ejecutar una animación ya creada, siempre se mostraran los mismos movimientos. También se busco que fuera multiplataforma ya que dentro de las facilidades de GeDA-3D esta considerada la posibilidad de trabajar en red.

Son pocos los motores que ofrecen estas características, algunos de ellos su programación es en Visual C++, otros en GCC, otros mas tenían la posibilidad de elegir entre alguna de esas dos, y un grupo pequeño en Java.

El software elegido es AgentFX2, este motor fue creado en Java, ofrece todas las características de un motor en tiempo real (colisiones, texturas, luces, cámaras, etc.), es multiplataforma, sus animaciones pueden ser predefinidas o creadas mediante código (animación de esqueletos), lo que nos permite tener control sobre el avatar.

El diseño de los caracteres puede ser realizado en varios tipos de editores de 3D, esto gracias a que cuenta con plug-in para ello. La ayuda que se puede obtener es muy completa, ya sea en tutórales con ejemplo, en la pagina o en foros.

AgentFX2 cuanta con una versión libre de prueba, la cual nos permite ver las ventajar que ofrece, para posteriormente obtener una licencia que nos permita trabajar libremente. Es un motor rápido, el diseño de los caracteres no depende de el sino del editor que se elija.

Para poder utilizarlo con facilidad se requieren conocimientos previos en Java y en herramientas de 3D, lo cual hace que solo un grupo de personas puedan utilizarlo. Por lo que se decidió tomar las facilidades de estas herramientas y diseñar un lenguaje que nos sirviera de comunicación entre la plataforma GeDA-3D y las herramientas seleccionadas en este estudio, sin embargo el diseño del lenguaje nos permitirá poder considerar algunas otras herramientas a las seleccionadas si es que ofrecen mayores facilidades o mejoras.

Capítulo 3 Lenguaje de Interfaz para Animaciones en 3D LIA·3D

Resumen

En este capítulo se describe el Lenguaje de Interfaz para Animaciones en 3D (LIA·3D). Este lenguaje es la interfase entre GeDA-3D y el sistema que permite desplegar la evolución de escenarios 3D.

3.1 Introducción

LIA·3D es un lenguaje que nos va permitir transmitir las instrucciones al sistema que se encarga de desplegar un ambiente 3D para que se visualice la evolución de un ambiente virtual. LIA·3D esta constituido principalmente en dos partes. La primera parte es la implementación del lenguaje que nos sirve para controlar los elementos del ambiente; la segunda parte es una arquitectura para la visualización de escenas 3D (AVE·3D). LIA·3D está implementado en XML [3.1], esta decisión se tomo puesto que XML esta volviéndose un estándar de comunicación con lo cual obtenemos una mayor facilidad en la comunicación con otras aplicaciones.

Para que el usuario (GeDA-3D) pueda utilizar LIA·3D no es necesario tener grandes conocimientos en 3D, ni en programación, solo necesita conocer el lenguaje y XML para formar las acciones que cada uno de los avatares deberá realizar en el ambiente virtual.

3.2 Lenguaje

LIA·3D se basa en XML ya que este estándar permite declarar estructuras de información las cuales están limitadas solo por los requerimientos de sus usuarios. Esta facilidad permite que LIA·3D integre constantemente nuevos desarrollos. Esto es necesario si queremos que GeDA-3D considere diferentes tipos de ambiente virtual.

Las acciones declaradas en el lenguaje tienen como objetivo evitar el mayor trabajo posible para el usuario. Por ejemplo para crear el escenario, el usuario no debe conocer como es que esta diseñado un avatar y/o el ambiente virtual, sino únicamente debe indicar que avatares se deben colocar en una determinada posición en el ambiente virtual de su elección.

Después de creado el ambiente virtual solo resta el manipular los avatares, para lo cual se utilizaran las acciones definidas dentro de LIA·3D. Las instrucciones indican a un avatar que partes de su esqueleto debe de modificar para obtener un resultado en particular.

Nuestro lenguaje permite que los avatares expresen emociones, para esto empleamos el estándar VHML, el cual reúne varios lenguajes que describen diferentes aspectos relacionados con la emociones de un avatar (animación facial, diálogos, etc.), la parte que utiliza LIA·3D son las animaciones de emociones y gesticulación.

Con la integración de las acciones definidas en LIA·3D y la parte de VHML integrada, se obtienen animaciones más convincentes. Con esta inclusión, un usuario de LIA·3D puede indicar a un avatar que acción realizar y con que estado de animo. Por ejemplo, dado un ambiente y un avatar que representa a una persona, mediante LIA·3D se puede indicar al avatar que camine triste en una dirección una cierta distancia.

VHML cuenta con una sección de animación del cuerpo, pero realizar una animación de la forma que está indicado, es complicado ya que el usuario para crear una animación debe indicar el movimiento de cada empalme del esqueleto que quiere animar. Un usuario de LIA·3D no necesita conocer el esqueleto solo necesita saber la

acción ya que LIA·3D se encarga del trabajo de manejar el esqueleto. Esto es el usuario solo necesita conocer LIA·3D, no necesita saber como se realiza una animación internamente. La estructura del lenguaje es sencilla a continuación se muestra un ejemplo de una acción:

```
<accion ident=1>
        <caminar>
                <pasos> 3 </pasos>
                <girar> -30 </girar>
        </caminar>
</accion>
```

En cada acción se indica el avatar que debe realizarla, así como los parámetros necesarios. Un evento o mensaje es la respuesta de LIA·3D hacia el cliente (GeDA-3D) el cual contiene el tipo de evento que se suscito con la acción que el cliente realizo, por ejemplo, si el cliente indico la acción de caminar a un avatar y este se colisiono con otro objeto 3D contenido en el ambiente virtual, LIA·3D responderá con un evento de colisión, por ejemplo:

```
<evento>
        <colision>
                <objeto 1> 1 </objeto 1>
                <objeto 2> 4 </objeto 2>
        </colision>
</evento>
```

Con este evento el usuario podrá hacer un seguimiento de sus acciones y saber si la acción que solicito se finalizo satisfactoriamente.

3.2.1 Acciones

El conjunto de las acciones esta divido principalmente en dos grupos, uno de ellos son las acciones de avatares, y el segundo grupo son las acciones de los objetos 3D. Las acciones de los avatares están divididas en keyframe [3.2] que son los que representan movimientos más complejos, i.e. movimientos coordinados de miembros del cuerpo, etc. Las acciones de los objetos son más sencillas, ya que son realizadas por objetos inanimados (puertas, cajas, etc.)

La organización de las acciones se muestra en la siguiente figura 3.1:

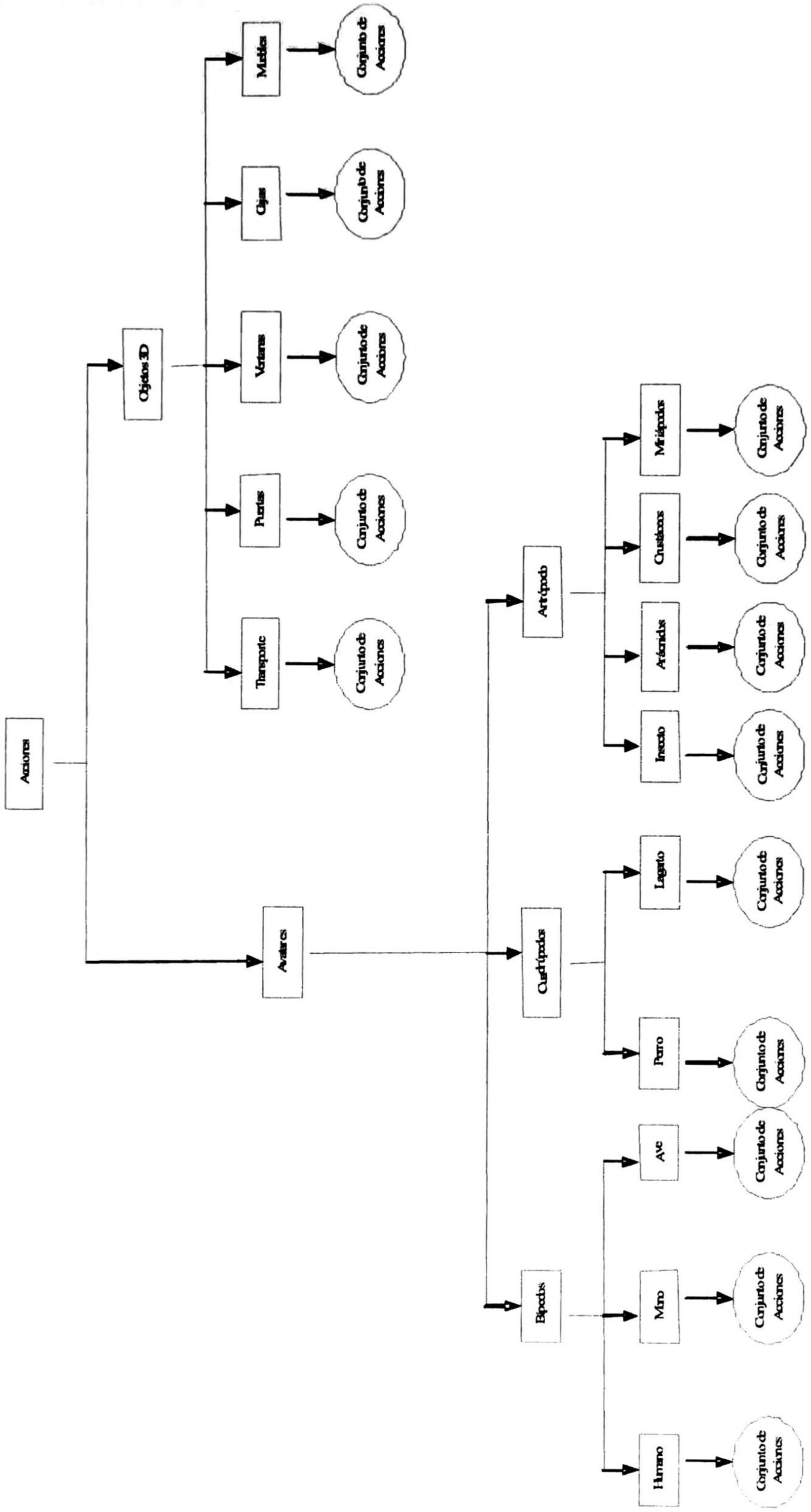


Fig. 3.1: Diagrama de Acciones.

Las acciones de los avatares a su vez están divididas en tipos de esqueletos. Esto hace que el usuario no necesite tener conocimiento de que tipo de avatar necesita manipular. Por ejemplo si el usuario quiere animar un avatar de perro que es un cuadrúpedo, solo necesita conocer el identificador numérico que tiene asignado y no preocuparse de cómo animar por ejemplo su caminar, esto es para el usuario es indiferente si es cuadrúpedo o bípedo.

El conjunto de acciones que cada tipo de avatar y/o objeto 3D específica las animaciones permitidas para cada uno de ellos. Las acciones se especifican de la misma manera que en los objetos, esto es, hay acciones comunes y las acciones específicas de cada tipo se realizan por especialización.

Los movimientos necesarios para obtener una acción son almacenados en una base de datos la cual esta creada en MySql [3.3], de esta forma si en algún momento se desea modificar la animación de una acción, tan solo se deben modificar los valores almacenados en esta base de datos.

Al igual que cualquier otro programa de animación, LIA·3D también maneja keyframe [3.2], estos no son necesarios que los conozca el usuario, de esta forma se reduce aun mas el trabajo del cliente.

El conjunto de acciones no esta limitado, es decir, puede estar en constante crecimiento. Para agregar una acción solo se deberá conocer el esqueleto con el cual se realizara la acción diseñada, y deberá elaborar una cadena XML con la siguiente estructura:

```
<accion ident=identificador>
        <nombre_de_la_accion>
                <parametro_1> valor_parametro_1</paramentro_1>
                <parametro_2> valor_parametro_2</paramentro_2>



                <parametro_n> valor_parametro_n</paramentro_n>
        </nombre_de_la_accion>
</accion>
```

En algunas ocasiones es necesario indicarle a un avatar su estado de animo, por ejemplo: si tenemos un avatar de un ser humano y deseamos que realice la acción de caminar, podemos indicarle su estado de animo, ya que una persona caminaría de forma diferente si esta cansada, triste, alegre, etc.

Para definir una acción y el estado de ánimo con el que se debe ejecutar, la cadena en XML deberá tener la siguiente estructura donde modo indica el estado de ánimo:

```
<accion ident=identificador>
        <nombre_de_la_accion>
                <parametro_1> valor_parametro_1</paramentro_1>
                <parametro_2> valor_parametro_2</paramentro_2>
```

```
                <parametro_n> valor_parametro_n</paramentro_n>
        </nombre_de_la_accion>
        <modo>
                <etiqueta_VHML>
                        <parametro> valor_parametro</parametro>
                </etiqueta_VHML>
        </modo>
</accion>
```

De la estructura anterior, una acción esta formada principalmente por dos partes: La primera es la acción a ejecutar, dentro de esta estarán los parámetros requeridos, solo podrá existir una acción en un bloque o cadena XML. La segunda parte es el ánimo del avatar, estas etiquetas son tomadas del lenguaje VHML [2.40] descrito brevemente a continuación. De este lenguaje se toma la estructura propuesta y se coloca dentro de las etiquetas "*modo*" para indicar que la acción a ejecutar deberá mostrarse con el estado de ánimo indicado.

VHML es un conjunto de lenguajes diseñados para permitir la manipulación de los avatares humanos. LIA·3D solo utiliza una parte del conjunto de lenguajes contenidos en VHML, estos lenguajes son descritos a continuación junto con las acciones de LIA·3D.

<accion> {arg1[, arg2]}

Descripción: contiene la acción a realizar por un avatar u objeto 3D y el estado de animo si es necesario.

Atributos: contiene el identificador numérico del avatar u objetos 3D que realizara la acción seleccionada. La forma de especificar el identificador es: ident = identificador, donde identificador es un valor entero.

Característica: solo puede existir una sola etiqueta de acción en un bloque de una cadena XML, es decir para indicar una segunda acción a un mismo avatar u objeto 3D, se deberá crear una nueva cadena XML con la nueva acción a realizar.

arg1: indicara el nombre de la acción a ejecutar por un determinado avatar.

arg2: indicara el estado de ánimo de un avatar al ejecutar una acción. Este argumento será especificado por medio de la etiqueta modo. Esta etiqueta es optativa.

Ejemplo:

```
<accion ident=1>
        <nombre_de_la accion>
```

<parametro_1> valor_parametro_1</paramentro_1>
<parametro_2> valor_parametro_2</paramentro_2>

<parametro_n> valor_parametro_n</paramentro_n>
</nombre_de_la_accion>
</accion>

<agregar>{arg1, arg2, arg3, arg4}

Descripción: agregara un nuevo avatar u objeto 3D al ambiente virtual, este deberá de existir dentro de la base de datos de Objetos3D, si no es así se regresara un evento indicándolo.

Parámetros: contiene cuatro parámetros los cuales son obligatorios.

Nombre (arg1): representa el identificador del avatar. Con este identificador se buscara en la base para saber su localidad y nombre físico.

<nombre> cadena </nombre>

Cadena es un conjunto de caracteres que hace referencia a un archivo.

Ejemplo:

<nombre> Rebeca </nombre>

Eje X (arg2): dará la posición del avatar en el eje x.

<eje_x> valor_x </eje_x>

Valor_x: es un flotante correspondiente al eje X.

Ejemplo:

<eje_x> 32.5 </eje_x>

Eje Y (arg3): dará la posición del avatar en el eje y.

<eje_y> valor_y </eje_y>

Valor_y es un flotante correspondiente al eje Y.

Ejemplo:

<eje_y> 32.5 </eje_y>

Eje Z (arg4): dará la posición del avatar en el eje z.

<eje_z> valor_z </eje_z>

Valor_z es un flotante correspondiente al eje Z.

Ejemplo:

<eje_z> 32.5 </eje_z>

Ejemplo:

<agregar>
<nombre> Carlos </nombre>
<eje_x> 10.5 </eje_x>
<eje_y> 0.3 </eje_y>
<eje_z> 30.0 </eje_z>
</agregar>

<caminar>{arg1[, arg2, arg3]}

Descripción: modifica principalmente los empales de las extremidades para lograr la animación de caminar, no importa que tipo de avatar este indicado, no puede aplicarse a un objeto 3D.

Parámetros: se podrá indicar si la forma de avanzar es por pasos o por una distancia específica:

Pasos (arg1):

<pasos> numero_de_pasos </pasos>

El numero_de_pasos es un valor entero.

Ejemplo:

<pasos> 4 </pasos>

Distancia (arg2):

<distancia> distancia_a_avanzar </distancia>

Distancia_a_avanzar es un valor flotante, indicando el número de píxeles a desplazar el avatar.

Ejemplo:

<distancia> 4.5 </distancia>

Girar (arg3): Otro de los parámetros es si el avatar debe girar para realizar la acción, si se desea que gire esto sucederá primero, su definición se muestra más adelante.

Ejemplo:

<caminar>
<pasos> 5</pasos>
</caminar>

<inicial>

Descripción: esta acción realiza una lista de los avatares u objetos 3D existentes en el escenario, así como sus posiciones en los tres ejes dentro del escenario 3D. Esta acción es la primera que manda el usuario, de esta forma conoce el estado del escenario 3D.

Parámetros: no cuenta con algún parámetro.

Ejemplo:

<inicial>
</inicial>

<bailar>{arg1}

Descripción: ejecuta una animación que representa a un avatar bailando. Modificara cada empalme del avatar. Su ejecución no afecta su posición dentro del escenario.

Parámetros: solo cuenta con un parámetro el cual es:

Pasos (arg1): indicara el número de repeticiones que se debe ejecutar la acción.

<pasos> num_pasos </pasos>

num_pasos valor numérico utilizado para repetir la animación.

Ejemplo:

<pasos>2</pasos>

Ejemplo:

<bailar>
<pasos>3</pasos>
</bailar>

<aqui>{arg1}

Descripción: mostrara una animación de un avatar que indica su posición. Modifica el brazo derecho así como la cintura. Extenderá su brazo derecho sobre la cabeza, llevándolo de arriba abajo, inclinando su torso de derecha a izquierda.

Parámetros: solo contiene uno el cual es:

Repetir (arg1): es utilizado para conocer la cantidad de veces que se ejecutara la animación.

<repetir>num_repetir</repetir>

num_repetir valor numérico entero que indica la cantidad de ciclos de la animación.

Ejemplo:

<repetir> 9 </repetir>

Ejemplo:

```
<aqui>
        <repetir>6</repetir>
</aqui>
```

<pulsar>{arg1}

Descripción: mostrara una animación con un avatar el cual empujara algo. Extenderá su brazo derecho hasta simular que presiona un objeto.

Parámetros: su único parámetro es:

Repetir (arg1): el valor incluido en esta etiqueta se utiliza para el número de repeticiones de la animación.

<repetir>num_repeticiones</repetir>

num_repeticiones valor numérico que se incluye en un ciclo para ejecutar la animación.

Ejemplo:

<repetir>1</repetir>

Ejemplo:

```
<pulsar>
        <repetir>3</repetir>
</pulsar>
```

<eliminar>

Descripción: borrara un objeto 3D o un avatar del escenario. Por medio del atributo id incluido en la etiqueta acción, se busca en el escenario y de elimina.

Parámetros: no cuenta con alguno.

Ejemplo:

<eliminar>
</eliminar>

<saltar>{arg1, arg2, arg3}

Descripción: crea una animación de saltar de un avatar. Esta acción modifica principalmente las extremidades, si es el caso de un avatar bípedo modificara o animara tanto sus extremidades superiores como las inferiores. Si es un cuadrúpedo se modificaran todas sus extremidades.

Parámetros: contiene tres parámetros, los cuales se describen a continuación:

Altura (arg1): indicara cuanto debe modificarse su posición respecto a su eje Y, este valor deberá indicarse con un número flotante.

<altura> elevar_avatar </altura>

elevar_avatar es un valor flotante con el cual se modificara la posición del avatar.

Ejemplo:

<altura> 3.5 </altura>

Distancia (arg2): con este paramento se modificara o avanzara sobre el eje X, el valor que contiene es un flotante.

<distancia> avanzar_avatar </distancia>

avanzar_avatar es requerido para indicar el valor que se modificara la posición del avatar.

Ejemplo:

<distancia> 5.8 </distancia>

Modo (arg3): es parámetro opcional es el estado de ánimo el cual es indicado según VHML. Si el estado de ánimo no es aplicable al avatar, se regresara un evento el usuario indicándole que no es posible aplicar un estado de ánimo a dicho avatar.

Ejemplo:

```
<saltar>
        <altura> 3.9 </altura>
        <distancia> 1.0 </distancia>
</saltar>
```

<correr>{agr1, arg2}

Descripción: indicara que la animación que se debe ejecutar es la que represente el correr de un avatar. Se podrá aplicar solo a los avatares, esta acción se representara dependiendo del tipo de avatar que el usuario eligió.

Parámetros: sus parámetros son:

Distancia (arg1): modificara su posición sobre el eje X, esto deberá ser indicado por medio de un valor flotante.

<distancia> distancia_avazar </distancia>

distancia_avanzar es un valor numérico el cual es utilizado para saber hasta donde se finalizara la animación.

Ejemplo:

<distancia> 7.0 </distancia>

Velocidad (arg2): indicara que tan rápido debe de mostrarse la animación. Su valor deberá ser entero.

<velocidad> rapidez </velocidad>

rapidez es un valor entero que es utilizado para la frecuencia de keyframes en la animación.

Ejemplo:

<velocidad> 5 </velocidad>

Ejemplo:

```
<correr>
        <distancia> 9.6 </distancia>
        <velocidad> 6 </velocidad>
</correr>
```

<nadar>{arg1, arg2}

Descripción: modificara el avatar de tal forma que represente que esta nadando, de esta forma gran parte del avatar será actualizado. Se podrá aplicar a avatares y no a objetos 3D.

Parámetros:

Distancia (arg1): modificara su posición sobre el eje X, esto deberá ser indicado por medio de un valor flotante.

<distancia> distancia_avazar </distancia>

distancia_avanzar es un valor flotante el cual es utilizado para saber hasta donde se finalizara la animación.

Ejemplo:

<distancia> 5.3 </distancia>

Velocidad (arg2): indicara que tan rápido debe de mostrarse la animación. Si valor deberá ser entero.

<velocidad> rapidez </velocidad>

rapidez es un valor entero que es utilizado para la frecuencia de keyframes en la animación.

Ejemplo:

<velocidad> 1 </velocidad>

Ejemplo:

```
<nadar>
        <distancia> 6.2 </distancia>
        <velocidad> 4 </velocidad>
</nadar>
```

<girar>{arg1}

Descripción: ya indicado el avatar se rotara según el valor del parámetro, esta rotación se realizara sobre el eje X. esta acción podrá ser incluida en otra acción como lo es caminar o correr, si esto pasa se ejecutara primero antes de la animación principal.

Parámetros: solo contiene un parámetro, el cual es:

Rotar (arg1): es un valor flotante el cual podrá ser positivo o negativo, si es positivo girara a la derecha sobre su eje Y.

<rotar> valor_rotacion </rotar>

valor_rotacion es flotante, si es negativo rotara a la izquierda.

Ejemplo:

<rotar> 6.2 </rotar>

Ejemplo:

```
<girar>
        <rotar> 2.9 </rotar>
</girar>
```

<abrir>{arg1}

Descripción: esta acción se podrá aplicar tanto a avatares como a un subconjunto de objetos 3D. Si se desea que la realice un avatar de una persona esta deberá extender el brazo para tocar la puerta y abrirla. Si se aplica a la puerta esta rotara para abrirse.

Parámetros: solo contiene un parámetro.

Porcentaje (arg1): es un valor flotante el cual indicara el porcentaje que deberá ser modificado el objeto 3d. Por lo tanto el máximo será un 100.0 y el mínimo 0.0. Solo deberán de ser valores positivos.

<porcentaje> abrir_porcentaje </porcentaje>

abrir_porcentaje es flotante, indica la magnitud de la modificación del objeto 3d.

Ejemplo:

<porcentaje> 56.7 </porcentaje>

Ejemplo:

<abrir>
<porcentaje> 2.3 </porcentaje>
</abrir>

<cerrar>{arg1}

Descripción: se podrá aplicar en avatares y objetos 3D, la acción en los avatares consistirá en extender una extremidad, para empujar (cerrar) un objeto 3D (puerta, ventana, etc.), en un objeto 3D es aplicable en aquellos que tengan una parte que se permita rotar (puerta, ventana, caja, etc.).

Parámetros: solo contiene rotar, descrita a continuación:

Rotar (arg1): contendrá un valor flotante positivo, el cual deberá ser mayor o igual a 0 y menor o igual a 100.

<rotar> valor_de_rotacion </rotar>

valor_de_rotacion es un flotante positivo, indicando el porcentaje que se cerrara la el objeto 3D.

Ejemplo:

<rotar> 7.6 </rotar>

Ejemplo:

```
<cerrar>
        <rotar> 6.7 </rotar>
</cerrar>
```

<subir_escalera>{arg1, arg2}

Descripción: podrá ejecutarse con algún avatar, modificando sus extremidades e inclinación. Si algún avatar no puede realizarla se le indicara al usuario que la acción no es aplicable al avatar que indico. Por lo cual el usuario deberá estar consiente que su avatar pueda realizar dicha acción, de esta forma ahorrar tiempo en la ejecución.

Parámetros: ya que existen dos tipos de escaleras principalmente, uno de sus parámetros indica que tipo de escalera se utilizara. Sus parámetros son:

Tipo (arg1): de esta forma el usuario podrá decidir que tipo de escalera es la que se utilizara.

<tipo> identificador_escalera </tipo>

Identificador_escalera: es un valor entero, si el valor es 1 será una escalera fija, el 2 indica una escalera movible. En el tipo 2 implica mover las manos para sujetarse del siguiente escalón.

Ejemplo:

<tipo> 2 </tipo>

Numero de escalones (arg2): podrá indicar cuantos escalones desea subir el avatar, si desea que suba toda la escalera el usuario deberá de conocer la cantidad de escalones que contiene.

<num_escalon> cantidad_escalones </num_escalon>

Cantidad_escalones valor entero no negativo, el limite será la cantidad de escalones de la escalera.

Ejemplo:

<num_escalon> 8 </num_escalon>

Ejemplo:

```
<subir_escalera>
        <tipo> 1 </tipo>
        <num_escalon> 20 </num_escalon>
</subir_esclera>
```

<bajar_escalera>{arg1, arg2}

Descripción: la animación mostrada será la de un avatar que baja las escaleras, al igual que subir_escaleras, solo pertenece aun subconjunto de los avatares, ya que por ejemplo una abeja no puede bajar las escaleras.

Parámetros: contiene dos parámetros, los cuales se describen a continuación:

Tipo (arg1): indica el tipo de escalera que se utilizara, los valores son iguales a subir_escalera. 1 escalera fija y 2 escalera movible.

<tipo> tipo_escalera </tipo>

Tipo_escalera es un valor entero con valor de 1 o 2.

Numero de escalones (arg2): será la cantidad de escalones a bajar, el limite será el tamaño de la escalera.

<num_escalones> cantidad_escalones </numero_escalones>

Cantidad_escalones valor entero que no superara el tamaño de la escalera del escenario.

Ejemplo:

```
<bajar_escalera>
        <tipo> 2 </tipo>
        <num_escalones> 44 </num_escalones>
<bajar_escalera>
```

<inclinarse>{arg1}

Descripción: modificara la parte superior de un avatar para hacer la animación de inclinarse. Dicha acción básicamente es parte del conjunto de los bípedos.

Parámetros: solo contiene un parámetro.

Porcentaje (arg1): indica el porcentaje a inclinarse podrá tener un valor igual o mayor a cero y menor o igual a 120. Este limite es para que la animación parezca lo mas real posible.

<porcentaje> valor_inclinacion </porcentaje>

Valor_porcentaje será un flotante no negativo, que entrara en el rango ya mencionado.

Ejemplo:

<porcentaje> 121.6 </porcentaje>

Ejemplo:

```
<inclinarse>
        <porcentaje> 5.9 </porcentaje>
</inclinarse>
```

<morder>{arg1, arg2[, arg3]}

Descripción: creara la animación de morder de un avatar, no es utilizada en objetos 3D. Manipulara parte del rostro como lo es la expresión y la quijada.

Parámetros: contiene tres parámetros dos de los cuales son obligatorios.

Tamaño (arg1): indicara que tanto se abrirá la boca (hocico) del avatar, será un valor flotante positivo.

<tamano> porcentaje_mordida </tamano>

Porcentaje_mordida es un flotante positivo, sus límites dependerán el avatar a utilizar.

Ejemplo:

<tamano> 8.8 </tamano>

Duración (arg2): se indicara con un entero no negativo y serán los milisegundos que durara la mordida.

<duracion> milisegundos </duracion>

Milisegundos valor entero, tomado para la finalización de la animación.

Ejemplo:

<duracion> 21 </duracion>

El parámetro de la emoción *(arg1)* es opcional y es descrito según VHML.

Ejemplo:

```
<mordida>
        <tamano> 14.6 </tamano>
        <duracion> 10 </duracion>
</mordida>
```

<sentarse>{arg1[, arg2]}

Descripción: actualizara el avatar rotando principalmente los empalmes de las rodillas, de esta forma el avatar tocara el objeto 3D sobre el cual se sentara.

Parámetros: solo contiene dos parámetros, el ánimo es uno de ellos y será opcional.

Porcentaje (arg1): indicara que tanto se rotaran los empalmes de las rodillas, de esta forma logrando la animación.

<porcentaje> valor_rotacion </porcentaje>

Valor_porcentaje es un flotante positivo, el cual es igual o mayor a cero y menor o igual a 100.

Ejemplo:

<porcentaje> 45 </porcentaje>

El ánimo *(arg2)* será aplicado siguiendo los lenguajes de VHML.

Ejemplo:

```
<sentarse>
    <porcentaje> 7 </porcentaje>
</sentarse>
```

<mover_extremidad>{arg1, arg2, arg3, arg4}

Descripción: si desea hacer una animación más elaborada se puede hacer mediante esta acción, por ejemplo después de sentarse se desea girar solo la cabeza, o si el usuario desea crear su propia animación puede hacer la secuencia por medio de esta acción.

Parámetros: contiene 4 parámetros obligatorios los cuales son:

Nombre del empalme (arg1): indicara que empalme es el que realizara dicha acción. Para esto el usuario deberá conocer los nombres de los empalmes del esqueleto que desea utilizar.

<nombre> nombre_empalme </nombre>

nombre_empalme es una cadena la cual es formada con el nombre del empalme, se debe tener cuidado con las mayúsculas y minúsculas.

<nombre> Cadera </cadera>

Eje_x (arg2): indica como se debe actualizar el eje x del empalme.

<eje_x> rotar_x </eje_x>

Rotar_x valor flotante aplicable al eje x del empalme seleccionado.

Ejemplo:

<eje_x> 5.7 </eje_x>

Eje_y (arg3): indica como se debe actualizar el eje x del empalme.

<eje_y> rotar_y </eje_y>

Rotar_y valor flotante aplicable al eje y del empalme seleccionado.

Ejemplo:

<eje_y> 0.0 </eje_y>

Eje_z (arg4): indica como se debe actualizar el eje z del empalme.

<eje_z> rotar_z </eje_z>

Rotar_z valor flotante aplicable al eje z del empalme seleccionado.

Ejemplo:

<eje_z> 9.1 </eje_z>

Ejemplo:

```
<mover_extremidad>
        <nombre> Cuello </nombre>
        <eje_x> 0.0 </eje_x>
        <eje_y> 10.0 </eje_y>
        <eje_z> 3.3 </eje_z>
</mover_extremidad>
```

A continuación se describen los dos lenguajes de VHML que se utilizan. El primer lenguaje es utilizado para describir emociones y el segundo para gestos faciales.

Lenguaje para Emociones EML (Emotion Markup Language)

El objetivo de EML (Emotion Markup Language) [2.40] es proveer los medios necesarios para realizar una animación de un avatar humano más real por medio de las emociones. EML toma en cuenta los elementos referentes a la emoción para que puedan ser tomados en cuenta en una animación. A continuación son descritas las etiquetas de las cadenas en XML que toma EML para describir las emociones.

Atributos de EML

Nombre	Descripción	Valor	default
duration	Especifica la duración en segundos o milisegundos que la emoción persistirá en el ser humano virtual.	▪ #s ▪ #ms	requerido para los elementos vacíos
intensity	Especifica la intensidad de esa emoción particular, por un valor descriptivo o por un valor numérico. El medio representa un valor numérico de cincuenta.	▪ Valor numérico ▪ low ▪ medium ▪ high	medium

Tabla 3.1: Atributos de las etiquetas de EML.

Los elementos siguientes constituyen EML, en cada uno de ellos se puede aplicar los atributos antes mencionados.

<afraid>

Descripción: Genera a un ser humano virtual que parece asustado. Las cejas se levantan y se tiran juntas, las cejas internas están dobladas hacia arriba y los ojos están tensos y alertas.

Ejemplo:

```
<afraid intensity="2">
</afraid>
```

<angry>

Descripción: Genera a un ser humano virtual que parece enojado. Las cejas internas se tiran hacia abajo y juntas, los ojos están de par en par abiertos y los labios se presionan cara a cara o se abren para exponer los dientes.

Ejemplo:

```
<angry>
</angry>
```

<confused>

Descripción: Genera a un ser humano virtual que parece confuso. Las cejas están dobladas hacia arriba, las cejas internas están teniendo gran movimiento y las esquinas de la boca son cercanas juntas.

Ejemplo:

```
<confused duration="4s" intensity="high" wait="2s"/>
```

<dazed>

Descripción: Genera a un ser humano virtual que parece deslumbrado. Las cejas se levantan levemente, los ojos abiertos algo más de par en par que normal y los labios se tiran levemente abajo y hacia fuera.

Ejemplo:

```
<dazed duration="10s"/>
```

<disgusted>

Descripción: Genera a un ser humano virtual que parece disgustado. Se relajan las cejas y los párpados.

Ejemplo:

```
<disgusted intensity="80">
</disgusted>
```

<happy>

Descripción: Genera a un ser humano virtual que parece feliz. Se relajan las cejas, la boca es abierta y las esquinas de la boca tiradas hacia los oídos.

Ejemplo:

```
<happy duration="7s" wait="2000ms"/>
```

<neutral>

Descripción: Genera a un ser humano virtual que parece neutral. Se relajan todos los músculos de la cara, los labios están en contacto, la boca es cerrada y la línea de los labios es horizontal.

Ejemplo:

```
<neutral wait="2s">
</neutral>
```

<sad>

Descripción: Genera a un ser humano virtual que parece triste. Las cejas internas están dobladas hacia arriba, los ojos son levemente cerrados y se relaja la boca.

Ejemplo:

```
<sad intensity="low">
</sad>
```

<surprised>

Descripción: Genera a un ser humano virtual que parece sorprendido. Se levantan las cejas, los párpados superiores están de par en par abiertos y se abre la quijada.

Ejemplo:

```
<surprised duration="2s" wait="500ms">
</surprised>
```

<default-emotion>

Descripción: El ser humano virtual conseguirá la emoción que se especifica en el elemento de la persona. Si no existe un elemento de la persona, la emoción que se predefine para el uso será utilizada.

Ejemplo:

```
<default-emotion>
</default-emotion>
```

Lenguaje para Gesticulaciones GML (Gesture Markup Language)

El lenguaje GML [2.40] sirve para describir las animaciones que puede realizar un avatar. Los elementos en GML describen gestos humanos bien conocidos.

Atributos de GML

Nombre	Descripción	Valor	default
duration	Especifica la duración en segundos o los milisegundos que la emoción persistirá en el ser humano virtual.	▪ #s ▪ #ms	requerido para los elementos vacíos
intensity	Especifica la intensidad de esa emoción particular, por medio de un valor descriptivo o por un valor numérico. El valor medio es representado mediante un valor numérico de igual a cincuenta.	▪ Valor numérico ▪ low ▪ medium ▪ high	medium

Tabla 3.2: Atributos de las etiquetas de GML.

Elementos de GML

El lenguaje GML está constituido por etiquetas, que son descritas a continuación, cada una de ellas contiene los atributos mencionados en la tabla anterior.

<agree>

Descripción: Sirve para indicar a un avatar humano expresar un "sí" o un acuerdo usando gesticulaciones. La gesticulación es un movimiento de la cabeza. El acuerdo o "si" está dividido en dos secciones que son: el aumento del empalme principal (alzar la cabeza) y su disminución (bajar la cabeza). Esto es, únicamente se altera el ángulo vertical de la cabeza durante la animación del elemento, la mirada fija todavía se enfoca adelante.

Nombre	Descripción	Valor	Default
repeat	Especifica cuántas veces debe ocurrir la acción.	integer	1

Tabla 3.3: Atributo de la etiqueta agree.

Ejemplo:

```
<agree duration="1000ms"/>
```

<disagree>

Descripción: Ordena a ser humano virtual expresar "no" o desacuerdo usando gesticula. La animación es una sacudida de la cabeza, que implica: primero el moverse a la izquierda, segundo hacia la derecha y en tercer lugar volver al plano central. El elemento afecta solamente la rotación horizontal de la cabeza y no se afecta ningunas otras características faciales.

Nombre	Descripción	Valor	Default
repeat	Especifica cuántas veces debe ocurrir la acción.	integer	1

Tabla 3.4: Atributo de la etiqueta disagree.

Ejemplo:

```
<disagree intensity="20"/>
</disagree>
```

<concentrate>

Descripción: Dirige a ser humano virtual que tiene una mirada que se concentra. Se bajan las cejas y los ojos están cerrados en parte.

Ejemplo:

```
<concentrate wait="2s"/>
</concentrate>
```

<emphasis>

Descripción: Acentúe las palabras en el texto hablado. Anima un cabeceo con las cejas que bajan en 21 píxeles.

Nombre	Descripción	Valor	Default
level	Especifica la fuerza del énfasis que se aplicará.	▪ reduced ▪ none ▪ moderate ▪ strong	moderate

Tabla 3.5: Atributos de la etiqueta emphasis.

Ejemplo:

```
<emphasis level="strong">
</emphasis>
```

<sigh>

Descripción: Ordena a ser humano virtual expresar un suspiro. Se soplan las mejillas y también se afectan las cejas, la cabeza y la boca.

Nombre	Descripción	Valor	Default
repeat	Especifica cuántas veces debe ocurrir la acción.	integer	1

Tabla 3.6: Atributo de la etiqueta sigh.

Ejemplo:

```
<sigh duration="2500ms" wait="2500ms"/>
```

<smile>

Descripción: Genera una expresión de un ser humano virtual sonriente. Se ensancha la boca y las esquinas se tiran hacia los oídos.

Ejemplo:

```
<smile intensity="low"/>
</smile>
```

<shrug>

Descripción: Mímico el facial y expresión del cuerpo para decir "no sé". La cabeza que inclinaba detrás, las esquinas de la boca se jalan hacia abajo y la ceja interna inclina hacia arriba.

Nombre	Descripción	Valor	Default
repeat	Especifica cuántas veces debe ocurrir la acción.	integer	1

Tabla 3.7: Atributo de la etiqueta shrug.

Ejemplo:

```
<shrug duration=”5000” intensity=”75”/>
```

3.2.2 Eventos

Si en algún caso surge un error al tiempo de ejecutar una acción se regresara un evento en XML indicando el tipo de error. De esta forma el usuario podrá conocer el estado del ambiente y tomar una decisión. La sintaxis para un evento es la siguiente:

```
<evento>
        <tipo_evento> valor_parametro </tipo_evento>
</evento>
```

La etiqueta <evento> contiene el evento surgido al ejecutar una acción. Solo podrá contener un solo evento por cadena XML. A continuación son descritos los tipos de eventos que pueden surgir dentro del escenario.

<colision>{arg1,arg2}

Descripción: cuando al ejecutar una acción esta provoca una colisión con otro objeto 3D, se activara este evento.

Parámetro: serán únicamente dos parámetros los cuales indicaran los objetos en colisión.

Objeto1 (arg1): contiene el identificador numérico de uno de los avatar en colisión.

Objeto2 (arg2): agrupa el identificador numérico del segundo avatar en colisión.

```
<evento>
        <objeto1> identificador_num </objeto1>
        <objeto2> identificador_num </objeto2>
</evento>
```

Identificador_num es el identificador de un objeto 3d o avatar que esta involucrado en la colisión.

Ejemplo:

```
<evento>
        <colision>
                <objeto1> 8 </objeto1>
                <objeto2> 1 </objeto2>
        </colision>
</evento>
```

<inicial>{agr1[, agr2, ... argn]}

Descripción: es la respuesta a la acción inicial. Creara una cadena en XML la cual incluye el nombre y posición de cada avatar u objeto 3D.

Parámetros: contiene un parámetro el cual agrupa otras etiquetas. Esto se describe a continuación:

Avatar (arg1): este agrupara los datos necesarios de un avatar, como es su nombre y posición en el escenario. Esta etiqueta se repetirá de acuerdo al número de objetos 3D y/o avatares que existan en el escenario 3D. La estructura de esta etiqueta se describe a continuación:

Nombre: contiene el nombre lógico de (los) objeto(s) 3D y/o avatar(es).

<nombre>cadena_nombre</nombre>

cadena_nombre cadena de caracteres que indica el nombre lógico.

Ejemplo:

<nombre>Roció</nombre>

posX: indica la posición en el eje x del avatar u objeto 3D.

<posX>valor_posicion</posX>

valor_posicion valor flotante en el que se encuentra el avatar u objeto 3D dentro del eje x.

Ejemplo:

<posX>10.1</posX>

posY: indica la posición en el eje y del avatar u objeto 3D.

<posY>valor_posicion</posY>

valor_posicion valor flotante en el que se encuentra el avatar u objeto 3D dentro del eje y.

Ejemplo:

<posY>0.0</posY>

posZ: indica la posición en el eje z del avatar u objeto 3D.

<posZ>valor_posicion</posZ>

valor_posicion valor flotante en el que se encuentra el avatar u objeto 3D dentro del eje z.

Ejemplo:

<posZ>3.0</posZ>

Ejemplo:

```
<avatar>
        <nombre>HombreVerde</nombre>
        <posX>20.0</posX>
        <posY>0.0</posY>
        <posZ>-23.0</posZ>
</avatar>
```

Ejemplo:

```
<evento>
    <inicial>
        <avatar>
            <nombre>Mariposa</nombre>
            <posX>0.2</posX>
            <posY>0.0</posY>
            <posZ>-0.5</posZ>
        </avatar>
        <avatar>
            <nombre>Perro</nombre>
            <posX>0.0</posX>
            <posY>0.0</posY>
            <posZ>8.2</posZ>
        </avatar>
    </inicial>
</evento>
```

<final_correcto>{arg1, arg2}

Descripción: cuando una acción finalice sin ningún contratiempo se regresara este evento al usuario.

Parámetros: serán dos con los cuales el usuario podrá identificar que acción se finalizó y que avatar u objeto 3d la realizo.

Ident (arg1): contendrá el identificador numérico del avatar u objeto 3D que se indico en la acción.

<ident> num_ident </ident>

Num_ident será un valor entero el cual es un identificador de un avatar u objeto 3D entro del ambiente virtual.

Ejemplo:

<ident> 8 </ident>

Accion (arg2): contiene el nombre de la acción que se solicito que se ejecutara.

<accion> nom_accion </accion>

Nom_accion es una cadena que indica el nombre de la acción que fue ejecutada.

Ejemplo:

<accion> nadar </accion>

Ejemplo:

```
<evento>
        <final_correcto>
                <ident> 2 </ident>
                <accion> correr </accion>
        </final_correcto>
</evento>
```

<ident_no_exist>{arg1, arg2}

Descripción: el identificador contenido en una acción no existe dentro del ambiente.

Parámetros: se describen a continuación:

Identificador (arg1): del avatar que se indico en una acción, el cual no existe.

<ident> num_identificador </ident>

Num_identificador es un valor numérico con el cual se hace referencia a un objeto 3D o a un avatar.

Ejemplo:

<ident> 9 </ident>

Acción (arg2): se podrá saber que acción contenía el identificador no valido, esto es útil en las ocasiones de que el usuario haya mandado varias acciones poder reconocer que acción es la errónea.

<accion> nombre_accion </accion>

Nombre_accion cadena que indica el nombre de la acción.

Ejemplo:

<accion> saltar </accion>

Ejemplo:

<evento>
<ident_no_exist>
<ident> 11 </ident>
<accion> correr </accion>
</inden_no_exist>
</evento>

<accion_no_valida>{arg1, arg2}

Descripción: si el usuario invoca una acción con un avatar el cual no la contiene, se le indicara por medio de este evento.

Parámetros: contiene dos parámetros los cuales son:

Accion (arg1): contendrá la acción no perteneciente al avatar u objeto 3D.

<accion> nom_accion </accion>

Nom_accion cadena de caracteres que señala el error.

Ejemplo:

<accion> abrir </accion>

Identificador (arg2): valor numérico que fue asignado a un avatar u objeto 3D.

<ident> num_identificador </ident>

Num_identificador valor entero no negativo.

Ejemplo:

```
<evento>
        <accion_no_valida>
                <accion> volar </accion>
                <indet> 4 </ident>
        </accion_no_valida>
</evento>
```

<no_empalme>{arg1, arg2}

Descripción: cuando un usuario desee solo modificar una parte del avatar u objeto 3D, por medio de sus empalmes, y alguno de estos no exista se regresara un evento de este tipo.

Parámetros: contendrá el empalme no existente y a quien fue asignado:

Empalme (arg1): indicara el nombre del empalme que no existe, este error puede surgir debido a que se hace distinción entre mayúsculas y minúsculas.

<empalme> nom_empalme </empalme>

Nom_empalme cadena de caracteres referente a un nombre de empalme.

Ejemplo:

<empalme> R der </empalme>

Identificador (arg2): avatar u objeto 3D el cual no contiene dicho empalme.

<ident> num_identificador </ident>

Num_identificador valor numérico que hace referencia a un objeto 3D o a un avatar.

Ejemplo:

<indet> 45 </indet>

Ejemplo:

```
<evento>
        <no_empalme>
                <empalme> cab </empalme>
                <ident> 2 </ident>
        </no_empalme>
</evento>
```

<no_coord>{arg1, arg2}

Descripción: debido a que un esqueleto tiene límites, esto es para evitar que sus movimientos realicen una animación no real. De esta forma si al realizar una animación moviendo empalmes se da una coordenada que supera los limites, se activara este evento.

Parámetros: indicaran que empalme y su avatar u objeto 3D.

Empalme (arg1): cadena con el nombre del empalme.

<empalme> nom_empalme </empalme>

Nom_empalme: empalme que sobrepasa al limite.

Ejemplo:

<empalme> Cuello </empalme>

Identificador (arg2): avatar u objeto 3D el cual no contiene dicho empalme.

<ident> num_identificador </ident>

Num_identificador valor numérico que hace referencia a un objeto 3D o a un avatar.

Ejemplo:

<indet> 45 </indet>

Ejemplo:

```
<evento>
    <no_coord>
        <empalme> Pelvis </empalme>
        <ident> 4 </ident>
    </no_coord>
</evento>
```

<no_agregado>{arg1}

Descripción: se provocara este evento cuando se desee agregar un objeto 3D o un avatar que no exista en la base de datos de Objetos3D.

Parámetros: solo contiene el nombre el objeto no existente.

Nombre (arg1): indicara el nombre del avatar que se quiso agregar.

<nombre> cadena </nombre>

Cadena es un conjunto de caracteres que hacen referencia al objeto no agregado.

Ejemplo:

<nombre> Blanca </nombre>

Ejemplo:

```
<evento>
        <no_agregado>
                <nombre> Idania </nombre>
        </no_agregado>
</evento>
```

<agregado>{arg1}

Descripción: se produce cuando el usuario dio la acción de agregación de un objeto. Regresara el identificador correspondiente.

Parámetros: solo contiene uno:

Ident (arg1): contendrá el identificador del nuevo avatar.

<ident> num_identificador </ident>

Num_identificador valor entero el cual es asignado por el modulo Render.

Ejemplo:

<ident> 5 </ident>

Ejemplo:

```
<evento>
        <agregado>
                <ident> 10 </ident>
        </agregado>
</evento>
```

3.3 Conclusiones

La principal aportación de LIA·3D no es su implementación ya que esta no es exhaustiva sino demostrativa. Sin embargo LIA·3D puede ser enriquecido de acuerdo a las necesidades especificas de las aplicaciones. Esto es, se pueden agregar nuevas acciones o eventos, o modificar las ya existentes. Una característica importante del lenguaje es que está implementado en XML, y puede emplearse con cualquier motor 3D y en otras aplicaciones diferentes a GeDA-3D que es para la cual se diseño sin ningún cambio.

Dentro del estado del Arte, los trabajos existentes con los que se podría comparar seria a nivel de arquitectura con los lenguajes utilizados en los juego 3D. LIA·3D tiene una arquitectura que permite crear cualquier tipo de ambiente o en todo caso juego, sin embargo los juegos existentes tienen una arquitectura que solo serviría para un juego en especifico.

Si uno desea crear un ambiente o un juego utilizando alguna librería como lo es OGRE [2.32] tendría que diseñarlo desde cero, utilizando LIA·3D el desarrollo tiene menos costo porque es mucho más fácil considerar las acciones que realizan las criaturas virtuales y los eventos posibles que surgen de estas acciones.

Capítulo 4 Arquitectura para Visualizar Escenas 3D AVE·3D

Resumen

Los objetivos de este capítulo son dos. Primero: describir la estructura del esqueleto humano, la manera en que fue dividido para visualizar las animaciones y emociones. El segundo objetivo es describir la arquitectura propuesta de AVE·3D, por medio de la descripción de cada módulo que la integran. Para esto, se muestra el flujo de datos mediante un ejemplo sencillo.

4.1 Introducción

El problema que nos ocupa es la animación de las acciones descritas en LIA·3D. Para resolver este problema proponemos una arquitectura para el lenguaje que interprete las acciones que conforman una animación. En este capitulo se describe la comunicación entre los módulos que conforman la arquitectura propuesta, así como la tarea realizada por cada uno de ellos.

4.2 Esqueleto Humano

El esqueleto de un ser humano contiene los empalmes necesarios para crear animaciones corporales y faciales bastante elaboradas. Las Figura 4.1- 4.5 muestra un esqueleto humano.

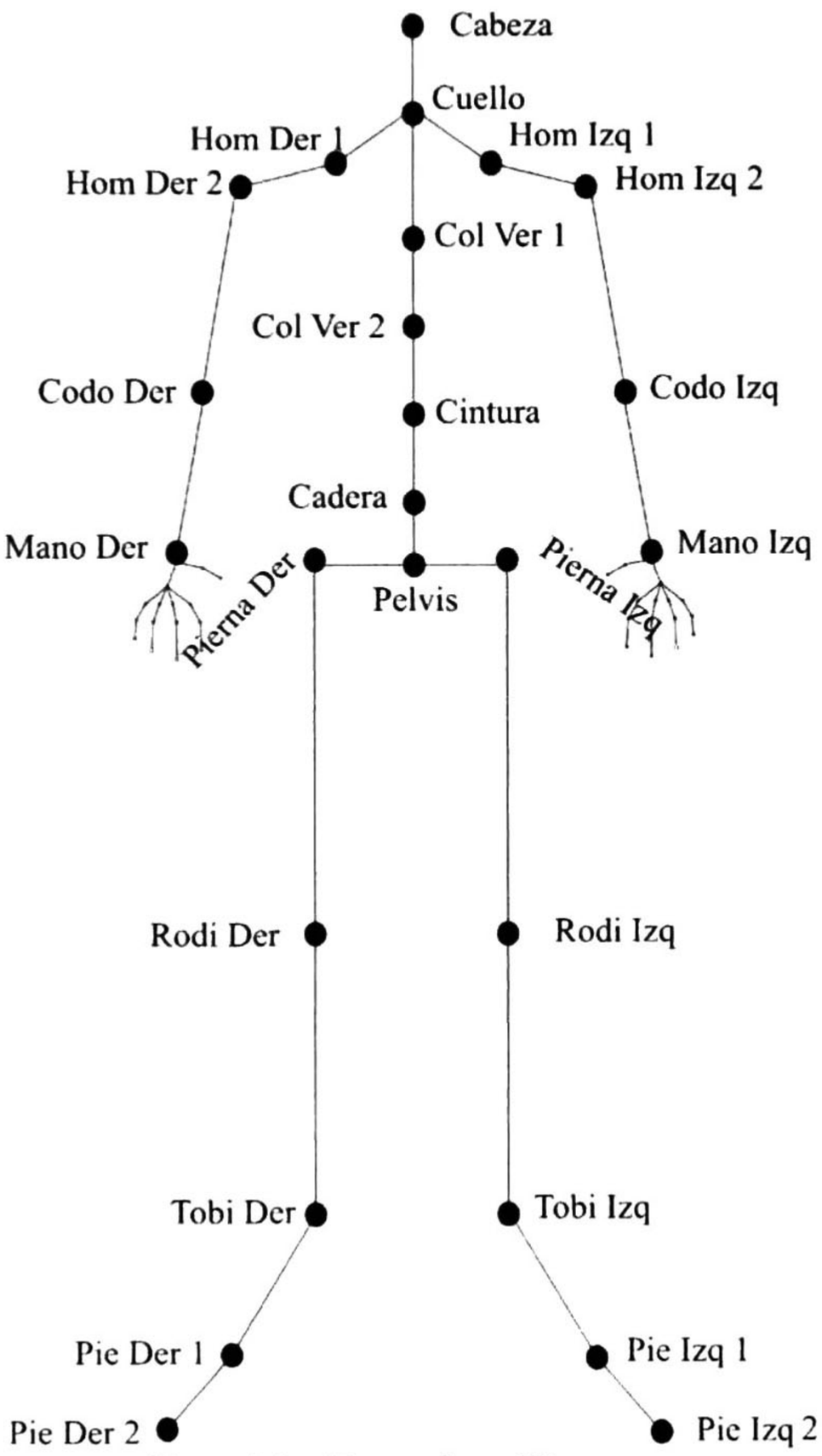


Fig. 4.1: Esqueleto Humano

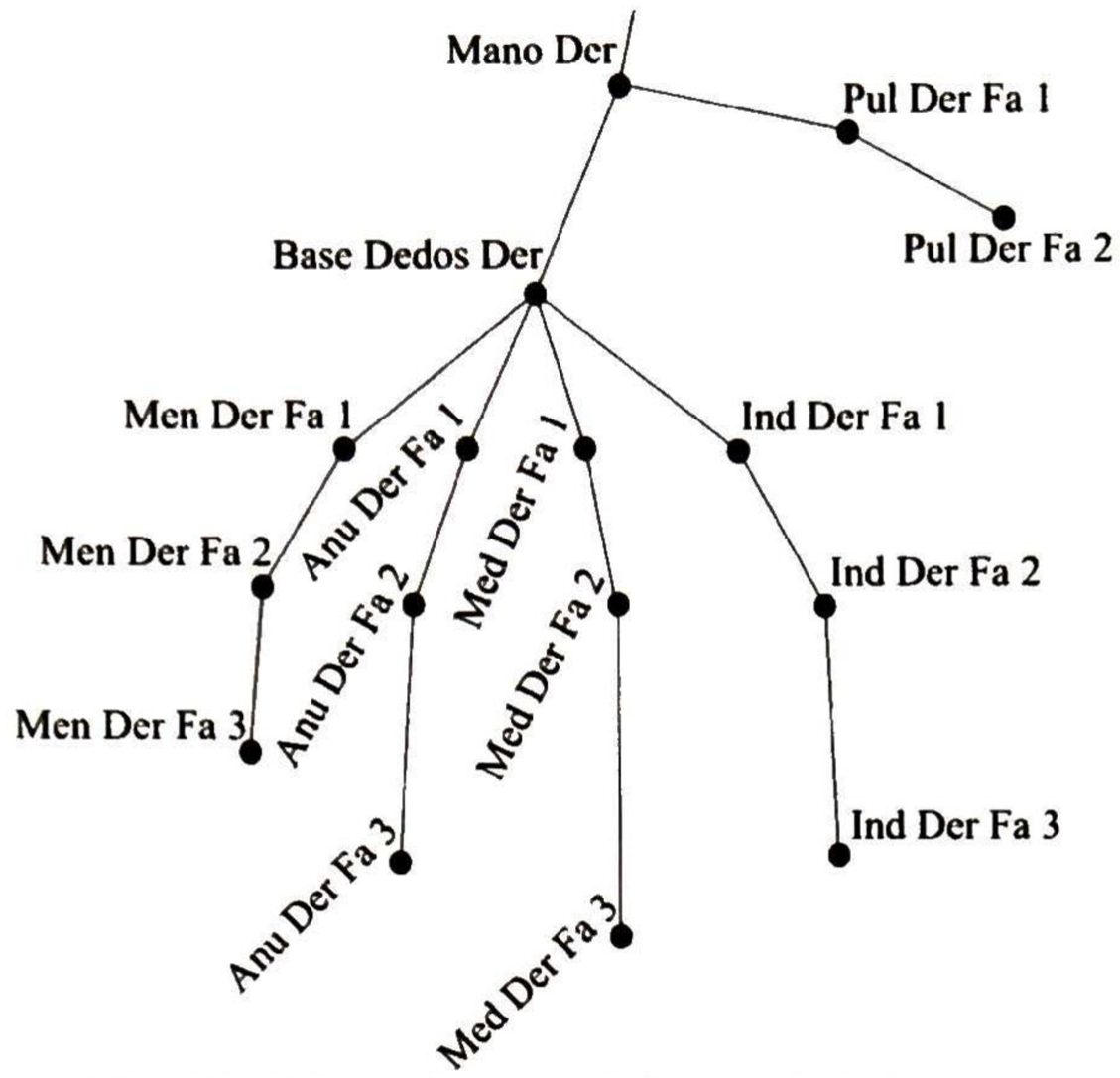


Fig. 4.2: Mano derecha del esqueleto humano

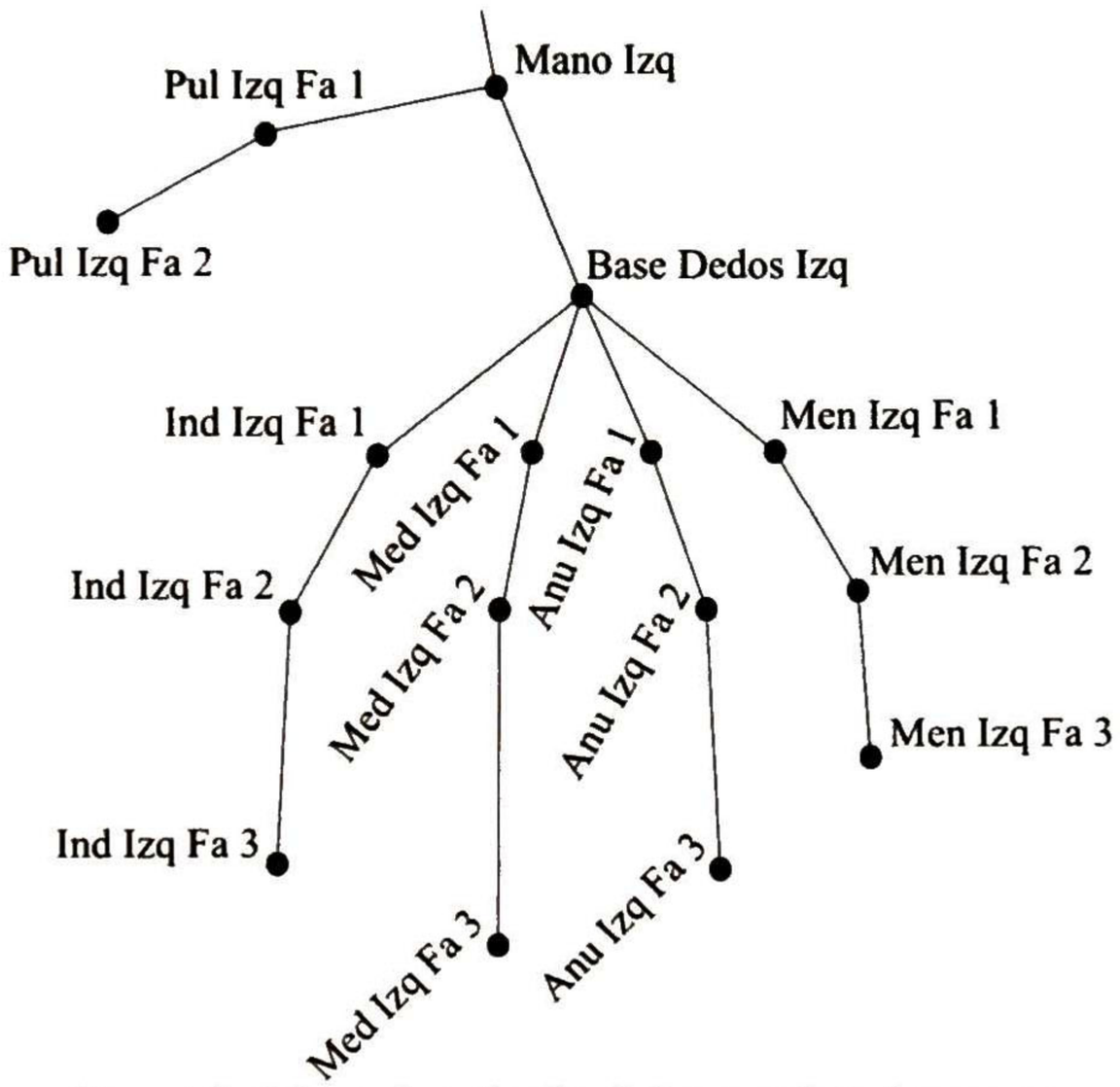


Fig. 4.3: Mano izquierda del esqueleto humano

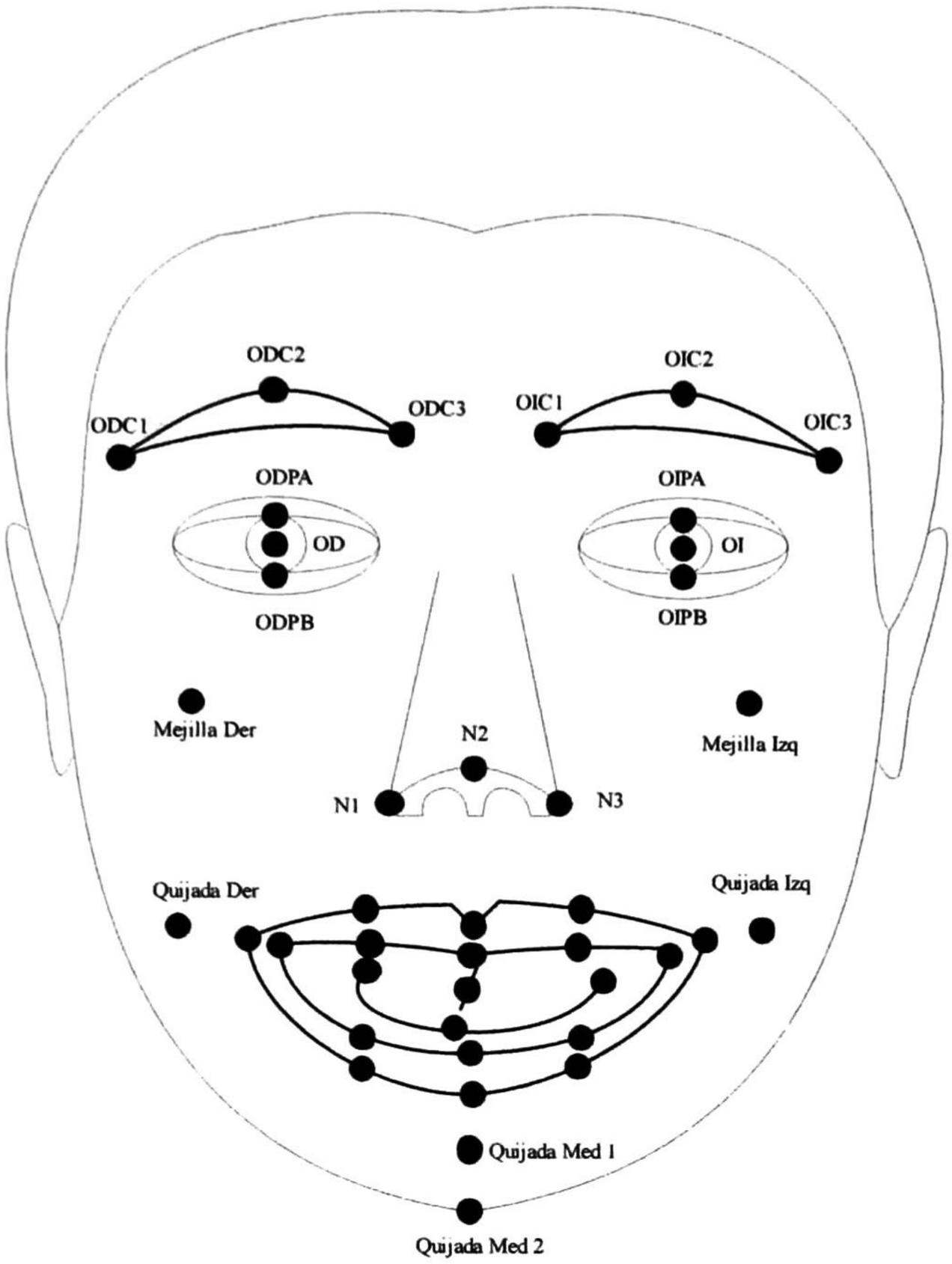


Fig. 4.4: Rostro Humano

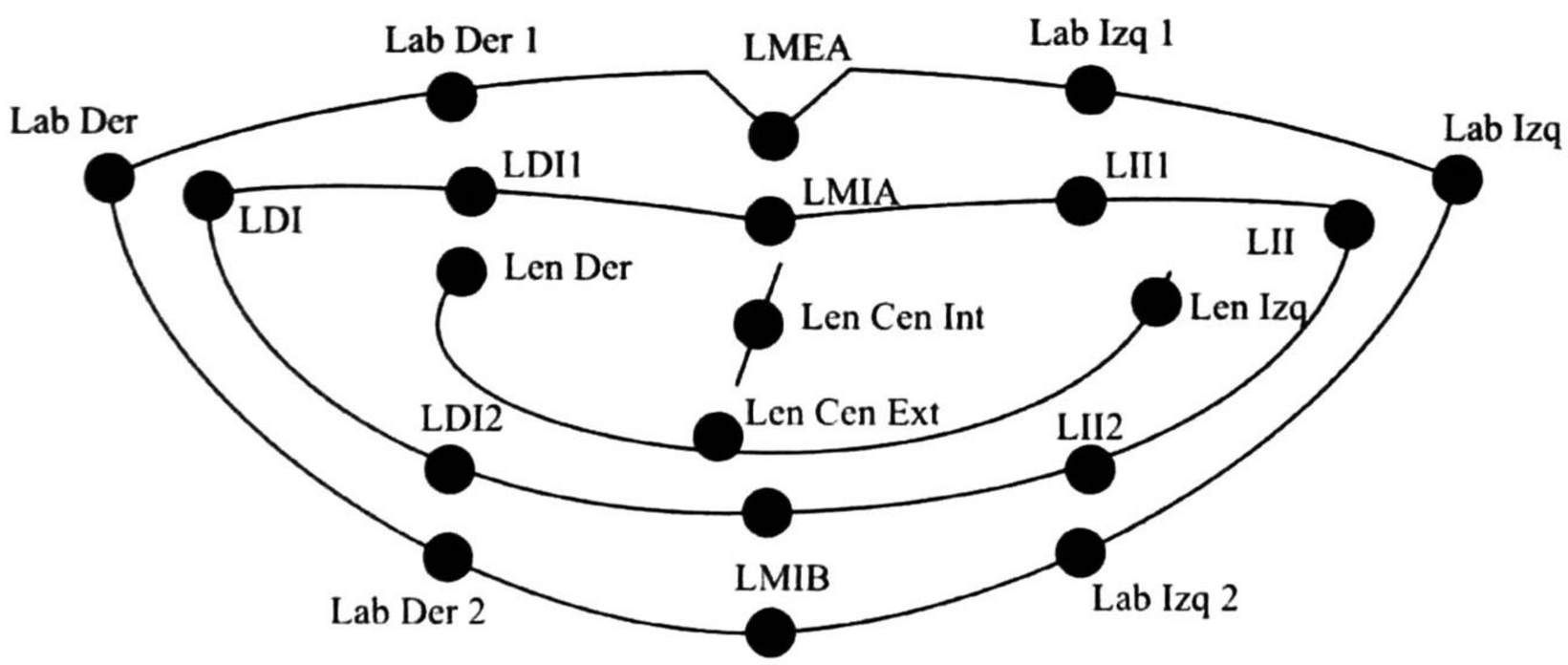


Fig. 4.5: Boca y lengua humana.

Un esqueleto considera los puntos principales para efectuar una animación, a estos puntos se les nombra empalmes del esqueleto. Los empalmes deben ser suficientes para crear una animación muy real. También es necesario considerar los empalmes de un esqueleto si se desea incrementar el conjunto de animaciones. Si se desea modificar un esqueleto, se puede tomar alguno como ejemplo a evolucionar o bien para diseñar a partir de el modelo uno nuevo.

4.3 Arquitectura

La Arquitectura para Visualizar Escenas 3D (AVE·3D) propuesta permite asignar acciones a objetos 3D o avatares para generar animaciones visuales. Las características de la arquitectura propuesta son:

- Independiente del lenguaje de implementación y del motor 3D empleado. La arquitectura puede implantarse en cualquier lenguaje y se pueden utilizar diferentes motores 3D.
- Está basada en módulos y es abierta. Esto quiere decir que se pueden modificar los módulos existentes o bien implementar nuevos que incrementen las funcionalidades de la arquitectura.
- Distribuida ya que la comunicación que emplea se basa en sockets [3.4] por lo cual una implantación puede considerar diferentes maquinas.
- La arquitectura permite incrementar fácilmente el conjunto de acciones que puede realizar un tipo de objeto 3D o avatar. Los pasos serian: 1) agregar las acciones al modulo del Esqueleto. 2) Modificar el modulo Acciones y por ultimo 3) agregar el tag correspondiente con las nuevas acciones del módulo en el módulo XmlRender.

La implementación actual de AVE·3D está Java lo cual permite trabajar en cualquier plataforma. El usuario podrá utilizar conjuntamente LIA·3D y AVE·3D el primero para modificación de escenarios 3D y AVE·3D para mostrar los cambios solicitados por LIA·3D. Los requerimientos de hardware y software del sistema son descritos en 4.5.

A continuación se hace una descripción de cada uno de los módulos de la arquitectura propuesta y mostrada en la figura 4.6.

Acciones: Seleccionara el tipo de avatar con el cual se realizara la acción deseada, de esta forma si en el ambiente existe un avatar de un ser humano y una ave se podrá distinguir como es que el avatar podrá llevar acabo dicha acción.

Esqueleto: en este modulo se especifican todas las acciones que podrá realizar un avatar, así como las acciones básicas de un esqueleto, como el mover una extremidad, avanzar, etc. Las acciones mas elaboradas son desarrolladas en el tipo de esqueleto, de esta forma se permite tener cualquier tipo de esqueleto que se desee.

EsqueletoX1 Xn: módulos donde se especializa en los movimientos dependiendo del tipo de esqueleto con el cual se desea realizar una acción, ya que no es lo mismo animar un caminar de un ser humano a un caminar de un cuadrúpedo.

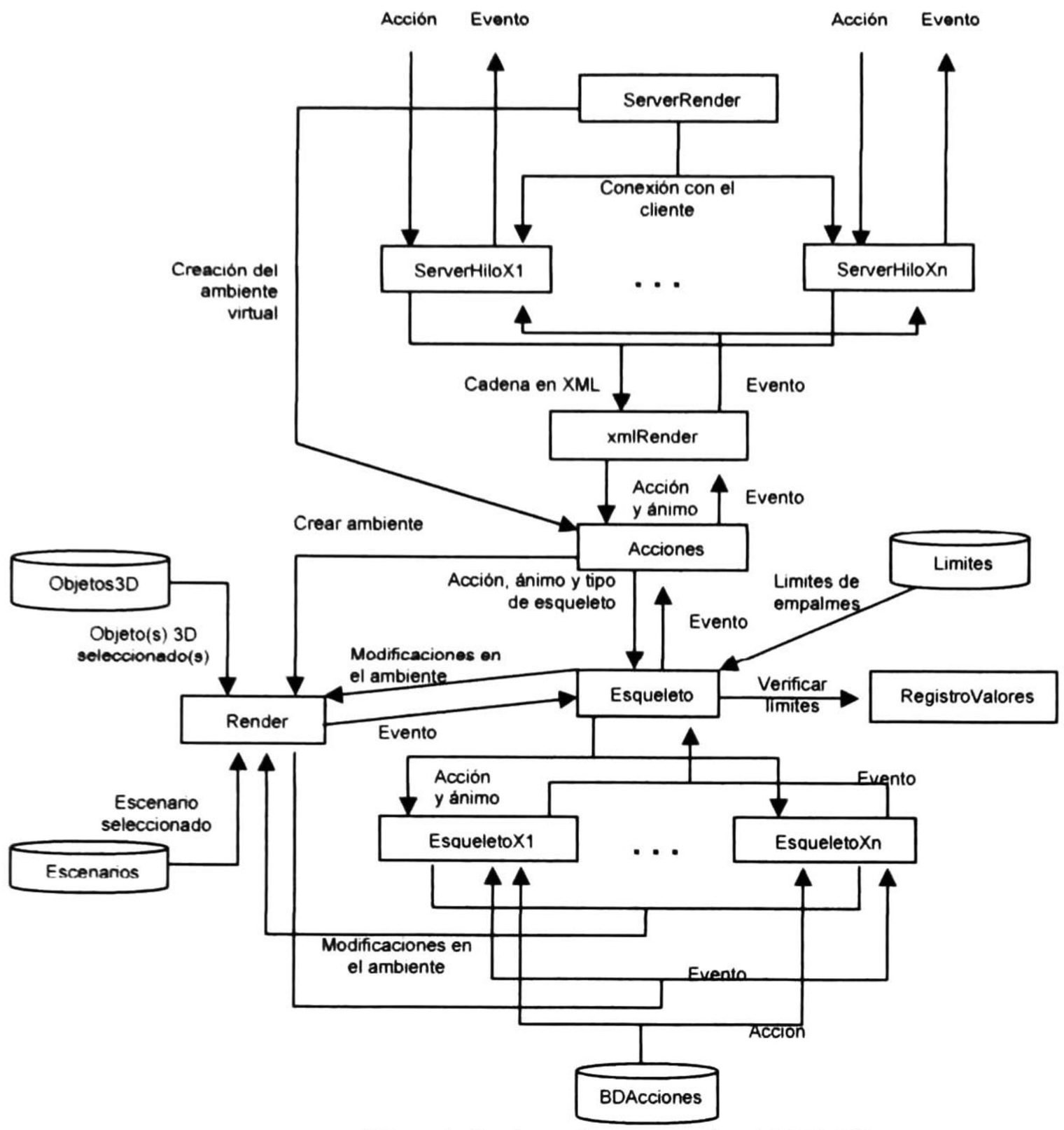


Fig. 4.6: Arquitectura de AVE·3D.

RegistroValores: contiene una estructura en la cual se podrá almacenar los limites de cada empalme de un esqueleto, no importando el tipo de esqueleto con el que se este trabajando. Su finalidad es evaluar los límites máximos y mínimos de cada empalme para evitar un movimiento erróneo, por ejemplo si a un avatar de un ser humano le indicamos que gire su cabeza más de 180° no lo permitirá ya que en los movimientos naturales de una persona no es posible.

Render: este modulo es el encargado de mostrar gráficamente todo lo acontecido dentro del ambiente virtual, así como cargar el ambiente y los objetos en 3D elegidos, el usuario es el que indicara la posición de cada avatar dentro del ambiente, esto es mediante un tipo de archivo el cual es leído al tiempo en que se carga el ambiente virtual.

ServerHilo X1 . . Xn: el número de estos módulos dependerá del número de usuarios a los que estén conectados al ambiente. Recibirá la acción en XML la cual será parseada para obtener así la acción y sus parámetros que se debe mostrar en el ambiente virtual.

ServerRender: crea el servidor del ambiente, así como también le indica al Render que debe crear el ambiente virtual. Se encarga de dar conexión a cada usuario que desee utilizar el ambiente virtual.

XmlRender: este modulo contiene el parse [3.5] para la cadena XML, la cual estará estructurada de acuerdo al lenguaje implementado, de esta forma se le indicara al Render que cambio en el ambiente deberá de mostrar.

Objetos3D: contiene todos los objetos 3D que el usuario podrá utilizar para diseñar su ambiente 3D, dentro de este conjunto puede existir cualquier objeto ya sea que contenga un esqueleto o no.

Escenarios: escenarios disponibles para la creación del ambiente virtual, los cuales deberá tener conocimiento de su existencia el cliente.

Limites: almacena los limites máximo y mínimos de un empalme de un esqueleto, esto es para que si el usuario ordena un movimiento sea lo mas realista posible.

BDAcciones: contiene los keyframe necesarios para realizar una acción, así como los movimientos involucrados. Cuando se cree una nueva acción, los movimientos es los ejes podrán ser guardados en esta base de datos o en todo caso en el modulo correspondiente, es recomendable se guarden en la base de datos para hacer mas ligero el código.

Un ejemplo del funcionamiento de la arquitectura es el siguiente cuando se desea agregar un objeto o avatar, las tareas son: Primero, crear un modulo de tipo *EsqueletoXn* el cual contendrá el conjunto de acciones que puede realizar ese esqueleto, este modulo creado hereda del módulo *Esqueleto* padre. El segundo paso es agregarlo al módulo de Acciones y por tercer y último paso, modificar el XmlRender para que acepte las acciones propias del nuevo esqueleto creado. Este sería el caso para poder agregar un nuevo objeto 3D[1] o avatar cuando ya existe un módulo de tipo esqueleto previamente.

4.4 Ejemplo del flujo de datos

Supongamos que un usuario desea participar en la evolución de un ambiente virtual mediante LIA·3D, su trabajo consistirá en seguir estos pasos:

1. Crear una aplicación que se comunique por medio de un socket al módulo ServerRender: Por medio de esta aplicación se podrá mostrar al usuario las acciones permitidas, los objetos 3D y avatares existentes en el ambiente virtual. El socket deberá conectarse al servidor (ServerRender) del ambiente por medio del puerto 4445, este creara un hilo (ServerHiloXn) para poder mantener la comunicación y así mandar acciones y recibir eventos.

2. Crear las cadenas en XML siguiendo la sintaxis de LIA·3D: Para cada acción deseada, se creara la correspondiente cadena XML. Por ejemplo: Consideremos el avatar de un ser humano el cual tiene el identificador 3,

[1] Por simplicidad estamos considerando que los objetos pueden realizar acciones, sin embargo esto puede ser modificado especificando únicamente las propiedades de los objetos, i.e. su masa, su elasticidad, etc. y los avatares al interactuar con ellos impondrán un efecto sobre estas propiedades.

y deseamos que gire a la izquierda 30 grados, camine 5 pasos y que esto se ejecute mostrando un estado de animo de enojo, la cadena XML correspondiente es la siguiente:

```
<accion ident=3>
        <angry> </angry>
        <girar>
                <rotar> 10.5 </rotar>
        </girar>
        <caminar>
                <pasos> 5 </pasos>
        </caminar>
</accion>
```

3. Mandar el XML al servidor: Una vez generada la cadena XML, esta debe enviarse por el socket creado al servidor (ServerHiloXn). La secuencia que ocurrirá es la siguiente:

    1. El módulo ServerHiloXN envía la cadena al módulo XmlRender. Al recibir este último la cadena la parsea para identificar la acción a ejecutar y sus parámetros. En este caso la acción caminar, enojado y girar.
    2. El módulo Acciones recibe la acción a ejecutar y por medio del identificador del avatar decide el tipo de esqueleto debe realizarla. A continuación indica al esqueleto que ejecute la acción indicando los parámetros. Para nuestro ejemplo el identificador 3 corresponde a un esqueleto de tipo humano, y las acciones permitidas están definidas en el módulo EsqueletoHumano.

        El modulo de EsqueletoXn hereda del módulo Esqueleto las acciones que varios avatares tengan en común. Si la acción es especifica de un avatar entonces se usa el modulo EsqueletoXn donde están especificadas las acciones especificas del avatar. Para nuestro ejemplo, EsqueletoHumano determina la acción caminar de un humano mediante la consulta a la tabla BDAcciones la cual contiene los movimientos de los empalmes involucrados, los cuales son regresados al modulo EsqueletoHumano. Las acciones deben identificarse para cada tipo de avatar ya que estas deben ser diferentes, por ejemplo las acciones de caminar para un caballo y un humano son diferentes.
    3. El modulo de Esqueleto o EsqueletoXn le indicara al Render que tipo de actualizaciones debe mostrar.

4. Escuchar al servidor para captar los eventos surgidos por las acciones realizadas. Supongamos en nuestro ejemplo, que la acción de caminar provoca un evento de colisión. Esto iniciara la siguiente secuencia:

1. El Render detectara la colisión y los objetos involucrados, una vez esta información determinada se empaqueta y es enviada al modulo de Esqueleto.
2. Esqueleto identificara que acción es la que provoco la colisión y es enviada al módulo Acciones
3. Acciones identifica el tipo de esqueletos para enviarlos al módulo XmlRender
4. XmlRender genera la cadena XML que corresponde al evento de colisión y la envía por medio del ServerHilo al cliente que envío la acción.

4.5 Requerimientos del Sistema

4.5.1 Hardware

Espacio en Disco duro: 60 MG
Memoria RAM: 256 MG
Procesador: 1.60 GHz
Acelerador de Gráficos: debe soportar OpenGL, 128 MG

4.5.2 Software

Sistema Operativo: Multiplataforma
Java: versión 1.4
Librerías: AgentFx versión 2.1.2 (Motor de 3D)
Base de Datos: MySql 4.1 y conector para java
Xml: Jdom

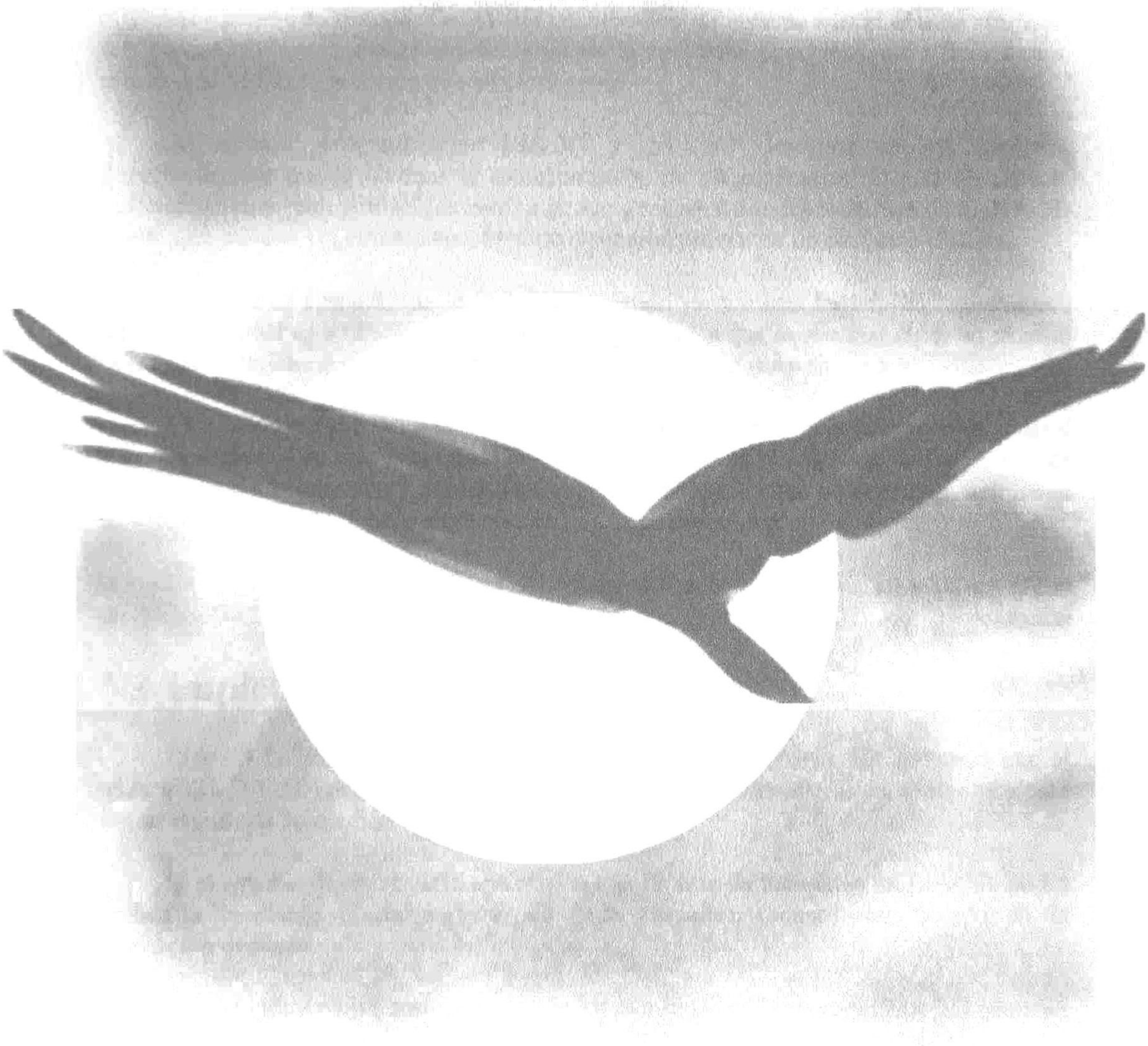
4.6 Conclusiones

La arquitectura AVE·3D propuesta cumple con las requerimientos de permitir la manipulación de avatares 3D necesarios para la simulación de escenas en 3D.

Además AVE·3D tiene características de ser abierta, distribuida y multiplataforma necesarias para el proyecto GeDA-3D del cual es parte este proyecto.

Actualmente esta programado en Java utilizando AgentFx como motor de 3D, Sin embargo la propuesta no esta atada a este motor sino que funciona con cualquier motor que permitan la animación por esqueletos de los avatares en tiempo de ejecución.

Capítulo 5 Resultados



Resumen

En este capítulo se mostraran los resultados obtenidos en esta tesis. El capítulo se divide en dos partes: La primera muestra el trabajo que realiza LIA·3D y la segunda parte es el funcionamiento de la arquitectura AVE·3D descrita en el capítulo 4.

5.1 Introducción

GeDA-3D es un sistema que ofrece facilidades 3D y está orientado a usuarios finales, es decir usuarios sin conocimiento de computación. GeDA-3D esta en desarrollo. Entre las partes que constituyen GeDA-3D está la de visualizar la evolución de un escenario. Este trabajo de tesis está orientado a generar esta parte del sistema y una de las características necesarias es que debe de realizar modificaciones en tiempo real del ambiente virtual.

Al iniciar este trabajo, GeDA-3D contaba con un sistema visual el cual realizaba una representación mediante una herramienta la cual daba poco realismo a las escenas, por lo cual se busco la forma de solucionar esto.

Al trabajar conjuntamente LIA·3D y AVE·3D descritos en los capítulos precedentes, se puede obtener la representación de un escenario 3D con un aspecto suficientemente real. Ambas herramientas son generales e independientes de GeDA-3D por lo que pueden ser empleadas conjunta o separadamente en un proyecto distinto.

La línea de aprendizaje de LIA·3D es rápido ya que esta basado en la estructura de XML, el cual es un lenguaje basado en etiquetas. Ya que la sintaxis fácil, un usuario con o sin experiencia puede manipular un escenario 3D en poco tiempo.

Otra ventaja del resultado de este trabajo es que tenemos una herramienta general y no especifica a un tipo específico de ambiente, como es el caso con las herramientas existentes para la creación y manipulación de escenarios 3D que no permiten fácilmente su modificación, esto debido a que tienen una tarea especifica.

En las siguientes secciones se describirá el funcionamiento de LIA·3D así como de AVE·3D esto médiate algunos ejemplos.

5.2 Implementación de LIA·3D

LIA·3D facilita el manejo de la evolución de un escenario 3D, evitando que el usuario (GeDA-3D) tenga que diseñar cada movimiento necesario de un avatar esto para lograr crear una animación.

Si el usuario desea realizar una acción que no este contemplada en LIA·3D, podrá realizarla mediante el movimiento de cada empalme (hueso) involucrado en la animación deseada.

Para tener un mejor conocimiento de los cambios en el escenario LIA·3D contiene un conjunto de eventos, los cuales pueden ayudar al usuario a tener mas control del escenario.

A continuación se mostrara un ejemplo de cómo es que funciona LIA·3D, con el objetivo de que sea mas claro su trabajo.

Los conocimientos que el usuario (GeDA-3D) debe tener es conocer un lenguaje de programación que permita la conexión por socket (el socket utilizado es 4445, la

dirección IP o el nombre de la maquina puede variar). Conocer un poco de XML, esto para poder interpretar los eventos recibidos. El conocer las acciones descritas en LIA·3D y que tipo de avatar las puede realizar evitara que el usuario reciba constantemente eventos indicándole que la acción no pertenece al avatar seleccionado.

Teniendo la aplicación que maneje las acciones y eventos de LIA·3D surgirán los siguientes pasos:

1.

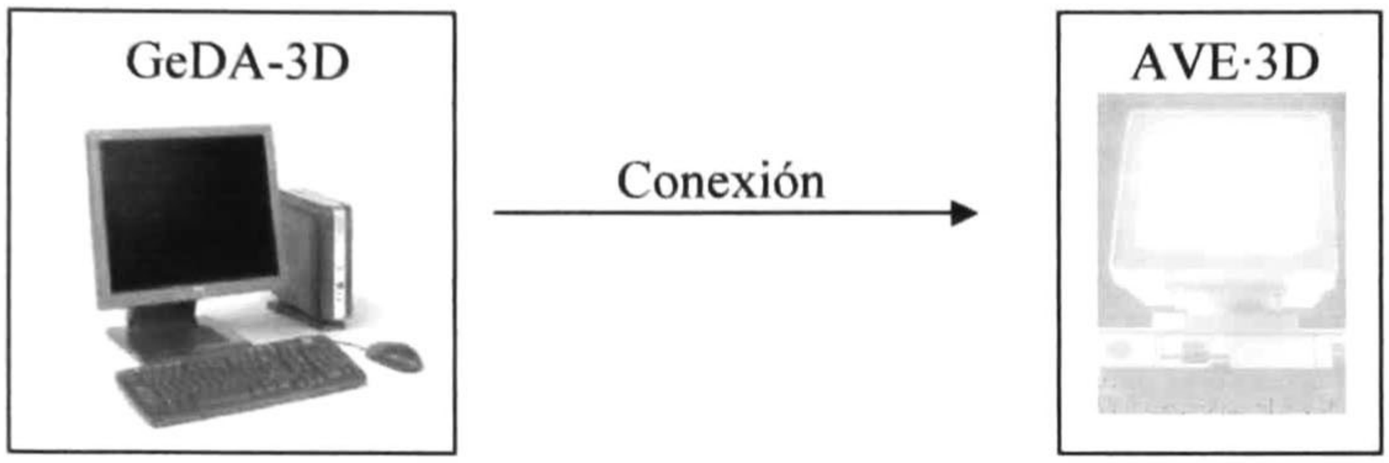


Fig. 5.1: Conexión con AVE·3D

Al iniciar la conexión con el Servidor de AVE·3D se creara un hilo para poder escuchar los eventos que surjan.

2.

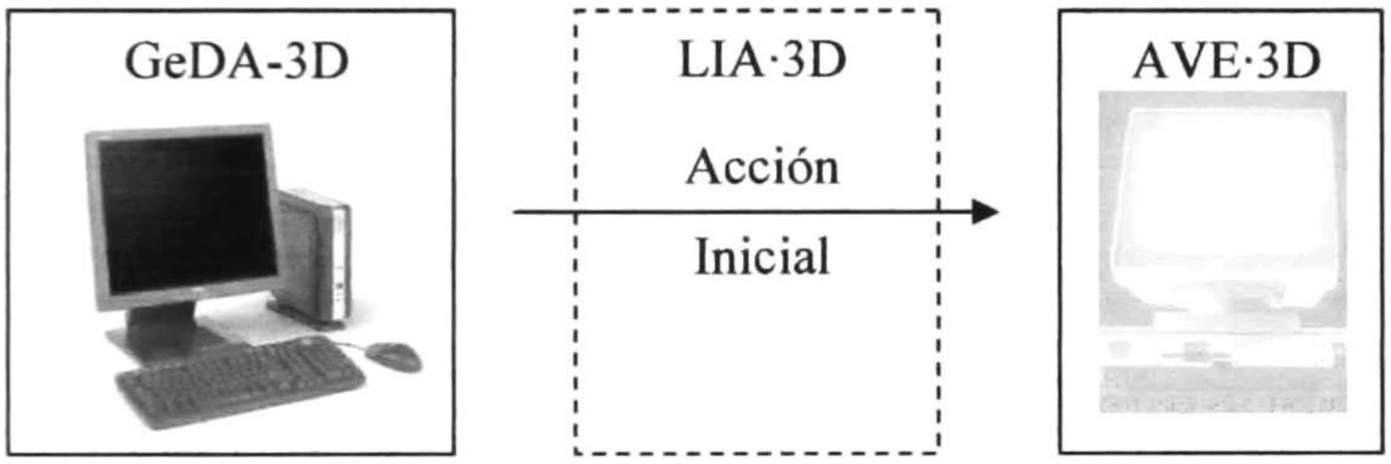


Fig. 5.2: Acción inicial

```
<accion>
        <inicial></inicial>
</accion>
```

Mediante la acción inicial GeDA-3D solicitara información del escenario 3D, la cual incluye el nombre y posición de los avatares u objetos 3D.

3.

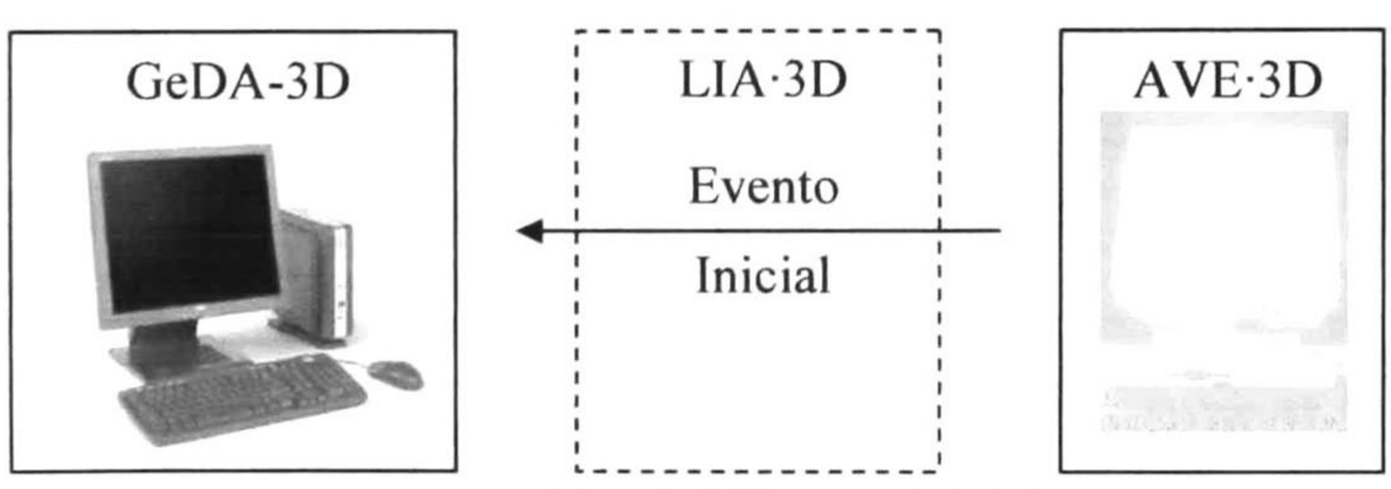


Fig. 5.3: Evento inicial

```
<evento>
        <inicial>
                <avatar>
                        <nombre>HombreVerde</nombre>
                        <posX>-30.2</posX>
                        <posY>0.0</posY>
                        <posZ>10.5</posZ>
                </avatar>
                <avatar>
                        <nombre>Chapulin</nombre>
                        <posX>-30.2</posX>
                        <posY>0.0</posY>
                        <posZ>10.5</posZ>
        </avatar>

        </inicial>
</evento>
```

AVE·3D responderá a la acción inicial enviando la información de los avatares u objetos incluidos en el escenario. La forma en que son mandados indica el numero de identificador asignado a cada avatar u objeto 3D.

4.

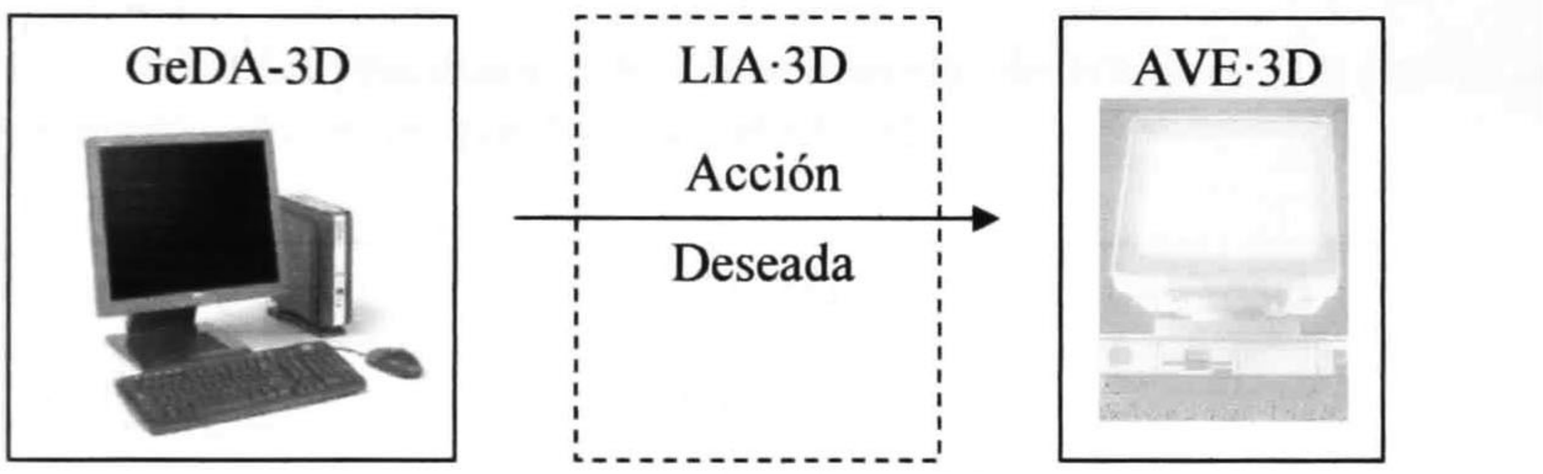


Fig. 5.4: Acción solicitada

```
<accion id=0>
        <expresion>
                <modo>
                        <afraid intensity=" medium">
                        </afraid>
                </modo>
        </expresion>
</accion>
```

Ya teniendo conocimiento de los avatares u objetos 3D, GeDA-3D podrá solicitar que se realice una acción en específico y quien la ejecutara.

5.

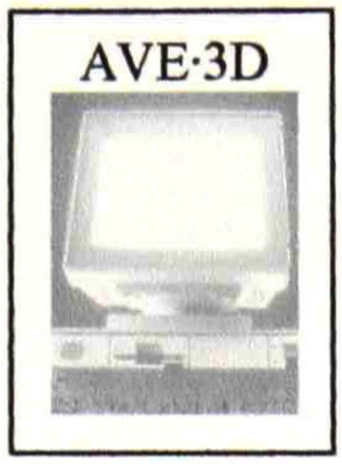


Fig. 5.5: Render de AVE·3D

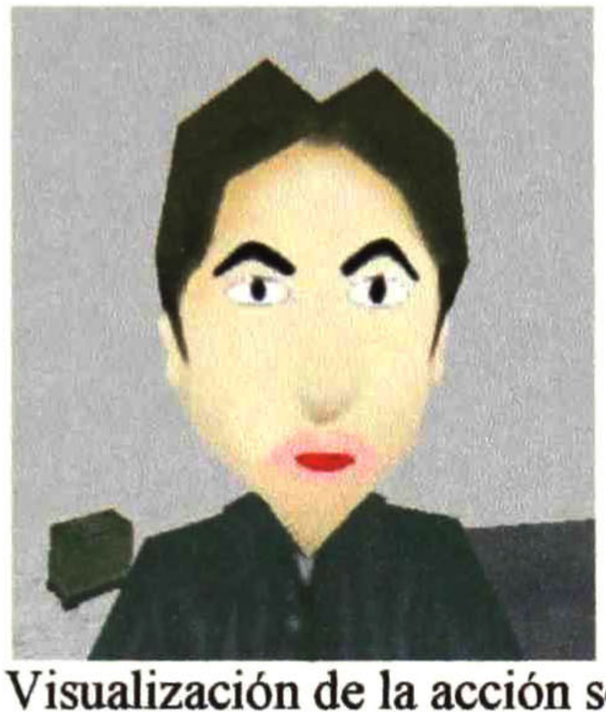


Fig. 5.6: Visualización de la acción solicitada.

AVE·3D procesara la acción requerida, de esta forma se podría generar un evento el cual es mandado a GeDA-3D.

6.

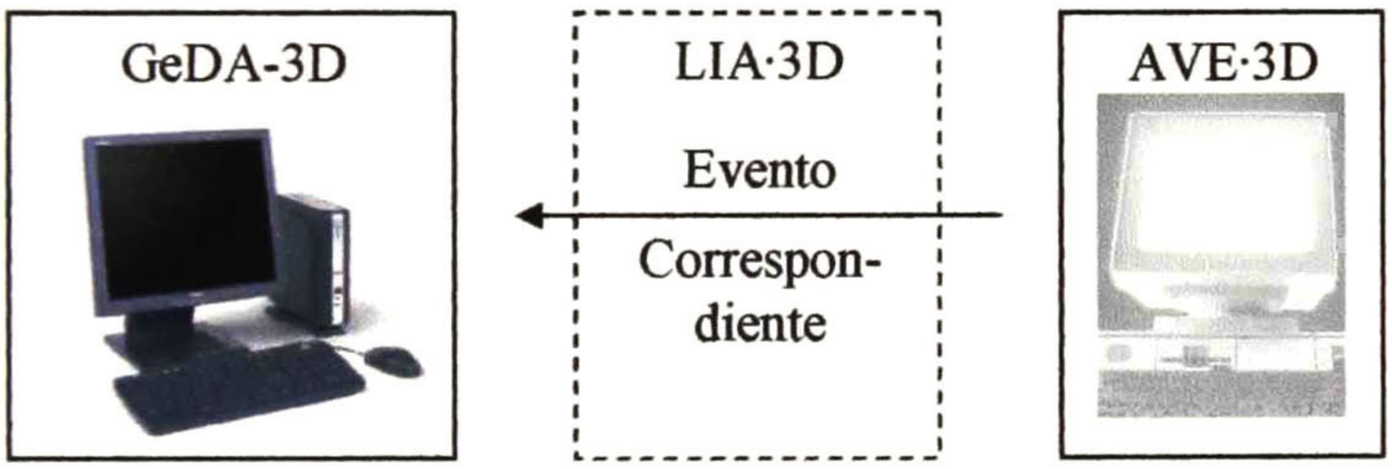


Fig 5.7: Evento resultante

```
<evento>
        <final_correcto>
                <ident>0</ident>
                <accion>expresion</accion>
        </final_correcto>
</evento>
```

El cliente (GeDA-3D) obtendrá un evento acorde a lo ocurrido con la acción que envió. De esta forma podrá seleccionar la siguiente acción.

La siguiente tabla muestra el tiempo requerido para algunas de las acciones. Las acciones elegidas son las que tienen los tiempos más representativos:

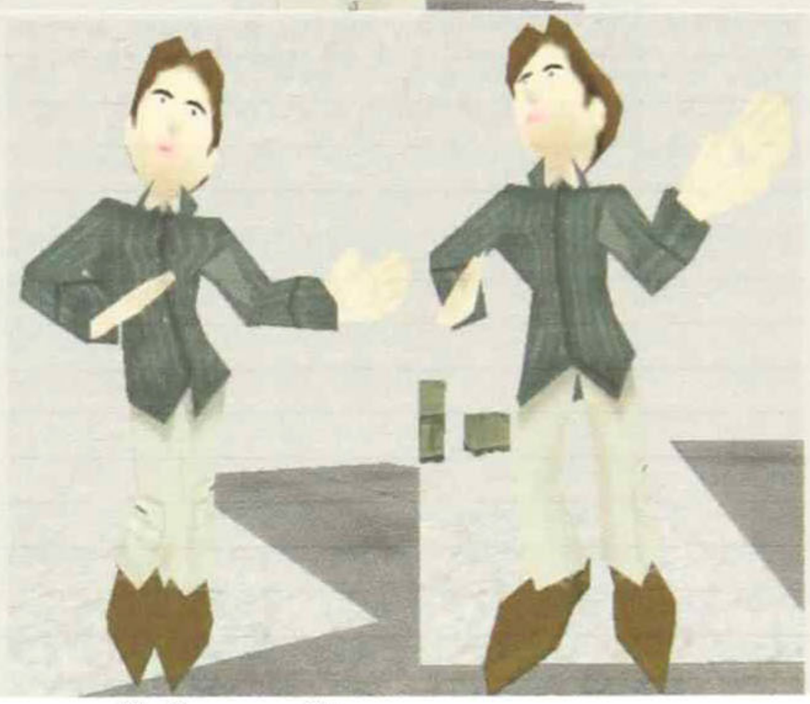
	Nombre	Tiempo	Imagen
1	Expresión	2 seg.	
2	Aquí	15 seg.	
3	Bailar	25 seg.	

Tabla 5.1: Tiempo requerido para dicha acción.

Debe tenerse en cuenta que la acción bailar toma mas tiempo ya que consta de un conjunto de movimientos. Estas pruebas se realizaron en un equipos que cumple con los requerimientos mínimos mencionados en el capitulo 4.

5.3 Implementación de AVE·3D

Esta arquitectura recibe como entrada las acciones de LIA·3D, estas acciones son interpretadas para modificar un escenario 3D. Debido a esta arquitectura el usuario de LIA·3D no debe conocer como es que se procesa internamente la acción de su elección.

AVE·3D puede ser modificado por una persona que tenga conocimientos en programación, a diferencia del usuario de LIA·3D. La arquitectura actualmente esta implementada en Java, una persona que tenga conocimientos en Java y XML podrá modificar tanto a AVE·3D como a LIA·3D.

Los ambientes virtuales existentes son una aplicación completa, los cuales no permiten ser aprovechados por algún otro grupo de trabajo. AVE·3D puede ser incluido en algún otro trabajo.

El funcionamiento interno de AVE·3D se basa en la comunicación por módulos lo cual hace un poco mas independientes los cambios en estos.

La siguiente tabla muestra el tiempo requerido para iniciar el servidor y la conexión:

	Iniciar	Tiempo estimado
1	Servidor de AVE·3D	8 seg.
2	Conexión con el Servidor	11 seg.

Tabla 5.2: Tiempo estimado para iniciar la conexión y servidor.

El tiempo de inicio del servidor y la conexión varían dependiendo de la cantidad de avatares u objetos 3D que contiene el escenario. En este caso se tomo el tiempo mínimo.

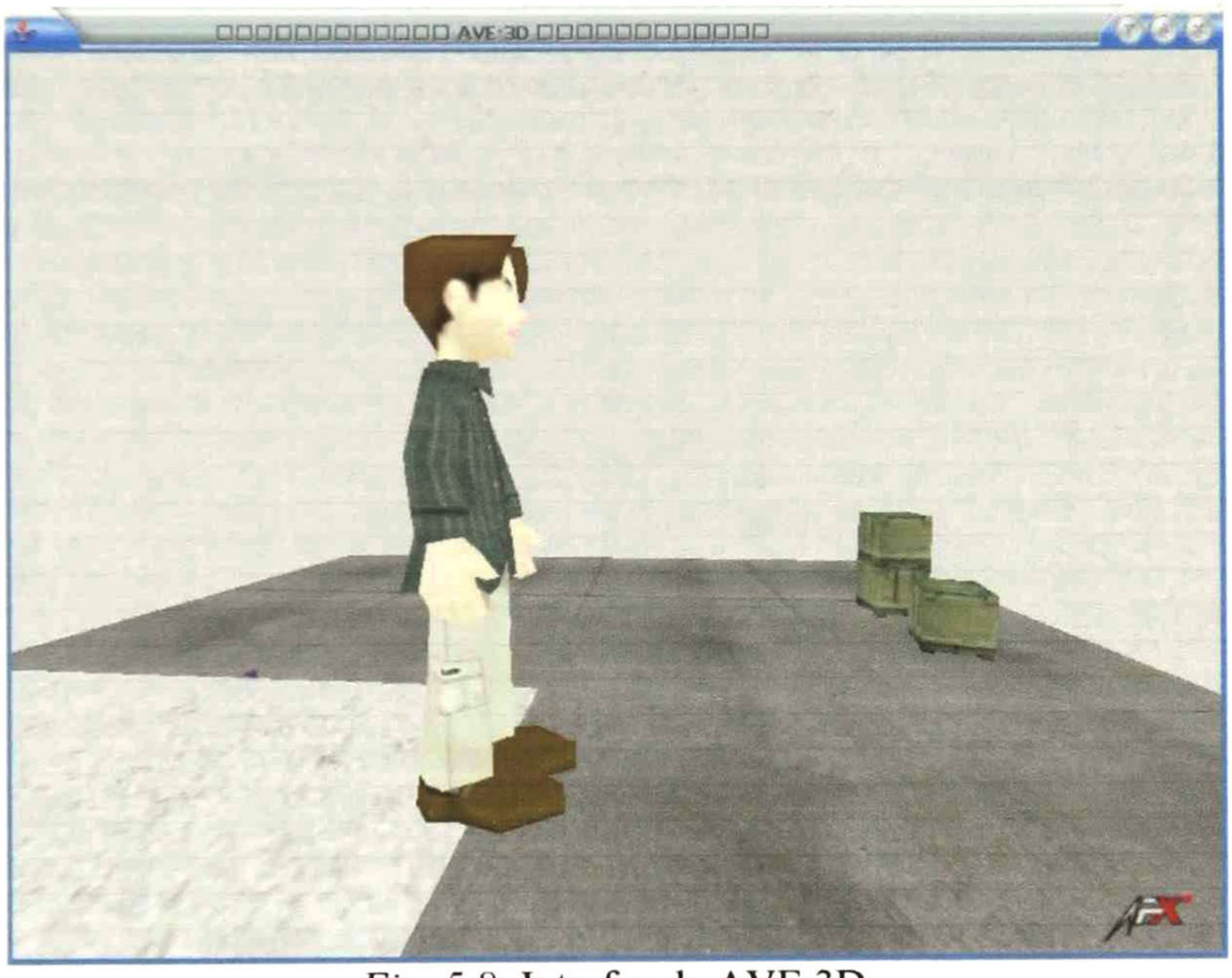


Fig. 5.8: Interfaz de AVE·3D.

Si en algún momento el usuario desea agregar un nuevo avatar a los ya existentes, podrá utilizar los esqueletos publicados. Debe diseñar el aspecto de su avatar y asignarlo al esqueleto.

Para esto se debe utilizar un editor de 3D (Maya, 3dMax, ver &cap2), es recomendable que se realice en Milkshape 3D ya que es el formato que acepta el motor de 3D (AgentFx). Milkshape 3D cuenta con varios importadores de los formatos mas utilizados en los juegos de video comerciales.

Para poder tener el avatar terminado, es decir, el diseño asignado al esqueleto, requiere de un tiempo considerable. Si se tiene conocimiento en Milkshape 3D podría tardar algunas horas en terminar el avatar. Esta demora es debida a que se deben asignar

a prueba y error las asignaciones del diseño al esqueleto para lograr que el diseño se mueva como se desea.

Cuando es finalizado el avatar, se puede diseñar una acción para dicho avatar. Este diseño también requiere tiempo, ya que se debe mover cada empalme a la posición deseada. Esto es a prueba y error.

La ventaja de esto es que para cuando se desee aplicar a otro avatar del mismo tipo ya no se tiene que hacer nuevamente la animación, como es el caso de las animaciones prediseñadas.

A continuación se describirá como es el trabajo de AVE·3D en cuatro situaciones, las cuales son:

1. Iniciar el servidor,
2. Realizar la conexión con el servidor,
3. Solicitar realizar la acción inicial, por ultimo,
4. Solicitar la ejecución de una acción, en este caso la acción bailar.

Al momento de iniciar el servidor se realiza la siguiente comunicación dentro de AVE·3D:

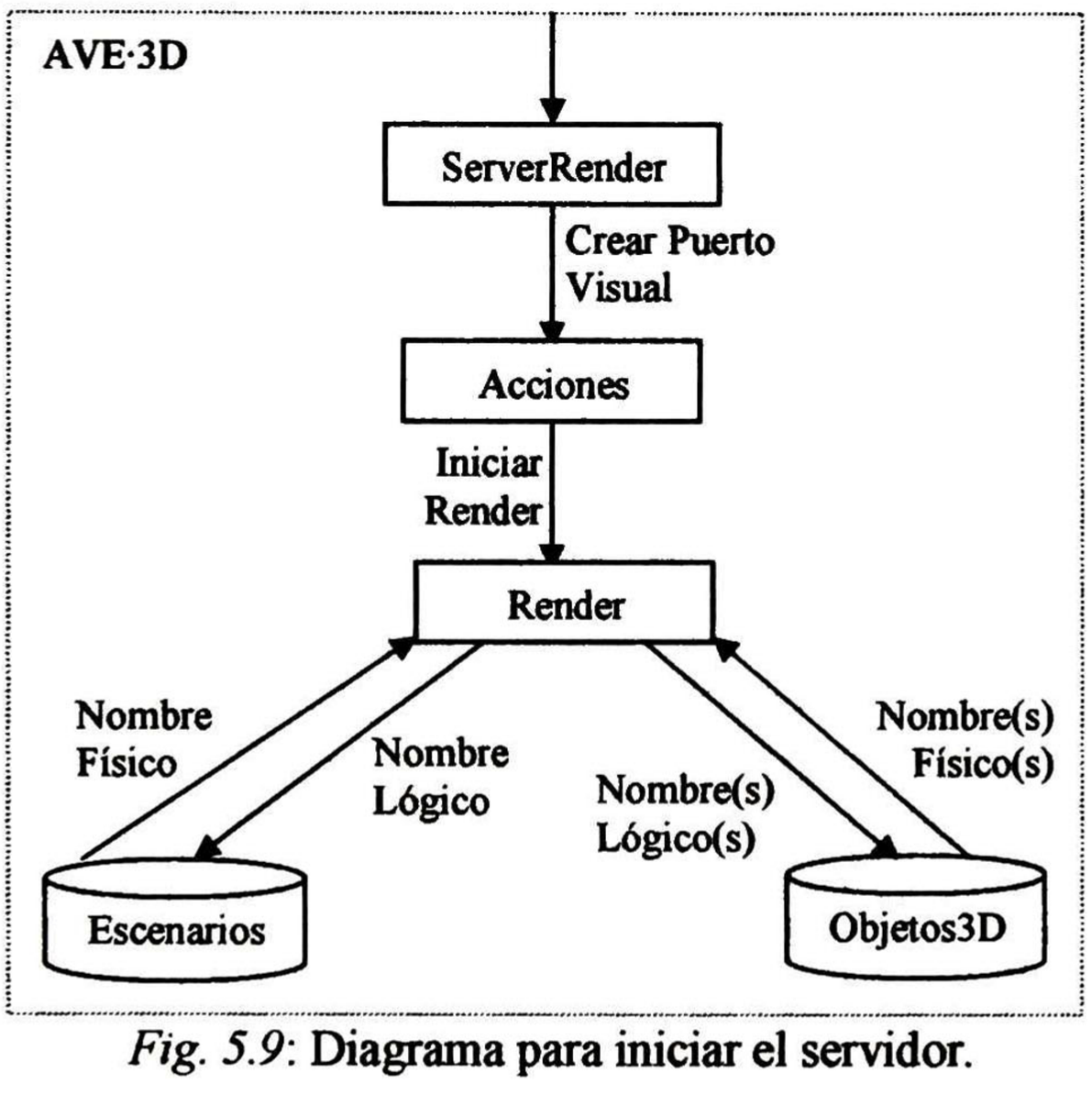


Fig. 5.9: Diagrama para iniciar el servidor.

Para iniciar el servidor se ejecuta el módulo ServerRender, el cual dará conexión a los nuevos clientes. El módulo ServerRender se comunica con el módulo de Acciones para solicitarle que se construya el puerto visual.

El puerto visual es desplegado por el módulo Render, este módulo lee un archivo especial, que contiene lo(s) avatar(es) y/o objeto(s) que formaran parte del escenario 3D, así como el escenario que se utilizara.

Los nombres incluidos en el archivo son nombres lógicos, estos son consultados en la base de datos Objetos3D, la cual contiene los nombres físicos y su ubicación. Esta información es requerida por el modulo Render.

El siguiente diagrama muestra el flujo de la información cuando se realiza la conexión con el servidor.

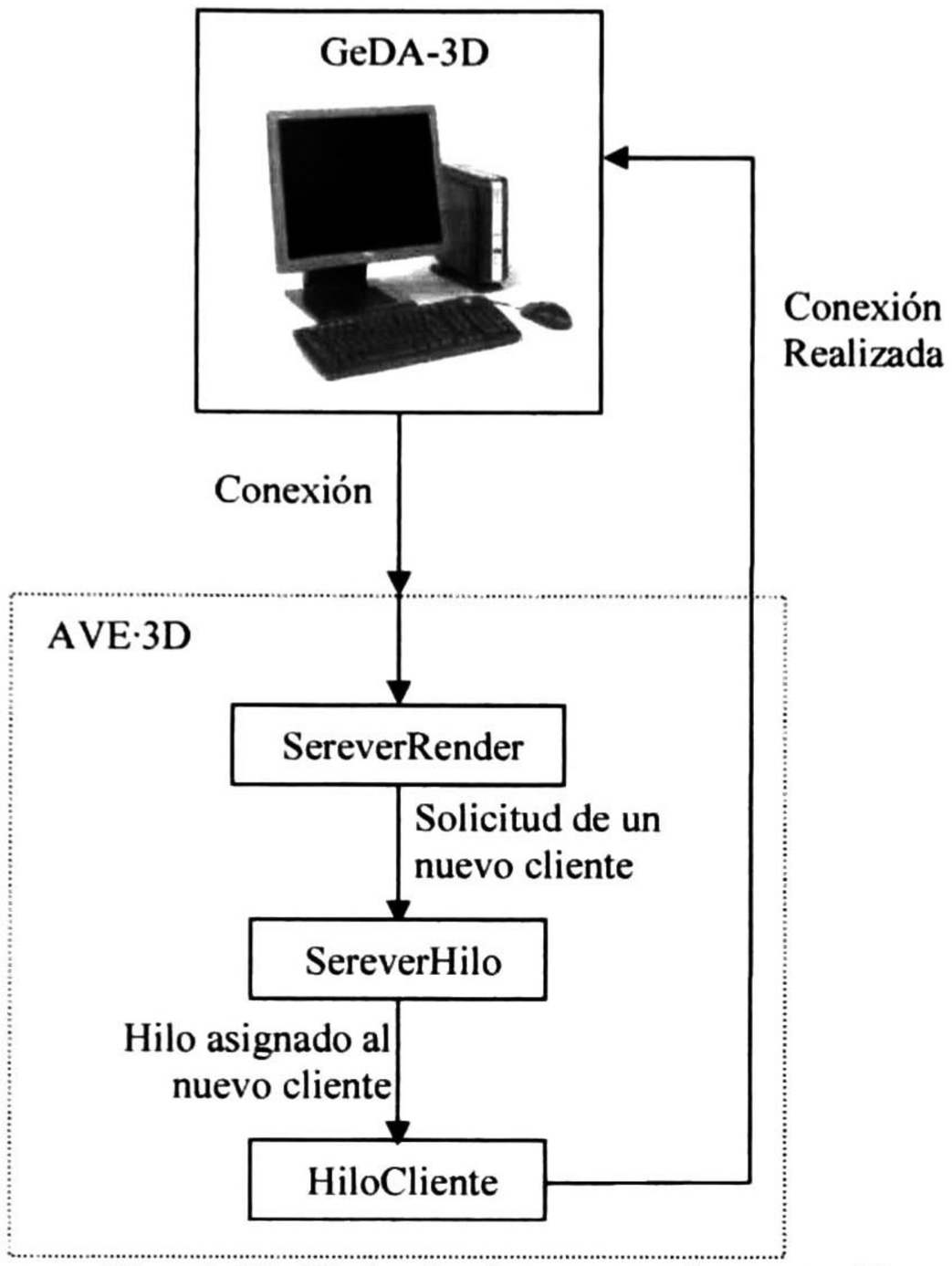


Fig. 5.10: Flujo de datos para la conexión.

El usuario debe de conectarse con el servidor por medio de un socket, esto le indicara al modulo ServerRender que debe solicitarle al modulo ServerHilo que se cree un nuevo hilo para mantener la comunicación con el usuario.

Después de realizar la conexión, el usuario (GeDA-3D) se comunicara mediante el hilo creado. De esta forma cada cliente conectado tendrá un hilo asignado.

Cuando se solicita la ejecución de la acción inicial, AVE·3D trabaja internamente de la siguiente forma descrita en la Fig. 4.11:

Cuando un cliente realiza la acción inicial es enviada por el socket y recibida por el hilo asignado previamente. De esta forma llega al modulo XmlRender el cual identifica la acción a ejecutar.

La acción inicial es mandada al modulo de Acciones, este modulo se comunica con el modulo de Render pidiéndole los objetos existentes. El modulo Render responde con el (los) nombre(s) y posición(es) del (los) objeto(s) existentes en el escenario 3D.

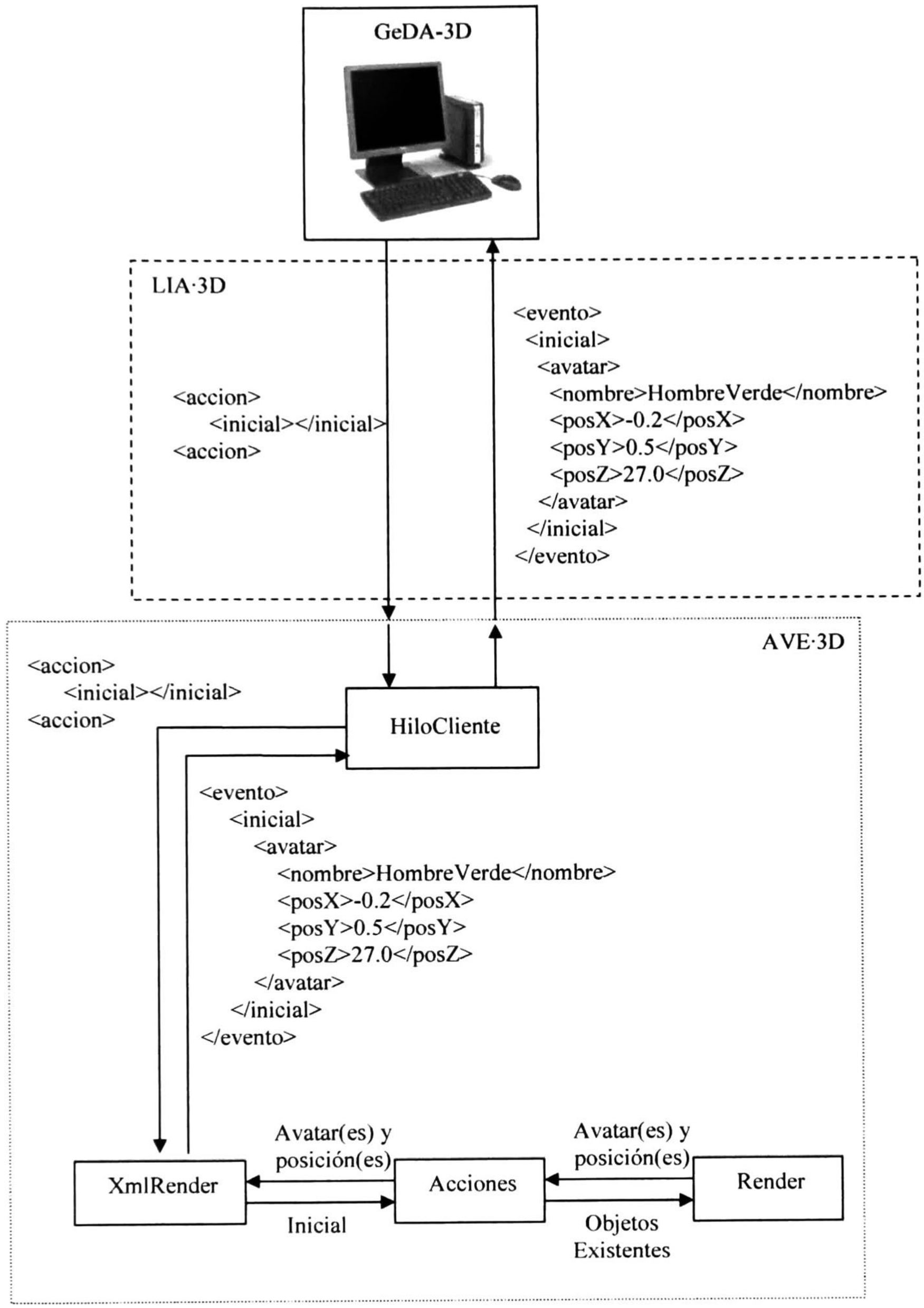


Fig. 5.11: Comunicación de módulos para la acción inicial.

Esta información es enviada de vuelta a través de varios módulos hasta el modulo XmlRender, el cual forma la cadena en XML equivalente al evento inicial. Una vez formado el evento inicial, este es enviado al hilo que sostiene la comunicación con el cliente.

Otro de los casos en que el usuario interactúa con AVE·3D, es cuando se ejecuta alguna acción contenida en LIA·3D. El siguiente diagrama muestra como ejemplo el caso cuando el usuario solicita la realización de la acción bailar a AVE·3D.

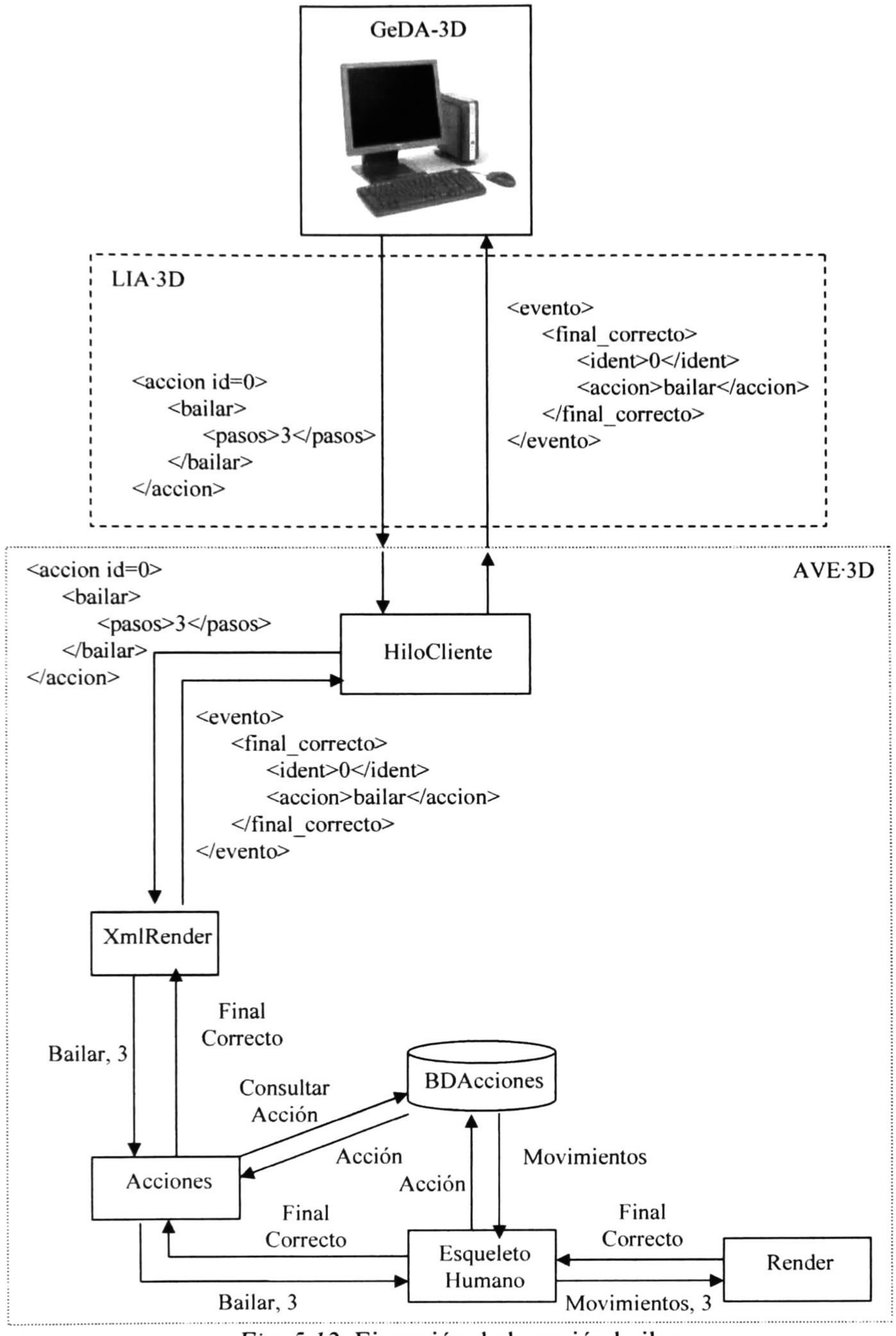


Fig. 5.12: Ejecución de la acción bailar.

La acción bailar es mandada por el socket, llegando al hilo asignado, este pasara la cadena XML al modulo XmlRender. El modulo XmlRender identificara la acción bailar.

El modulo Acciones recibe la acción y consulta la base de datos BDAcciones para verificar que pertenece al conjunto de acciones del avatar u objeto 3D indicado en la cadena XML.

Si la acción pertenece al avatar u objeto 3D, se elige el modulo Esqueleto correspondiente. En nuestro ejemplo se selecciona el modulo EsqueletoHumano.

El modulo EsqueletoHumano consultara la base de datos BDAcciones para obtener los movimientos necesarios para realizar la acción bailar. Estos movimientos se pasan al modulo Render, el cual muestra visualmente los cambios indicados según los movimientos recibidos.

Cuando la acción se realiza sin ningún contratiempo, se indica que finalizo correctamente. Esto llega al modulo XmlRender, de acuerdo a este final se estructura el evento de final correcto. Este evento es descrito en LIA·3D.

De esta misma forma se ejecutaran las acciones. Si existe algo que impida que se realice la acción se generara un evento para indicarle al usuario (GeDA-3D) que la acción solicitada no fue realizada.

5.4 Conclusiones

Podríamos decir que los usuarios que manejan estas dos aplicaciones se dividen en dos:

1. Cuando solo se desea utilizar LIA·3D y AVE·3D, no requiere de mucho trabajo. Estas dos aplicaciones están diseñadas para ahorrar trabajo al usuario final. Se pensó en un constante crecimiento, por esta razón LIA·3D esta basada en XML y AVE·3D esta dividida en módulos.
2. Cuando el usuario desea enriquecer alguna o ambas aplicaciones. El tipo de usuario en este caso requiere de más conocimientos en programación que el descrito anteriormente.

Las acciones descritas en LIA·3D son algunas de las que puede ejecutar un avatar de un ser humano, los parámetros incluidos en cada una de ellas son los mínimos para poder ejecutar la acción. Es posible detallar los parámetros y así incrementar el realismo de las acciones.

Los módulos de AVE·3D pueden incrementarse como es el caso de los que heredan del modulo Esqueleto, de esta forma se puede acoplar al tipo de avatar que se desee usar.

Al tener una arquitectura da la libertad de migrar la herramienta a otro motor de 3D que de más realismo en los escenarios 3D. Si en un futuro se desea realizar tal cambio, es recomendable que el motor de 3D este programado en Java para seguir garantizando que la aplicación es multiplataforma.

Capítulo 6 Conclusiones y Trabajo futuro

Resumen

La finalidad de esta sección es describir nuestras conclusiones obtenidas y el trabajo futuro para ampliar los alcances de nuestra propuesta.

6.1 Conclusiones

El objetivo es realizar un estudio para proveer a GeDA-3D con un módulo que permitiera hacer el rendering en tiempo real de una descripción de una escena en 3D fue alcanzado. El trabajo realizado para obtener este objetivo puede ser dividido dos partes principalmente que son: 1. El diseño de un lenguaje de interfaz que comunicara los requerimientos de animación de GeDA-3D y 2. El diseño e implementación de sistema que se encarga de realizar el rednering

1. El lenguaje que fue diseñado lo nombramos LIA·3D de ***Lenguaje de Interfaz para Animaciones en 3D***. Esté lenguaje satisface los requerimientos establecidos para nuestro proyecto. Las características principales del lenguaje son:

1. Tener una sintaxis fácil y en lo posible seguir un estándar. Para esto se estudiaron varios lenguajes y se selecciono que nuestro lenguaje fuera del tipo XML ya que esto facilitaría la comunicación y además es de rápida curva de aprendizaje.
2. Debe ser útil para manipular cualquier tipo de escenario.
3. Debe ser extensible, esto es que puedan agregar nuevas animaciones para nuevos tipos de criaturas virtuales que se representen en el ambiente.

2. El objetivo del sistema AVE·3D ***(Arquitectura para Visualización de Escenas 3D)*** es mostrar las animaciones descritas en LIA·3D. Este sistema interpreta las cadenas XML para convertirlas en comandos que son ejecutados por el motor 3D que en nuestro caso fue AgentFx.

AgentFx es el motor de 3D que se elegido por que cumple con las características que se deseaban (animación de esqueletos, portabilidad, manejo de cámaras, etc.). Sin embargo se puede emplear algún otro motor 3D que realice animaciones por esqueletos que es el tipo de animación que nos permite tener control de cada parte del avatar u objeto 3D. Además el motor seleccionado debe permitir la manipulación del esqueleto mediante código en tiempo real.

Con el resultado de este trabajo, los detalles en un escenario 3D dependen del diseño elaborado en un editor de 3d (en este caso Milkshape 3D), por lo cual se puede diseñar escenarios, avatares y objetos 3D con un aspecto un tan real como queramos.

Para indicar las modificaciones al escenario 3D no es necesario que GeDA-3D este en la misma computadora, ya que su comunicación es por medio de un socket. Esta característica permite que GeDA-3D este programada en otro lenguaje de programación que no sea Java.

Actualmente no todo el conjunto de acciones y eventos descritos en LIA·3D, esta programados dentro de AVE·3D, esto debido a que se requiere de tiempo para detallar cada movimiento de una avatar u objeto 3D.

Los avatares, objetos y escenarios incluidos en la base de datos, son pocos debido a que para el diseño de cada uno de ellos se requiere tener un buen nivel de conocimiento en un editor de 3D.

Los objetos 3D (puertas, ventanas, pelotas, etc.) cuentan con un conjunto de acciones para facilitar el manejo de ellos. Al tener este conjunto de acciones se evita el calculo de sus movimientos mediante formulas de física.

6.2 Trabajo futuro

- Incrementar el conjunto de acciones en AVE·3D para otro tipo de esqueleto que no sea el humano, que es el que esta implementado actualmente.
- Incrementar el número de avatares, objetos y escenarios. Esto es diferentes avatares para humanos y objetos para detallar más los escenarios.
- Hacer el puerto visual distribuido para que se tenga un rendering distribuido en diferentes maquinas.
- Incrementar las acciones y eventos descritos en LIA·3D. Actualmente se tienen solo ciertas acciones de los humanos y ninguna para otro tipo de avatar.
- Realizar continuamente una revisión de los motores de 3D, para poder mejorar la representación del puerto visual y el tiempo del rendering.
- Incrementar las funciones ofrecidas por AVE·3D, por ejemplo crear el escenario en tiempo de ejecución.

Referencias

[1.1] Un Editor de Ambientes Virtuales para GeDA-3D, Hugo Iván Piza Dávila. Septiembre 2002.
[1.2] A Platform to Design and Run Dynamic Virtual Environments. H. Iván Piza, Fabiel Zúñiga, Félix F. Ramos,Multi-Agent Systems Development Group
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional
Guadalajara, Jal., México
[1.3] 3D Emotional Agent Architecture. Félix F. Ramos, Luis Razo, Alma V. Martinez, Fabiel Zúñiga, Hugo I. Piza. Multi-Agent Systems Development Group. Innovative Internet Community Systems (I^2CS) 2005.
[1.4] Adaptive Algorithms Useful to Generate Behaviours in Dynamic Virtual Environments. Fabiel Zúñiga Gallegos. Septiembre 2002.
[1.5] http://www.java.com
[1.6] http://java.sun.com/products/java-media/3D/
[2.1] http://www.alias.com/eng/index.shtml
[2.2] http://www4.discreet.com/3dsmax/
[2.3] http://www.artofillusion.org/
[2.4] http://www.beanshell.org/
[2.5] http://www.gamemaker.nl/
[2.6] http://www4.discreet.com/gmax/
[2.7] http://www.swissquake.ch/chumbalum-soft/
[2.8] http://www.radonlabs.de/
[2.9] http://www.erain.com/
[2.10] http://www.agency9.se
[2.11] http://www.blitzbasic.com/
[2.12] http://cal3d.sourceforge.net/
[2.13] http://www.conitec.net/a4info.htm
[2.14] http://www.scriptsearch.com/cgi-bin/jump.cgi?ID=673
[2.15] http://www.caligari.com/gamespace/
[2.16] http://www.macromedia.com/software/flash/
[2.17] http://www.python.org/
[2.18] http://www.genesis3d.com/
[2.19] http://msdn.microsoft.com/visualc/
[2.20] http://www.borland.com/
[2.21] http://www.borland.com/us/products/delphi/index.html
[2.22] http://msdn.microsoft.com/vbasic/
[2.23] http://www.idx3d.ch/idx3d/idx3d.html
[2.24] http://irrlicht.sourceforge.net/
[2.25] http://www.opengl.org/
[2.26] http://www.microsoft.com/windows/directx/default.aspx
[2.27] http://www.jazz3d.co.uk/
[2.28] http://jmonkeyengine.com/
[2.29] http://www.blender.org/
[2.30] http://www.jpct.net/
[2.31] http://ode.org/
[2.32] http://www.ogre3d.org/
[2.33] http://panda3d.org/
[2.34] http://www.powerrender.com/
[2.35] http://www.realityfactory.ca/v4/
[2.36] http://www.garagegames.com/
[2.37] http://ugs3d.sourceforge.net/

[2.38] http://udn.epicgames.com/Main/WebHome
[2.39] http://www.yafray.org/
[2.40] http://www.vhml.org/
[2.41] http://vrlab.epfl.ch/research/S_BAML.PDF
[2.42] http://www.thedayaftertomorrow.com/
[2.43] http://troymovie.warnerbros.com/
[3.1] http://www.xml.com/
[3.2]
http://www.lemonteam.com/wiki/index.php?pagename=GlosarioDeGrafismo&PHPSESSID=fdf51a42971ca610badcd0f75405f57c
[3.3] http://www.mysql.com/
[3.4]
http://www.google.com.mx/search?hl=es&biw=1003&q=%22Gesture+Markup+Language%22&meta=
[3.5] http://www.tugurium.com/gti/termino.asp?tr=parse

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL I.P.N.
UNIDAD GUADALAJARA

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó la tesis

Lenguaje para Animación de Criaturas Virtuales

del (la) C.

Alma Verónica MARTÍNEZ GONZÁLEZ

el día 26 de Agosto de 2005.

Dr. Federico Sandoval Ibarra
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Luis Ernesto López Mellado
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Félix Francisco Ramos Corchado
Investigador CINVESTAV 2B
CINVESTAV Unidad Guadalajara