UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DIVISIÓN DE ESTUDIOS DE POSTGRADO
POSTGRADO EN COMPUTACIÓN

"Desarrollo de un sistema multiagentes para el control de las interacciones físicas entre los objetos y avatares de un Ambiente Virtual 3D Distribuido"

Autor: M. Sánchez
Tutor: D. Hernández

Trabajo de grado presentado ante la ilustre Universidad de Los Andes como requisito parcial para optar al grado de *Magister Scientiae en Computación*

Mérida, Abril 2012

Desarrollo de un sistema multiagentes para el control de las interacciones físicas entre los objetos y avatares de un Ambiente Virtual 3D Distribuido

Ing. Manuel B. Sánchez

Proyecto de Grado — Maestría en Ciencias de la Computación, 196 páginas

Resumen: En el presente trabajo de grado se presenta el desarrollo de un sistema multiagentes para el control de las interacciones entre los objetos y avatares de un ambiente virtual 3D distribuido. Para llevar a cabo la investigación se realizó, en primer lugar, un análisis que permitió definir el motor físico a ser usado por el agente encargado de controlar la actividad física del ambiente virtual dinámico.

Posteriormente, se emprendió el desarrollo del sistema multiagentes, el cual está compuesto por el Agente de Iniciación, el Agente de Atención al Cliente, el Agente de Sincronización, el Agente Gestor de Datos, el Agente Repositorio, el Agente de Visualización, el Agente Receptor de mensajes y el Agente Emisor de Información. La interacción entre estos agentes, permite: almacenar y recuperar la información del ambiente virtual dinámico, reaccionar ante una acción física, sincronizar y visualizar el estado del ambiente virtual dinámico, entre otros.

Con respecto al Agente de Control Físico, se puede decir que el mismo es capaz de diferenciar entre objetos estáticos o dinámicos, así como también, toma en cuenta las propiedades físicas de los objetos para efectuar la reacción cuando un objeto interactúa con otro. El sistema multiagentes fue diseñado usando la metodología MultiAgent Systems for INtegrated Automation (MASINA) y el Lenguaje de Modelado Unificado (UML); Los prototipos de prueba del agente fueron desarrollados usando la metodología propuesta por (Hernández, Barrios, & Gutierrez, 2010) denominada: "Metodología para el desarrollo de un ambiente virtual dinámico (MAVD).

Palabras Clave: Sistema multiagentes, agente, motor de renderizado, motor físico, modelado 3d, animación esquelética, sincronización, comunicación en red

Dedicatoria

A mi madre, a mi esposa, hijos y toda mi familia quienes con su apoyo me dan la fortaleza para seguir cumpliendo las metas que me trazo día a día.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Agradecimientos

En primer lugar quiero agradecer a Dios Todopoderoso, por darme la fe, la salud, la sabiduría y el valor suficiente para llevar a cabo este proyecto.

A mi madre, María Sánchez por sus valiosos consejos y por enseñarme desde muy niño el valor que tiene la educación y a esforzarme por alcanzar mis metas.

A mi esposa, Sonia Duarte por brindarme su amor, su comprensión y apoyo para finalizar este trabajo.

A mis hijos, Yeray Alejandro y Ángela Sarahí quienes me prestaron el tiempo que les pertenecía para terminar y quienes con sus risas y travesuras hicieron mis días más gratos, motivándome cada día a seguir adelante.

A mis hermanos y hermanas, por la confianza que han depositado en mí.

A mi tutor, el Profesor Domingo Hernández por su asesoría y dirección en el desarrollo de está investigación.

A los moderadores y usuarios de los foros de Ogre3D y Newton Game Dynamics, quienes resolvieron mis dudas en varias oportunidades.

A mis amigos y todas aquellas personas que de una u otra forma, colaboraron o participaron en la realización de esta investigación, hago extensivo mi más sincero agradecimiento.

INTRODUCCIÓN

En la actualidad, vivimos en un mundo donde interactuamos casi a diario con Ambientes Virtuales 3D distribuidos o no, ya sea juegos, software educativo, software científico, software para controlar maquinaria de forma virtual, software para simulaciones, entre otros.

Según (Hernandez H., 2001), los Ambientes virtuales dinámicos (AVD) son en efecto sistemas distribuidos, donde los usuarios comparten un estado y cuyo propósito fundamental es proveer a los usuarios la ilusión de que todos están observando las mismas cosas e interactuando entre sí en tiempo real.

Los AVD son de gran importancia, debido a las ventajas que brindan, ya que las personas pueden adquirir experiencia en ciertas áreas sin tener que desplazarse del lugar donde residen ni poner en peligro sus vidas, y a su vez, pueden interactuar con los demás usuarios que visiten el sitio.

Un ejemplo de ello podría ser: visita a un museo virtual, en el cual se pueda hablar con las demás personas que visitan el sitio, y observar, tocar, golpear, o incluso arrojar los objetos (obras de arte, fósiles, minerales, rocas, etc.) presentes en el mismo, dando así mayor realismo a la escena, ya que las personas podrán tener la sensación de haber estado realmente en el sitio.

Para que el AVD sea lo más real posible, es necesario, no sólo crear un escenario con texturas impresionante, sino que es más importante aún, agregar un modelo basado en principios físicos, que dé la sensación de estar inmerso en el mundo real (Bourg, 2002).

En este trabajo se emprende el desarrollo de un sistema multiagentes con la tarea fundamental de controlar las interacciones físicas entre objetos y avatares de un AVD 3D, con el fin de aumentar el realismo del ambiente, y hacerlo más inmersivo.

Capítulo 1. El Problema

En este capítulo se definen los antecedentes que es la explicación de cómo se originó el interés por el problema, se realiza la definición del problema donde se establece de manera concreta la situación que se ha investigado, se explica de manera detallada la importancia de realizar la investigación y se definen los objetivos, metodología, alcance y estructura del documento.

1.1. Antecedentes

(Ramos N., Larios D., Cervantes C., & Leriche V., 2007) en su artículo "Creación de Ambientes Virtuales Inmersivos con Software Libre" explican los elementos que caracterizan a una aplicación de realidad virtual inmersiva, y cómo es posible desarrollarlas con el uso de bibliotecas de software libre. Se realiza una breve descripción de las bibliotecas, sus funcionalidades y cómo se utilizan en la creación de aplicaciones tales como: visualizadores para ambientes arquitectónicos, espacios para tratamiento de fobias en psicología y ambientes para enseñanza de lenguas extranjeras.

(Moctezuma, 2006) en su tesis de maestría presenta un sistema para la manipulación en tiempo real de objetos deformables virtuales sin retroalimentación de fuerzas. Los objetos se construyen en base a mallas. En cada arista de la malla se acopla un sistema mecánico simple, compuesto por una masa, un resorte y un amortiguador, sistema que le da a la malla su capacidad deformable. El modelo deformable se encuentra bajo la acción de un modelo rígido el cual se controla a través de un guante.

Dentro de los recorridos en escenarios virtuales se cita (La Tumba de Nefertari) en este recorrido, se observa con bastante detalle el lugar tal cual es en la realidad; solo existen objetos estáticos, y no cuenta con un modelo físico real de la escena,

ni tampoco se puede interactuar con las demás personas que estén visitando el lugar.

Algunos autores han investigado en esta línea con fines comerciales, entre ellos se encuentran:

Tradky: Es un software que permite crear y editar ambientes virtuales tridimensionales que pueden ser visitados desde cualquier computador que posea conexión a Internet, ofrece la creación de ferias virtuales en 3 dimensiones, museos, ciudades, galerías de arte, entre otros. Este ambiente carece de interacción física (Museo Virtual, 2009).

1.2. Planteamiento del Problema.

El hombre, desde sus inicios, se ha interesado por establecerse en un lugar y explotar los recursos de una determinada región, tratando de hacer el menor esfuerzo posible para llevar a cabo una determinada tarea. Esta característica, ha hecho que se desarrollen tecnologías que le permitan cumplir su labor de manera automatizada y con poca intervención humana. Sin embargo, existen áreas que han sido poco desarrolladas y necesitan de la manipulación del hombre para poder obtener un buen resultado y otras en la cuales, tal vez, nunca se pueda prescindir del recurso humano.

En algunas de las áreas mencionadas, las labores pueden representar un riesgo para las personas que se desenvuelven en el medio, sobre todo para personal poco capacitado. Por otra parte, capacitar personal directamente en el área puede resultar desastroso; es allí donde viene a jugar un papel importante los Ambientes Virtuales en 3D, ya que por medio de ellos es posible capacitar nuevo personal sin riesgo alguno, y una vez que éste adquiera cierta experiencia se pueda realizar un adiestramiento directamente en el lugar de trabajo reduciendo considerablemente

los riesgos de sufrir algún accidente por inexperiencia.

Existen otras áreas, donde las labores deben hacerse de forma coordinada, es decir, se requiere de la intervención de más de un usuario en forma conjunta para llevar a cabo una tarea específica. En este caso se puede hacer uso de un AVD 3D, en el cual los usuarios se conectan y cada uno se encarga de efectuar una operación que incide en las operaciones que realizan los demás usuarios. Por ejemplo, si se desea enviar una misión espacial al planeta Marte; es sabido, que de ese planeta se conoce muy poco, y si se quisiera aterrizar una nave espacial en algún sector del planeta y llegase a ocurrir un fallo, sería un fracaso total para la misión, ya que no sólo se tiene una gran pérdida económica, sino que más importante aún, estaríamos arriesgando vidas humanas; para evitar un desastre, sería necesario realizar una simulación del aterrizaje en el planeta; para ello usaremos un Ambiente Virtual 3D Distribuido, ya que cada astronauta debe encargarse de un grupo de controles de la cabina, y cuando un astronauta accione un control específico, habrá una incidencia en el trabajo que estén realizando los demás.

Otro campo donde se podría usar los AVD 3D, es el área militar, ya que podría servir para entrenar equipos de pilotos de vuelo sin arriesgar la vida de los mismos. También es útil en el área turística, permitiendo a las personas con poco tiempo o de bajos recursos económicos, visitar diversos lugares del planeta sin tener que salir de su hogar.

Como se puede ver en los ejemplos expuestos anteriormente, los AVD 3D tienen una amplia gama de aplicaciones y permiten no solo abaratar costos, sino que a su vez existe la posibilidad de adquirir experiencias sin poner en peligro la vida humana, e incluso evitar el traslado de las personas. Sin embargo, en ninguna de las aplicaciones mencionadas anteriormente, se obtendría un verdadero aprendizaje, si el Ambiente Virtual 3D, luce poco real; Según (Bourg, 2002) para que un ambiente virtual tenga una apariencia realista, es necesario considerar un

modelo físico del mismo, ya que, si en una simulación de vuelo no se toma en cuenta la gravedad, la fuerza de roce producida por el aire, colisión con otros objetos, sino, que sólo se consideran las características propias del avión, la sensación de estar volando realmente sería muy poca y al momento en que el piloto se enfrente a un vuelo real, notaría un cambio drástico y se encontraría en una situación totalmente desconocida para él.

Este proyecto busca resolver el problema de la inclusión del modelo físico en AVD 3D, mediante el desarrollo de un sistema multiagentes que se encargue de observar los objetos y usuarios que intervienen en la escena, detectando y ejecutando las interacciones físicas que se generen entre ellos y a su vez sincronizando la información en cada uno de los monitores de los usuarios que se encuentren conectados. El agente encargado del modelo físico debe responder de acuerdo a las características físicas de los objetos que interactúan, y se reaccionará de una manera u otra, ya que no es lo mismo patear un balón, que patear un bloque de 8 kg, el primero describirá una trayectoria curva, mientras que para el segundo la trayectoria será recta o posiblemente no cambie de posición. Estás características físicas deberán almacenarse en una base de datos, junto con las demás características visuales de cada objeto presente en la escena.

1.3. Justificación

Los AVD permiten representar en la computadora cualquier ambiente conocido o imaginado. Sin embargo, la mayoría de las herramientas que se usan para crear estos ambientes omiten el modelo físico del sistema, lo que conlleva a tener una realidad virtual incompleta.

Por otra parte, a pesar de que existen herramientas como Newton (Jerez & Suero, Newton Game Dynamics, 2003), y Ode (Smith, 2007), que permiten realizar simulaciones físicas e incorporarlas a un Ambiente Virtual, se necesita un grado alto de conocimiento en el desarrollo de ambientes virtuales 3D ya sea en OpenGl,

Java3D, Ogre3D, entre otros, lo que dificulta el uso por parte de usuarios de poca experiencia.

Las herramientas de desarrollo de ambientes virtuales no sincronizan la vista del ambiente a los usuarios conectados de forma automática, sino, que el desarrollador del AVD debe encargarse de hacerlo por su cuenta, aumentado así la complejidad del desarrollo del AVD. Estás herramientas tampoco cuentan con un gestor de base de datos para guardar las características de los objetos del ambiente, y los datos de los usuarios que se conectan, dificultando así la administración de este tipo de ambientes y la posibilidad de mantener características comunes a varios objetos, tales como el tipo de material, dimensiones, ubicación inicial en el ambiente, entre otros.

1.4. Objetivos

1.4.1. Objetivo General

Desarrollar un sistema multiagentes para el control de las interacciones físicas entre los objetos y avatares de un Ambiente Virtual 3D Distribuido.

1.4.2. Objetivos Específico

- Analizar las diversas técnicas usadas para la construcción de Ambientes Virtuales 3D.
- Realizar un estudio de las técnicas usadas en la actualidad para la construcción de Agentes.
- Identificar, analizar y comparar las herramientas o motores físicos usados con mayor frecuencia, para incorporar modelos físicos en Ambientes Virtuales 3D.

- Definir en base al análisis realizado de los motores físicos, la conveniencia de utilizar un motor físico ya desarrollado o programar el motor físico acorde a las necesidades del sistema.

- Emprender el diseño del sistema multiagentes que controlará las interacciones físicas de los objetos dinámicos presentes en el AVD.

- Desarrollar un prototipo operativo basado en agentes que permita probar y validar el correcto funcionamiento del SMA diseñado previamente.

1.5. Metodología

Debido a las características del problema estudiado en esta investigación, es necesario el uso de dos metodologías, que permitan llevar un mejor control en el diseño y desarrollo tanto de los agentes como del software que será usado para probar que los agentes cumplen a cabalidad con sus objetivos. A continuación se detallan las metodologías seleccionadas para tal fin.

1.5.1. Primera Fase: Diseño de los Agentes

De acuerdo a (Aguilar, Bessembel, Cerrada, Hidrobo, & Narciso, 2008) MASINA es una extensión del modelo orientado a objetos MAS-CommonKADS, y se basa en el mismo ciclo de desarrollo, con importantes modificaciones que permiten incorporar comportamientos inteligentes.

MASINA es una extensión de MAS-CommonKADS, la cual consta de las fases de conceptualización, análisis, diseño, codificación y pruebas, integración, y operación y mantenimiento (Aguilar et al., 2003).

(Rios B., Cerrada, Narciso, Hidrobo, & Aguilar, 2008) sostienen que la definición de los Agentes que conforman el sistema, se realiza en la fase de conceptualización, utilizando diagramas de casos de uso y actividades de UML,

mientras que la especificación detallada de los agentes se realiza en la fase de análisis por medio del modelo de agente, modelo de tarea, modelo de inteligencia, modelo de coordinación y modelo de comunicación.

En el desarrollo del proyecto se presentará en la fase de conceptualización el Agente de Iniciación, el Agente de Atención al Cliente, el Agente de Sincronización, el Agente Gestor de Datos, el Agente Repositorio, el Agente de Visualización, el Agente Receptor de Mensajes y el Agente Emisor de Información, descritos por diagramas de interacción en la fase de análisis. También se desarrollarán algunos diagramas de clase y de secuencias correspondientes a la fase de diseño.

1.5.2. Segunda Fase: Desarrollo del Ambiente Virtual Dinámico

Según (Hernández, Barrios, & Gutierrez, 2010) MAVD es un método para el diseño e implementación de ambientes virtuales dinámicos en donde puedan intervenir agentes inteligentes.

La metodología MAVD propone cinco grandes procesos como son: Dirección, Preproducción, Diseño, Producción y Postproducción (Hernández, et al., 2010)

El uso de este método permitirá al autor alcanzar la realización de un AVD 3D que cumpla con las exigencias de los usuarios finales, el cual incorporará un conjunto de agentes, a los cuales se les asignarán tareas que permitirán crear el AVD, iniciar sesión en el ambiente, sincronizar las vistas, realizar acciones físicas, entre otros.

1.6. Resultados Esperados

Al finalizar el presente proyecto, se esperan obtener los siguientes resultados:

La construcción de un sistema multiagentes para el control de las interacciones entre los objetos físicos y avatares de un Ambiente Virtual Distribuido, este sistema se encargará de controlar el AVD, desde su almacenamiento, recuperación, visualización y sincronización, los agentes del sistema deben realizar la observación de todos los objetos dinámicos presentes en el mismo y al detectar una interacción entre ellos, realizar una consulta en la base de datos, donde se deben encontrar las características de los objetos en cuestión, posteriormente ejecutará la reacción producida por su interacción física, para luego sincronizar las vistas de cada usuario, acorde a las nuevas posiciones de los objetos.

El usuario administrador podrá crear los escenarios almacenando la información del mismo en una base de datos y deberá indicar las características físicas de cada objeto, para que el Agente de Control Físico responda de acuerdo a esta información.

Se espera tener un prototipo operativo que permita a un usuario crear cuentas, crear avatares, iniciar sesión, establecer comunicación con otros usuario por medio de chat, moverse a través del ambiente, iniciar acciones físicas como: patear, golpear o empujar objetos; todo esto bajo una arquitectura cliente servidor que permita a usuarios de diversas localidades conectarse al ambiente sin importar su ubicación.

El modelo físico a ser considerado en el ambiente no incluye cuerpos deformables o líquidos, solo se basará en la física de cuerpos rígidos y la física newtoniana, sin embargo, se consideraran algunas propiedades físicas de los objetos como son, elasticidad, fricción estática, fricción dinámica y dureza del cuerpo.

1.7. Estructura del Documento

Capítulo 1, El Problema. Se definen los antecedentes, problema, justificación, objetivos, metodología y resultados esperados. Describiendo de dónde y por qué surge el problema, lo que intenta solucionar y la forma en que se desarrollara el proyecto.

Capítulo 2, Marco teórico. Contiene los fundamentos teóricos y el estado del arte de los diferentes lenguajes y métodos, para la creación de objetos 3D y ambientes virtuales necesarios para el estudio y desarrollo de los objetivos del proyecto.

Capítulo 3, Diseño. Se muestra cómo se ha diseñado cada uno de los agentes, el prototipo de prueba, los avatares, el proceso de detección de colisiones, sincronización, diseño de datos, entre otros.

Capítulo 4, Implementación y Pruebas. En este capítulo se detalla el proceso que se ha seguido para implementar los elementos diseñados en el capítulo anterior, tales como: datos, avatares, detección de colisiones, etc. También se resumen las pruebas realizadas al sistema.

Capítulo 5, Conclusiones. En este capítulo se discuten y analizan los resultados obtenidos del desarrollo del proyecto y posteriormente se exponen las conclusiones y recomendaciones para futuras investigaciones.

Capítulo 2. Marco Teórico

En este capítulo se describen los fundamentos teóricos necesarios para el entendimiento y comprensión del proyecto, así como también las diferentes arquitecturas y los lenguajes usados para el desarrollo de los ambientes virtuales.

2.1. Selección del Lenguaje de Programación

(Gallego & Llinás, 2000) definen el lenguaje de programación como un lenguaje empleado por el programador para dar a la computadora las instrucciones necesarias que permiten dar solución a un problema determinado. Estás instrucciones constituyen el llamado programa fuente.

De acuerdo con (Briggs, 2007) los computadores, como los humanos, tienen más de un lenguaje. Entre ellos podemos listar: PASCAL, FORTRAN, BASIC, ADA, C, C++, C#, Java, Prolog, Lisp, entre otros.

Para efectos de esta investigación se hará uso del leguaje de programación C++ ya que por medio de él, se podrán aprovechar los pilares de la programación orientada a objetos como son: Herencia, Abstracción, Encapsulamiento y Polimorfismo para crear un software flexible en el cual se puedan incorporar actualizaciones de manera rápida y sin tener que modificar el programa completo. Además, debido a que en C++ la memoria se maneja de manera manual, podremos tener un mejor aprovechamiento de la misma, lo que posiblemente influya en un mejor rendimiento del software. Otra razón por la cual se elige C++ frente a otros lenguajes orientado a objetos como Java, es que por medio de C++ se genera un programa objeto, en código de máquina, lo que también maximizará el rendimiento de la aplicación, ya que al momento de poner en ejecución el software, las instrucciones no tienen que ser interpretadas nuevamente, antes de ser enviadas al procesador de la computadora.

2.2. Ambiente Virtual

(Ramos N. et. al, 2007) establece que los AVD son una representación tridimensionales de espacios reales o imaginarios, generados por computadora, con los que el usuario puede interactuar y que le producen la sensación de estar dentro del mundo real.

(Badano, 2008) citando a Edward Castronova, profesor asociado de Telecomunicaciones en la Universidad de Indiana, economista, y uno de los pioneros en el estudio de los mundos virtuales como objeto de investigación, define un Mundo virtual como:

> *"Un mundo virtual es un programa de computadora con tres características fundamentales:*
>
> *- **Interactividad:** existe en una computadora, pero puede ser accedido remota y simultáneamente (a través de una conexión a Internet, por ejemplo) por un gran número de personas. Los comandos que cada persona ingresa en el sistema afectan los comandos de otras personas.*
>
> *- **Física:** Las personas acceden al programa a través de una interface que simula un ambiente físico visto en primera persona, en la pantalla de sus monitores; este ambiente es generalmente gobernado por las reglas naturales de la tierra y esta caracterizado por la escasez de recursos.*
>
> *- **Persistencia:** El programa continúa corriendo, sin importar si alguien lo está usando o no; recuerda la ubicación de las personas y las cosas, así como también la propiedad sobre los objetos."* (Castronova, 2001)

Así, Castronova hace su definición de un mundo virtual, basándose netamente en las características fundamentales para la existencia del mismo.

Por otro lado, (Hernandez H., 2001) expone que los ambientes virtuales están compuestos por los siguientes elementos:

> *"**Entidad:** Es cualquier objeto que conforma la escena de un ambiente virtual. Este objeto puede ser estático o responder a leyes de evolución o movimiento. Las entidades pueden ser compartidas por múltiples usuarios que se encuentren compartiendo el mundo virtual y se encuentra sincronizada por un agente inteligente.*
>
> ***Avatar**: Es una entidad que representa la perspectiva que tiene un cliente o usuario del ambiente virtual. El avatar permite que el usuario lleve el control en el ambiente virtual en tiempo real y pueda interactuar con él.*
>
> ***Agente**: Es un programa que permite controlar un objeto cuando es compartido por dos o más avatares. El agente permite sincronizar las interacciones entre los distintos avatares y los objetos que conforman el ambiente virtual."*

Hernández por su parte, menciona los elementos que deben estar presentes en un ambiente virtual, basándose en el paradigma de agentes. Desde su perspectiva, un mundo virtual es un sistema multiagentes que tienen como tarea fundamental sincronizar la información entre las entidades y avatares que componen el ambiente. La Figura 2.1 es representativa de esta definición.

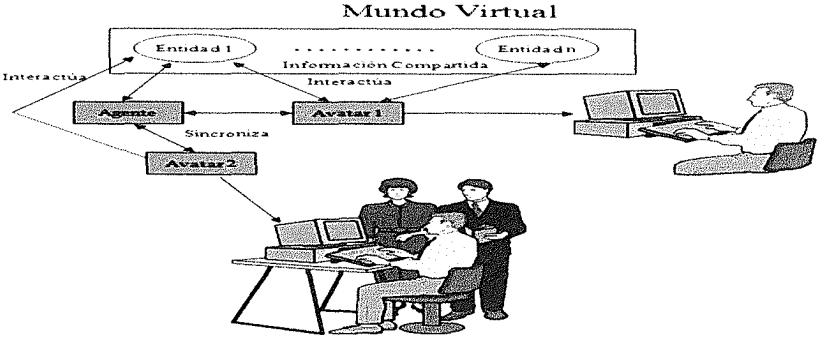

Figura 2.1. Modelo Conceptual de un Ambiente Virtual.

Fuente: (Hernandez H., 2001)

2.3. Clasificación de los Ambientes Virtuales

Los ambientes virtuales se pueden clasificar en Ambientes Virtuales Inmersivos, Ambientes Virtuales Colaborativos y Ambientes Virtuales Dinámicos.

2.3.1. Ambientes Virtuales Inmersivos

En este tipo de ambientes, los usuarios pueden interactuar con el ambiente, produciendo la sensación de estar dentro de éste.

2.3.2. Ambientes Virtuales Colaborativos (AVC)

Se define como un sistema multi-usuario que soporta explícitamente el trabajo cooperativo. Es aquel donde todos o un grupo de usuarios trabajan activa y concurrentemente sobre algún elemento compartido para alcanzar una meta común.

2.3.3. Ambientes Virtuales Dinámicos (AVD)

Son, en efecto, sistemas distribuidos donde los usuarios comparten un estado y cuyo propósito fundamental es proveer a los usuarios la ilusión de que todos están observando las mismas cosas e interactuando entre sí en tiempo real.

2.4. Agentes

De acuerdo a (Aguilar, Rivas, & Cerrada, 2010) la definición de agente es un tema de mucha polémica y controversia. El problema es que, aunque el término sea ampliamente utilizado por muchas personas, desafía los intentos de establecer una definición única y universalmente aceptada. A continuación se muestran

algunas de las definiciones más aceptadas.

Según (Weiss, 1999) "Un agente es un sistema computacional que está situado en un ambiente, y es capaz de tomar acciones autónomas en ese ambiente con el fin de cumplir sus objetivos de diseño".

(Wooldridge & Jennings, 1995) por su parte dice que "Un agente es un sistema computacional autónomo y flexible, que es capaz de actuar en un entorno".

Por otra parte (Russell & Norvig, 2003) exponen que "Un agente es cualquier cosa capaz de percibir su medioambiente con la ayuda de sensores y actuar en ese medio ambiente utilizando actuadores". La Figura 2.2 ejemplifica esta definición.

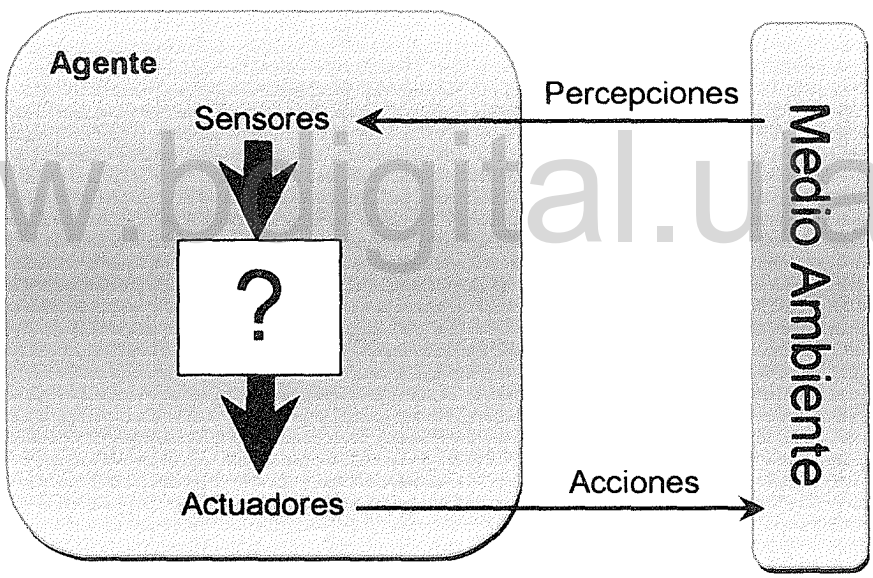

Figura 2.2. Interacción de los agentes con el medio ambiente.
Fuente: (Russell & Norvig, 2003)

Algunos autores, entre ellos (Aguilar et al., 2010) y (Weiss, 1999) han optado por definir una serie de propiedades que caracterizan a los agentes. Entre estas propiedades se tienen:

- **Autonomía:** Weiss dice que la autonomía es la noción central de los

agentes, y argumenta que los agentes son autónomos si poseen la capacidad de tener un comportamiento propio, y reaccionar a los estímulos externos basados en su estado interno, sin la intervención humana ni de otros sistemas externos.

- **Sociabilidad**: los agentes son capaces de interaccionar con otros agentes (humanos o no) a través de un lenguaje de comunicación entre agentes. Una sociedad de agentes es un grupo de agentes que interactúan, se comunican, conversan, "piensan", y actúan en conjunto para lograr un objetivo común.

- **Reactividad**: los agentes son capaces de percibir estímulos de su entorno (recibir una señal, o percibir un cambio de estado en el ambiente), y reaccionar a dichos estímulos.

- **Proactividad**: los agentes no son sólo entidades que reaccionan a un estímulo, sino que tienen un carácter emprendedor, y pueden actuar guiados por sus objetivos.

- **Movilidad**: capacidad que tiene un agente de trasladarse desde un nodo a otro, dentro de un sistema distribuido.

- **Veracidad**: suposición de que un agente no comunica información falsa a propósito.

- **Racionalidad**: asunción de que un agente actúa de forma racional, intentando cumplir sus objetivos si son viables. Un agente puede razonar acerca de lo que percibe, a fin de definir una acción óptima.

- **Adaptabilidad**: esta característica está relacionada con el aprendizaje que un agente puede lograr, y con su capacidad para cambiar su propio

comportamiento basado en este aprendizaje.

2.4.1. Tipos de Agentes

(Russell & Norvig, 2003) definen 5 tipos de agentes de acuerdo a las capacidades de los mismos. De esta forma se tienen los agentes reactivos simples, agentes reactivos basados en modelos, agentes basados en objetivos, agentes basados en utilidad y agentes que aprenden.

2.4.1.1. Agentes reactivos simples.

Este tipo de agentes selecciona las acciones sobre la base de las percepciones actuales, ignorando las percepciones históricas. Aplica reglas de condición/acción con la forma **SI** condición **ENTONCES** acción, se debe aparear cada percepción o combinación de estas en un conjunto completo de reglas especificando la acción a tomar en cada caso. En la Figura 2.3 se tiene el diagrama que describe el comportamiento de este tipo de agentes.

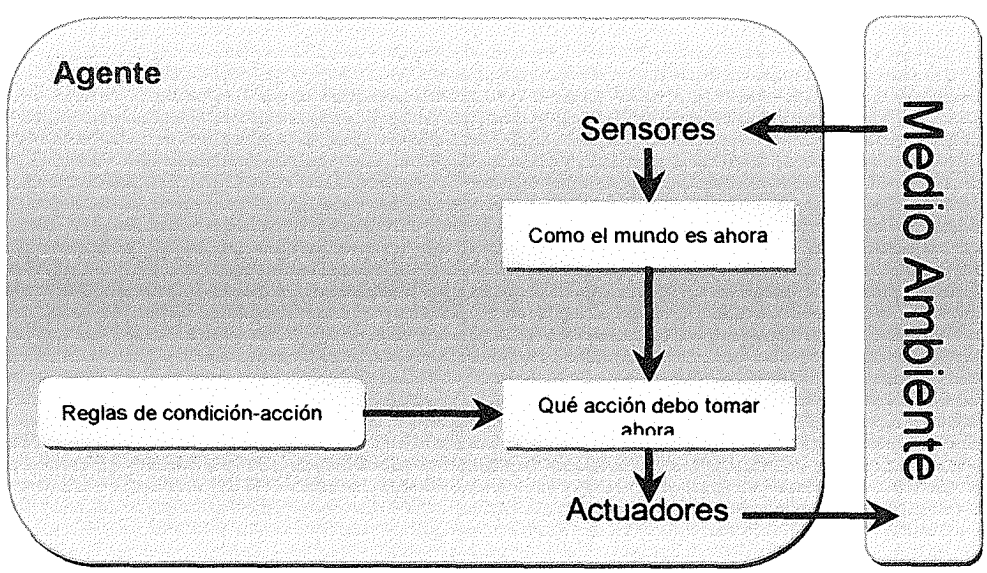

Figura 2.3. Diagrama esquemático de un agente reactivo simple
Fuente: (Russell & Norvig, 2003)

Este tipo de agentes posee una memoria muy limitada; además, el agente funcionará sólo si el entorno es totalmente observable, ya que de no serlo pueden encontrarse situaciones en las cuales el agente no tenga una regla de condición acción para actuar.

2.4.1.2. Agentes reactivos basados en modelos.

Los agentes reactivos basados en modelos mantienen la historia de todo lo que han percibido en su entorno, de este modo puede reflejar alguno de los aspectos no observables del estado actual. Opera encontrando una regla cuya condición coincida con la situación actual y luego procede a efectuar la acción que corresponda a la regla.

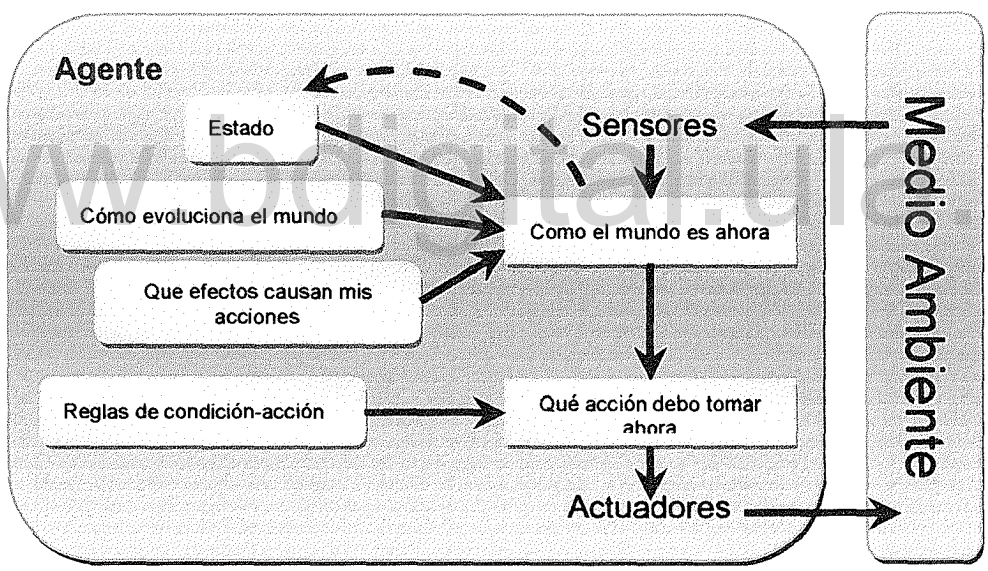

Figura 2.4. Agente reactivo basado en modelo.
Fuente: (Russell & Norvig, 2003)

En la Figura 2.4 se observa el diagrama de un agente reactivo basado en modelo, observamos cómo el agente mantiene un estado interno que está siendo alterado de acuerdo a la percepción actual del mundo. Además, para poder mantener el estado interno del agente, es necesario saber cómo evoluciona el mundo independientemente del agente y cómo las acciones tomadas por el agente

afectan su entorno.

Un inconveniente que puede presentar este tipo de agentes, es que en ocasiones el estado actual del mundo no es suficiente para decidir qué hacer, ya que al igual que el agente reactivo simple, puede encontrarse con situaciones en las cuales no se puede tomar una decisión.

2.4.1.3. Agentes basados en objetivos

Este tipo de agentes mantiene una o varias metas que describen las situaciones que son deseables, lo cual ayuda al agente a tomar decisiones correctas. La toma de decisiones de este tipo de agente es fundamentalmente diferente de las reglas de condición-acción, ya que se hace necesario tener en cuenta consideraciones sobre el futuro que permitirán definir qué acción deberá ejecutar el agente para lograr el objetivo. La Figura 2.5 intenta expresar el concepto de un agente basado en objetivos.

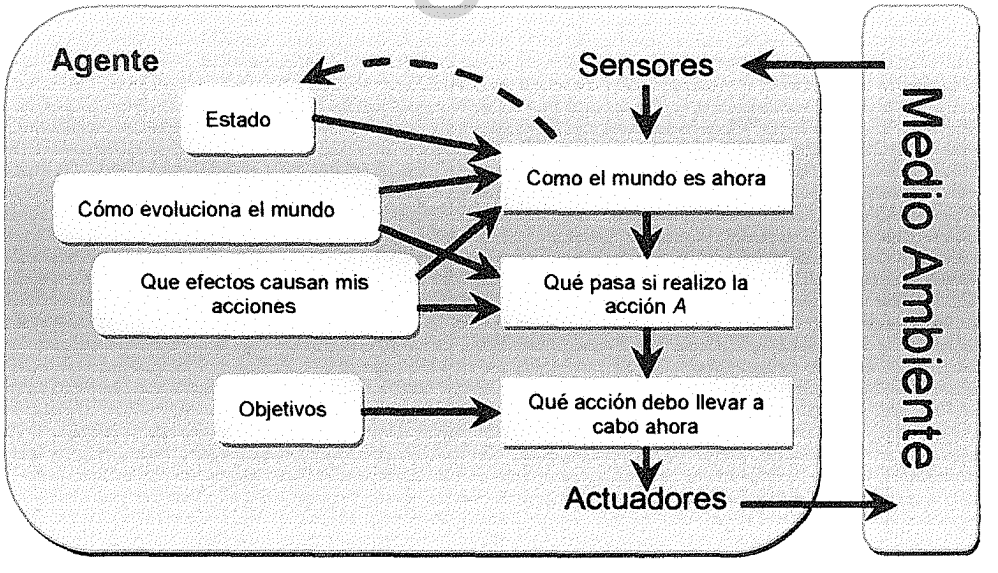

Figura 2.5. Agente basado en objetivos y en modelo

Fuente: (Russell & Norvig, 2003)

Este tipo de agentes a pesar que el mismo llega a cumplir su objetivo, pueden

existir muchas maneras de lograrlo y no se asegura que las acciones que haya tomado el agente son la manera más óptima de cumplir su meta.

2.4.1.4. Agentes basados en utilidad

Este tipo de agentes busca alcanzar su meta maximizando la calidad de su comportamiento. Para hacer distinción de cual estado es más óptimo se basa en una función de utilidad.

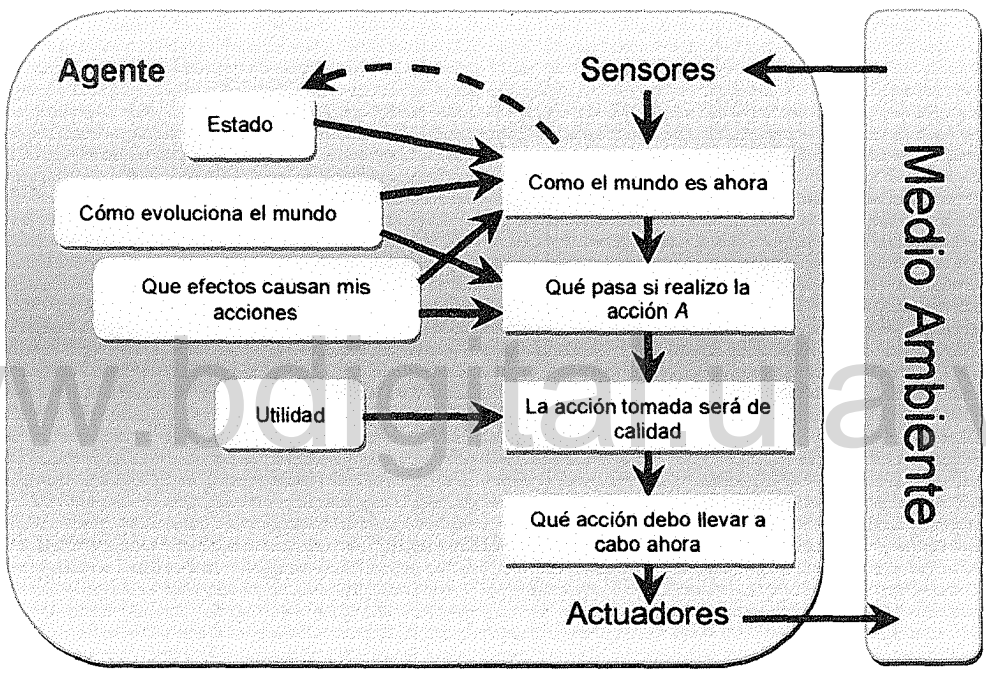

Figura 2.6. Agente basado en utilidad y en modelos
Fuente: (Russell & Norvig, 2003)

La función de utilidad proyecta un estado (o una secuencia de estados) en un número real, que representa un nivel de calidad. La definición completa de una función de utilidad permite tomar decisiones racionales en dos tipos de casos en los que las metas son inadecuadas. Primero, cuando haya objetivos conflictivos, y sólo se puedan alcanzar algunos de ellos, la función de utilidad determina el equilibrio adecuado. Segundo, cuando haya varios objetivos por los que se pueda guiar el agente, y ninguno de ellos se pueda alcanzar con certeza, la utilidad

proporciona un mecanismo para ponderar la probabilidad de éxito en función de la importancia de los objetivos. En la Figura 2.6 se observa el comportamiento de un Agente basado en utilidad y en modelos.

2.4.1.5. Agentes que aprenden

En muchas áreas de la inteligencia artificial, los agentes que aprenden son el método más adecuado para crear sistemas novedosos, ya este tipo de agentes puede operar en ambientes inicialmente desconocidos y eventualmente podrán ser más eficientes que si sólo utilizase un conocimiento inicial.

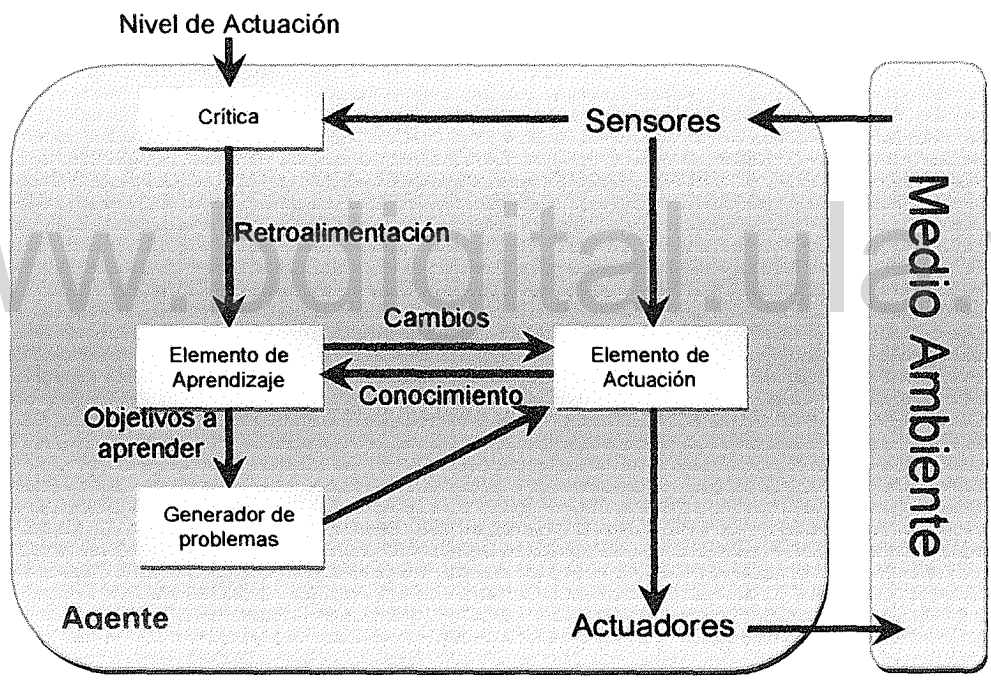

Figura 2.7. Modelo general para agentes que aprenden

Fuente: (Russell & Norvig, 2003)

La Figura 2.7 presenta un modelo para agentes que aprenden. Destacan cuatro componentes principales: a) **Elemento de Aprendizaje**, se encarga de hacer mejoras a las acciones. b) **Elemento de actuación,** es responsable de seleccionar las acciones externas (este elemento es lo que antes se consideraba como agente completo). c) **Crítica,** este componente retroalimenta al elemento de

aprendizaje, determinando cómo se debe modificar el elemento de actuación, para proporcionar mejores resultados en el futuro. d) **El generador de problemas,** es responsable de sugerir acciones que guiarán al agente hacia experiencias nuevas e informativas, lo que conlleva a descubrir acciones mejores a largo plazo.

2.4.2. Los Agentes en Ambientes Virtuales Dinámicos

Los Agentes son de vital importancia en los ambientes virtuales dinámicos ya que ellos son los encargados de sincronizar la vista de los usuarios de manera automática, cuando el ambiente o alguna entidad presente en el mismo cambia su estado.

Por otra parte, los agentes serán los encargados de atender solicitudes de inicio de sesión de usuarios, detección de colisiones entre entidades, ejecutar acciones a solicitud del usuario (caminar, correr, golpear, entre otros), y finalmente, pero no menos importante, la sincronización de la información entre los usuarios presentes en el AVD.

2.5. Modelado 3D

El proceso de modelado es una simplificación de un objeto para su posterior estudio o representación. Así, podemos hablar de modelos matemáticos que simplifican fenómenos físicos, o modelos meteorológicos para la predicción del tiempo atmosférico, etc. Un modelo geométrico define la información sobre la forma (geometría) de un determinado objeto. Las simplificaciones que se realicen en su definición vendrán determinadas por diferentes factores como el método de representación utilizado, operadores empleados o nivel de detalle.

Según (Gónzales & Vallejo, s.f.) se puede definir el proceso de modelado geométrico tridimensional como el encargado de crear modelos consistentes que

puedan ser manejados algorítmicamente en una computadora. Este proceso de construcción se aborda en diferentes etapas, partiendo típicamente de entidades básicas y aplicando una serie de operadores sobre ellas. Estas entidades básicas pueden ser primitivas geométricas (calculadas de forma algorítmica o mediante una ecuación matemática) u obtenidas mediante un dispositivo de captura (escáner 3D). En la Figura 2.8 y la Figura 2.9 se pueden observar dos objetos, el primero modelado a partir de primitivas básicas y el segundo obtenido por medio de un escáner 3D.

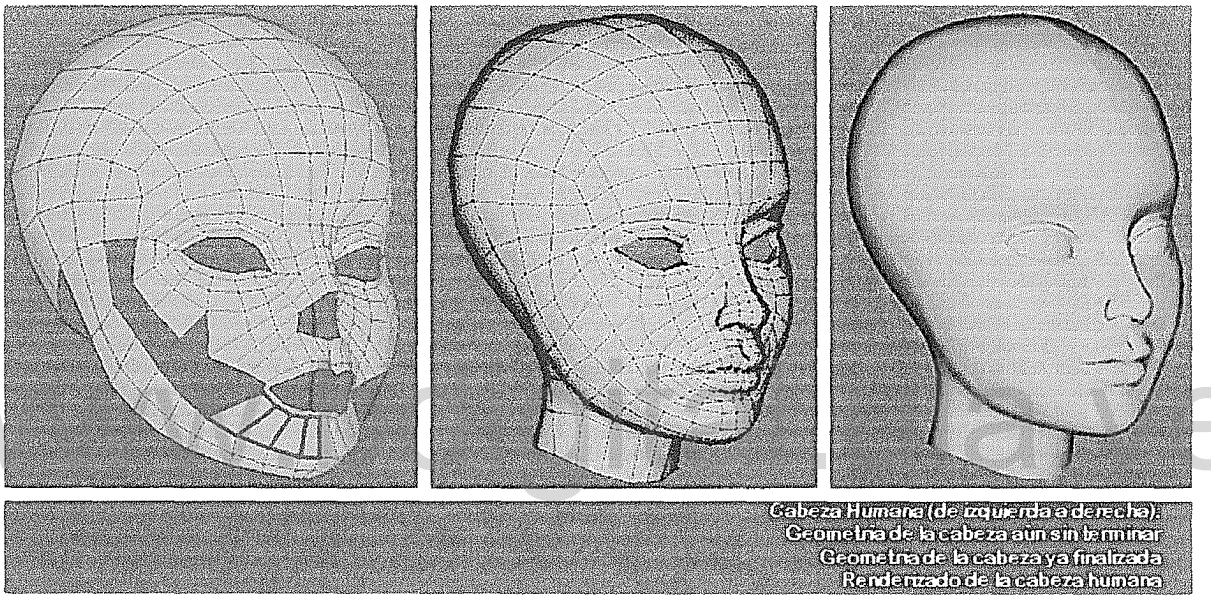

Figura 2.8. Modelado de una cabeza humana a partir de primitivas geométricas.

Fuente: Propia

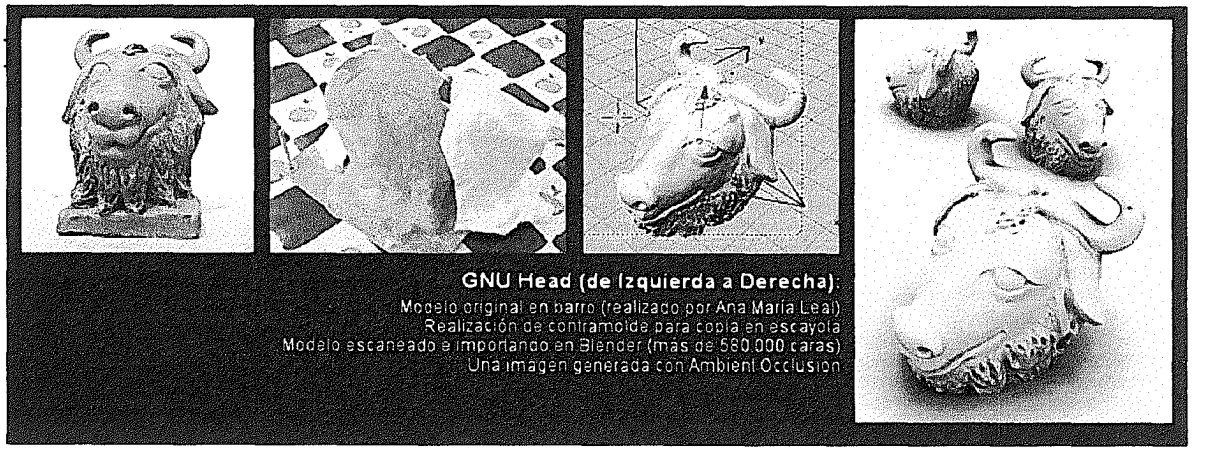

Figura 2.9. Modelado de GNU Head Obtenido con un escáner laser.

Fuente: (Gónzales & Vallejo, s.f.)

Al modelar un objeto tridimensional es habitual comenzar con **primitivas geométricas** básicas, y aplicar posteriormente operadores y transformadores de los elementos que forman esas primitivas. Estas primitivas habitualmente están definidas mediante una descripción algorítmica (una función de un programa), y pueden ser distintas según el programa empleado. En la Figura 2.10 se muestran algunas primitivas que pueden utilizarse en los editores tridimensionales más usados.

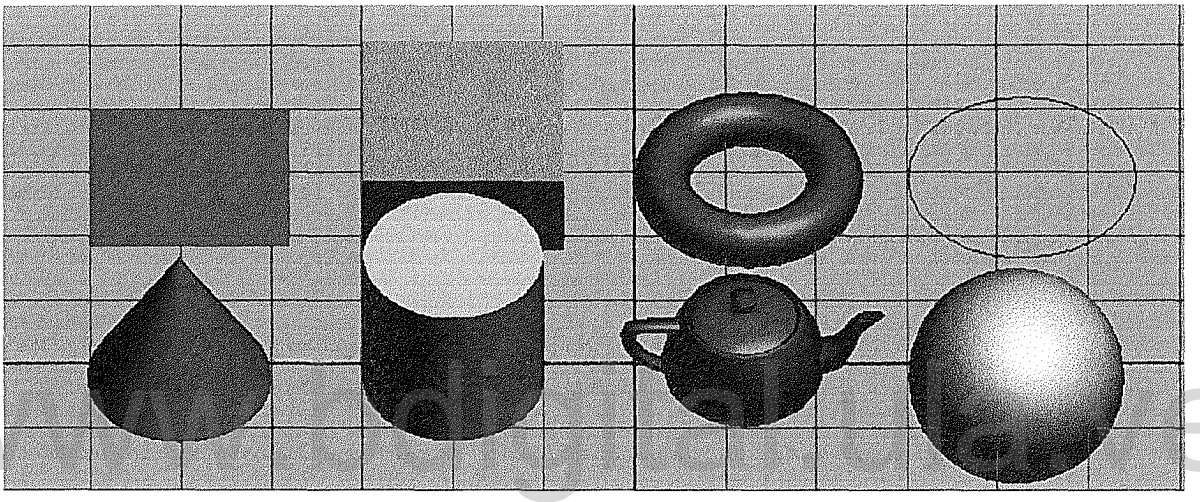

Figura 2.10. Algunas Primitivas para modelado tridimensional.
Fuente: Propia

2.5.2. Componentes de un Modelo Tridimensional

Un modelo tridimensional se compone de lo siguientes elementos:

2.5.2.1. La malla (Mesh)

La parte más importante del modelo es la malla, ya que sin ella, el modelo no existe. La malla es una agrupación de caras, cada una con un número de vértices específicos, las cuales dan forma al modelo y que pueden ser editados y renderizados. En la Figura 2.8 se puede observar como la malla está formada por polígonos (paralelogramos de color rojo) de 4 vértices, los cuales, al estar

ubicados en diferentes posiciones definen la forma del modelo.

2.5.2.2. Textura

Otra parte de gran importancia en el modelo es la textura. Ésta es la imagen que definirá cómo ha de lucir la superficie del modelo al ser renderizado.

En la Figura 2.11 se observa un modelo tridimensional al cual se le ha agregado una textura con apariencia metálica y que además da la sensación de ser un objeto viejo debido a que se encuentra corroído.

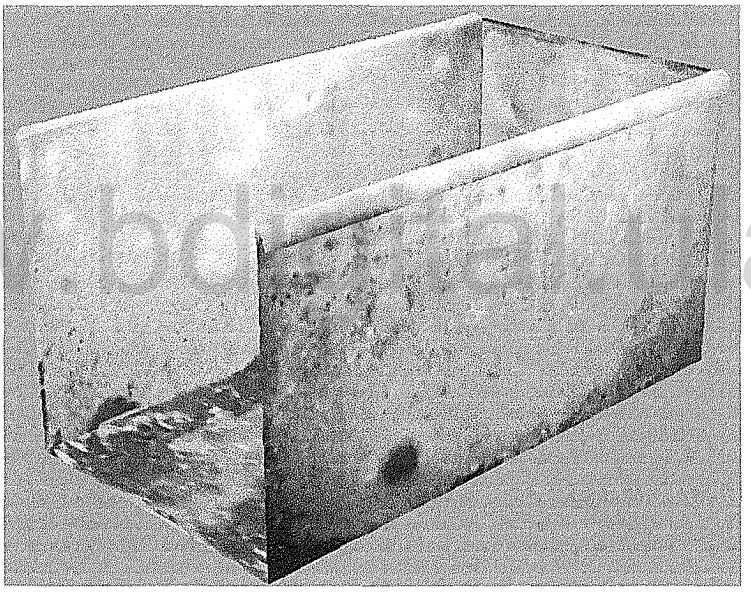

Figura 2.11. Modelo tridimensional de una caja con una textura de apariencia metálica
Fuente: Propia

2.5.2.3. Esqueleto

El esqueleto es una estructura de huesos que se acopla en modelos de animales o personas, el cual es usado para realizar animaciones esqueléticas, de tal manera que al mover un hueso, también se moverán los vértices de la malla asociados a ese hueso, en forma similar a como ocurre en la realidad.

La Figura 2.12 muestra, un modelo 3D de una mujer y su esqueleto, se puede observar que al mover el esqueleto también se mueve el modelo. Mediante esta técnica evitamos la animación por cuadro, en la cual es necesario tener el modelo en diferentes posiciones e ir cambiando el mismo a medida que avanza la animación, lo que trae como consecuencia un alto costo de memoria.

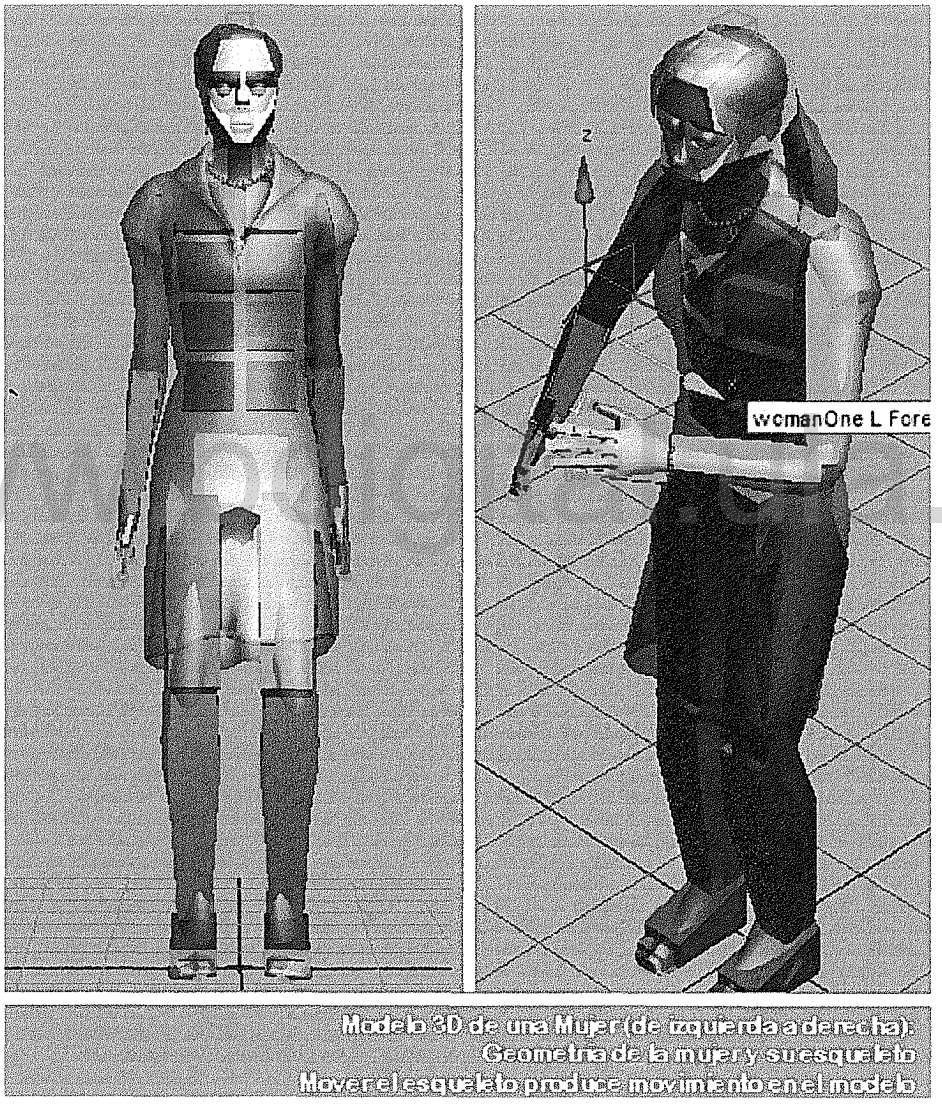

Figura 2.12. Esqueleto en un modelo 3D de una mujer

Fuente: propia

2.5.2.4. Animación

La animación es un punto importante cuando se quiere modelar un ambiente que represente la realidad lo mejor posible. En tal sentido, si en el ambiente que se modela, están presentes entidades como personas, animales, entre otros, es necesario crear animaciones para caminar, correr, agarrar cosas, etc. Así, cuando el personaje ejecute una acción, se reproducirá la animación correspondiente, obteniendo un comportamiento más real que si se reubicase simplemente el modelo sin emitir ninguna animación.

Cuando se animan personajes, es común realizar una animación corta, que se pueda reproducir en ciclo, de tal forma que el personaje aparente estar realizando una animación continua, cuando realmente, está reproduciendo la misma animación una y otra vez.

2.6. API para Gráficos por Computadora

API es la abreviatura de *Aplication Programming Interface*. Un API es una serie de servicios o funciones que el Sistema Operativo ofrece al programador, como por ejemplo, imprimir un carácter en pantalla, leer el teclado, escribir en un archivo de disco, etc. Visto desde la perspectiva del código máquina, el API aparece como una serie de llamadas, mientras que si lo vemos desde la perspectiva de un lenguaje de alto nivel, el API aparece como un conjunto de procedimientos y funciones.

Existen APIs especializadas para facilitar los procesos en todas las etapas de la generación de gráficos por computadora, estas APIs pueden hacer uso de las capacidades gráficas del hardware aprovechando las ventajas del mismo; a tal punto que algunas tarjetas gráficas incluyen una versión particular de este tipo de APIs. Entre las APIs para Gráficos más usadas tenemos:

2.6.1. OpenGL

OpenGL es una interfaz de software para el uso del hardware gráfico. La interfaz contiene cerca de 120 comandos, que son usados para especificar operaciones necesarias para producir aplicaciones tridimensionales interactivas (Neider, Davis, & Woo, 1994).

De acuerdo con (Buss, 2003) OpenGL es independiente del hardware ya que su interfaz ha sido implementada en muchas plataformas y sistemas operativos. Buss dice que la ventaja principal de usar OpenGl es que el mismo es ampliamente aceptado por la mayoría de las industrias de diseño de hardware para gráficos.

OpenGL fue desarrollado por Silicon Graphics Inc. (SGI) en 1992 y ha sido usado en aplicaciones CAD, realidad virtual, visualización científica, simuladores de vuelo, video juegos, entre otros.

2.6.2. Direct3D

(Walsh, 2003) expone que Direct3D puede ser descrito como un sistema operativo para gráficos. Su núcleo provee de funciones que sirven de interfaz con los dispositivos gráficos, liberando al programador de las complicaciones y peligros que pudiera traer la dependencia con el hardware. Direct3D también provee un conjunto de servicios que ayudan al programador a implementar gráficos 3D en la computadora; en este sentido, Direct3D es similar a otras interfaces como OpenGl, con la diferencia que Direct3D ha sido desarrollado por Microsoft Corporation para ser usado en su sistema operativo.

2.6.3. Java3D

De acuerdo con (Davison, 2005) El API de Java3D provee una colección de constructos de alto nivel para crear, renderizar y manipular escenas

tridimensionales, compuestas de geometrías, materiales, luces, sonidos, entre otros. Java3D fue desarrollado por Sun Microsystems, y al momento de escribir este informe se encontraba en su versión 1.5.1.

Java3D en sí, es un envoltorio (*wrapper*) para las interfaces OpenGL y Direct3D, que provee de una interfaz que permite la programación de gráficos en el lenguaje Java, usando un enfoque orientado a objetos.

2.6.4. Ogre3D

Acrónimo del inglés Object-Oriented Graphics Rendering Engine (Ogre-Team, 2011) es un motor de renderizado 3D orientado a objetos, independiente de la plataforma, que a pesar de haber sido escrito en el lenguaje de programación C++, puede ser usado con otros lenguajes de alto nivel como C#, VB.NET, Phython y Java.

Sus bibliotecas evitan la dificultad de la utilización de capas inferiores de librerías gráficas como OpenGL y Direct3D, y además, proveen una interfaz basada en objetos del mundo y otras clases de alto nivel. Además, cuenta con su propio formato de archivo para modelos tridimensionales, en el cual se puede incorporar el esqueleto del modelo, así, como las animaciones del mismo, facilitando la carga de los modelos y la reproducción de sus animaciones.

Ogre3D es software libre, licenciado bajo MIT y ha sido utilizado en algunos videojuegos comerciales, como por ejemplo **Ankh** (Deck 13 Interactive, 2006) y **Earth Eternal** (Iron Realms Entertainment, 2007).

En la presente investigación se hará uso de la API de Ogre3D, debido a que es orientada a objetos y además, se integra de manera directa con el lenguaje de programación C++, ya que está API ha sido escrita en este mismo lenguaje.

En la Figura 2.13 se observa un cuadro del video de 3 minutos "Turin: before the city" renderizado usando la API de ogre3D como motor de renderizado interno y algunas otras herramientas como Hydrax para el agua y VueInfinite para el cielo. El video es una producción realizada para la ciudad de Turin y el museo TORINO por el grupo ASALab.

Figura 2.13. Paisaje del video "Turin: before the city" renderizado usando Ogre3D
Fuente: http://vimeo.com/26159068

2.6.4.2. Estructura de una escena en Ogre3D

Como se puede observar en la Figura 2.14 Ogre3D mantiene una estructura de árbol que permite ubicar los objetos en la escena.

El Nodo Raíz del árbol es llamado **RootSceneNode,** a este nodo se van agregando los demás nodos de la escena. Las entidades, avatares, cámaras y luces son las hojas del árbol, lo que quiere decir que a una entidad no se le pueden agregar otros nodos objetos. Sin embargo, un nodo puede ser padre de

otros nodos u objetos.

Cada nodo del árbol mantiene sus transformaciones respecto a su nodo padre, es decir, que para obtener la posición u orientación real del objeto deben sumarse las transformaciones aplicadas a cada uno de sus ancestros.

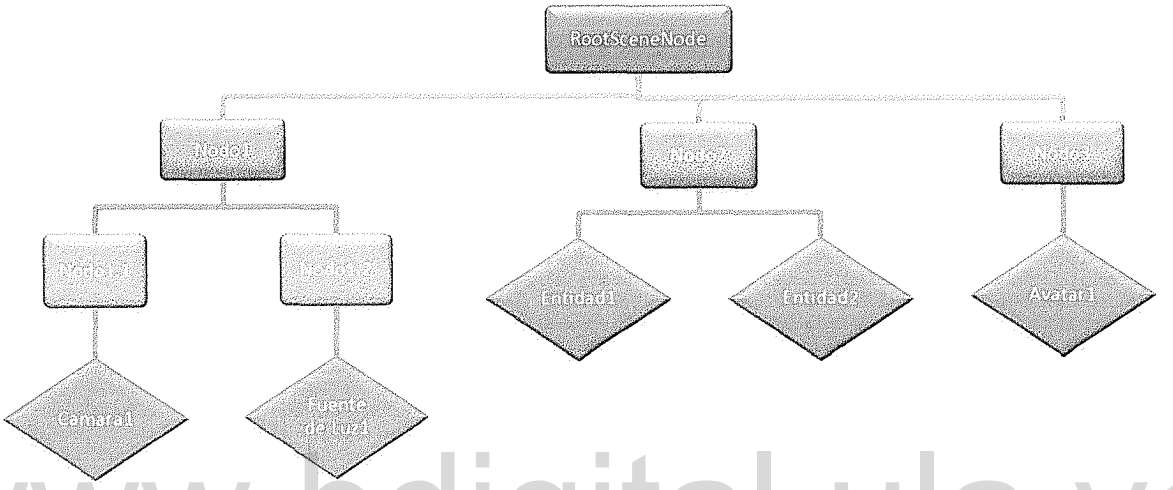

Figura 2.14. Estructura de una Escena en Ogre3D
Fuente: Propia

Por ejemplo, suponga que la Cámara1 está ubicada en el punto (0, 100, 0) y la Fuente de Luz1 en el punto (100, 300, -20), si se aplica una traslación al nodo1 de 20 unidades en el eje X positivo, entonces la nueva ubicación de la Camara1 será (20, 100, 0), mientras que la de la Fuente de Luz1 será (120, 300, -20).

2.7. Motor Físico

Según (Millington, 2007) un motor físico es una pieza de código común que conoce la física general del sistema, pero que no ha sido programado con los detalles específicos de cada aplicación. Millington dice que el motor físico básicamente se encarga realizar los cálculos matemáticos necesarios para simular la física, pero no sabe que se necesita para simularla. Por ejemplo, si se tiene un

motor físico que implemente las ecuaciones para simular el lanzamiento de un proyectil y se desea simular la trayectoria de una flecha, es necesario indicarle al motor físico las características de la flecha, de tal forma que se pueda adaptar las ecuaciones genéricas del lanzamiento de proyectil para simular la trayectoria de la flecha.

El motor físico es capaz de realizar simulaciones de ciertos sistemas físicos como la dinámica del cuerpo rígido, el movimiento de un fluido y la elasticidad.

Los motores físicos han sido altamente empleados en el campo de los videojuegos para modelar el mundo real. También se han usado en menor escala para realizar simulaciones con fines científicos, modelando procesos de alta complejidad, que requieren una gran cantidad de cálculos y una alta precisión numérica.

2.7.1. Detección de colisiones.

(Moctezuma, 2006) en su tesis de maestría dice que la detección de colisiones consiste en determinar si dos objetos se encuentran en contacto así como el momento y el lugar exacto donde se dió dicho contacto. Generalmente, para detectar una colisión, cada uno de los objetos que se aproximan se considera en el interior de una caja imaginaria cuyo tamaño es el mínimo necesario para contenerlo, y la intersección de estas cajas se entiende como colisión.

La Figura 2.15 muestra una simulación donde una bala choca con una pared de ladrillos, los cuales están puestos unos sobre los otros sin ningún tipo de pegamento. Se puede observar la reacción producida en la pared al momento de la colisión.

El objetivo principal de esta investigación, es la incorporación de un modelo físico de cuerpos rígidos a un ambiente virtual dinámico, el cual debe ser controlado por un agente universal. Es por ello que es de especial interés el estudio de algunos

de los motores físicos existentes. A continuación se nombraran algunos de ellos.

2.7.2. PhysX

PhysX es un motor físico multiplataforma desarrollado por NVIDIA Corporation que soporta un amplio rango de dispositivos, que van desde un dispositivo iPhone de Apple Inc. hasta procesadores con múltiples núcleos. A pesar que PhysX es un SDK que ha sido diseñado para programadores de videojuegos, este es usado en gran medida por investigadores, educadores y por desarrolladores de aplicaciones para simulación, debido a que se pueden realizar simulaciones en tiempo real con un alto grado de precisión. Algunas de las características que ofrece son: detección de colisiones en forma discreta y continua, *raycasting*, modelo físico para cuerpos rígidos, fluidos y partículas, así como controladores para vehículos y avatares (Nvidia Corporation, 2011).

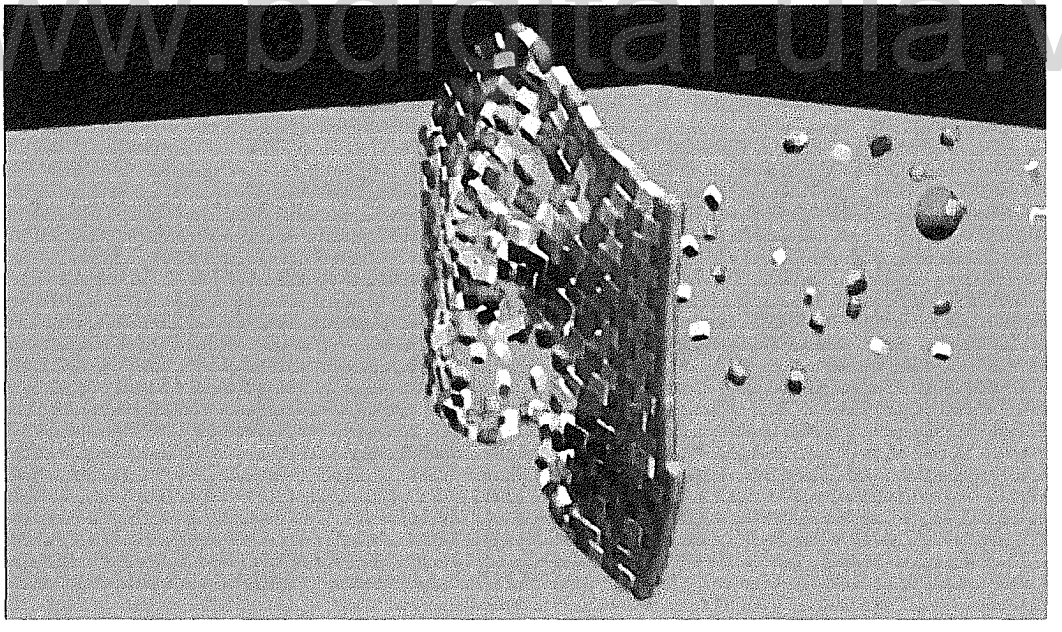

Figura 2.15. Detección de colisiones entre una bala y una pared de ladrillos.
Fuente: http://newtondynamics.com/forum/viewtopic.php?f=14&t=5260

Se han desarrollado varias librerías que permiten el uso de PhysX en Ogre3D,

entre ellos tenemos: NxOgre, OgrePhysX (Caphalor, 2011)

2.7.3. Havok

Según (Havok, 2011) Havok es un motor físico que provee un mecanismo robusto de detección de colisiones y simulación física. Havok trabaja sobre una arquitectura multi-hilo y puede ser ejecutado en múltiples plataformas. Ha sido optimizado para ambientes como Xbox 360 © Microsoft Corporation, Play Station 3 de Sony Computer Entertainment Inc, Wii © Nintendo of America Inc, Nintendo 3DS © Nintendo of America Inc, entre otros.

Al momento de desarrollar esta investigación no se consiguen librerías que permitan la integración con Ogre3D.

2.7.4. Open Dynamics Engine (ODE)

(Smith, 2007) asegura que ODE es una librería de código abierto, que provee un alto grado de rendimiento para la simulación de cuerpos rígidos. Es estable, independiente de la plataforma, además de ser fácil de acoplar con el lenguaje de programación C/C++. Provee varios tipos de uniones entre cuerpos e integra la detección de colisiones con fuerzas de fricción. ODE es útil para simular vehículos, objetos en ambientes de realidad virtual y avatares. Actualmente, es usado por varios juegos para computadoras, herramientas para edición 3D y simulación.

Existe una librería que permite la integración entre ODE y Ogre3D, sin embargo, este proyecto se encuentra un poco abandonado, la librería se llama: OgreODE.

2.7.5. Newton Game Dynamics

De acuerdo con (Jerez & Suero, Newton Game Dynamics, 2011) Newton Game

Dynamics es un proyecto de código abierto que ha sido liberado bajo la licencia zlib/libpng (Wikipedia, 2011), que integra soluciones para simulación en tiempo real de ambientes físicos.

El API de newton provee manejo de escenario, detección de colisión, comportamiento dinámico, además de ser de código reducido, rápido, estable y fácil de usar (Jerez & Suero, Newton Game Dynamics, 2011).

Se puede integrar fácilmente con Ogre3D por medio de una librería conocida como: OgreNewt.

2.7.6. Selección del Motor Físico para el proyecto

Existiendo tantos motores físicos en el mercado y tan pocos estudios comparativos entre ellos, no es sencillo seleccionar uno para ser usado en la investigación; se optará por estudiar las propiedades de cada uno de ellos y seleccionar aquel que permita una fácil integración con el API de gráficos seleccionado previamente (Ogre3D) lo que facilitará el desarrollo del proyecto, que el API sea libre y de código abierto, lo que permitirá realizar adaptaciones al código en caso de requerirlo y finalmente, que se mantenga actualizado constantemente, para corregir los errores de versiones previas.

Es importante resaltar que además de los motores físicos ya estudiados, existen algunas otras librerías que permiten incorporar modelos físicos en ambientes virtuales 3D. Sin embargo, no serán consideradas en este estudio debido a que su integración con el motor de renderizado Ogre3d es muy baja.

En la siguiente tabla se resumen los criterios establecidos para la comparación de los motores físicos y que permiten definir cuál ha de ser usado para el propósito de éste proyecto.

Tabla 2.1. Comparación entre diversos motores físicos.

Motor Físico	Facilidad de Integración con Ogre3D	Código Abierto	Actualizado
PhysX	Media	No	Si
Havok	Baja	No	Si
ODE	Media	Si	No
Newton	Media	Si	Si

Es importante dejar claro que en el estudio, no se tomó en cuenta la capacidad de rendimiento y estabilidad de los motores físicos, ya que no se cuenta con los recursos de tiempo y dinero necesarios para realizar dicha comparación. Por otra parte, los estudios que se encuentran en Internet no pertenecen a una fuente de información confiable, ya que generalmente quienes hacen los estudios son los mismos creadores de las librerías.

En la Tabla 2.1 se puede observar que el motor físico que cumple con todas las características del estudio es **Newton Game Dynamics;** por esa razón, en este proyecto, se hará uso de este motor físico y específicamente de su *Wrapper*: **OgreNewt** el cual facilitara la integración con el API gráfico.

2.7.7. Primitivas de colisión

(Jerez & Suero, Newton Game Dynamics, 2011) en la wiki de Newton Game Dynamics exponen que la primitiva de colisión representa una forma aproximada de la malla del modelo 3D, la cual es usada, en vez de la malla, para determinar la posición y orientación del cuerpo en la simulación, esto permite separar la simulación física de la visualización gráfica, acelerando el rendimiento de dicha simulación.

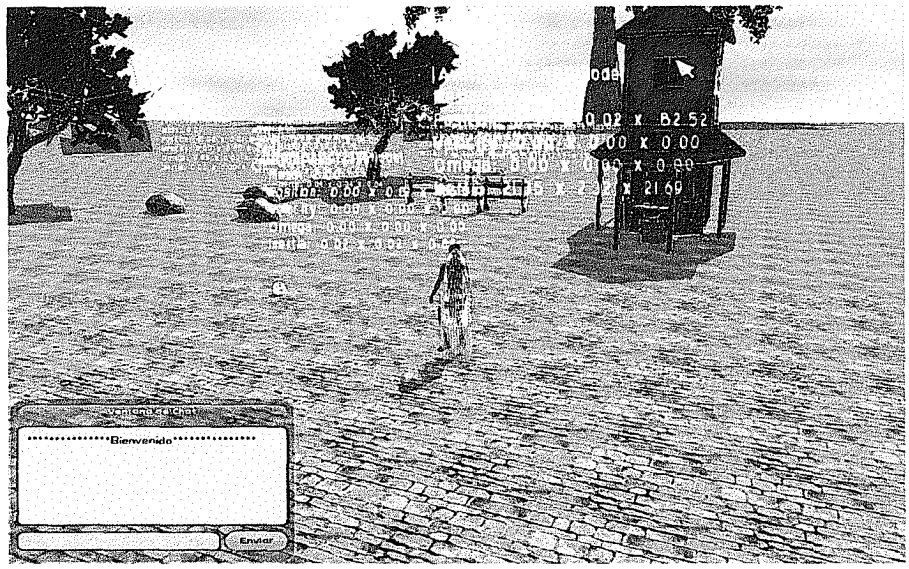

Figura 2.16. Algunas primitivas de colisión

Fuente: propia

En la Figura 2.16 se muestran algunas primitivas de colisión usadas en una simulación física, se puede observar que el avatar es representado por un elipsoide, el balón por una esfera, los árboles casas y piedras por un tipo de primitiva especial denominado árbol de colisión.

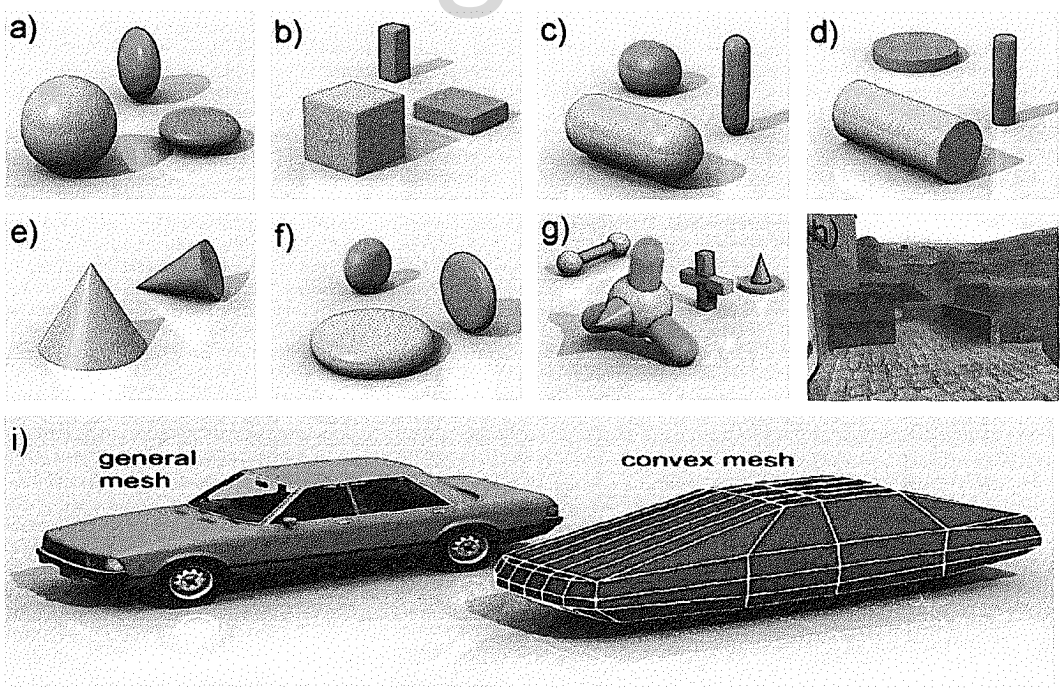

Figura 2.17. Primitivas de colisión disponibles en los motores físicos

Fuente: propia

En la Figura 2.17 se observan algunas de las primitivas disponibles en Newton Game Dynamic y otros motores físicos, de izquierda a derecha y de arriba hacia abajo tenemos: a) primitiva tipo esfera b) primitiva tipo caja, c) primitiva tipo capsula d) primitiva tipo cilindro, e) primitiva tipo cono f) primitiva tipo cilindro biselado g) primitiva compuesta h) primitiva tipo escena (árbol de colisión) i) primitiva de cuerpo convexo.

En la Wiki de Newton Game Dynamics existe una imagen que sirve de ayuda al programador de la simulación física, para decidir el tipo de primitiva de colisión que debe ser usada dependiendo de las características del modelo tridimensional y de las propiedades físicas del mismo en el ambiente. En la siguiente figura se muestra este proceso.

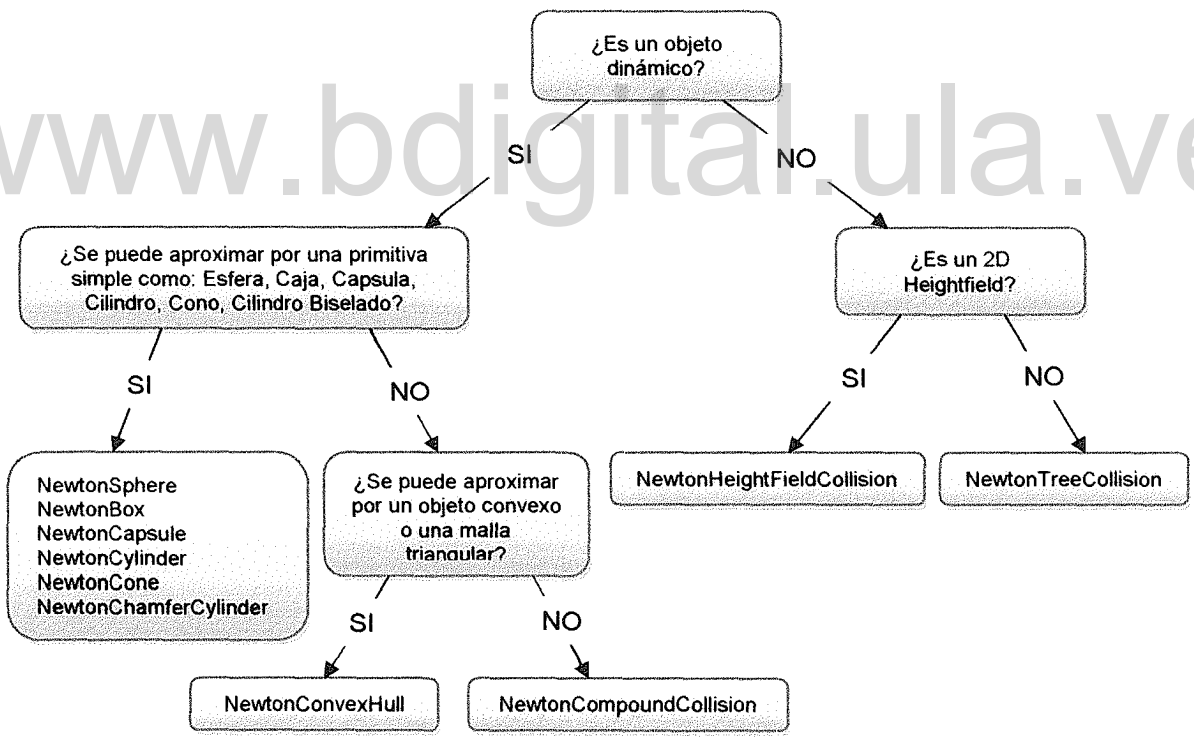

Figura 2.18. Guía para escoger el tipo de primitiva de colisión
Fuente: http://newtondynamics.com/wiki/index.php5?title=Collision_primitives

Capítulo 3. Diseño

En este capítulo se presenta la forma como se han diseñado cada uno de los elementos presentes en el proyecto, entre ellos: el diseño de agentes, diseño de datos, avatares, entre otros.

3.1. Primera Parte: Diseño de los Agentes

El objetivo principal de esta investigación es el desarrollo de un sistema multiagentes para el control de las interacciones físicas entre los avatares y objetos físicos de un ambiente virtual dinámico. Es por ello que la primera fase de esta investigación se basa en el diseño de los agentes que conforman la plataforma multiagentes.

Para el diseño del SMA se hará uso de la metodología MASINA (Aguilar et al., 2008) centrándonos principalmente en las tres primeras fases de esta metodología, dejando la fase de codificación y pruebas para ser desarrollada en la segunda parte del presente proyecto, ya que allí es donde se crea el escenario virtual que permite poner en práctica los agentes diseñados.

En la Figura 3.1 se observa el modelo de los agentes que componen el sistema, los cuales han sido clasificados de acuerdo a si están ubicados en el servidor o en el cliente. Se tienen 5 agentes en el lado del servidor (CustomerService, Bootstrapping, PhysicalControl, Synchronization y DataManger), de los cuales el agente CustomerService se replicará por cada cliente que se conecte al ambiente; mientras que del lado del cliente se tienen 4 agentes (Visualization, Receiver, Broadcast y Repository). Cada uno de estos agentes serán descritos siguiendo las fases de la metodología MASINA.

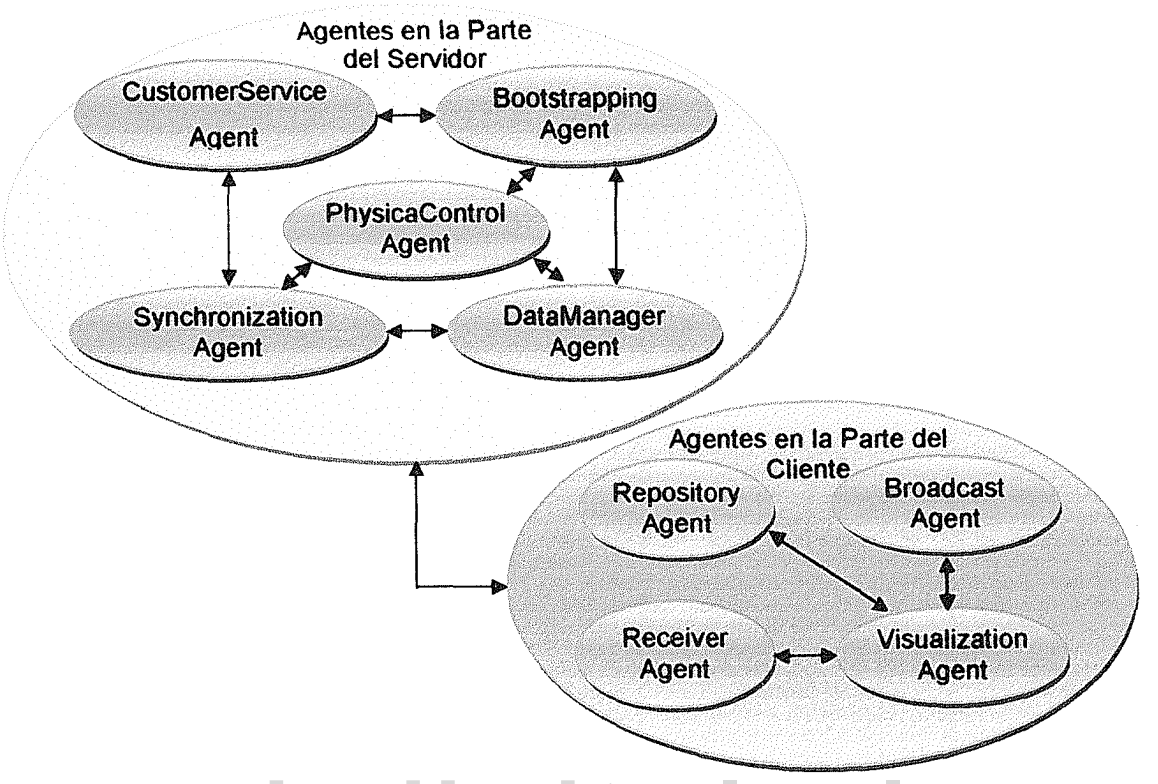

Figura 3.1. Modelo del Sistema Multiagentes para el Control de las Interacciones Físicas entre Objetos y Avatares en un Ambiente Virtual 3D Dinámico[1]
Fuente: propia

3.1.2. Fase I: Conceptualización

En esta fase de la metodología, se definen los servicios requeridos del sistema y quienes lo requieren, también se propone una arquitectura preliminar del SMA y se definen los requisitos que debe cumplir el agente, así como las actividades que llevan a cabo los agentes para cumplir con ellos.

3.1.2.1. Agente Bootstrapping (Agente de Iniciación)

Es un agente de tipo reactivo, se encarga de percibir la configuración inicial del Ambiente Virtual y ejecutar las acciones correspondientes para visualizar el mundo

[1] Los nombres de los agentes fueron escritos en ingles debido a que el investigador realizó el código y diseño en este idioma para acortar los nombres de las clases en los diagramas.

correctamente. También tiene como tarea despertar a los demás agentes para que comiencen a percibir información; otra de sus funciones es indicar a cada Agente **CustomerService** sobre los objetos iniciales que conforman el Ambiente virtual.

Este agente reside en el servidor del Ambiente Virtual Distribuido. En la Figura 3.2 se observa el diagrama de casos de uso del agente Bootstrapping, el actor Administrador es el encargado de iniciar el AVD (El administrador inicia el servidor) y una vez creado, se despiertan los demás agentes, para posteriormente pasar a esperar conexiones de usuario. La descripción de los casos de uso se puede observar en la Tabla 3.1, en la Tabla 3.2 y en la Tabla 3.3

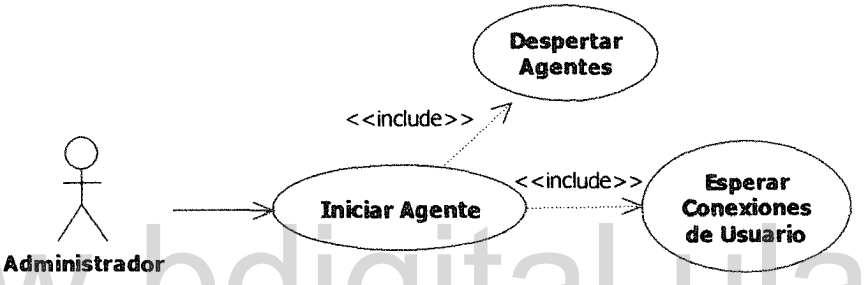

Figura 3.2. Diagrama de casos de uso para el Agente Bootstrapping

Tabla 3.1. Descripción del caso de uso Iniciar Agente

Caso de Uso	Iniciar Agente
Descripción	Iniciar el Servidor del Ambiente Virtual
Pre-condición	Existencia de un mundo virtual en el repositorio de datos
Actores	Administrador
Condición de fracaso	Error de acceso a los datos en el repositorio
Condición de éxito	Objetos del AVD cargados correctamente desde el repositorio

Se definen dos servicios para el agente, uno de ellos se encarga de iniciar el AVD y es activado por el Administrador, el otro tiene como tarea principal esperar conexiones de usuarios, y crear un nuevo agente **CustomerService** para que se encargue de atender las solicitudes del usuario entrante. Una vez realizada esta

tarea, el servicio se reiniciará para comenzar nuevamente con la espera. El diagrama de actividades para este agente se puede observar en la Figura 3.3

Tabla 3.2. Descripción del caso de uso Despertar Agentes

Caso de Uso	Despertar Agentes
Descripción	Se encarga de despertar a los demás agentes del Ambiente Virtual
Pre-condición	Mundo virtual cargado correctamente
Actores	Administrador
Condición de fracaso	Error de comunicación entre agentes
Condición de éxito	Respuesta satisfactoria por parte de los demás agentes

Tabla 3.3. Descripción del caso de uso Esperar Conexiones de Usuario

Caso de Uso	Esperar Conexiones de Usuario
Descripción	Atiende la solicitud de conexión de un usuario e informa al Agente Sincronización sobre la existencia del mismo.
Pre-condición	Agentes del SMA en ejecución
Actores	Administrador, Usuario
Condición de fracaso	Error de comunicación entre agentes o falla de conexión
Condición de éxito	Respuesta satisfactoria del Agente Sincronización

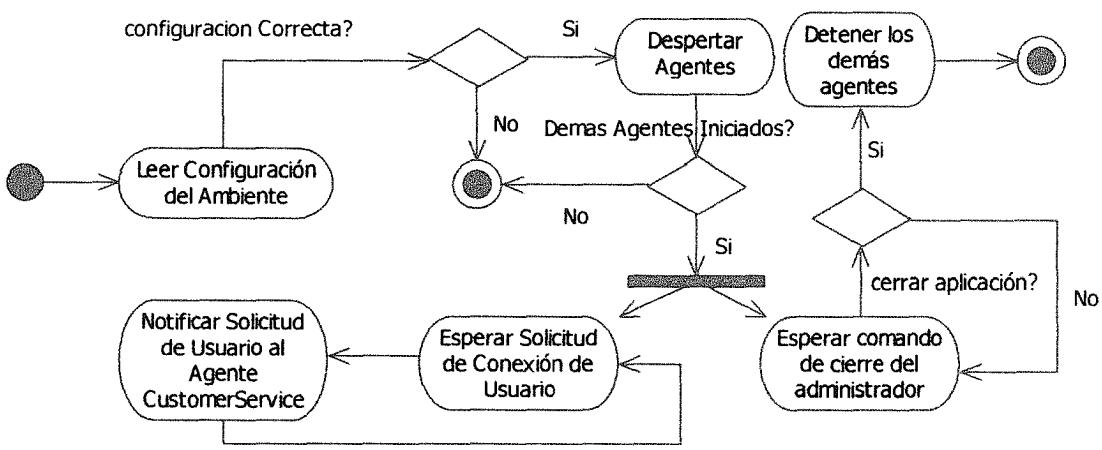

Figura 3.3. Diagrama de actividades del Agente Bootstrapping

3.1.2.2. Agente CustomerService (Agente de Atención al Cliente)

Es un agente reactivo, encargado de dar respuesta a las solicitudes realizadas por el cliente, cada vez que un agente del lado del cliente requiera de alguna información en el servidor, debe realizar una solicitud al agente CustomerService, el cual, una vez procesada la solicitud, enviará el resultado al agente solicitante.

El agente reside en el servidor y presenta un servicio, encargado de procesar y dar respuestas a las solicitudes que realice el usuario, este agente se comunica directamente con el Agente DataManager para solicitar la información necesaria que permita dar respuesta a la solicitud del usuario. El diagrama de casos de uso del agente se observa en la Figura 3.4.

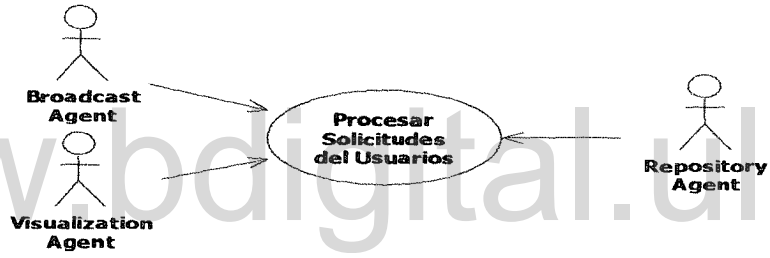

Figura 3.4. Diagrama de Casos de Uso para el Agente CustomerService

La descripción del caso de uso expuesto en el diagrama anterior, se pueden observar en la Tabla 3.4. Este caso de uso es activado únicamente por los agentes BootstrappingAgent, RepositoryAgent y VisualizationAgent.

Tabla 3.4. Descripción del caso de uso procesar solicitudes del usuario

Caso de Uso	Procesar Solicitudes del Usuario
Descripción	Atiende y da respuesta a las solicitudes de los Agentes en el lado del Cliente
Pre-condición	DataManager Agent activo.
Actores	Broadcast Agent, Visualization Agent, Repository Agent
Condición de fracaso	Acción solicitada no presente en la lista de solicitudes posibles
Condición de éxito	Solicitud procesada

Las actividades que lleva a cabo el agente para cumplir con sus objetivos se muestran en la Figura 3.5.

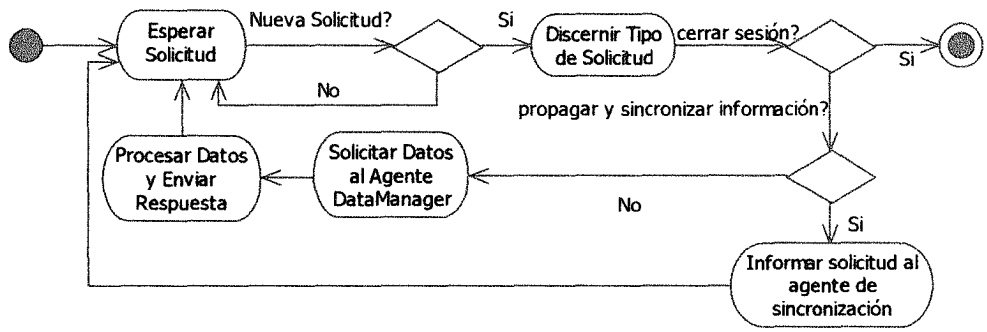

Figura 3.5. Diagrama de actividades para el Agente CustomerService

3.1.2.3. Agente DataManager (Agente Gestor de Datos)

Este agente tiene como tarea principal servir de interfaz entre la base de datos y los demás agentes del sistema. De tal manera, que cada vez que un agente necesita alguna información de la base de datos, debe comunicarse con el agente DataManager, el cual se encargará de ejecutar las instrucciones necesarias, para guardar, eliminar, actualizar o recuperar la misma.

La ventaja que tiene este agente, es que sólo él conoce el lenguaje de comunicación con la base de datos, evitando que los demás agentes deban aprender otro lenguaje. Por otra parte, si en un futuro se decide cambiar el Sistema Manejador de Base de Datos, solo será necesario modificar el agente DataManager para que acepte el nuevo lenguaje, evitando tener que realizar modificaciones en todos los otros agentes y haciendo más flexible la incorporación de actualizaciones al sistema.

En la Figura 3.6 se presenta el diagrama de casos de uso para el agente DataManager y en la Tabla 3.5 se resume la descripción del mismo. Se puede ver que este agente se encarga de procesar la solicitud de datos del Agente

CustomerService y del agente PhysicalControl. El Agente presta un único servicio, que se encarga de procesar las solicitudes de datos provenientes de los demás agentes del SMA.

Figura 3.6. Diagrama de casos de uso para el Agente DataManager

Tabla 3.5. Descripción del caso de uso Procesar Solicitud de Datos

Caso de Uso	Procesar Solicitud de Datos
Descripción	Procesa las solicitudes de datos realizadas al agente.
Pre-condición	Conexión a Base de Datos correcta
Actores	CustomerService Agent, Physical Control Agent
Condición de fracaso	No se puede establecer la conexión a la base de datos
Condición de éxito	Solicitud procesada

El diagrama de las actividades que debe realizar el agente para cumplir con sus tareas se presenta en la Figura 3.7, Aquí se observa como el agente se encuentra en un ciclo recibiendo y procesando las solicitudes de datos que realizan los demás agentes, el ciclo termina cuando se cierra la aplicación servidora.

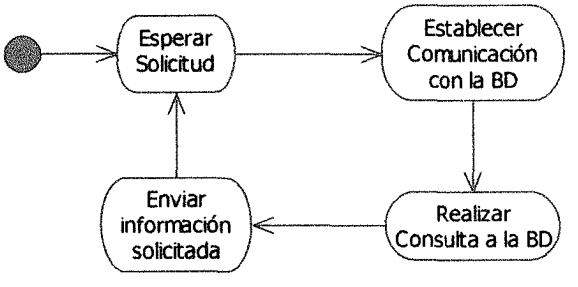

Figura 3.7. Diagrama de Actividades para el Agente DataManager

3.1.2.4. Agente Synchronization (Agente de Sincronización)

El agente Synchronization es un agente basado en modelos que mantiene en su memoria la información de la ubicación, y dirección de cada objeto presente en la escena. El diagrama de casos de uso de este agente se presenta en la Figura 3.8, se observa que el agente tendrá como tarea fundamental propagar la información a uno o más usuarios del sistema, con la finalidad de mantener sincronizados los datos. La descripción para este caso de uso se presenta en la Tabla 3.6

Figura 3.8. Diagrama de Casos de Uso para el Agente Synchronization.

Tabla 3.6. Descripción del caso de uso Propagar Información

Caso de Uso	Propagar Información
Descripción	Propagar la información de los usuarios a los demás a fin de sincronizar el sistema.
Pre-condición	Existan usuarios conectados
Actores	CustomerService Agent, PhysicalCobtrol Agent
Condición de fracaso	No se puede establecer la comunicación.
Condición de éxito	Información propagada.

Se define un servicio, para este agente, este servicio recibe solicitudes de sincronización y establece todo lo necesario para que la información sea propagada a los demás usuarios. En la Figura 3.9 se pueden observar las actividades que debe llevar a cabo el agente Synchronization para cumplir con las metas propuestas.

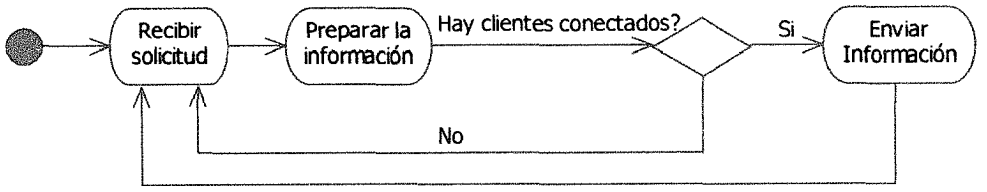

Figura 3.9. Diagrama de actividades para el Agente Synchronization

3.1.2.5. Agente Repository (Agente Repositorio)

El agente Repository, es un agente de tipo reactivo, que se encuentra del lado del cliente. Este agente sirve como repositorio de datos temporales, manteniendo en memoria información referente a los avatares de un usuario u objetos de un ambiente virtual. Sin embargo, no se considera como un agente con modelo, ya que no utiliza la información para tomar decisiones.

Este agente se comunica con el CustomerService para hacer solicitudes de información, como los datos del usuario, avatares y objetos de sistema, entre otros. El agente tiene un objetivo similar al del agente DataManager, solo que el primero no pide información directamente al Sistema Manejador de Base de datos, sino que la solicitud la realiza al CustomerService, para que éste pida luego la información al DataManager, y por último el agente CustomerService, mandará los resultados al Agente Repository quien los dirigirá hacia el agente que haya hecho la solicitud inicialmente, guardando en su memoria un cache de la información, para posteriores consultas.

La Figura 3.10 muestra cómo el agente Visualization es quien realiza la solicitud de datos al agente Repository. La descripción de este caso de uso se puede observar en la Tabla 3.7

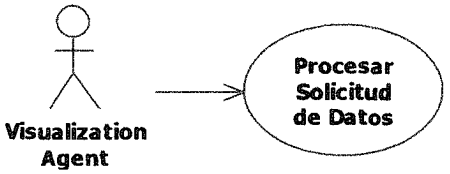

Figura 3.10. Diagrama de caso de uso del Agente Repository

Tabla 3.7. Descripción del caso de uso procesar solicitud de datos

Caso de Uso	Procesar solicitud de datos
Descripción	Carga los datos desde el servidor y replicar la información al agente solicitante
Pre-condición	Usuario conectado
Actores	Visualization Agent
Condición de fracaso	No se puede establecer la comunicación al servidor, o información inexistente.
Condición de éxito	Información recuperada y replicada

El agente Repository presenta un servicio, que se encarga de responder a las solicitudes de datos del agente de Visualization. Las actividades que desarrolla el agente para cumplir con este servicio se muestra a continuación.

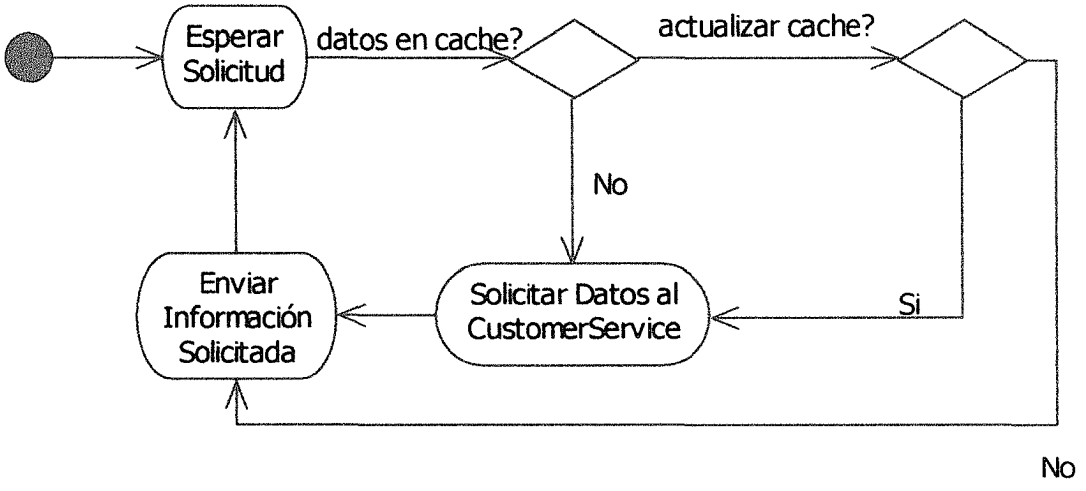

Figura 3.11. Diagrama de actividades para el Agente Repository

3.1.2.6. Agente Vizualization (Agente de Visualización)

El agente Visualization es el encargado de todo lo que tiene que ver con la presentación gráfica de los objetos y avatares en el ambiente virtual. Establece una interfaz de comunicación con el usuario del ambiente, ya que es este agente quien recibe las órdenes directamente del usuario, para posteriormente redirigirlas hacia el agente encargado de procesar dicha orden.

Como se puede observar en la Figura 3.12, este agente presenta mucha más funcionalidad que los agentes antes vistos. El caso de uso Actualizar Vista, es activado por el mismo agente de visualización, y esto lo hace de forma automática mediante un hilo que constantemente renderiza los objetos del ambiente en sus nuevas ubicaciones.

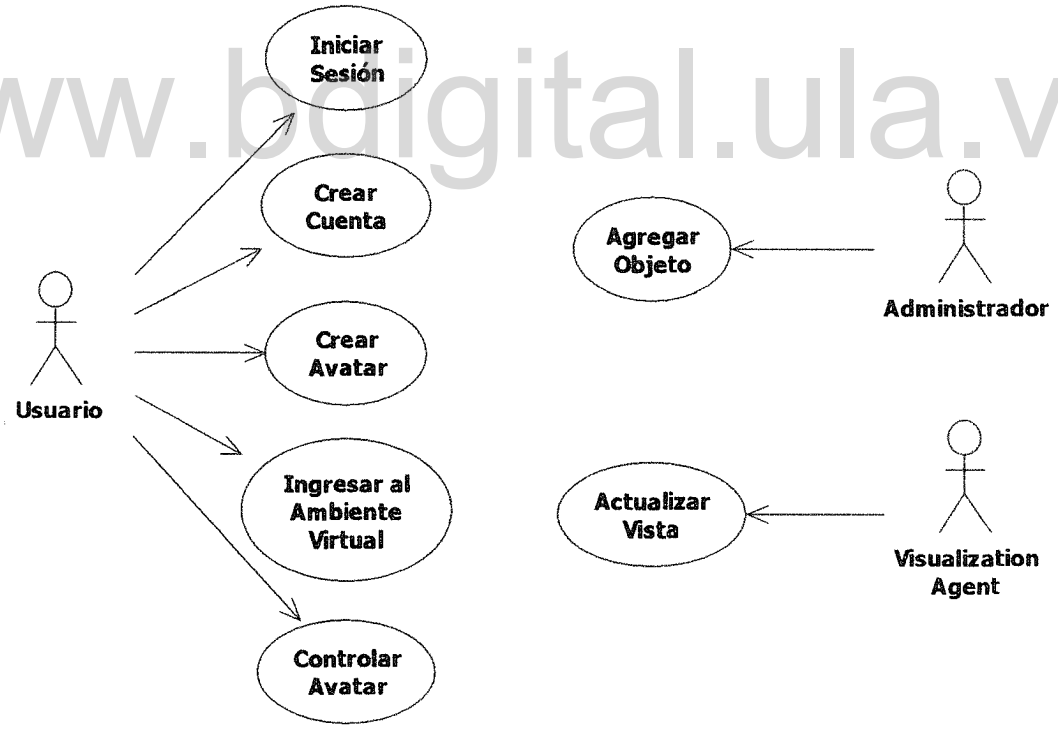

Figura 3.12. Diagrama de Casos de Uso para el Agente Visualization

La descripción de los casos de uso expuestos en el diagrama anterior se puede

observar en la Tabla 3.8, la Tabla 3.9, la Tabla 3.10, la Tabla 3.11, la Tabla 3.12, la Tabla 3.13 y la Tabla 3.14.

Tabla 3.8. Descripción del caso de uso iniciar sesión

Caso de Uso	Iniciar Sesión
Descripción	Se encarga de validar que el usuario exista en el sistema y que sus datos son correctos
Pre-condición	Servidor en ejecución
Actores	Usuario
Condición de fracaso	No se puede establecer la comunicación al servidor, o datos inválidos
Condición de éxito	Datos correctos.

Tabla 3.9. Descripción del caso de uso crear cuenta

Caso de Uso	Crear Cuenta
Descripción	Crea un nuevo usuario.
Pre-condición	Servidor en ejecución
Actores	Usuario
Condición de fracaso	No se puede establecer la comunicación al servidor, o datos incorrectos.
Condición de éxito	Cuenta de usuario creada.

Tabla 3.10. Descripción del caso de uso crear avatar

Caso de Uso	Crear avatar
Descripción	Crea un nuevo avatar para el usuario activo.
Pre-condición	Usuario conectado.
Actores	Usuario
Condición de fracaso	No se puede establecer la comunicación al servidor, o datos incorrectos, o número de avatares por usuario, superados.
Condición de éxito	Avatar creado.

Para este agente será necesario definir un servicio, encargado de esperar las órdenes del usuario. El diagrama de actividades para este agente se presenta en la Figura 3.13.

Tabla 3.11. Descripción del caso de uso ingresar al ambiente

Caso de Uso	Ingresar al ambiente
Descripción	Carga la información del ambiente, así como los objetos y demás avatares conectados.
Pre-condición	Usuario conectado y Avatar de usuario Exista.
Actores	Usuario
Condición de fracaso	No se puede establecer la comunicación al servidor.
Condición de éxito	Ambiente virtual, presentado y avatar listo para recibir ordenes.

Tabla 3.12. Descripción del caso de uso controlar avatar

Caso de Uso	Controlar Avatar
Descripción	Recibe la orden que el usuario quiera dar al avatar y la dirige al agente que corresponda.
Pre-condición	Usuario conectado y dentro del ambiente virtual.
Actores	Usuario
Condición de fracaso	Orden no se puede ejecutar.
Condición de éxito	Orden Ejecutada

Tabla 3.13. Descripción del caso de uso agregar objeto

Caso de Uso	Agregar Objeto
Descripción	Agrega nuevos objetos al ambiente virtual distribuido.
Pre-condición	Administrador conectado y dentro del ambiente.
Actores	Administrador
Condición de fracaso	No se puede establecer la comunicación al servidor.
Condición de éxito	Objeto agregado al ambiente.

Tabla 3.14. Descripción del caso de uso actualizar vista

Caso de Uso	Actualizar Vista
Descripción	Repinta la vista de la cámara del usuario cada cierto tiempo, dependiendo de los cuadros por segundos que sean estipulados.
Pre-condición	Usuario conectado y ambiente virtual en ejecución.
Actores	Visualization Agent
Condición de fracaso	Las capacidades del computador donde se ejecuta la aplicación no soporten el ambiente.
Condición de éxito	Vista actualizada

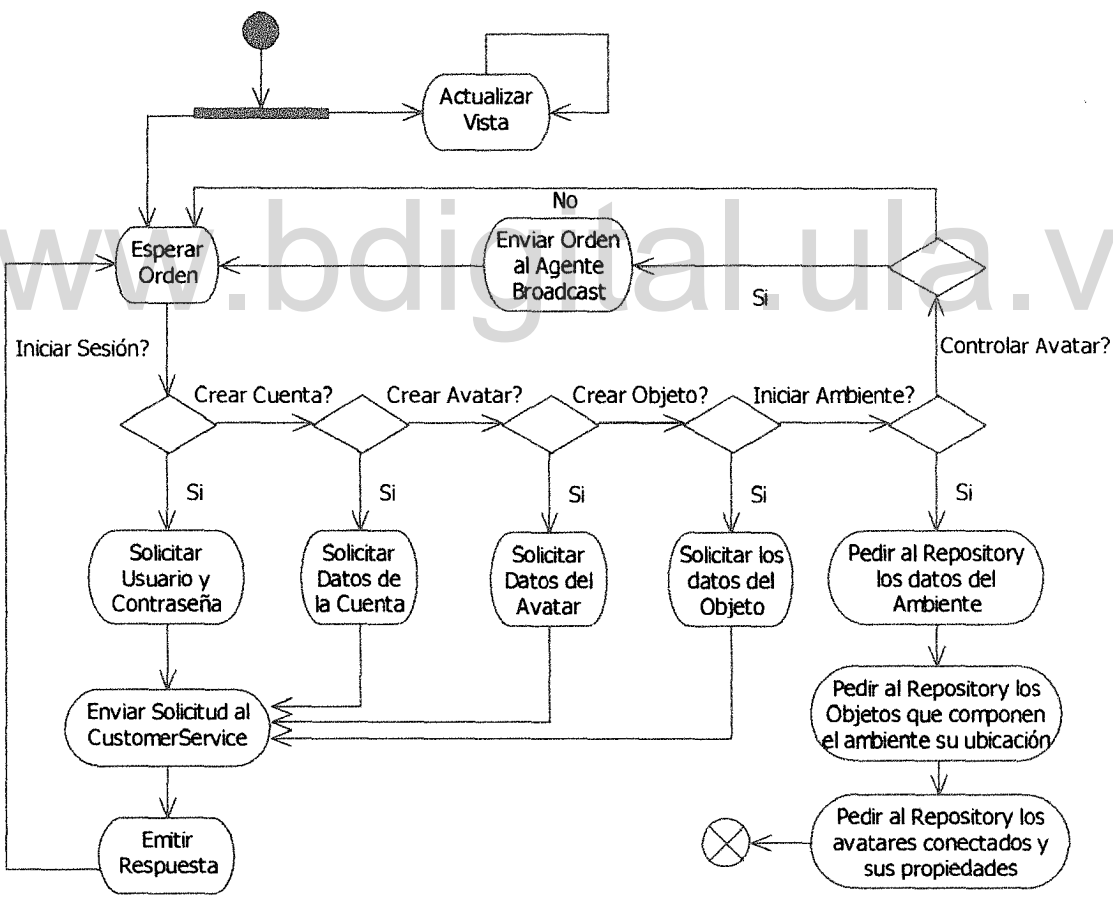

Figura 3.13. Diagrama de Actividades para el Agente Visualization

3.1.2.7. Agente PhysicalControl (Agente de Control Físico)

El agente PhysicalControl es un agente con modelo que se encarga del control de las interacciones físicas entre los avatares y objetos físicos del ambiente virtual 3d dinámico. Además, este agente es quien se encarga de incorporar a la lista de simulaciones físicas las acciones que serán ejecutadas por el avatar que esté controlando el usuario.

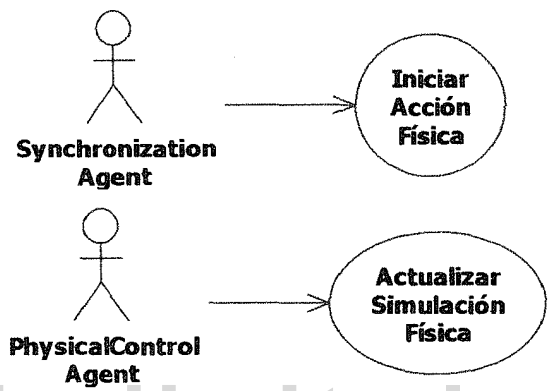

Figura 3.14. Diagrama de Casos de uso del Agente PhysicalControl

Tabla 3.15. Descripción del caso de uso iniciar acción física

Caso de Uso	Iniciar Acción Física
Descripción	Inicia la acción física en el avatar que controla el usuario.
Pre-condición	Usuario conectado y ambiente virtual en ejecución.
Actores	Synchronization Agent
Condición de fracaso	Acción física no permitida o no se puede iniciar físicamente.
Condición de éxito	Acción iniciada.

En la Figura 3.14 se presenta el diagrama de casos de uso del Agente PhysicalControl, se observa como el Agente Synchronization realiza la solicitud al agente Físico para que inicie o incorpore a la simulación la acción física solicitada por el usuario, y es el mismo agente físico quien posteriormente actualiza la simulación para calcular o limitar las posiciones de los objetos y avatares del AVD.

La descripción de estos casos de uso se puede ver en la Tabla 3.15 y la Tabla 3.16

Tabla 3.16. Descripción del caso de uso actualizar simulación física

Caso de Uso	Actualizar
Descripción	Actualiza la simulación física en un tiempo preestablecido.
Pre-condición	Usuario conectado y ambiente virtual en ejecución.
Actores	PhysicalControl Agent
Condición de fracaso	Error al actualizar la física del ambiente
Condición de éxito	Ambiente actualizado, posiciones de objetos recalculadas.

Se define un servicio para el agente el cual se encarga de procesar las solicitudes que realice el agente Visualization. El diagrama de actividades para este agente se muestra en la Figura 3.15.

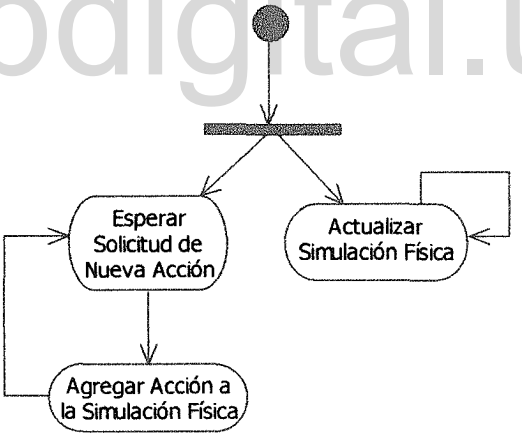

Figura 3.15. Diagrama de Actividades para el agente PhysicalControl

3.1.2.8. Agente Broadcast (Agente de Emisor de Información)

El agente Broadcast en conjunto con el agente Receiver son los encargados de mantener sincronizado el ambiente virtual en la parte del cliente. Cuando se agrega una nueva acción a la simulación física, se le anuncia al agente Broadcast,

el cual, se comunica con el agente CustomerService para solicitar que esa acción sea informada a los demás usuarios conectados, al ambiente, el CustomerService se comunicará a su vez con el agente Sincronizatión del servidor para que se encargue de llevar a cabo el proceso de información. En la Figura 3.16 se presenta el diagrama de casos de uso para el agente Broadcast, y su descripción se muestra en la Tabla 3.17

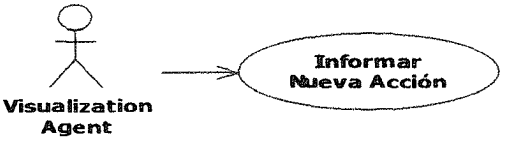

Figura 3.16. Diagrama de Casos de Uso para el Agente Broadcast

Tabla 3.17. Descripción del caso de uso informar nueva acción

Caso de Uso	Informar nueva acción
Descripción	Se encarga de informar al CustomerService sobre el inicio de una nueva acción física por parte del avatar del usuario.
Pre-condición	Usuario conectado y ambiente virtual en ejecución.
Actores	Visualization Agent
Condición de fracaso	No se puede establecer la comunicación con el CustomerService
Condición de éxito	Nueva acción Informada

Se propone un servicio para el agente, el cual es encargado de recibir las solicitudes realizadas por el agente Visualization y posteriormente informar al CustomerService. En la Figura 3.17 se puede observar el diagrama de actividades que ejecuta el agente Broadcast para cumplir con sus tareas.

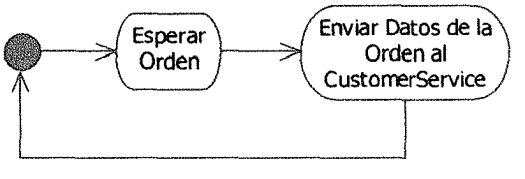

Figura 3.17. Diagrama de actividades del Agente Broadcast

3.1.2.9. Agente Receiver (Agente Receptor de Mensajes)

El agente Receiver se encarga de recibir las órdenes iniciadas por otros usuarios en el ambiente virtual. Estás ordenes son enviadas por el Agente Synchronization que es el encargado en el lado del servidor de propagar las órdenes informadas por el usuario. Una vez que el agente Receiver, obtiene la información la remite al agente de Visualization, para que se encargue de iniciar esas órdenes en el ambiente local. En la Figura 3.18 se muestra el diagrama de casos de uso para este agente y su descripción en la Tabla 3.18.

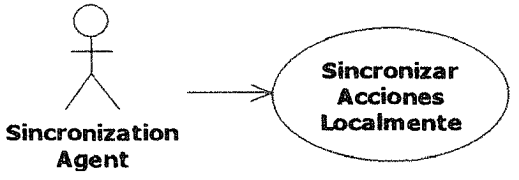

Figura 3.18. Diagrama de Casos de Uso para el Agente Receiver

Para este agente se define un único servicio, encargado de recibir la información y pasarla posteriormente al agente que corresponda. El diagrama de actividades que describe este proceso se puede observar en la Figura 3.19

Tabla 3.18. Descripción del caso de uso sincronizar acciones localmente

Caso de Uso	Sincronizar acciones localmente
Descripción	Recibe las ordenes a ser sincronizadas desde el agente Synchronization y las remite al agente de Visualization o PhysicalControl según el caso
Pre-condición	Usuario conectado y ambiente virtual en ejecución.
Actores	Synchronization Agent
Condición de fracaso	No se pueden recibir los datos o el objeto a sincronizar no existe en el ambiente local
Condición de éxito	Sincronización iniciada

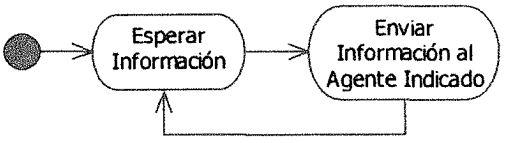

Figura 3.19. Diagrama de actividades de Agente Receiver

3.1.3. Fase II: Análisis

En esta fase se presentara el modelo de comunicación y coordinación del SMA, los cuales definen el comportamiento del agente. También se muestran los actos de habla de cada uno de los agentes que componen el SMA. Cabe destacar que por cuestiones de tiempo la descripción detallada de cada una de las tareas de los agentes, así como cada uno de los actos de habla no se realizó y por tal motivo no se anexa en este documento.

3.1.3.1. Modelo del Agente Bootstrapping

La Tabla 3.20 muestra el modelo de agente para el Agente Bootstrapping, se especifica el objetivo principal del agente y se describen sus dos servicios.

Tabla 3.19. Modelo de tareas para el agente Bootstrapping

Servicios	Tareas
Iniciar Ambiente Virtual Dinámico	T1. Recibir solicitud para iniciar el agente T2. Cargar del archivo de configuración los datos del ambiente virtual a cargar T3. Despertar Agentes de sincronización y datos T4. Iniciar servicio que procesa solicitudes de conexiones de usuario
Esperar conexiones de usuario	T1. Recibir solicitud de conexión. T2. Crear un nuevo agente CustomerService para que atienda al usuario T3. Delegar tarea de atención de usuario al CustomerService recién creado.

Tabla 3.20. Modelo de Agente para el Agente Bootstrapping

Agente		
Nombre		Agente Bootstrapping
Posición		SMA lado del servidor
Componentes		No aplica
Marco de referencia		No aplica
Descripción del agente		Se encarga de iniciar el servidor del AVD, despertar al resto de agentes para que estén pendientes de procesar solicitudes y esperar por nuevas conexiones de usuario.
Objetivos del Agente		
Nombre		Iniciar servidor
Descripción		Recibe la solicitud para cargar el AVD y comenzar a prestar servicio a los usuarios del AVD
Parámetro de entrada		Solicitud de inicio por parte del administrador
Parámetro de salida		Notificación de AVD creado correctamente
Condición de activación		Recepción de solicitud de inicio
Condición de finalización		AVD cargado y en espera de usuarios entrantes
Condición de éxito		El AVD cargado corresponde al solicitado por el administrador
Condición de fracaso		Error de comunicación entre los actores.
Ontología		Ontología iniciación de ambientes virtuales
Servicios del Agente		
Nombre		Iniciar AVD
Descripción del Servicio		El agente recibe la solicitud para inicial el AVD por parte del administrador. El agente procede a dar aviso a los demás agentes para que procesen cualquier solicitud entrante, luego se queda a la espera de los usuarios del AVD.
Tipo de Servicio		Dual
Parámetros de entrada		Ninguno
Parámetros de salida		Notificación de AVD cargado
Propiedades del Servicio		
Nombre	**Valor**	**Descripción**
Calidad	0-100	Porcentaje de la calidad del servicio en función del tiempo de respuesta
Auditable	1	Capacidad de diagnosticar la calidad del servicio
Confiabilidad	0-100	Certificación de respuesta
Nombre		Atender conexiones de usuario
Descripción del Servicio		El agente recibe la solicitud de conexión de parte del usuario, crea una instancia de CustomerService y delega la función en este nuevo agente
Tipo de Servicio		Dual
Parámetros de entrada		Solicitud de conexión al AVD
Parámetros de salida		Notificación de status de la conexión
Propiedades del Servicio		
Nombre	**Valor**	**Descripción**
Calidad	0-100	Porcentaje de la calidad del servicio en función del tiempo de respuesta
Auditable	1	Capacidad de diagnosticar la calidad del servicio
Confiabilidad	0-100	Certificación de respuesta
Capacidad del Agente		
Habilidades del agente		Gestionar servicios de comunicación
Rep. del Conocimiento		No Aplica
Lenguaje de Comunicación		El agente se comunica por medio de los métodos de la clase y XML.

3.1.3.2. Modelo de Tareas para el Agente Bootstrapping

En la Tabla 3.19 se muestra la relación de las tareas que ejecuta el Agente, asociadas a cada uno de los servicios del mismo.

3.1.3.3. Modelo de Coordinación y Comunicación para el Agente Bootstrapping

La Tabla 3.21 y la Tabla 3.22 muestran el modelo de conversación para cada uno de los servicios que ofrece el agente. Este modelo se describe en los diagramas de secuencias de las Figura 3.21 y Figura 3.20

Tabla 3.21. Modelo de conversación del Agente Bootstrapping Servicio Iniciar AVD

CONVERSACIÓN: Solicitud de Servicio Iniciar AVD	
Objetivo	Iniciar el servidor del AVD y despertar los agentes del SMA
Agentes participantes	Agente Bootstrapping, Agente DataManager, Agente Synchronization
Iniciador	Administrador
Actos de habla	Solicitar iniciar AVD, Iniciar Agentes del SMA
Precondición	Existir una solicitud para iniciar el servidor del AVD
Condición de terminación	No se pueda establecer comunicación con los demás agentes
Descripción	El Administrador inicia una conversación que permite al Agente Bootstrapping poner en línea el servidor del AVD.

Tabla 3.22. Modelo de conversación del Agente Bootstrapping servicio atender conexiones

CONVERSACIÓN: Solicitud atender conexiones de usuario	
Objetivo	Establecer conexión entre la vista del usuario y el servidor del AVD
Agentes participantes	Agente Bootstrapping, Agente CustomerService, Usuario, Agente Visualization
Iniciador	Usuario
Actos de habla	Solicitud de conexión por parte del usuario, Iniciar Agentes de Visualización, Iniciar CustomerService
Precondición	Servidor en ejecución
Condición de terminación	Usuario siendo atendido por su CustomerService
Descripción	En esta conversación el Usuario indica al Agente iniciador que cree una vista en su monitor y que permita conectarse al servidor, el Agente Bootstrapping crea un CustomerService que se encarga de atender al usuario.

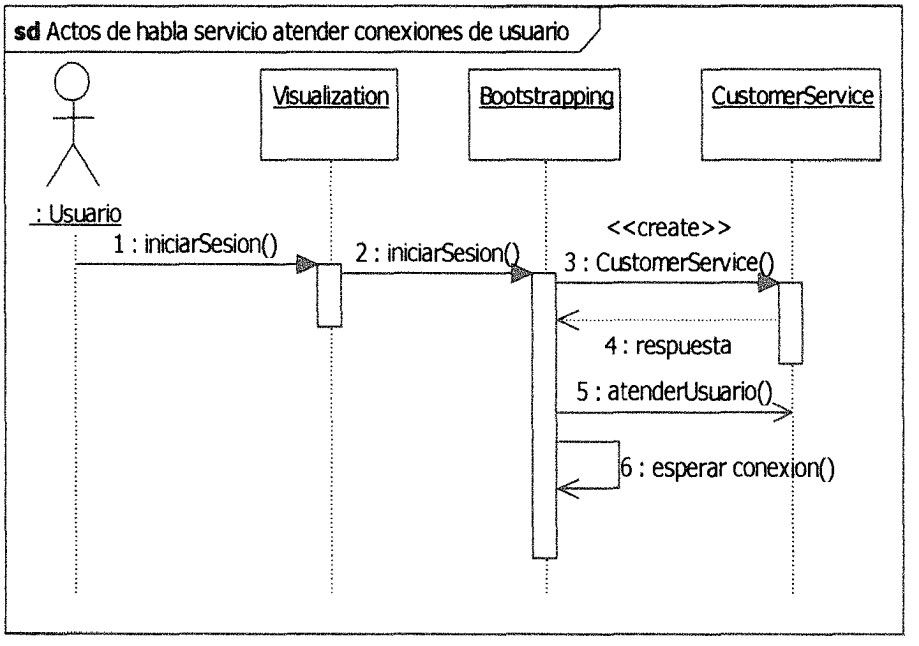

Figura 3.20. Actos de Habla para el Servicio atender conexión de usuario

3.1.3.4. Modelo del Agente DataManager

El modelo de Agente para el agente DataManager se muestra en la Tabla 3.24, así como su objetivo principal y servicios.

3.1.3.5. Modelo de Tareas para el Agente DataManager

El conjunto de tareas que debe llevar a cabo el agente se muestran en la Tabla 3.23

Tabla 3.23. Modelo de tareas para el agente DataManager

Servicios	Tareas
Recuperar Datos	T1. Recibir solicitud de datos T2. Anexar solicitud a la cola de tareas T3. Extraer la próxima solicitud de la cola y determinar la cadena de consulta a enviar a la BD T4. Enviar la consulta a la bd y obtener la información T5. Enviar información al agente solicitante.

Tabla 3.24. Modelo de Agente para el Agente DataManager

Agente		
Nombre		Agente DataManager
Posición		SMA lado del servidor
Componentes		No aplica
Marco de referencia		No aplica
Descripción del agente		Se encarga de procesar las solicitudes de datos realizadas por los demás agentes que se encuentran del lado del servidor, sirviendo de interfaz entre estos y la base de datos.
Objetivos del Agente		
Nombre		Comunicarse con la base de datos y recuperar información
Descripción		Servir de interfaz entre los agentes del sistema y el Sistema Gestor de Base de Datos
Parámetro de entrada		Datos a recuperar
Parámetro de salida		Datos recuperados
Condición de activación		Recepción de solicitud de datos
Condición de finalización		Solicitud procesada
Condición de éxito		Datos recuperados
Condición de fracaso		Datos no existentes
Ontología		Ontología comunicación con base de datos
Servicios del Agente		
Nombre		Recuperar Datos
Descripción del Servicio		El agente recibe la solicitud de datos por parte del agente CustomerService, se comunica con la base de datos para obtener la información y la retorna al agente solicitante
Tipo de Servicio		Dual
Parámetros de entrada		Datos a recuperar
Parámetros de salida		Datos recuperados
Propiedades del Servicio		
Nombre	**Valor**	**Descripción**
Calidad	0-100	Porcentaje de la calidad del servicio en función del tiempo de respuesta
Auditable	1	Capacidad de diagnosticar la calidad del servicio
Confiabilidad	0-100	Certificación de respuesta
Capacidad del Agente		
Habilidades del agente		Capacidad para comunicarse con el sistema manejador de base de datos
Rep. del Conocimiento		No Aplica
Lenguaje de Comunicación		El agente se comunica por medio de los métodos de la clase y XML.

3.1.3.6. Modelo de Coordinación y Comunicación para el Agente DataManager

El modelo de coordinación del agente para el servicio recuperar datos se observa en la Tabla 3.25 y el diagrama de secuencias de la Figura 3.22 describe los actos de habla que se producen en la comunicación.

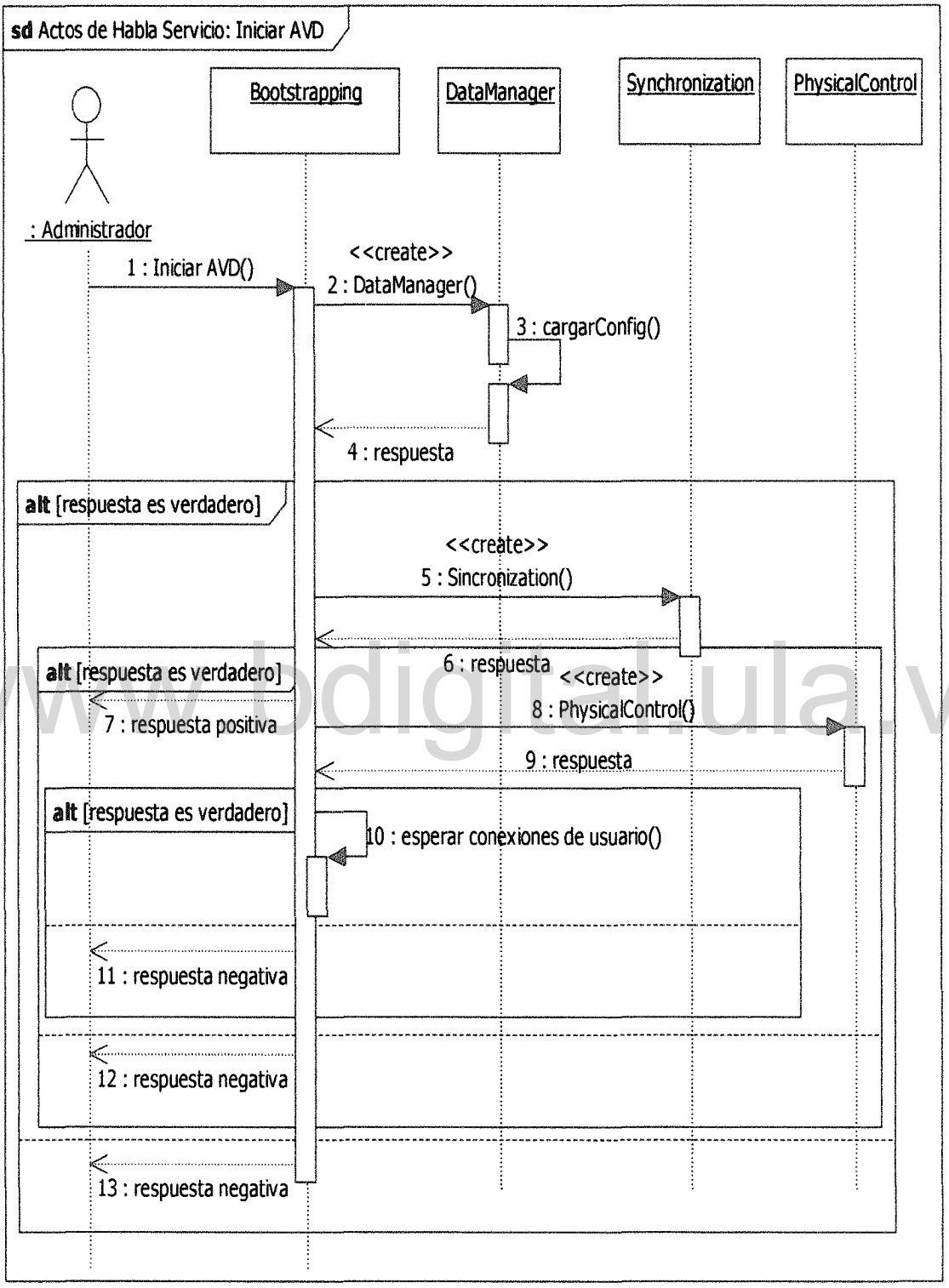

Figura 3.21. Actos de habla para el servicio Iniciar AVD

Tabla 3.25. Modelo de conversación del Agente DataManager servicio recuperar datos

CONVERSACIÓN: Solicitud recuperar datos	
Objetivo	Recuperar un conjunto de datos desde el servidor de base de datos.
Agentes participantes	Agentea CustomerService, DataManager y PhysicalControl
Iniciador	Agente CustomerService, Agente PhysicalControl
Actos de habla	Solicitud de datos, establecer comunicación con la BD, recuperar información.
Precondición	Servidor en ejecución y conexión a BD establecida
Condición de terminación	Datos recuperados
Descripción	En esta conversación el CustomerService realiza la solicitud de datos al agente DataManager, el cual convierte esa comunicación en consultas SQL para la base de datos, ejecuta la consulta en la base de datos y retorna la información recuperada en formato XML

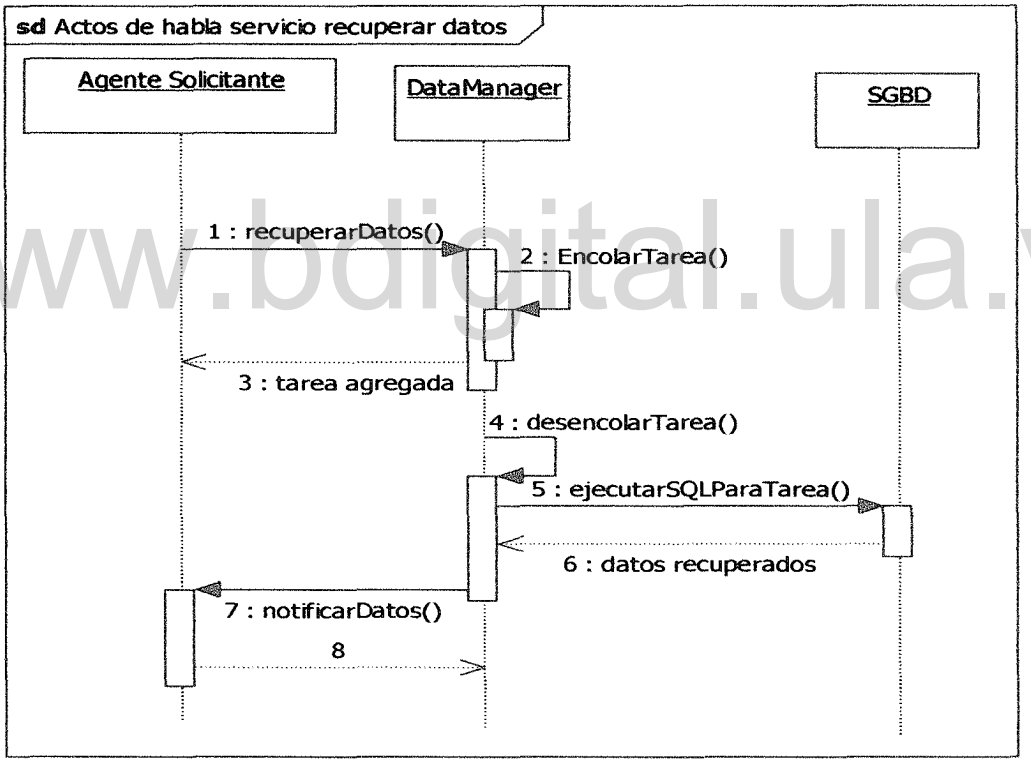

Figura 3.22. Actos de habla para el servicio recuperar datos del agente DataManager

3.1.3.7. Modelo de Agente para el Agente CustomerService

El modelo de Agente para el CustomerService se muestra en la Tabla 3.26.

Tabla 3.26. Modelo de Agente para el Agente CustomerService

Agente		
Nombre		Agente CustomerService
Posición		SMA lado del servidor
Componentes		No aplica
Marco de referencia		No aplica
Descripción del agente		Se encarga de atender las solicitudes de datos y sincronización realizadas por los agentes en el cliente; en sí, este agente representa una interfaz de comunicación entre los agentes del lado del servidor y los agentes del lado del cliente, por esa razón existen un agente CustomerService por cada usuario que se conecta al AVD.
Objetivos del Agente		
Nombre		Comunicar cliente y servidor
Descripción		Servir de interfaz entre los agentes del lado del cliente y los agentes del lado del servidor
Parámetro de entrada		Información a recuperar o sincronizar
Parámetro de salida		Información recuperada o sincronizada
Condición de activación		Recepción de solicitud de datos o sincronización
Condición de finalización		Solicitud procesada
Condición de éxito		Datos recuperados o propagados
Condición de fracaso		Datos no existentes
Ontología		Ontología comunicación entre agentes
Servicios del Agente		
Nombre		Procesar acción
Descripción del Servicio		Recibe la acción desde el cliente y la procesa. Si es una solicitud de datos pide los mimos al agente DataManager para posteriormente enviar la información al cliente, si es una solicitud de sincronización de acción delega el trabajo al agente Synchronization
Tipo de Servicio		Dual
Parámetros de entrada		Acción a procesar
Parámetros de salida		Datos recuperados o sincronizados
Propiedades del Servicio		
Nombre	**Valor**	**Descripción**
Calidad	0-100	Porcentaje de la calidad del servicio en función del tiempo de respuesta
Auditable	1	Capacidad de diagnosticar la calidad del servicio
Confiabilidad	0-100	Certificación de respuesta
Capacidad del Agente		
Habilidades del agente		Capacidad de comunicación en red.
Rep. del Conocimiento		No Aplica
Lenguaje de Comunicación		El agente se comunica por medio de los métodos de la clase, XML y funciones para comunicación en red.

3.1.3.8. Modelo de Tareas para el Agente CustomerService

El conjunto de tareas que debe llevar a cabo el agente para cumplir con su objetivo se muestran en la Tabla 3.27

Tabla 3.27. Modelo de tareas para el agente CustomerService

Servicios	Tareas
Procesar acción	T1. Recibir solicitud para procesar acción T2. Definir el tipo de acción requerida T3. Si la acción es de sincronización delegar la función al agente de sincronización T4. Si la acción es solicitud de datos, pedir datos al DataManager T5. Enviar Datos al cliente

3.1.3.9. Modelo de Coordinación y Comunicación para el Agente de CustomerService

El modelo de coordinación del agente CustomerService para el servicio procesar acción se observa en la Tabla 3.28 y el diagrama de secuencias que describe los actos de habla en la Figura 3.23.

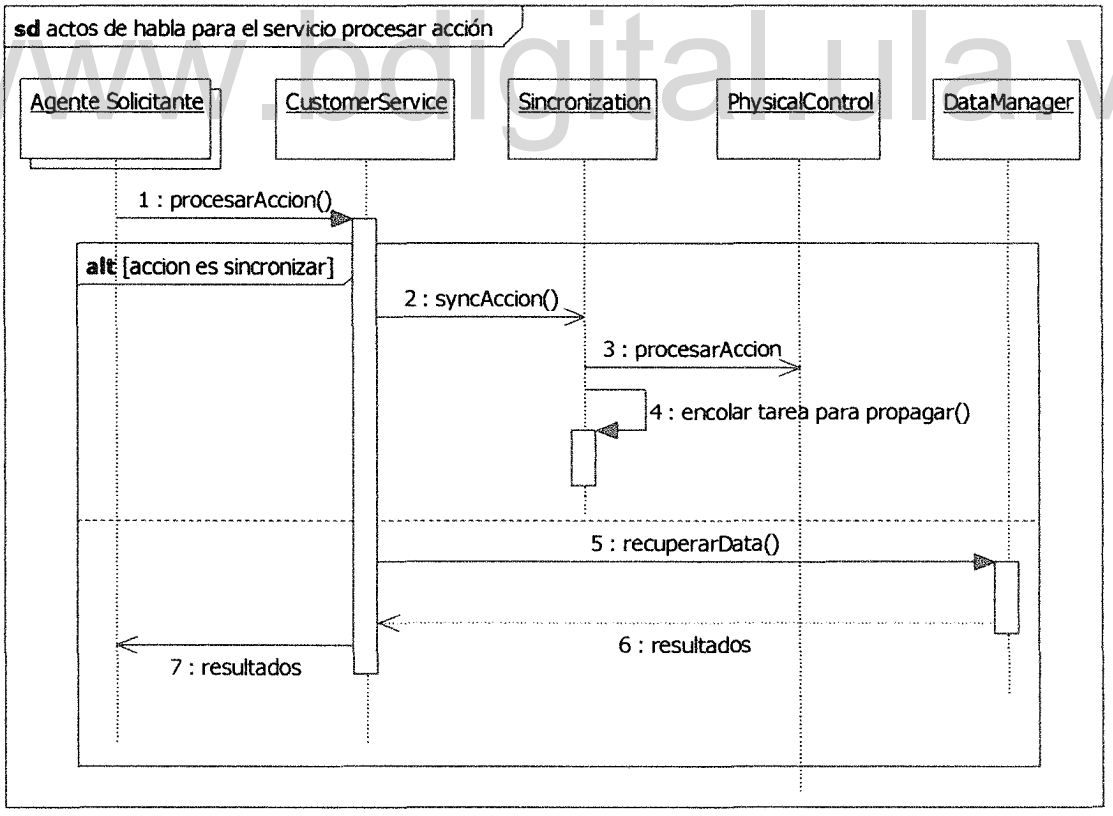

Figura 3.23. Actos de habla para el servicio procesar acción del agente CustomerService

Tabla 3.28. Modelo de conversación del Agente CustomerService servicio procesar acción

CONVERSACIÓN: Solicitud procesar acción	
Objetivo	Recuperar datos desde el agente DataManager y enviarlos al agente solicitante en el cliente, Si la acción es de sincronización delegar la función al Agente Synchronization
Agentes participantes	Agente CustomerService, Agente DataManager, Agente Repository, Agente Visualization, Agente Synchronization
Iniciador	Agente Visualization, Agente Repository o Agente Bootstrapping
Actos de habla	Realizar solicitud, revisar tipo de acción, delegar acción, recuperar datos, reenviar infomación.
Precondición	Usuario conectado
Condición de terminación	Datos recuperados
Descripción	Esta conversación es iniciada por alguno de los agente en el lado del cliente, el CustomerService delegara funciones al agente Synchronization de ser el caso o pedirá datos al agente DataManager para posteriormente dar respuesta al agente solicitante.

3.1.3.10. Modelo del Agente Synchronization

El modelo del agente para el Agente de Synchronization se especifica en la Tabla 3.30, así como su objetivo principal y servicios que ofrece

3.1.3.11. Modelo de Tareas para el Agente de Synchronization

Las tareas que debe realizar el agente para llevar a cabo el proceso de sincronización se lista en la Tabla 3.29 para el servicio sincronizar datos.

Tabla 3.29. Modelo de tareas para el agente Synchronization

Servicios	Tareas
Procesar acción	T1. Recibir solicitud de sincronización T2. Agregar la solicitud a la cola de tareas T3. Extraer la solicitud de la cola T4. Enviar datos a ser sincronizados a los agentes Receiver de cada cliente. T5. Extraer siguiente solicitud de sincronización T6. Ejecutar la tarea T4 mientras hayan datos que sincronizar en la cola de tareas.

Tabla 3.30. Modelo de Agente para el Agente Synchronization

Agente		
Nombre		Agente Synchronization
Posición		SMA lado del servidor
Componentes		No aplica
Marco de referencia		No aplica
Descripción del agente		El agente se encarga de propagar las acciones iniciadas por un usuario hacia los demás usuarios presentes en el AVD.
Objetivos del Agente		
Nombre		Sincronizar información
Descripción		Sincronizar las acciones entre los usuarios del AVD
Parámetro de entrada		Acciones a sincronizar
Parámetro de salida		Ninguno
Condición de activación		Recepción de solicitud de sincronización
Condición de finalización		Solicitud procesada
Condición de éxito		Datos propagados a los clientes
Condición de fracaso		Datos no pueden ser enviados a los clientes
Ontología		Ontología comunicación entre agentes y sincronización de información
Servicios del Agente		
Nombre		Sincronizar datos
Descripción del Servicio		Recibe la acción a ser sincronizada desde alguno de los clientes y propaga esas acciones a los demás usuarios del AVD, permitiendo que el AVD se mantenga sincronizado
Tipo de Servicio		Dual
Parámetros de entrada		Acción a sincronizar
Parámetros de salida		Ninguno
Propiedades del Servicio		
Nombre	**Valor**	**Descripción**
Calidad	0-100	Porcentaje de la calidad del servicio en función del tiempo de respuesta
Auditable	1	Capacidad de diagnosticar la calidad del servicio
Confiabilidad	0-100	Certificación de respuesta
Capacidad del Agente		
Habilidades del agente		Capacidad de comunicación en red y sincronización de información
Rep. del Conocimiento		No Aplica
Lenguaje de Comunicación		El agente se comunica por medio de los métodos de la clase y funciones para comunicación en red.

3.1.3.12. Modelo de Coordinación y Comunicación para el Agente de Synchronization

El modelo de coordinación del agente Synchronization para el servicio sincronizar datos se observa en la Tabla 3.31 y el diagrama de secuencias que describe los actos de habla en la Figura 3.24.

Tabla 3.31. Modelo de conversación del Agente Synchronization, servicio sincronizar datos

CONVERSACIÓN: Solicitud sincronizar datos	
Objetivo	Propagar las acciones que deben ser sincronizadas por cada uno de los usuarios del Ambiente Virtual Dinámico.
Agentes participantes	Agente CustomerService, Agente Synchronization, Agente Broadcast, Agente Receiver
Iniciador	Agente CustomerService o Agente PhysicalControl
Actos de habla	Realizar solicitud, solicitar datos de la acción, propagar accion
Precondición	Usuario conectado
Condición de terminación	Datos propagados
Descripción	Esta conversación es iniciada por el CustomerService cuando recibe una acción de sincronización desde el cliente, CustomerService delega la acción al agente de Sincronización, quien obtendrá los datos a sincronizar para posteriormente propagar la acción hacia los demás clientes.

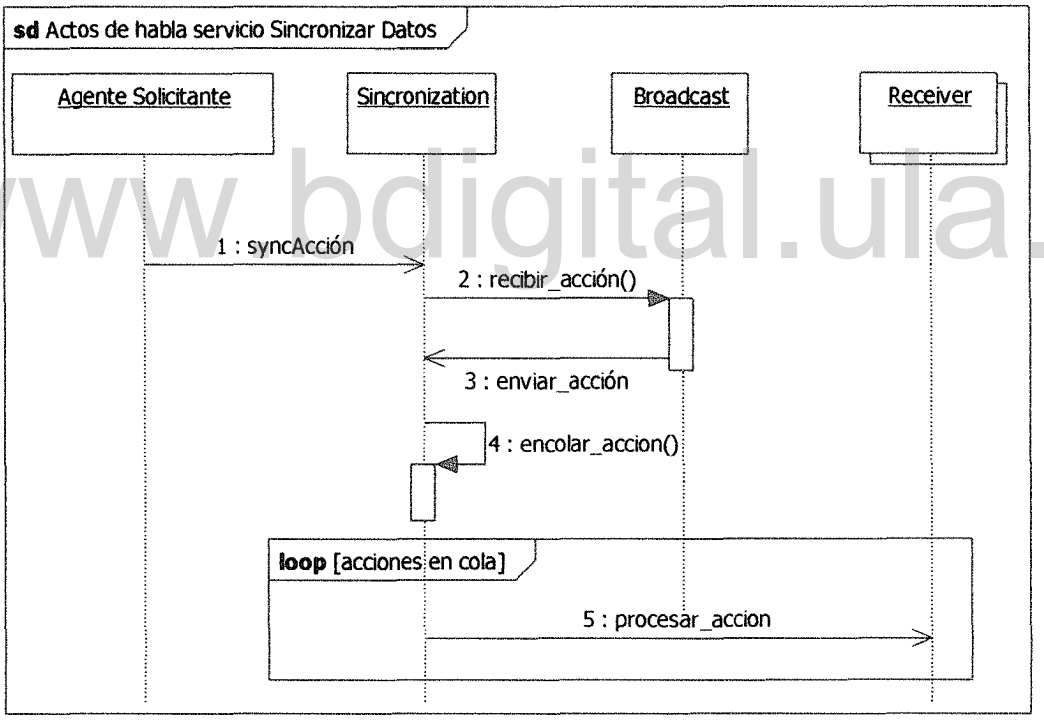

Figura 3.24. Actos de habla para el servicio sincronizar datos del agente Synchronization

3.1.3.13. Modelo del Agente para el Agente Repository

A continuación se describe el modelo de agente para el agente Repository.

Tabla 3.32. Modelo de Agente para el Agente Repository

Agente		
Nombre		Agente Repository
Posición		SMA lado del cliente
Componentes		No aplica
Marco de referencia		No aplica
Descripción del agente		El agente sirve como repositorio temporal de datos, manteniendo en una especie de cache los datos que traiga desde el servidor, a fin de agilizar la búsqueda de los datos, evitando los retrasos que pueda ocasionar el trafico de red al tener que buscar la información en el servidor central.
Objetivos del Agente		
Nombre		Servir de memoria cache
Descripción		Mantiene los datos de forma temporal en la memoria
Parámetro de entrada		Datos a localizar
Parámetro de salida		Datos localizados
Condición de activación		Solicitud de datos
Condición de finalización		Solicitud procesada
Condición de éxito		Datos localizados
Condición de fracaso		Datos no pueden ser localizados
Ontología		Ontología sincronización de información
Servicios del Agente		
Nombre		Recuperar datos
Descripción del Servicio		Se encarga de localizar los datos para los agentes solicitantes, ya sea que tenga que buscarlos en el servidor o en su propia memoria.
Tipo de Servicio		Dual
Parámetros de entrada		Datos a recuperar
Parámetros de salida		Datos recuperados
Propiedades del Servicio		
Nombre	**Valor**	**Descripción**
Calidad	0-100	Porcentaje de la calidad del servicio en función del tiempo de respuesta
Auditable	1	Capacidad de diagnosticar la calidad del servicio
Confiabilidad	0-100	Certificación de respuesta
Capacidad del Agente		
Habilidades del agente		Capacidad de memorizar y sincronizar información
Rep. del Conocimiento		No Aplica
Lenguaje de Comunicación		El agente se comunica por medio de los métodos de la clase y funciones para comunicación en red.

3.1.3.14. Modelo de Tareas para el Agente de Repository

El modelo de tareas para el Agente Repository se resume en la tabla Tabla 3.33, allí se muestran las tres tareas que lleva a cabo el agente para cumplir con su función.

Tabla 3.33. Modelo de tareas para el agente Repository

Servicios	Tareas
Recuperar datos	T1. Recibir solicitud de recuperación de datos T2. Recuperar datos desde el servidor T3. Recuperar datos desde la memoria temporal

3.1.3.15. Modelo de Coordinación y Comunicación para el Agente de Repository

En la Tabla 3.34 se muestra el modelo de coordinación del agente Repository para el servicio recuperar datos y el diagrama de secuencias que describe los actos de habla se observa en la Figura 3.25

Tabla 3.34. Modelo de conversación del Agente Repository, servicio recuperar datos.

CONVERSACIÓN: Solicitud recuperar datos	
Objetivo	Recuperar y mantener una cache de los datos requeridos por el Agente de visualización del lado del cliente.
Agentes participantes	Agente CustomerService, Agente Repository, Agente Visualization
Iniciador	Agente Visualization
Actos de habla	Realizar solicitud, enviar datos, buscar datos en cache
Precondición	Usuario conectado
Condición de terminación	Datos recuperados
Descripción	La Conversación la inicia el Agente Visualization, para solicitar información referente a los objetos y avatares presentes en un ambiente virtual dinámico.

3.1.3.16. Modelo de Agente para el Agente Visualization

El modelo del agente para el Agente Visualization se especifica en la Tabla 3.36, así como su objetivo principal y servicios que ofrece.

3.1.3.17. Modelo de Tareas para el Agente de Visualization

Las tareas que debe realizar el agente para llevar a cabo las órdenes emitidas por el usuario se listan en la Tabla 3.35 para el servicio procesar órdenes de usuario.

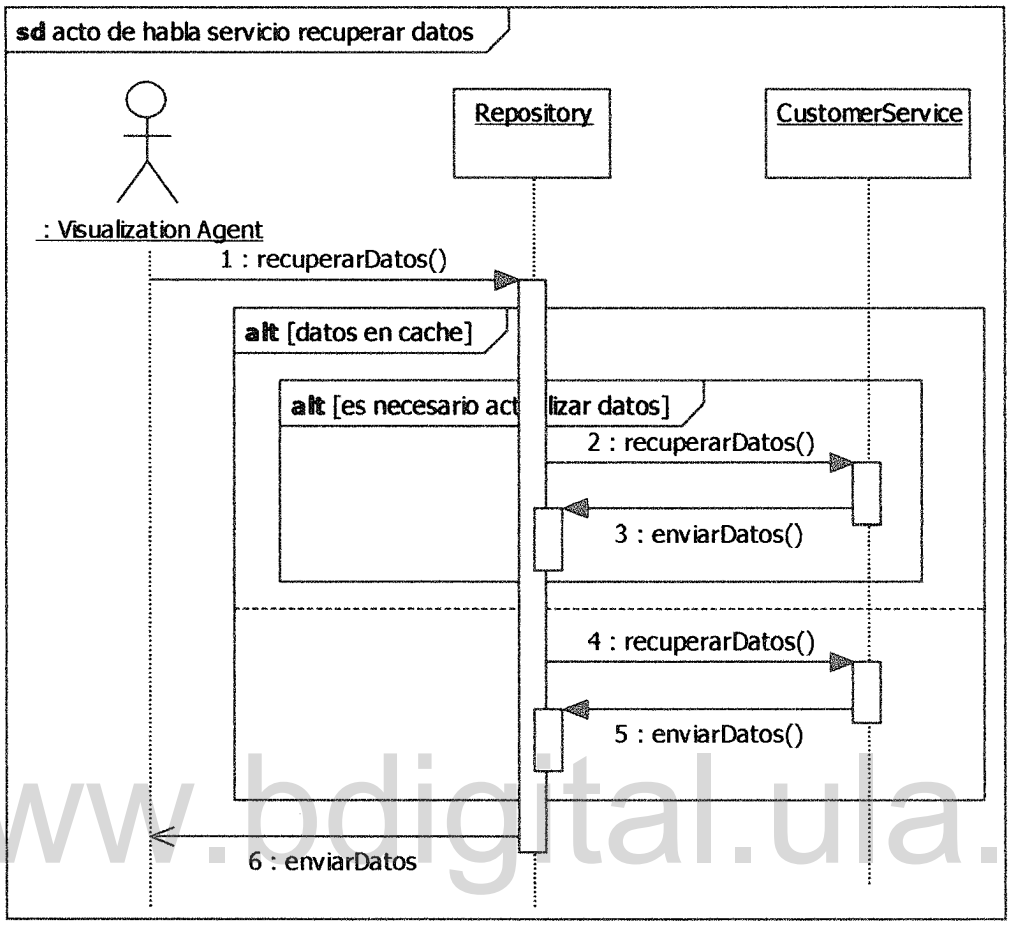

Figura 3.25. Actos de habla servicio recuperar datos del agente Repository

Tabla 3.35. Modelo de tareas para el agente Visualization

Servicios	Tareas
Procesar ordenes de usuario	T1. Recibir orden del usuario T2. Propagar la orden al agente Broadcast

3.1.3.18. Modelo de Coordinación y Comunicación para el Agente de Visualization

El modelo de coordinación del agente para el servicio procesar ordenes de usuario se observa en la Tabla 3.37 y el diagrama de secuencias de la Figura 3.26 describe los actos de habla que se producen en la comunicación.

Tabla 3.36. Modelo de Agente para el Agente Visualization

Agente		
Nombre		Agente Visualization
Posición		SMA lado del cliente
Componentes		No aplica
Marco de referencia		No aplica
Descripción del agente		El agente se encarga de recibir las órdenes del usuario y remitirlas al agente PhysicalControl para que se encargue de ejecutarlas. A la vez, el agente debe repintar la escena para actualizar los cambios que hayan en la misma.
Objetivos del Agente		
Nombre		Recibir ordenes
Descripción		Recibe las ordenes del usuario
Parámetro de entrada		Orden a efectuar
Parámetro de salida		Ninguno
Condición de activación		Orden por parte del usuario
Condición de finalización		Orden remitida al agente Broadcast
Condición de éxito		Orden remitida al agente Broadcast
Condición de fracaso		Orden no se puede remitir
Ontología		Ontología obtención de ordenes
Servicios del Agente		
Nombre		Procesar ordenes de usuario
Descripción del Servicio		Se encarga de recibir las órdenes del usuario y ponerlas a disposición del agente Broadcast para que sean propagadas
Tipo de Servicio		Dual
Parámetros de entrada		Orden a ejecutar
Parámetros de salida		Ninguno
Propiedades del Servicio		
Nombre	**Valor**	**Descripción**
Calidad	0-100	Porcentaje de la calidad del servicio en función del tiempo de respuesta
Auditable	1	Capacidad de diagnosticar la calidad del servicio
Confiabilidad	0-100	Certificación de respuesta
Capacidad del Agente		
Habilidades del agente		Capacidad para atender ordenes
Rep. del Conocimiento		No Aplica
Lenguaje de Comunicación		El agente se comunica por medio de los métodos de la clase y eventos.

3.1.3.19. Modelo de Agente para el Agente PhysicalControl

En la Tabla 3.39 se puede observar el resumen del modelo de agente para el Agente PhysicalControl, allí mismo se especifica el objetivo principal que tiene el agente, así como el único servicio que ofrece.

Tabla 3.37. Modelo de conversación del Agente Visualization, servicio procesar órdenes.

CONVERSACIÓN: Solicitud procesar ordenes de usuario	
Objetivo	Atender las ordenes del usuario
Agentes participantes	Agente Visualization, Agente Broadcast
Iniciador	Usuario
Actos de habla	Recibir datos, procesar datos, enviar datos
Precondición	Usuario conectado
Condición de terminación	Orden remitida
Descripción	La conversación tiene lugar cuando un usuario desea dar una orden al avatar que controla.

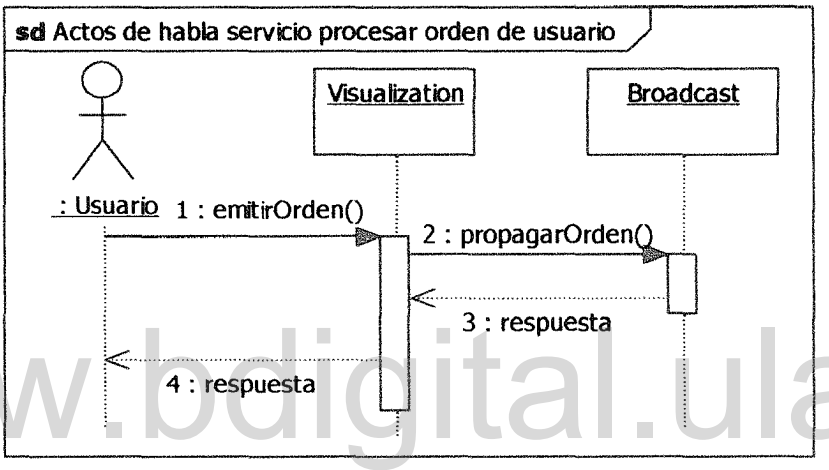

Figura 3.26. Actos de habla servicio procesar ordenes de usuario del agente Visualization

3.1.3.20. Modelo de Tareas para el Agente PhysicalControl

La Tabla 3.38 muestra la lista de tareas que ejecuta el agente PhysicalControl para cumplir con sus objetivos.

3.1.3.21. Modelo de Coordinación y Comunicación para el Agente PhysicalControl

El modelo de coordinación del agente se observa en la Tabla 3.40. El diagrama de secuencias que muestra el desarrollo de los actos de habla se puede observar en la Figura 3.27.

Tabla 3.38. Modelo de tareas para el agente PhysicalControl

Servicios	Tareas
Simular acción	T1. Recibir acción a simular T2. Preparar datos de acuerdo a la acción T3. Verificar si la acción es físicamente posible T4. Calcular trayectorias y puntos de contacto T5. Agregar la acción a la simulación física. T6. Actualizar la simulación física T7. Solicitar sincronizar nueva acción

Tabla 3.39. Modelo de Agente para el Agente PhysicalControl

Agente		
Nombre		Agente PhysicalControl
Posición		SMA lado del cliente
Componentes		No aplica
Marco de referencia		No aplica
Descripción del agente		El agente se ocupa de agregar nuevas acciones a las simulaciones físicas en caso de ser posibles, también se encarga de actualizar las simulaciones físicas actuales y de detectar las colisiones entre los objetos y avatares presentes en el AVD.
Objetivos del Agente		
Nombre		Agregar acción física
Descripción		Agrega una nueva acción física a la simulación controlada por el agente.
Parámetro de entrada		Acción a agregar
Parámetro de salida		Ninguno
Condición de activación		Solicitud por parte del agente Synchronization
Condición de finalización		Acción agregada a la simulación
Condición de éxito		Acción agregada a la simulación
Condición de fracaso		Acción no es posible físicamente
Ontología		Ontología simulación física
Servicios del Agente		
Nombre		Simular acción
Descripción del Servicio		Simula físicamente una acción iniciada por el usuario sobre su avatar
Tipo de Servicio		Dual
Parámetros de entrada		Orden a ejecutar
Parámetros de salida		Ninguno
Propiedades del Servicio		
Nombre	**Valor**	**Descripción**
Calidad	0-100	Porcentaje de la calidad del servicio en función del tiempo de respuesta
Auditable	1	Capacidad de diagnosticar la calidad del servicio
Confiabilidad	0-100	Certificación de respuesta
Capacidad del Agente		
Habilidades del agente		Capacidad para simular acciones físicas
Rep. del Conocimiento		No Aplica
Lenguaje de Comunicación		El agente se comunica por medio de los métodos de la clase y XML.

Tabla 3.40. Modelo de conversación del Agente PhysicalControl, servicio simular acción.

CONVERSACIÓN: Solicitud simular acción	
Objetivo	Agregar nuevas acciones a la simulación física
Agentes participantes	Agente Synchronization, Agente PhysicalControl
Iniciador	Agente Synchronization
Actos de habla	Recibir acción, simular acción, sincronizar acción
Precondición	Usuario conectado
Condición de terminación	Acción añadida a la simulación
Descripción	La conversación inicia cuando el Agente Synchronization remite la acción para que el agente PhysicalControl se encargue de agregarla a la simulación física. Una vez añadida la acción el agente la remite al agente Synchronization para que sea sincronizada.

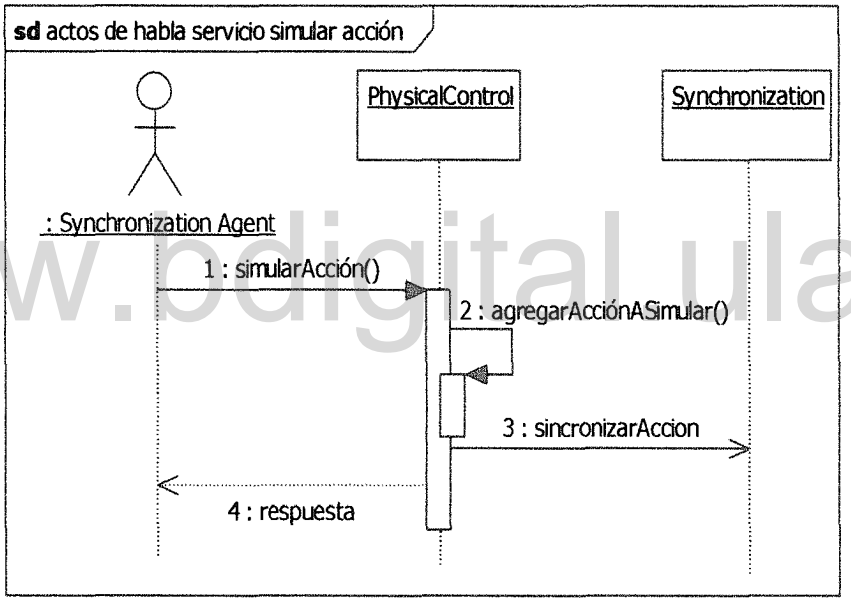

Figura 3.27. Actos de habla servicio simular acción del Agente PhysicalControl

3.1.3.22. Modelo de Agente para el Agente Broadcast

En la Tabla 3.41 se puede observar el resumen del modelo de agente para el Agente Broadcast, donde se especifica la descripción del agente, su objetivo principal, así como el único servicio que ofrece y demás propiedades del agente en estudio.

Tabla 3.41. Modelo de Agente para el Agente Broadcast

Agente		
Nombre		Agente Broadcast
Posición		SMA lado del cliente
Componentes		No aplica
Marco de referencia		No aplica
Descripción del agente		El agente se encarga de enviar las acciones hacia el servidor para que el agente Synchronization se encargue de propagarlas a los demás clientes del AVD.
Objetivos del Agente		
Nombre		Remitir acciones
Descripción		Remite las acciones iniciadas por el avatar del usuario local hacia el servidor, para que sean propagadas a los demás usuarios del Ambiente Virtual Dinámico.
Parámetro de entrada		Acciones a sincronizar
Parámetro de salida		Ninguno
Condición de activación		Solicitud por parte del agente Visualization
Condición de finalización		Acción remitida al servidor
Condición de éxito		Acción remitida al servidor
Condición de fracaso		No se puede establecer la comunicación con el servidor
Ontología		Ontología comunicación en red
Servicios del Agente		
Nombre		Informar acciones de usuario
Descripción del Servicio		Se encarga de recibir las solicitudes del agente Visualization y remitirlas al servidor para que sean propagadas a los demás clientes, de tal forma que la información sea sincronizada.
Tipo de Servicio		Dual
Parámetros de entrada		Acción a sincronizar
Parámetros de salida		Ninguno
Propiedades del Servicio		
Nombre	**Valor**	**Descripción**
Calidad	0-100	Porcentaje de la calidad del servicio en función del tiempo de respuesta
Auditable	1	Capacidad de diagnosticar la calidad del servicio
Confiabilidad	0-100	Certificación de respuesta
Capacidad del Agente		
Habilidades del agente		Capacidad para comunicación en red
Rep. del Conocimiento		No Aplica
Lenguaje de Comunicación		El agente se comunica por medio de los métodos de la clase y funciones de comunicación en red.

3.1.3.23. Modelo de Tareas para el Agente Broadcast

La Tabla 3.42 muestra la lista de tareas que ejecuta el agente Broadcast para cumplir con sus objetivos.

Tabla 3.42. Modelo de tareas para el agente Broadcast

Servicios	Tareas
Informar acciones de usuario	T1. Recibir acción a informar T2. Preparar datos para ser enviados al servidor T3. Informar al servidor la solicitud T4. Enviar datos al servidor

3.1.3.24. Modelo de Coordinación y Comunicación para el Agente Broadcast

El modelo de coordinación del agente se observa en la Tabla 3.43. El diagrama de secuencias que muestra el desarrollo de los actos de habla para el servicio Informar acciones de usuario se puede observar en la Figura 3.28.

Tabla 3.43. Modelo de conversación del Agente Broadcast, servicio Informar acciones.

CONVERSACIÓN: Solicitud Informar acciones de usuario	
Objetivo	Informar al servidor las acciones emprendidas por el usuario para que sean sincronizadas con los demás clientes del AVD.
Agentes participantes	Agente Visualization, Agente Broadcast, Agente CustomerService, Agente Synchronization
Iniciador	Agente Visualization
Actos de habla	Recibir acción, remitir acción, sincronizar acción
Precondición	Usuario conectado
Condición de terminación	Acción remitida al servidor
Descripción	La conversación tiene lugar cuando el agente Visualization recibe la solicitud para ejecutar una acción por parte del usuario, en ese momento debe avisar al agente Broadcast para que remita la información al servidor del tal forma que se pueda sincronizar con los demás usuarios.

3.1.3.25. Modelo de Agente para el Agente Receiver

El modelo de agente para el Agente Receiver se especifica en la Tabla 3.44, así como su objetivo principal, servicios que ofrece, habilidades y demás características del agente.

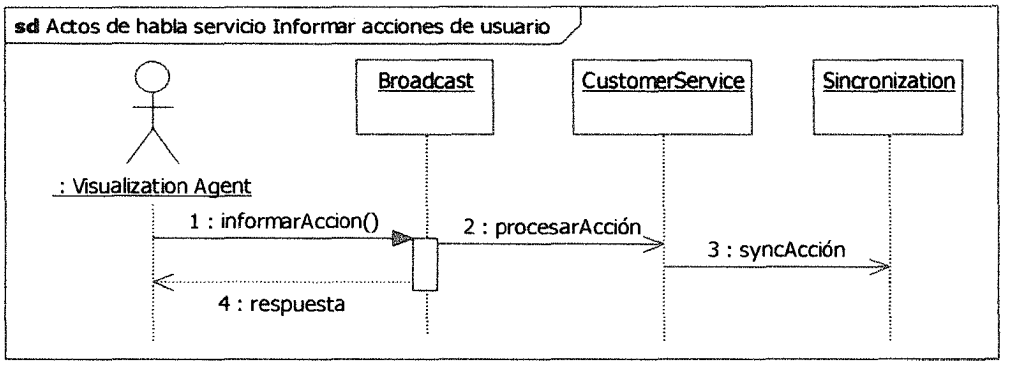

Figura 3.28. Actos de Habla Informar acciones de usuario agente Broadcast

Tabla 3.44. Modelo de Agente para el Agente Receiver

Agente		
Nombre		Agente Receiver
Posición		SMA lado del cliente
Componentes		No aplica
Marco de referencia		No aplica
Descripción del agente		El agente Receiver es el encargado de recibir las acciones iniciadas por otros usuarios del sistema y que deben ser sincronizadas.
Objetivos del Agente		
Nombre		Sincronizar acciones externas
Descripción		Sincroniza las acciones que han sido iniciadas en el AVD por otros usuarios.
Parámetro de entrada		Acciones a sincronizar
Parámetro de salida		Ninguno
Condición de activación		Solicitud por parte del agente Synchronization
Condición de finalización		Acción recibida
Condición de éxito		Acción sincronizada
Condición de fracaso		No se puede sincronizar la acción
Ontología		Ontología comunicación en red y sincronización
Servicios del Agente		
Nombre		Procesar acciones a sincronizar
Descripción del Servicio		Recibe las acciones iniciadas por otros usuarios del AVD.
Tipo de Servicio		Dual
Parámetros de entrada		Acción a sincronizar
Parámetros de salida		Ninguno
Propiedades del Servicio		
Nombre	**Valor**	**Descripción**
Calidad	0-100	Porcentaje de la calidad del servicio en función del tiempo de respuesta
Auditable	1	Capacidad de diagnosticar la calidad del servicio
Confiabilidad	0-100	Certificación de respuesta
Capacidad del Agente		
Habilidades del agente		Capacidad para comunicación en red y sincronizar información
Rep. del Conocimiento		No Aplica
Lenguaje de Comunicación		El agente se comunica por medio de los métodos de la clase y funciones de comunicación en red.

3.1.3.26. Modelo de Tareas para el Agente Receiver

La lista de tareas que ejecutadas por el agente Receiver se observa en la Tabla 3.45.

Tabla 3.45. Modelo de tareas para el agente Receiver

Servicios	Tareas
Procesar acciones a sincronizar	T1. Recibir acciones T2. Informar acciones

3.1.3.27. Modelo de Coordinación y Comunicación para el Agente Receiver

En la Tabla 3.46 se muestra el modelo de coordinación del agente Receiver. El diagrama de secuencias que describe los actos de habla para el servicio Procesar acciones a sincronizar se puede observar en la Figura 3.29.

Tabla 3.46. Modelo de conversación del Agente Receiver, servicio procesar acciones.

CONVERSACIÓN: Solicitud Procesar acciones a sincronizar	
Objetivo	Poner en ejecución local las acciones iniciadas por otros usuarios del ambiente virtual dinámico
Agentes participantes	Agente Visualization, Agente Synchronization, Agente Receiver
Iniciador	Agente Synchronization
Actos de habla	Recibir acción, ejecutar acción
Precondición	Usuario conectado
Condición de terminación	Acción iniciada localmente
Descripción	La conversación inicia cuando el Agente Synchronization solicita sincronizar las acciones iniciadas por otros usuarios del sistema, para lo cual el agente Receiver debe informar de este hecho al Agente Visualization.

3.1.4. Fase III: Diseño

En esta fase se presenta el diagrama de clases donde se detallan las relaciones y asociaciones que existen entre los diferentes objetos que componen el Sistema Multi Agentes. Este diagrama se genera a partir de los componentes del SMA detallados en la fase de análisis.

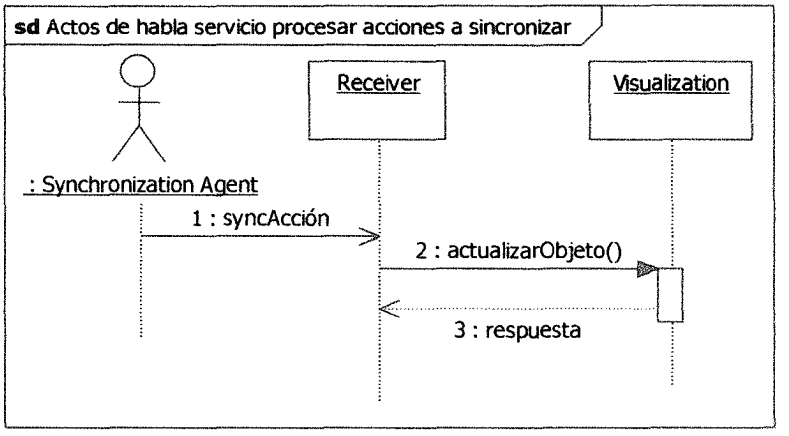

Figura 3.29. Actos de habla servicio procesar acciones a sincronizar agente Receiver

En la Figura 3.30 y la Figura 3.31 se observa el diagrama de clases de los Agentes del SMA, donde se pueden apreciar las relaciones existentes entre los mismos, todos los agentes heredan de la clase abstracta Thread y deben sobrescribir los métodos abstractos presentes en esa clase.

Por motivos de espacio fue necesario dividir el diagrama de clases SMA en dos partes, ya que el mismo no cabe en una sola página, y no se podrían observar los detalles del diagrama.

La metodología MASINA propone el uso de TDSO para describir las clases del sistema, sin embargo, en esta investigación se omitirá el uso de esta herramienta, ya que la dinámica de las clases se puede observar en los diagramas de secuencia antes descritos y el diagrama de clases de la Figura 3.30.

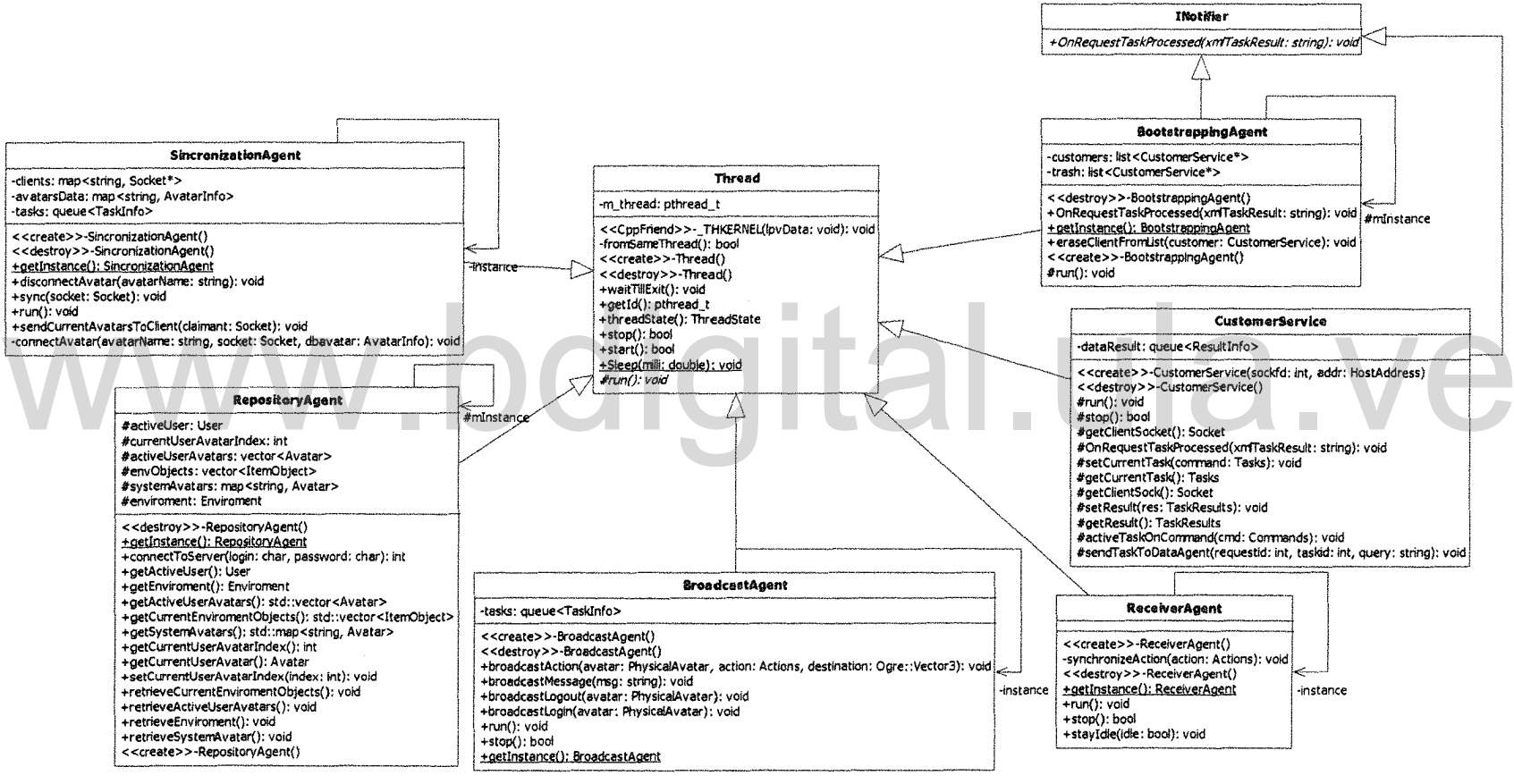

Figura 3.30. Diagrama de clases del Sitema Multi-Agentes Parte 1/2

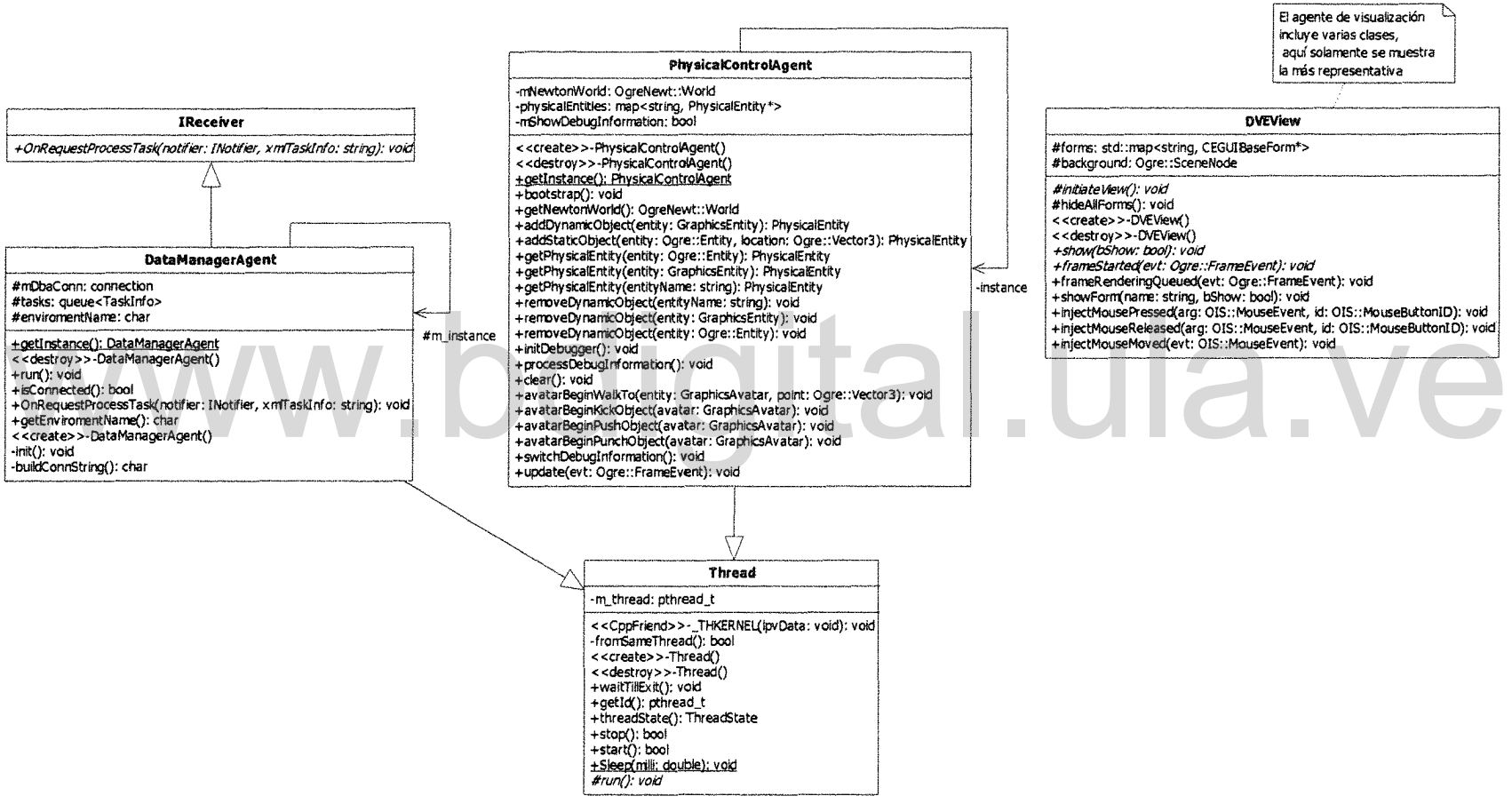

Figura 3.31. Diagrama de clases para el sistema multi-agentes Parte 2/2

Para finalizar esta fase, se presenta en la Figura 3.32 el diagrama de distribución del sistema multiagentes, el cual sirve como resumen de los modelos desarrollados en la primera parte de este capítulo.

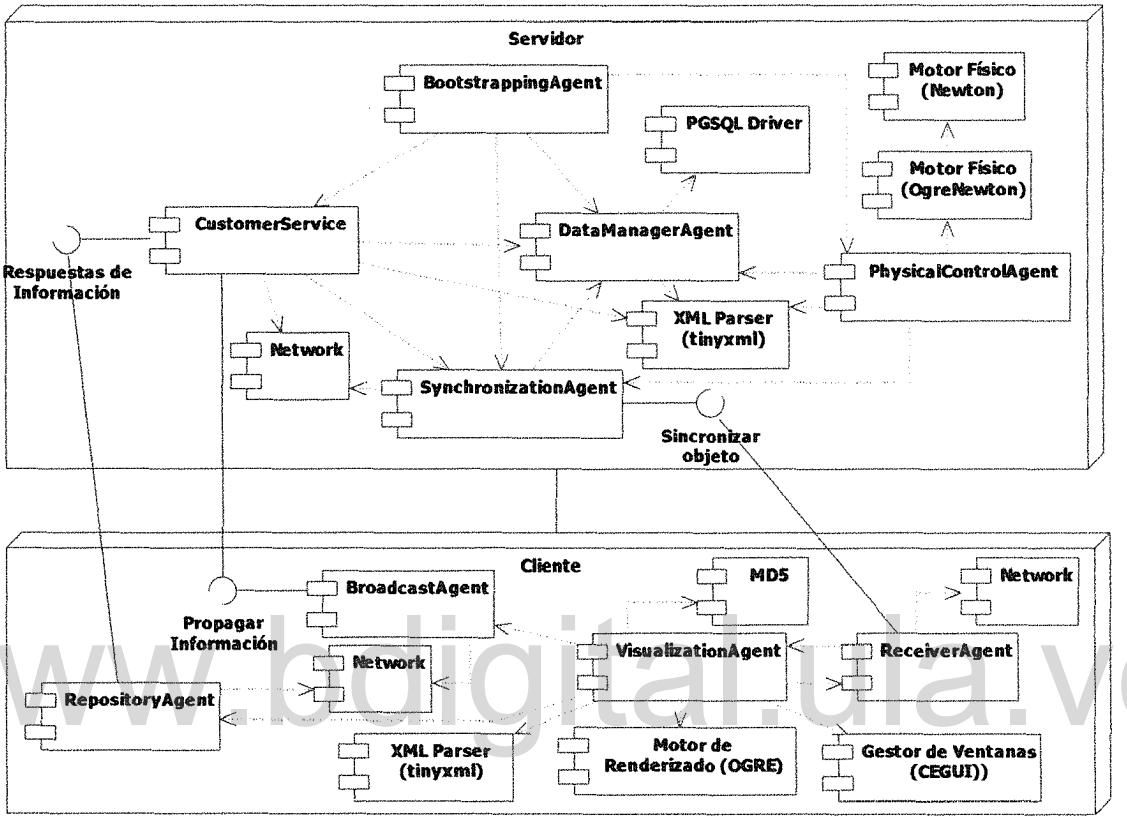

Figura 3.32. Diagrama de distribución del sistema multiagentes

En el diagrama anterior se observan dos nodos, el servidor y el cliente, y se detallan los componentes que existen en cada uno de ellos, se observa en el servidor los agentes Bootstrapping, CustormerService, Datamanager, Synchronization y PhysicalControl y en el cliente persisten los agentes Visualization, Repository, Receiver y Broadcast.

En este diagrama también se aprecia que el agente Bootstrapping se comunica con todos los demás agentes, para activarlos cuando sea requerido, y que el agente DataManager usa los componentes para comunicación con la base de datos y análisis de datos en formato XML.

Por otro lado, el agente de PhysicalControl depende de los componentes Motor Físico y análisis de datos en XML y se comunica con el Agente Synchronization y DataManager.

El Agente Synchronziation realiza solicitud de datos al agente DataManager y envía información de datos que deben ser sincronizados a los agentes Receiver en los clientes. De la misma manera el agente CustomerService da respuesta a las solicitudes de información realizadas por el agente Repository, y a la vez sirve como intermediario entre el agente Broadcast y el agente Synchronization cuando se requiere propagar una acción emprendida (patear, golpear, caminar, enviar mensaje) por el avatar de un usuario y que debe ser sincronizada a los demás clientes.

En el cliente, el agente de visualización es quien se encarga de activar a los demás agentes, por eso tiene relación con todos ellos, a la vez el agente Receiver una vez que recibe los datos de los objetos que deben ser sincronizados, se comunica con el agente Visualization, para que actualice la vista del usuario.

El agente de visualization usa los componentes para renderizar modelos 3d, un gestor de ventana que sirve de interfaz entre el usuario y el sistema y un analizador de datos en formatos XML.

El agente Broadcast, recibe las solicitudes acciones por parte del usuario (patear, golpear, empujar, caminar, enviar mensaje) y da parte al Agente CustomerService para que propague está información tanto al agente PhysicalControl como a los demás usuarios que se encuentren conectados al AVD.

Finalmente, el agente Repository es quien mantiene de manera local los datos del usuario conectado al sistema, así como los datos de los objetos y avatares, información que debe actualizar realizando solicitudes al agente CustomerService cuando sea requerido.

3.2. Segunda Parte: Diseño del AVD de Prueba

Para esta parte se hace uso de la metodología para el desarrollo de un ambiente virtual dinámico (MAVD) propuesta por (Hernández et al., 2010).

Esta metodología consta cuatro faces: Preproducción, Diseño, Producción y Post-Producción todas ellas vinculadas mediante la *Dirección* del lider del proyecto. Como primer paso se inicia por conformar el equipo de trabajo y designar el director o lider del proyecto.

A continuación se detalla el proceso metódico que se ha seguido a través de las fases de la métodología. En este capítulo se hace enfásis en las fases de Preproducción y Diseño, dejando las fases de Producción y Post-Producción para el Capítulo 4.

3.2.1. Dirección

De acuerdo con la metodología MAVD, el primer paso para emprender el desarrollo de un AVD es la conformación del equipo de trabajo y definir quien será el lider del proyecto.

En el presente proyecto, el equipo de trabajo está conformado por una sola persona (el investigador), por tal motivo, será éste quien a su vez se encargue de las funciones de director (lider) de desarrollo del proyecto, así como de fungir de desarrollador y usuario final (cliente) del sistema.

3.2.2. Preproducción

En esta fase se hace necesario describir los objetos, escenarios, así como la funcionalidad que tendrán los avatares en el ambiente virtual dinámico. En esta fase se deben cumplir varias actividades, las cuales se presentan a continuación:

3.2.2.1. Conceptualización de la aplicación

La primera tarea a realizar en esta actividad es la de analizar e identificar las ideas preliminares de los clientes. A continuación se presenta la lista de requisitos iniciales:

a) El sistema debe contener una vista, para iniciar sesión, otra para elegir el avatar con el que se desea ingresar al AVD y otra donde el avatar pueda interactuar con el AVD.
b) Se limitara el número de avatares por usuario a cinco, debido a que en la pantalla de inicio del usuario deben mostrarse todos sus avatares para que el mismo elija cual usará en el ambiente. para ingresar al AVD el usuario debe haber creado al menos un avatar.
c) El sistema debe ser una plataforma para crear ambientes virtuales simples y genericos.
d) El ambiente virtual creado con la herramienta debe contener una única escena donde se presentan y se desenvuelven los avatares y demás objetos del ambiente.
e) No existen una historia predefinida para el AVD, sino que más bien es un lugar de reunión para los usuarios que se conecten al AVD.
f) El ambiente virtual debe estar compuesto por un área plana que será tomada como el piso del ambiente.
g) Debe iniciar con un modelo de una estructura que posea escaleras, paredes, habitaciones, entre otros.
h) El administrador del sistema puede agregar otros objetos al AVD, como son: arboles, barriles, etc.
i) Los usuarios del AVD deben poder comunicarse por medio de un chat.
j) Los avatares del sistema deben tener la apariencia de una persona (hombre o mujer).
k) Los avatares deben interactuar con los objetos del ambiente mediante acciones de patear, empujar o golpear objetos.

l) Los avatares deben ser controlados mediante órdenes dadas con el ratón de la computadora, para caminar, golpear, patear y empujar.

m) Los avatares deben mostrar una animación cuando se ejecuten cada una de sus acciones.

n) Para la parte física, los objetos presentes en el AVD, deben contar con asignación de propiedades físicas para los materiales, de tal forma que se pueda obtener comportamientos diferentes para diversos materiales, ya que no es igual lanzar una pelota de goma, que lanzar una pelota de cuero.

o) La información sobre los objetos que deben estar presentes en el AVD, debe ser permanente, de tal forma que cada vez que sea iniciado el servidor del AVD, pinte los objetos que fueron agregados previamente.

Una vez generada la lista de requisitos se debe realizar la descripción de la evolución de los escenarios. El prototipo de prueba que se propone desarrollar en este capítulo contará con un único escenario, por tal motivo no existe evolución ni cambio de un escenario a otro.

La siguiente tarea a realizar, consiste en describir los objetos a crear en 3D. Inicialmente se contara con un conjunto reducido de objetos, los cuales pueden ser ampliados por el administrador del sistema. En la Tabla 3.47 se resumen los objetos que serán creados inicialmente.

3.2.2.2. Identificación de las interacciones del ambiente con los usuarios

A continuación se procede a identificar los roles de los distintos usuarios que intervienen en el AVD.

En el AVD existen dos roles fundamentales, el rol que cumple el usuario administrador y el rol que cumple el usuario cliente del ambiente virtual dinámico. La Figura 3.33 muestra los roles de los usuarios en el sistema.

Tabla 3.47. Descripción de los objetos 3D a crear

Objeto	Descripción
Piso	El piso del AVD estará representado por un plano ubicado en (0,0,0) y con una textura con apariencia de piedra.
Ambientes Interiores	Se pueden crear uno o dos ambientes interiores, que pueden ser casas o castillos.
Arbol	Se necesitan dos arboles, de tamaños y características diferentes.
Otros objetos	Se necesitarán algunos otros objetos como cajas, barriles, balones, etc. Para que el usuario pueda interactuar con ellos.

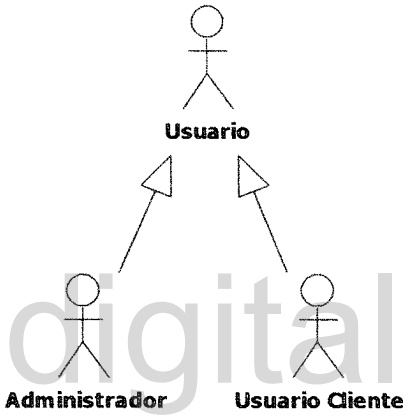

Figura 3.33. Modelo de actores del sistema

Otras de las tareas que se deben realizar en esta actividad es definir el prototipo funcional del sistema, eventos y relaciones con los actores. Sin embargo, esta descripción ya se ha realizado en la primera parte de este capítulo (Diseño de los Agentes), de tal forma que los diagramas de caso de uso y de actividades del sistema se pueden observar en la sección 3.1 de este capítulo, así como los eventos y relaciones entre los actores y el sistema.

A continuación se hace una caracterización de avatares. Para nuestro caso, y como se especificó en el documento de requisitos (ver sección 3.2.2.1), los avatares deben ser lo más similar a una persona; deben contar con animaciones de estado para cada acción específica del avatar, las acciones que puede realizar

un avatar son (quieto, caminar, golpear, patear, empujar y girar).

Con respecto a su vinculación con los usuarios, un usuario debe tener la capacidad de crear avatares y como se mencionó en el documento de requisitos un usuario solo puede crear cinco avatares, para que todos sean mostrados en la pantalla de selección de avatar al mismo tiempo. El usuario puede crear cualquier avatar disponible en el AVD; para ello, el sistema debe contar con un conjunto de avatares que puedan ser compartidos por todos los usuarios del sistema.

3.2.2.3. Identificar las entidades estáticas y dinámicas de las escenas del ambiente virtual

En la Tabla 3.48 se muestra un resumen de las entidades físicas del sistema. Estás entidades son la base inicial, y posteriormente el administrador del sistema podrá agregar nuevas entidades.

Tabla 3.48. Entidades del sistema y tipo de estructura física.

Entidad	Estática	Dinámica
Piso	☒	☐
Ambiente Interior	☒	☐
Arbol	☒	☐
Balones y bolas, caja, barriles y obstaculos	☐	☒
Avatar	☐	☒

3.2.2.4. Estudio de viabilidad y riesgos del AVD

Una de las primeras tareas a realizar en esta actividad es el análisis de las tecnologías y herramientas para la construcción de AVD. El investigador expone este estudio en el Capítulo 2, por esa razón omitiremos este paso de la metodología.

3.2.3. Diseño

De acuerdo con (Hernández et al, 2010) en esta fase se realiza el diseño del AVD bajo tres visiones fundamentales que son: *la visión geométrica y artística del ambiente virtual*, *la visión de interacción* y *la visión de diseño de software*. La metodología describe el conjunto de actividades que han de realizarse bajo cada visión.

Para la investigación algunas actividades no son representativas ya que no se está diseñando un ambiente virtual específico, sino más bien un ambiente virtual genérico que sirve de base para diseñar ambientes virtuales; por tal motivo, estás actividades serán omitidas. Ejemplo de ello, es la actividad: "Diseñar los planos de los escenarios del mundo"

3.2.3.1. Diseñar los objetos que forman parte de una escena

En esta actividad como primera tarea, se realizaron búsquedas de objetos tridimensionales en repositorios de objetos en Internet, encontrando en primer lugar los objetos que serán utilizados como avatares del sistema. En la Figura 3.34 se pueden observar los objetos conseguidos en esta primera búsqueda.

Continuando con la búsqueda de objetos, se pasó a revisar los modelos usados en los ejemplos que proporciona el motor físico Newton Game Dynamics, allí se consiguieron otros objetos que se presentan en la Figura 3.35.

Los objetos que se observan en la Figura 3.36 fueron recopilados del juego OpenFrag (OpenFrag, 2011).

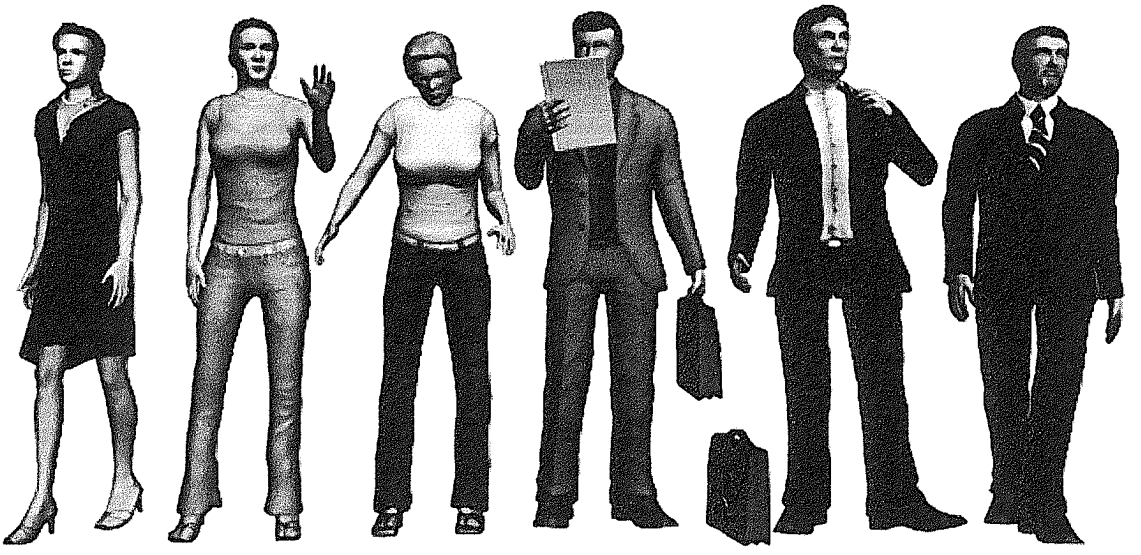

Figura 3.34. Render de algunos avatares que podrán ser usados en el sistema

Fuente: propia

Figura 3.35. Render de dos castillos extraídos de los ejemplos de Newton Game Dynamics

Fuente: propia

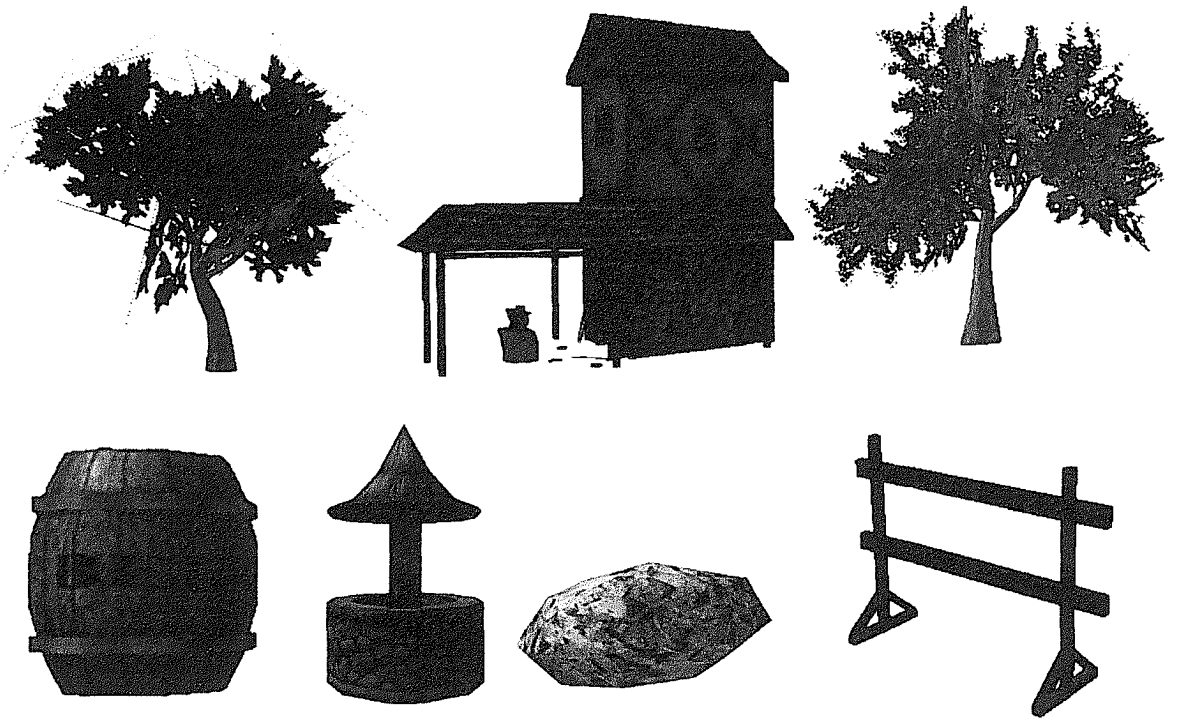

Figura 3.36. Diversos objetos 3D recuperados del juego software libre openfrag

Fuente: propia

Finalmente, los objetos que no se encontraron en la web fueron diseñados por el autor de esta investigación siguiendo la guía descrita por (Gahan, 2011) y (Luca de Tena, 2007). Estos objetos se muestran en la Figura 3.37.

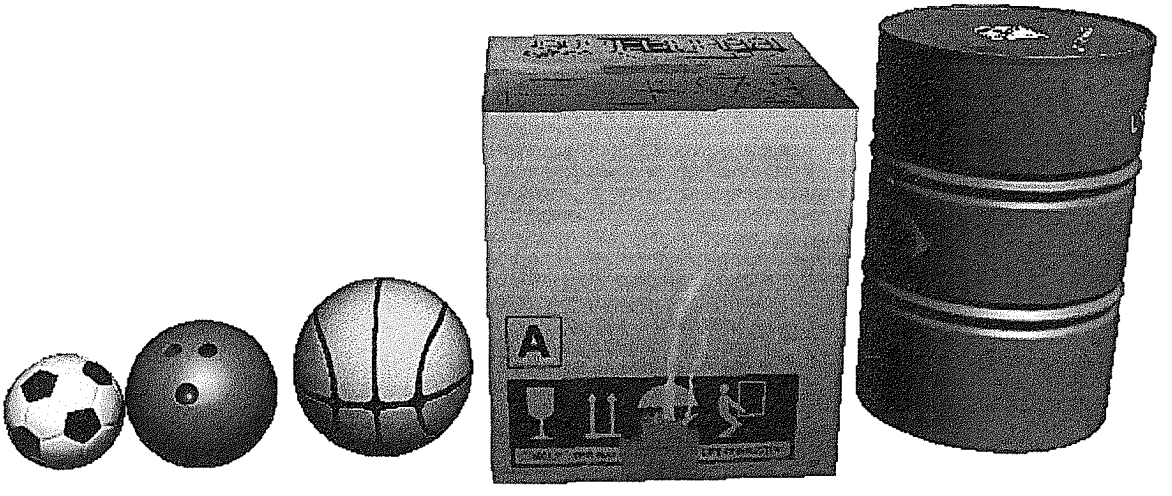

Figura 3.37. Otros objetos tridimensionales creados por el investigador

Fuente: propia

Una vez que se tienen los objetos, se pasó a realizar las animaciones en los avatares. Debido a que no se cuenta con los equipos para captura de movimientos, el investigador generó las animaciones cuadro por cuadro usando la técnica de animación esquelética y un editor tridimensional, tratando en lo posible, que el resultado de la animación se vea bastante natural. Posteriormente, los objetos y sus animaciones fueron exportados a un formato soportado por Ogre3D.

El conjunto de animaciones resultantes tienen un total de 2208 cuadros, es decir 2208 posiciones diferentes del avatar. Éstas animaciones fueron guardadas en un archivo de captura de movimiento (BIP) para que posteriormente puedan ser aplicadas a los demás avatares, reduciendo así el esfuerzo al animar todos los avatares. En la figura 3.37 se muestran algunas poses de las animaciones resultantes

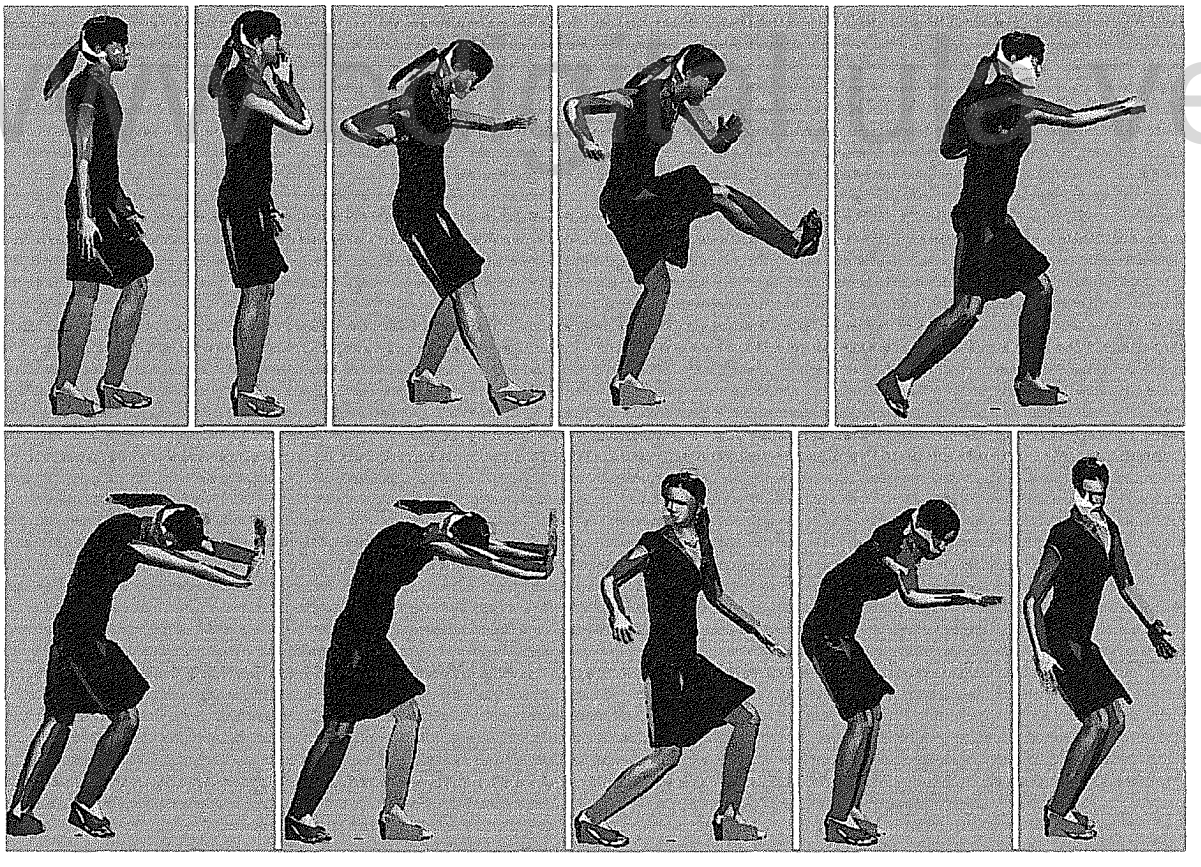

Figura 3.38. Poses de las animaciones realizadas en el avatar

Fuente: propia

Explicando la Figura 3.38, tenemos de izquierda a derecha y de arriba hacia abajo a) pose corresponde a la animación de caminar b) pose de la animación que se activa cuando el avatar está inactivo, la cual se denomina regularmente animación de ambiente c) pose de la animación de una patada corta d) pose de la animación de patada larga e) pose de la animación de un golpe directo f) pose de la animación de empujar g) pose de otra animación de empujar donde el avatar camina con ambas piernas h) pose de la animación de caminar hacia atrás i) pose de una segunda animación de ambiente j) pose de una tercera animación de ambiente.

3.2.3.2. Identificar eventos

Los eventos son activados por el usuario sobre el avatar, el cual puede iniciar una acción de caminar o alguna acción que cambie el estado físico de un objeto dinámico.

En la Figura 3.39 se muestra el diagrama de estados para el avatar y en la Figura 3.40 el diagrama de estados para los objetos dinámicos.

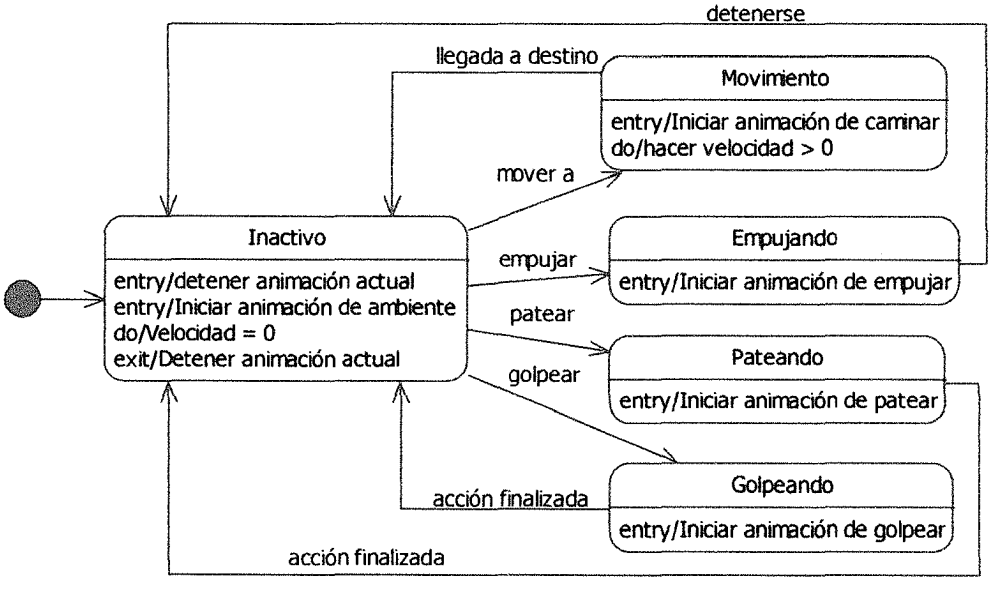

Figura 3.39. Diagrama de estados para el objeto avatar

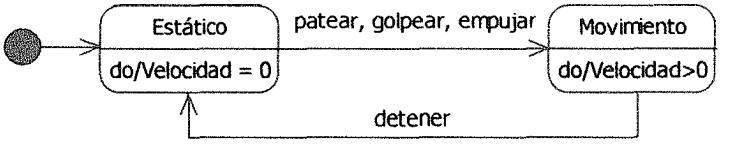

Figura 3.40. Diagrama de estados para los objetos dinámicos

3.2.3.3. Diseño de luces y cámaras

Como se ha mencionado en otras oportunidades, nuestro sistema es un software que permite crear ambientes virtuales pequeños y simples. En esta primera versión el sistema solo soporta una luz de ambiente, que da la sensación de estar en pleno día.

Con respecto a la cámara, el sistema desarrollado presenta una cámara en tercera persona, que se mueve a medida que el avatar controlado por el usuario cambia de posición. Ésta cámara puede alejarse o acercarse algunas unidades con respecto al avatar.

3.2.3.4. Diseño del esquema conceptual de datos

Como primer paso se hizo una investigación de las tecnologías software libre existes para el manejo de base de datos; y en base a esta investigación se realizó la selección del gestor de datos que será usado.

En la Tabla 3.49 se muestra un cuadro comparativo de dichas tecnologías. Este estudio fue realizado por (Luna, Aguayo, & Rossodivita, 2005). Se puede observar que en líneas generales los Sistemas Manejadores de Base de Datos (DBMS) tienen gran similitud en cuanto a su estructura, la diferencia radica en el manejo de los usuarios, las sentencias de código, la seguridad y la integridad de los datos, entre otros.

Tabla 3.49. Cuadro Comparativo entre SGBD en Arquitecturas OpenSource

	MySQL	PostgreSQL	MaxDB	Firebird	Ingres
Versión	5 8.x	7.5	1.5.x	Ingres	R3
Plataformas	Linux, Solaris, HP-UX, MacOs, AIX, SCO, IRIX, FreeBSD, NetBSD, OpenBSD, Windows, SDI, DEC, OS/2, Compaq Tru64, Novell NetWare	Linux, Solaris, HP-UX, AIX, IRIX, FreeBSD, OpenBSD, NetBSD, MacOs, SCO OpenServer, SCO Unixware, BeOS, BSDI, Compaq Tru64, QNX, Windows	Linux, Solaris, HPUX, AIX, Windows	Linux, Solaris, freeBSD, HP-UX, MacOS, Windows	Linux, Solaris, HP-UX, AIX, Compaq Tru64, OpenVMS Windows
SQL Standard	Medio	Alto	Medio	Alto	Medio
Velocidad	Media/Alta	Media	?	Media/Alta	Media/Alta
Estabilidad	Alta/Muy Alta	Alta	Media/Alta	Media	?
Integridad de Datos	Sí	Sí	Sí	Sí	Sí
Seguridad	Alta	Media/Alta	Media	?	Alta
Soporte de Vistas	Sí (sin índices)*	Sí	Sí	Sí	Sí
Soporte de Esquemas	Sí*	Sí	Sí	No	Sí
Soporte de procedimientos almacenados	Sí (no permite algunas instrucciones)*	Sí (pl/pgSQL, pl/Perl, pl/TCL, pl/Python, pl/sh)	Si	Si	Si
Interfaces de programación	ODBC, JDBC, C/C++, .NET/Mono, ADO.Net, OLEDB, Delphi, Perl, Python, PHP, Embedded (C precompiler), Embeded in Java	ODBC, JBDC, C/C++, Embeded SQL (in C), Tcl/Tk, Perl, Python, PHP, .NET	ODBC ,JDBC, C/C++, Precompiler (Embedded SQL), Perl, Python, PHP	ODBC, JDBC, C/C++, PHP, Python, Perl, Kylix, Delphi, .NET/Mono, ADO	ODBC, JDBC, C/C++, .NET, Perl, Python, PHP, Cobol, Fortran
Tipos de tablas	InnoDB (default), MYISAM, BerkeleyDB, MERGE, Derived	1 Tipo	EVT, tablas procedimen tales, vistas actualizables y tablas derivadas	?	BTREE, ISAM, HASH, HEAP, HEAPSORT, PARTITION
Balanceo de cargas	Sí	No	No	No	No
Tablespaces	Sí	Sí	No	?	No
Clustering	Sí	No	No	No	Sí

Fuente: (Luna et al., 2005)

Finalmente, al comparar PostgreSQL y MySQL, se observa que son pocas las diferencias; sin embargo, el investigador se inclina a elegir PostgreSQL ya que tiene mejor conocimiento sobre este manejador, que sobre los demás DBMS.

Una vez realizada la elección del DBMS a ser usado, se pasó a elaborar el modelo relacional de la base de datos para los objetos persistentes. Este modelo se muestra en la Figura 3.41. Allí se puede observar, que los objetos que se necesitan mantener de manera persistente son los Usuarios, Avatares, Objetos del sistema (ItemObject), la información del Ambiente y las características de los materiales; las demás tablas presentes en el modelo provienen de las relaciones de herencia y dependencia entre objetos.

Para poder elaborar el modelo relacional de la base de datos se partió del diagrama de clases del sistema (son dos diagramas de clases el del servidor y el del cliente). Este diagrama es bastante amplio, ya que se hizo uso de algunos patrones de diseño como el patrón de máquinas de estado, patrón de diseño en solitario, y el patrón de diseño decorativo, buscando en lo posible de crear un sistema flexible y susceptible a modificaciones. Debido al tamaño del diagrama se hace necesario presentarlo en varias figuras y algunas clases serán repetidas en algunas de ellas, para poder mostrar todas las relaciones entre clases.

Los diagramas de clases para el servidor son mostrados en la Figura 3.42, Figura 3.43, Figura 3.44 y Figura 3.45, mientras que el diagrama de clases para la aplicación cliente lo componen la Figura 3.47, la Figura 3.48 y la Figura 3.49.

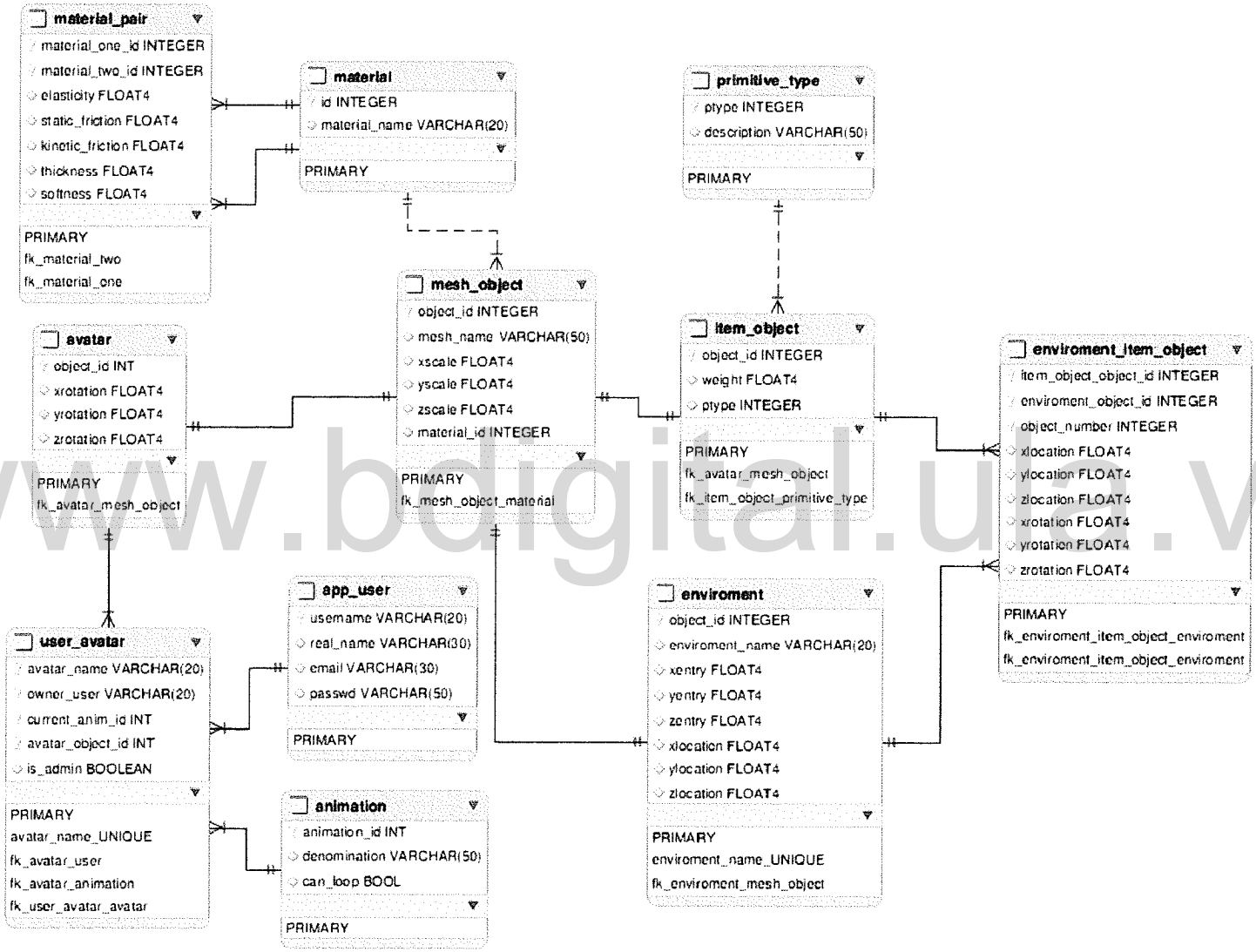

Figura 3.41. Modelo relacional de la base de datos

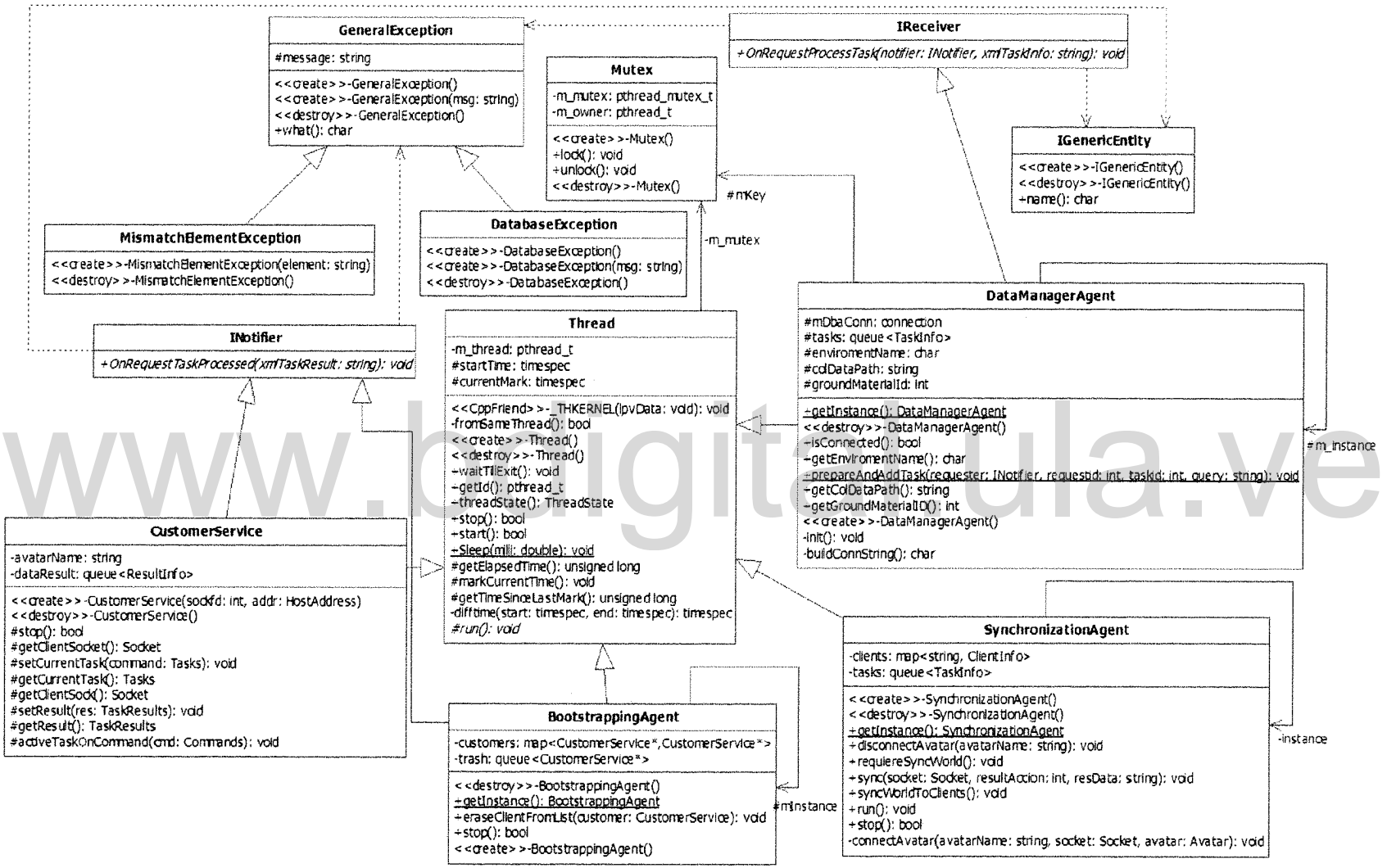

Figura 3.42. Diagrama de clases para el programa servidor vista, de agentes y excepciones generales

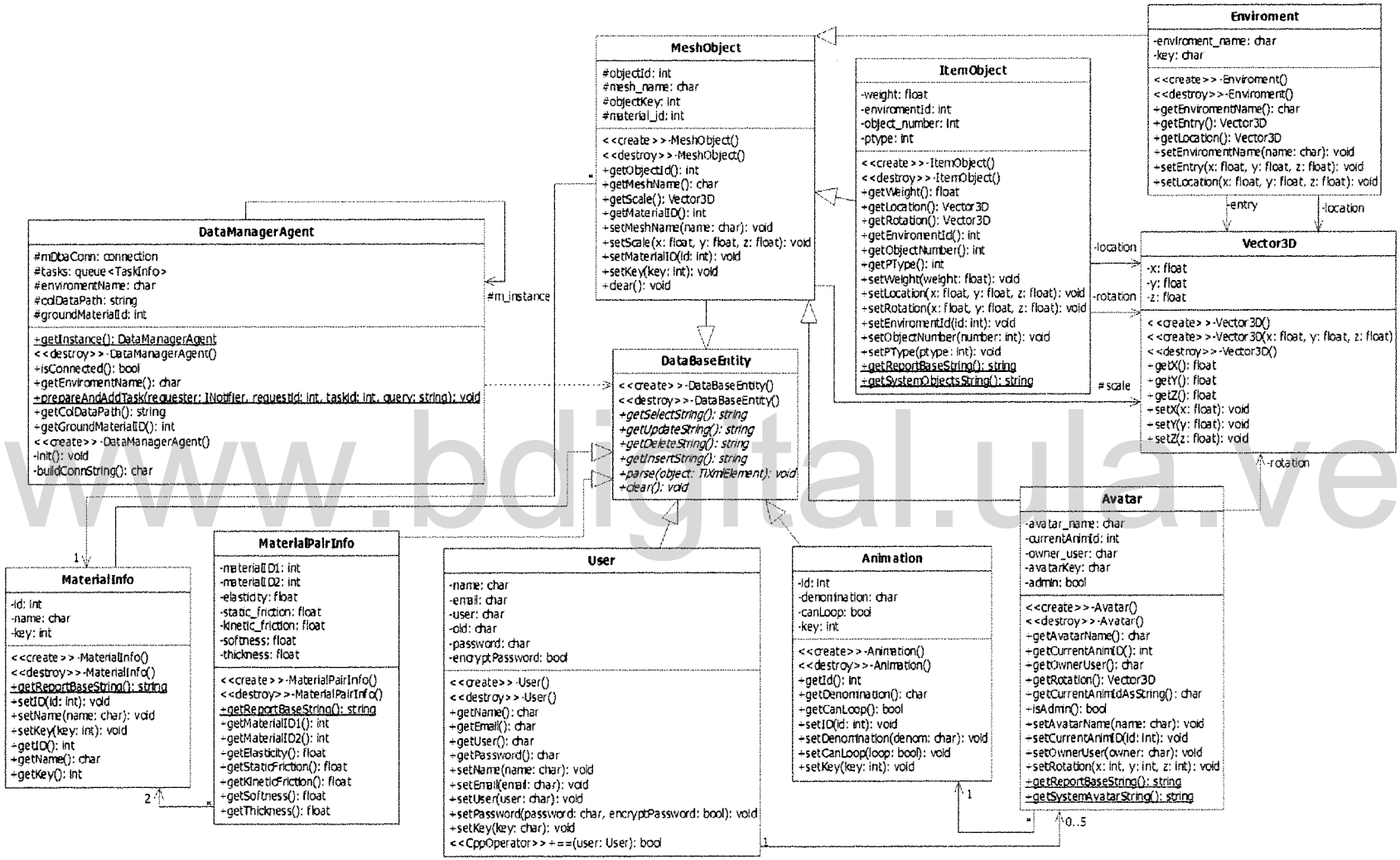

Figura 3.43. Diagrama de clases para el programa servidor, vista de datos persistentes

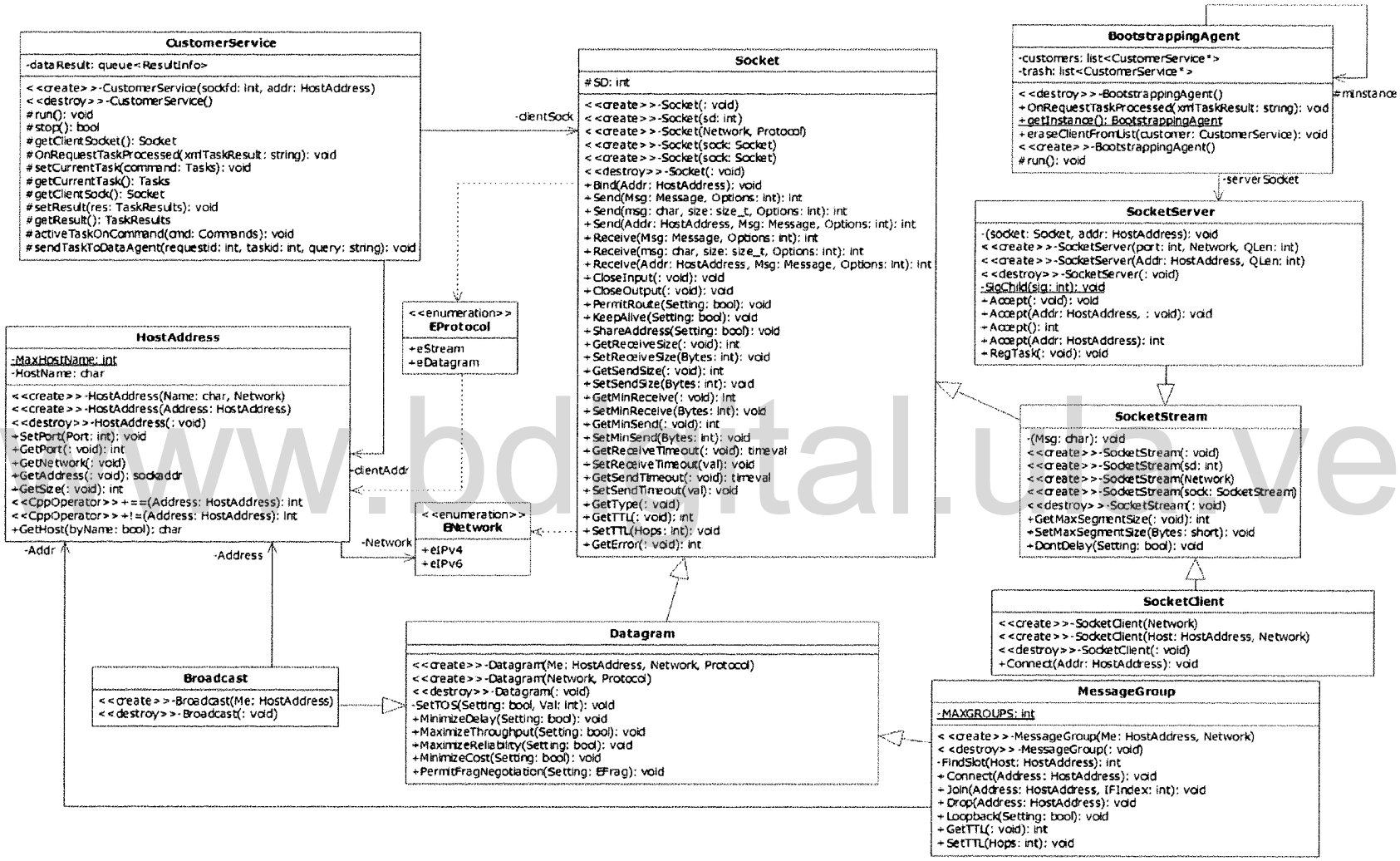

Figura 3.44. Diagrama de clases para el programa servidor, vista de comunicación para red

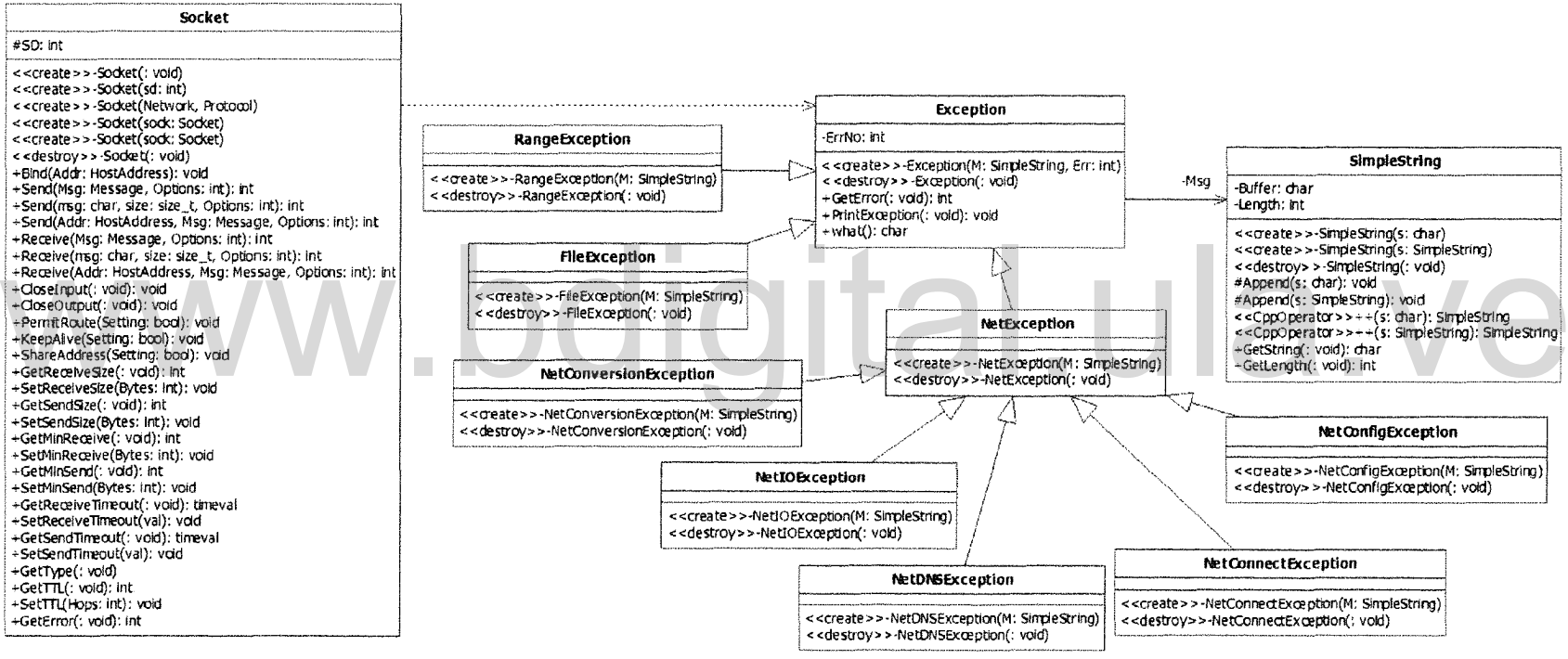

Figura 3.45. Diagrama de clases para el programa servidor, vista de excepciones de red

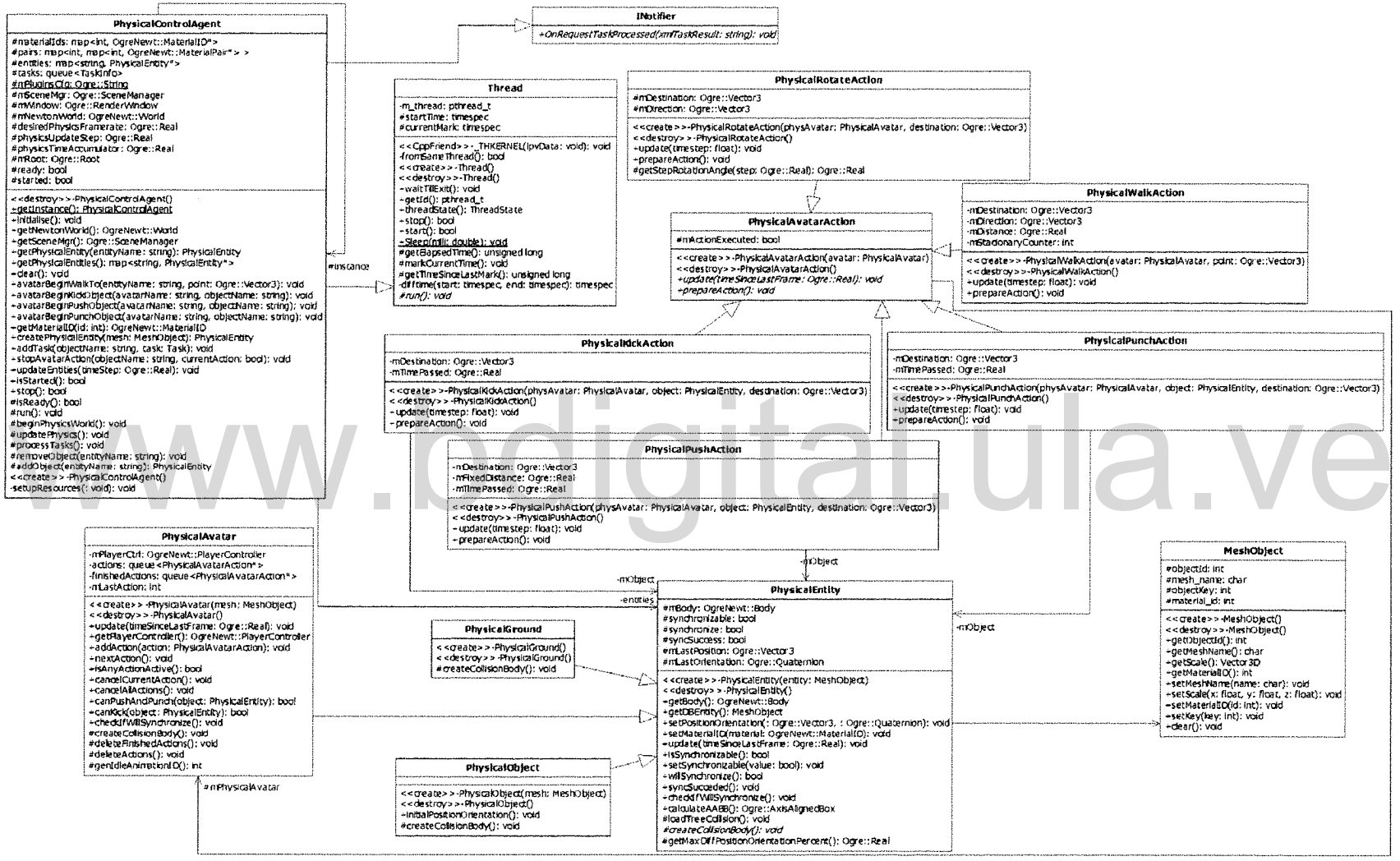

Figura 3.46. Diagrama de clases para el programa servidor, vista agente físico y objetos físicos

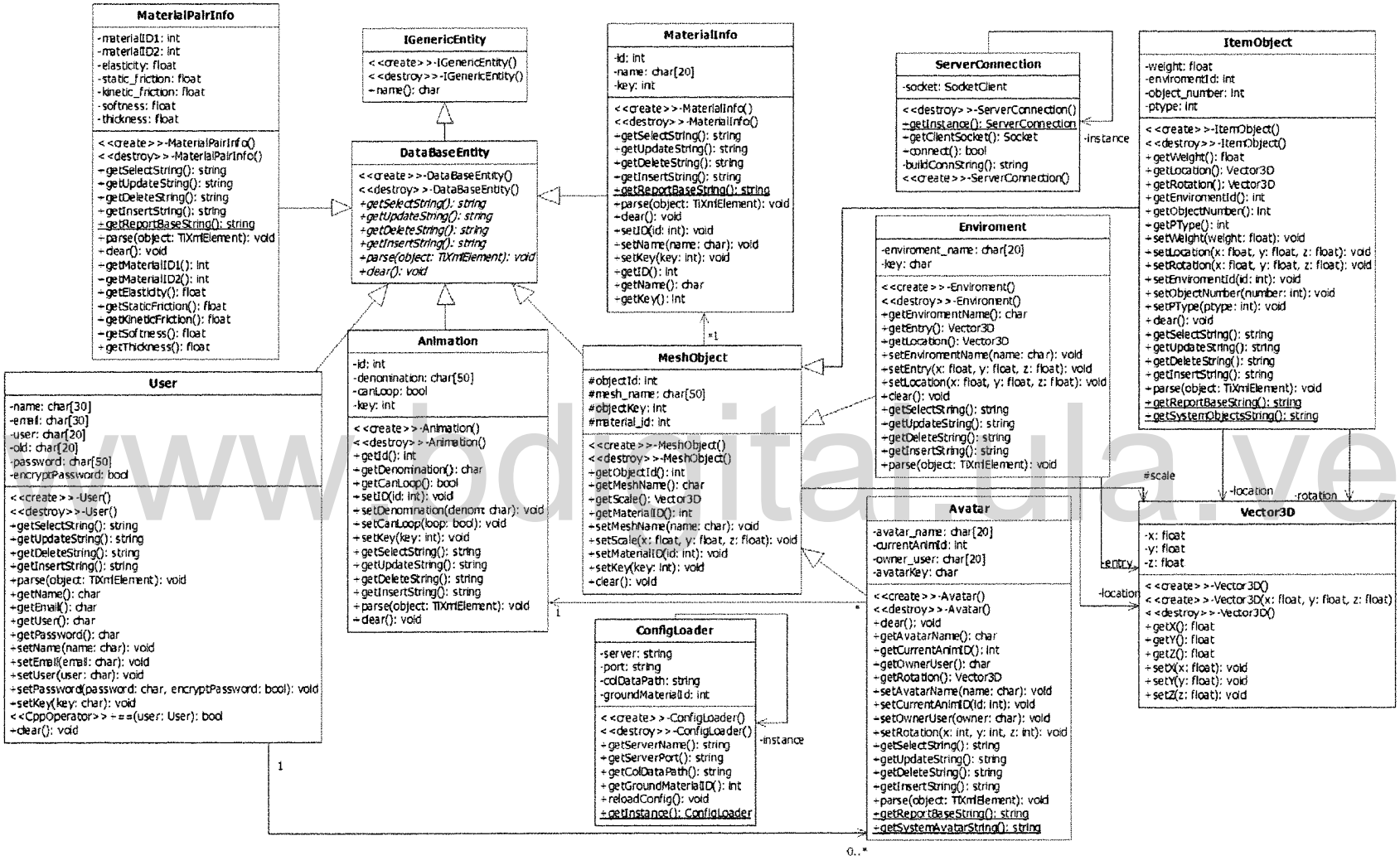

Figura 3.47. Diagrama de clases para el programa cliente, vista de datos persistentes, configuración y comunicación en red

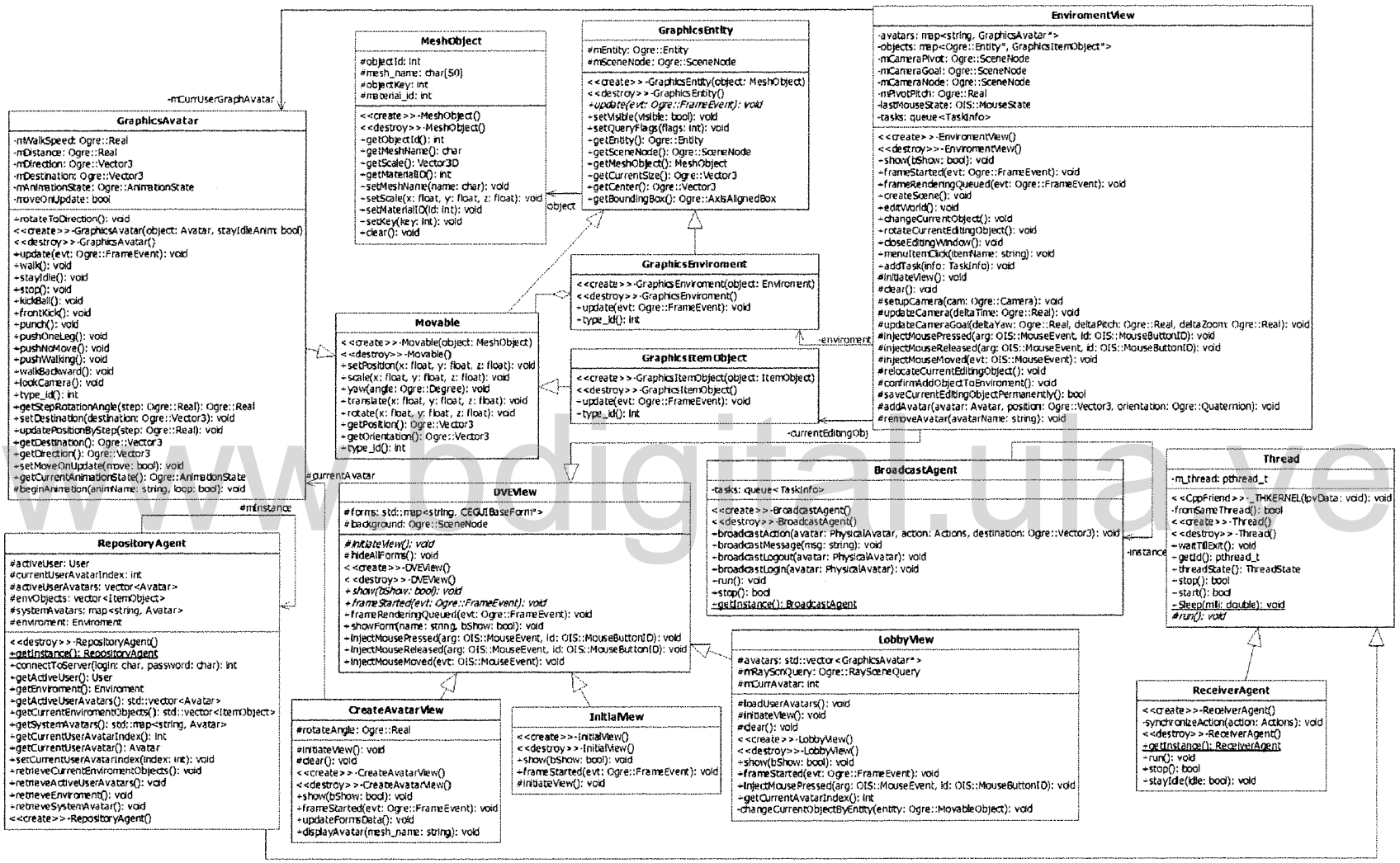

Figura 3.48. Diagrama de clases para el programa cliente, vista de objetos gráficos y agentes

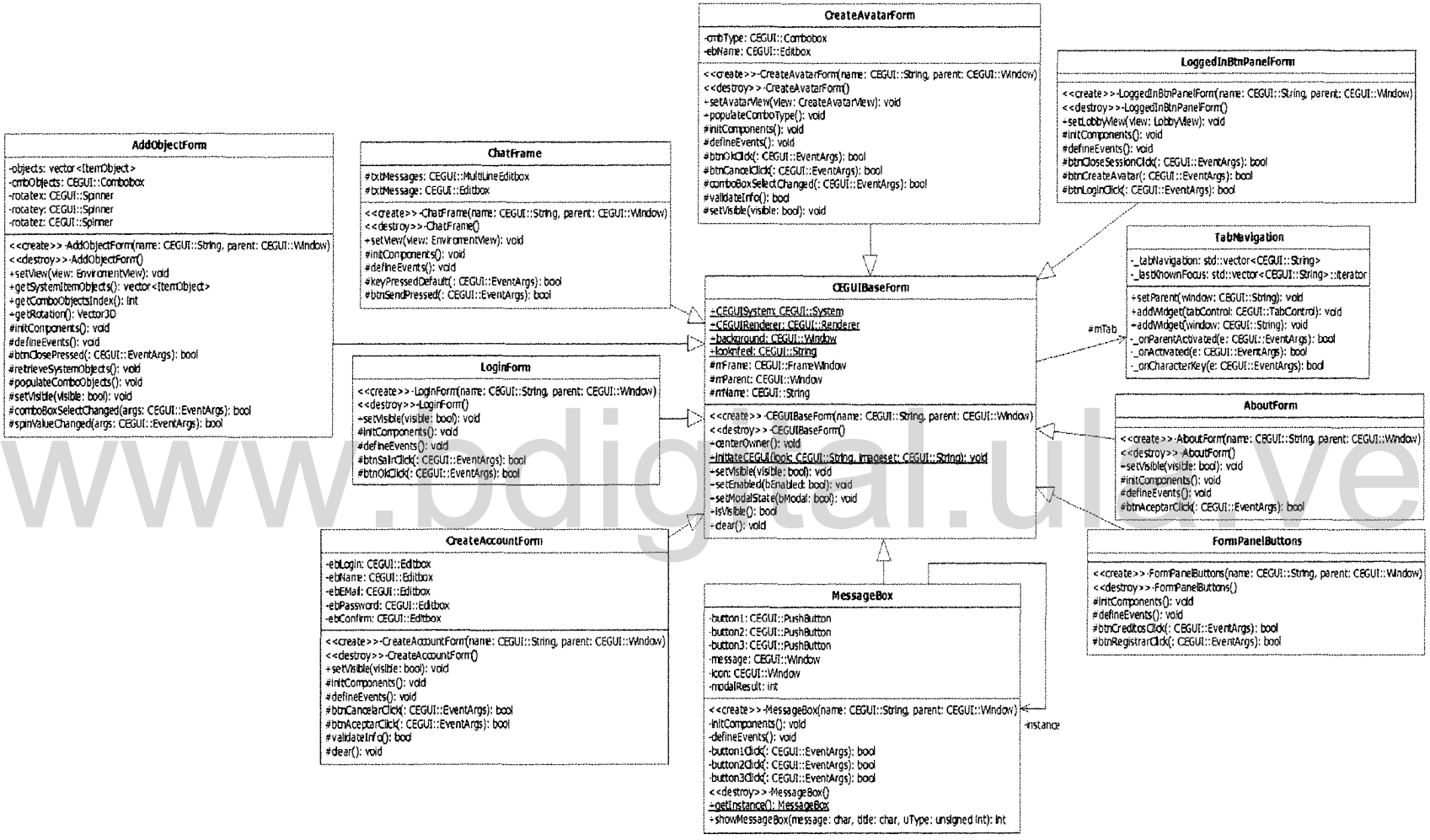

Figura 3.49. Diagrama de clases para el programa cliente, vista de ventanas y diálogos

Capítulo 4. Implementación

En este capítulo se presenta en detalle la forma como se implementó cada uno de los elementos necesarios para el desarrollo de la herramienta para construcción de ambientes virtuales dinámicos, para esto se puso en práctica las fases de producción y post-producción de la metodología MAVD.

4.1. Producción

De acuerdo con (Hernández et al., 2010) ha de construirse el AVD, integrando los objetos que forman parte del mismo, para lograr la funcionalidad propuesta en la fase de preproducción. Además, han de codificarse y probarse individualmente los modulos de software que forman el sistema; también es necesario implantar el modelo de datos en el DBMS seleccionado para tal fin.

4.1.1. Instalar la plataforma de desarrollo de la aplicación

Como primera actividad para esta fase de la metodología, se debe instalar la herramienta de desarrollo de la aplicación. El investigador realizó este proyecto bajo la plataforma Linux, en vista que la mayoría de componentes que integran el sistema, son también de software libre.

Los pasos que se han seguido para la instalación de las herramientas de desarrollo en Linux se pueden observar en los anexos del A al E.

4.1.2. Desarrollo Agentes

En el Capítulo 3 se describió la forma como se diseñaron los agentes que forman parte del sistema; en esta parte del capítulo se muestra cómo se hizo la implementación en el lenguaje de programación C++.

Como primera fase para el desarrollo de los agentes se realizó una investigación que permitió localizar algunos componentes que eran necesarios para el desarrollo del sistema; Los componentes localizados forman un conjunto de librerías desarrolladas en C++ bajo el paradigma de la Programación Orientada a Objetos.

La primera librería que se consiguió, es una librería orientada objetos para comunicación en red; el componente se usa para comunicar los agentes del cliente con los agentes del servidor. Este componente se encuentra en el libro Programación de Socket Con Linux de Sean Walton (Walton, 2001) y el mismo ha sido liberado bajo la licencia GPL.

El segundo componente es una librería desarrollada por (Thilo, 2011) que permite crear hash MD5 de una cadena de caracteres. Este componente fue usado para aumentar la seguridad del ambiente virtual, de tal forma, que la contraseña del usuario no es enviada como texto plano por la red, sino que en cambio, se envía su hash MD5, lo cual disminuye la posibilidad de que alguna persona malintencionada pueda obtener de manera ilegal la contraseña de un usuario desde los datos enviados por la red.

Luego, se consiguió una librería que sirve para realizar el análisis de un archivo XML. Esta librería se conoce como *tinyxml* y fue desarrollada originalmente por (Thomanson, 2010). La librería es de vital importancia para el proyecto, ya que el agente DataManager recibe las tareas y emite los resultados en formato XML.

También se consiguió la librería libpq de PostgreSQL, la cual permite realizar consultas a la base de datos PostgreSQL usando el lenguaje de programación C++. Esta librería es usada por el agente DataManager para enviar y extraer información de la base de datos.
El investigador hizo algunas modificaciones menores a algunas de las librerías descritas anteriormente, que permitió adaptarlas a las necesidades de desarrollo

del ambiente virtual dinámico.

Una vez obtenidas las librerías necesarias para la implementación de los agentes se procedió a codificar cada uno de ellos, siguiendo los diagramas de clases actividades y secuencias presentados en el Capítulo 3.

4.1.3. Construcción de la BD

La construcción de la base de datos se realizó usando los asistentes de la herramienta pgAdmin III que viene incluida con la instalación de PostgreSQL. Las instrucciones SQL que se llevaron a cabo para cumplir con esta actividad, se exponen en el Anexo I.

4.1.4. Diseño y ejecución de pruebas de componentes

Antes de integrar los componentes es necesario probarlos a fin de asegurarse que funcionan correctamente de forma individual. La primera prueba fue realizada en el componente de conexión a la base de datos "libpq", el resultado de esta prueba se puede observar en la Tabla 4.1.

Tabla 4.1. Pruebas realizadas al componente de base de datos "libpq"

Componente de conexión a base de datos "libpq"	
Prueba	**Resultado**
Conexión a la Base de datos	Correcto
Guardar datos	Correcto
Recuperar datos	Correcto
Actualizar datos	Correcto

Es importante resaltar que estos resultados, se obtuvieron después de varias pruebas; si una prueba resultaba en estado erróneo, se hacía la depuración del error, hasta conseguir que el resultado de la prueba fuera correcto.

La siguiente prueba fue realizada sobre el componente de comunicación en red. El resumen de esta prueba se muestra en la Tabla 4.2.

Tabla 4.2. Pruebas realizadas al componente de comunicación en red

Componente de comunicación en red	
Prueba	**Resultado**
Conexión al servidor	Correcto
Espera de clientes (listen)	Correcto
Enviar objetos	Correcto
Recibir objetos	Correcto
Desconexión	Correcto

Se pasó a realizar la prueba en el componente de codificación MD5 para verificar su funcionalidad. Para validar si el resultado es correcto, se generaron códigos de textos usando la función MD5 de PosgresSQL, y luego, ese mismo texto se codificó usando el componente MD5 para C++. Posteriormente, se compararon ambos resultados, con lo cual se validó que el componente funciona correctamente. Esta prueba se muestra en la Tabla 4.3.

Tabla 4.3. Pruebas realizadas al componente de codificación MD5

Codificación MD5	
Prueba	**Resultado**
Obtener Hash de una cadena y validar resultado respecto al MD5 de PostgreSQL	Correcto

Se prosigue con las pruebas realizadas al componente tinyxml. Para este componente se realizaron pruebas que validan, si el componente realiza correctamente la lectura y creación del archivo xml (ver Tabla 4.4).

Tabla 4.4. Pruebas realizadas al componente tinyxml

Componente tinyxml	
Prueba	**Resultado**
Analizar archivo para lectura	Correcto
Recorrer estructura XML	Correcto
Crear estructura XML	Correcto

Una vez que se probaron los componentes fundamentales para comunicación entre agentes, se hizo la prueba de animación de los avatares, revisando si se ejecutaban correctamente en Ogre3D. Para ello, se usó la herramienta OgreMeshy, la cual permitió abrir los modelos que habían sido exportados previamente al formato de archivo de Ogre3D y revisar su visualización correcta, así como cada una de las animaciones integradas en el modelo. La Tabla 4.5 muestra el resultado de estas pruebas.

Tabla 4.5. Pruebas realizadas al avatar

Animaciones del Avatar	
Prueba	**Resultado**
Animación caminar	Correcto
Animación patear	Correcto
Animación golpear	Correcto
Animación patada frontal	Correcto
Animación caminar hacia atrás	Correcto
Animación empujar con una pierna	Correcto
Animación empujar con ambas piernas	Correcto
Animación de Ambiente 1	Correcto
Animación de Ambiente 2	Correcto
Animación de Ambiente 3	Correcto

Habiendo realizado las diferentes pruebas en cada uno de los componentes encontrados, se continuó con la integración estos componentes en los agentes, para darles las propiedades de comunicación entre ellos, así como con la base de datos.

Los agentes del sistema no se crearon en una sola iteración de la fase de producción, sino que se hicieron varias iteraciones.

En la primera iteración, solo se permitía el inicio de sesión del usuario, la creación de cuentas y la creación de avatares para el usuario. Los Agentes programados en esta iteración fueron: *Bootstrapping*, *CustomerService*, *DataManager*, *Repository* y *Visualization*.

En la segunda iteración, se incorporó la conexión del avatar al ambiente virtual aún sin física y sin sincronización. No se incorporan nuevo agentes ya que los existentes cubrían las nuevas funcionalidades a ser incorporadas.

Para la tercera iteración, se incorporó la física entre objetos (detección de colisiones), así como la sincronización de las vistas de los usuarios por parte del agente de sincronización. Se crean los agentes *PhysicalControl*, *Synchronization*, *Receiver* y *Broadcast*.

Finalmente en la cuarta iteración, se programa la realización de acciones por parte del usuario, aquí también hubo que modificar los agentes *Synchronization*, *Receiver* y *Broadcast*, así como el agente *PhysicalControl*, a fin de que estás nuevas características fueran tomadas en cuenta al momento de realizar la sincronización. Es importante señalar que el Agente Synchronization solo realiza la sincronización de un objeto en caso que el mismo haya cambiado sus propiedades físicas (posición o rotación), de lo contrario el objeto permanecerá sin ser sincronizado, lo que permite aliviar el tráfico de datos por medio de la red.

Al finalizar la cuarta iteración ya se cumplían todos los requerimientos del sistema. Así que se detiene el proceso iterativo y se continúa con la siguiente fase de la metodología.

4.2. Postproducción

Según (Hernández et al., 2010) en esta fase se comprueba que el sistema creado cumple realmente con los requisitos establecidos en la fase de preproducción. Para ello, se diseñó un conjunto de pruebas que permitieron llevar a cabo la verificación del sistema.

A continuación se presentan cada una de las pruebas realizadas al sistema, las cuales permitirán certificar el cumplimiento de los requisitos solicitados.

La primera prueba que se hizo permite verificar que el sistema cumple con el requisito "a)" del documento de requisitos presentado en el Capítulo 3. En la Figura 4.1, la Figura 4.2, la Figura 4.3, la Figura 4.4, la Figura 4.5, la Figura 4.6 y la Figura 4.8, se muestran cada una de las vistas gráficas que posee el sistema con lo cual se certifica el cumplimiento de este requisito.

Luego, se verificó el cumplimiento del requisito "b)" de la lista de requisitos. En la Figura 4.4 se observa cómo cada usuario tiene hasta un máximo de cinco espacios para ubicar sus avatares, mientras que en la Figura 4.7 se puede ver que el sistema emite un mensaje de error si se intenta iniciar sesión sin haber creado al menos un avatar.

Para verificar que los requisitos "c)" y "h)" se cumplen, se presentan la Figura 4.9 y la Figura 4.10. En la primera figura se muestra la pantalla de edición del ambiente y la lista de objetos que pueden ser agregados al mismo, mientras que en la segunda, se observa el momento en que se intenta agregar un tambor a la escena, lo cual es hecho por un usuario con privilegios de administrador.

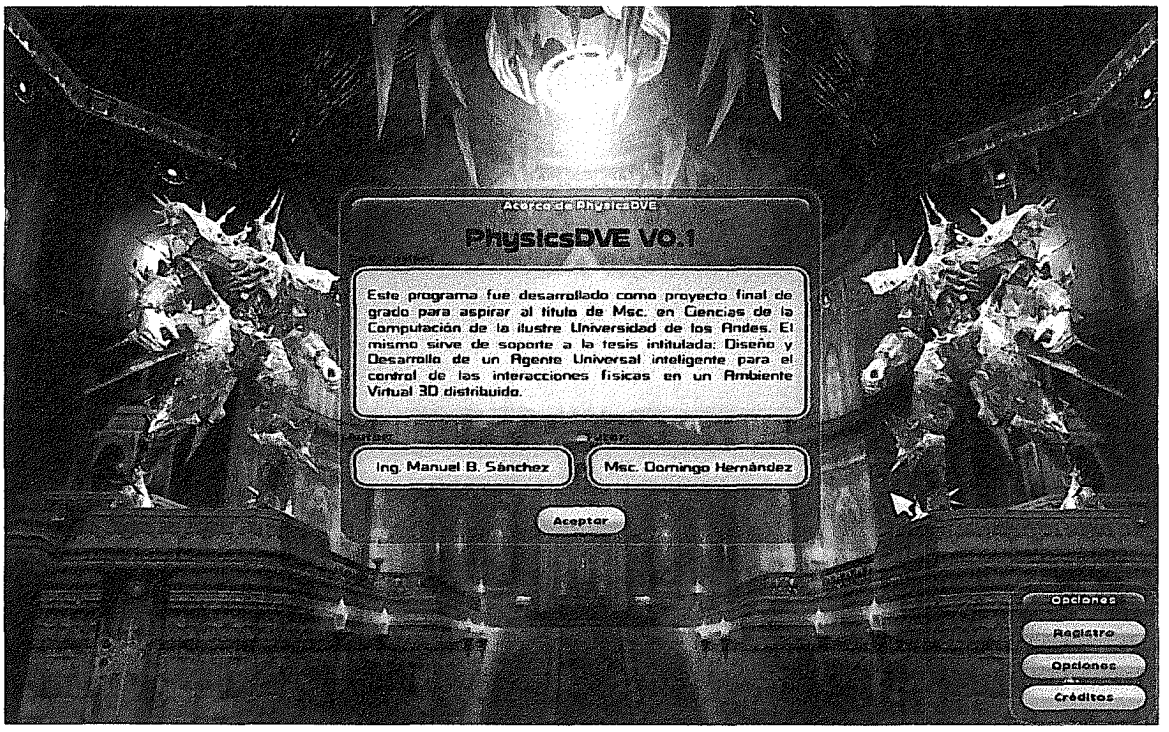

Figura 4.1. Vista inicial de la aplicación cliente (créditos)

Fuente: propia

Figura 4.2. Vista de inicio de sesión de la aplicación cliente

Fuente: propia

Figura 4.3. Registro de un nuevo usuario

Fuente: propia

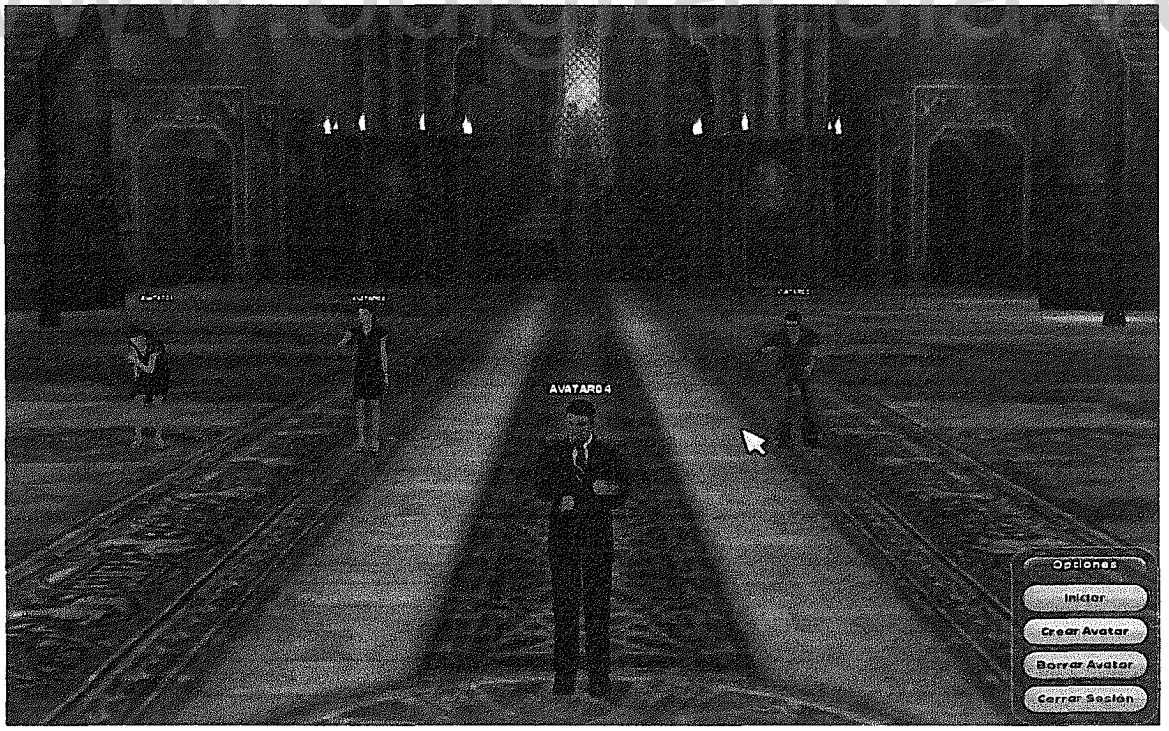

Figura 4.4. Vista de Selección de Avatares de usuario

Fuente: propia

Figura 4.5. Vista de creación de avatares (avatar femenino)

Fuente: propia

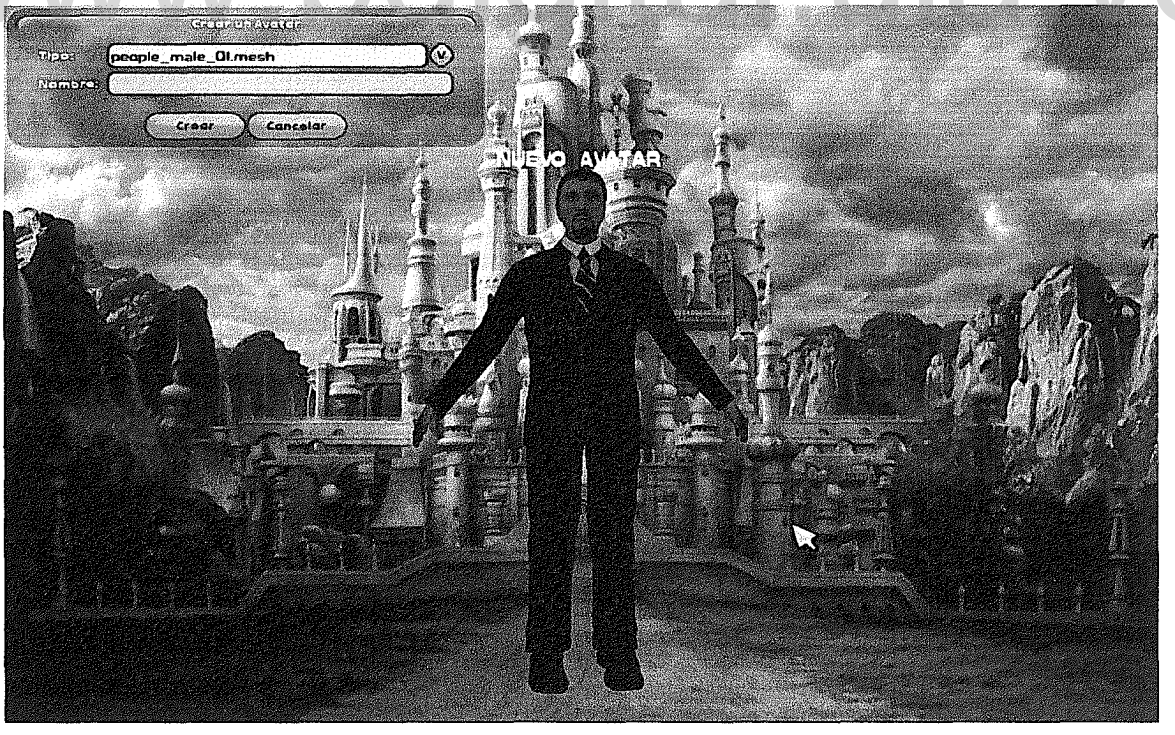

Figura 4.6. Vista de creación de avatares (avatar masculino)

Fuente: propia

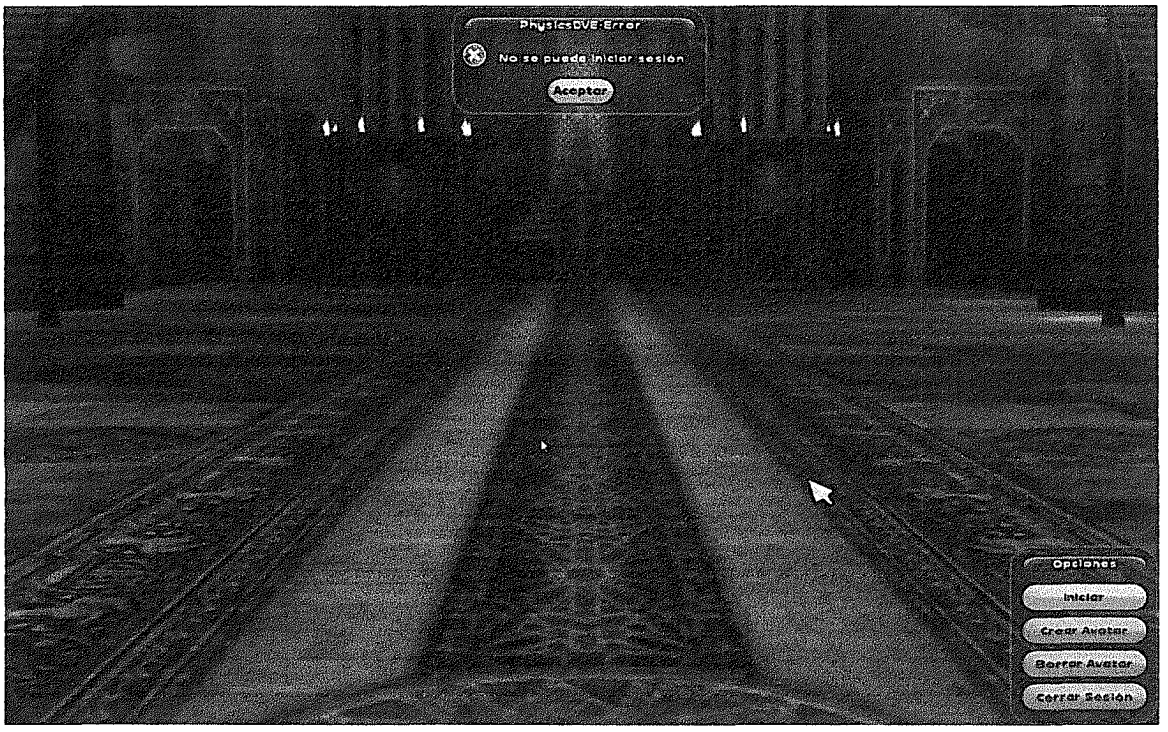

Figura 4.7. Mensaje recibido cuando se intenta iniciar sesión sin haber creado avatares

Fuente: propia

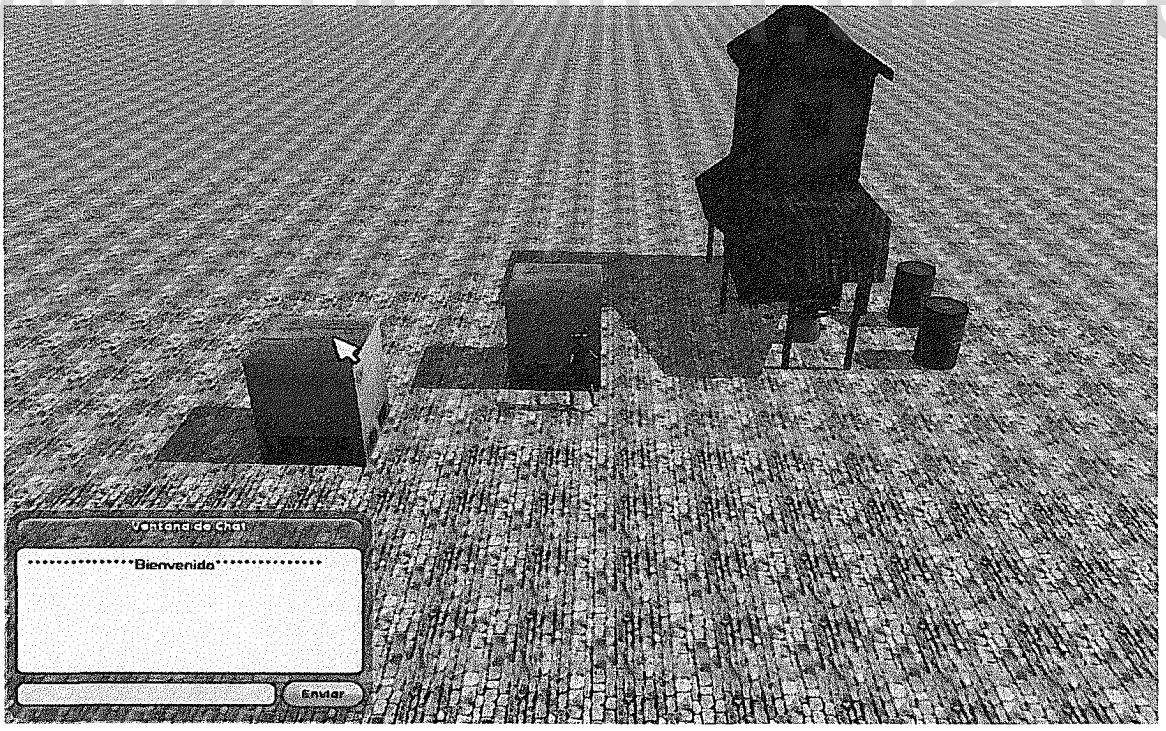

Figura 4.8. Vista de Ambiente

Fuente: propia

Figura 4.9. Edición del ambiente virtual (Lista de objetos que se pueden agregar)

Fuente: propia

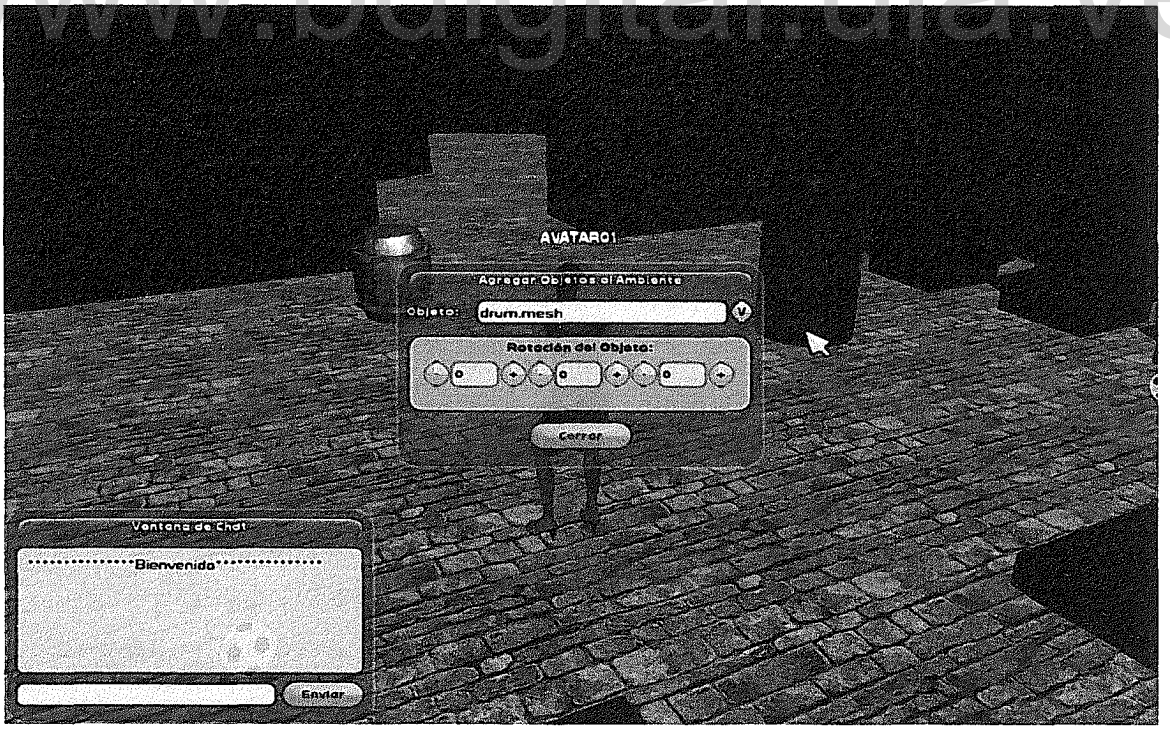

Figura 4.10. Edición del ambiente virtual (Agregando un barril al ambiente virtual)

Fuente: propia

Para los requisitos "d)", "e)", "f)", "g)" e "i)" se puede verificar su cumplimiento en la Figura 4.11 y Figura 4.12. En la primera de estas dos figuras vemos como los avatares se encuentran reunidos en una estructura, compuesta por paredes, escaleras, columnas, entre otros. También se observa que los avatares están manteniendo una conversación por medio de un chat.

En la segunda imagen de las mencionadas, se observa que el resto del ambiente virtual está compuesto por un piso de una gran dimensión, en donde se pueden ubicar otros objetos que componen el AVD.

Por otra parte, en estas dos imágenes se puede apreciar que los personajes no se ven envueltos en una historia que deben desarrollar para poder avanzar a otras áreas del ambiente, sino que cada personaje puede moverse libremente por el ambiente, siempre y cuando exista un camino físicamente posible hasta su lugar de destino.

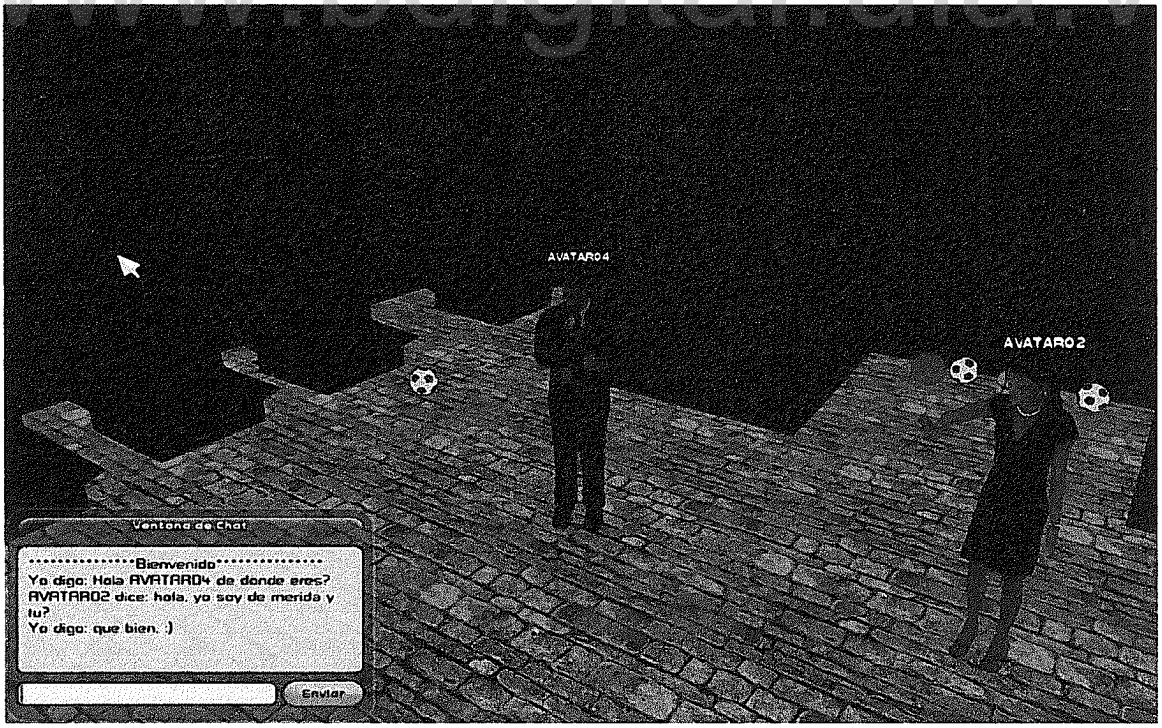

Figura 4.11. Comunicación de avatares por medio del chat

Fuente: propia

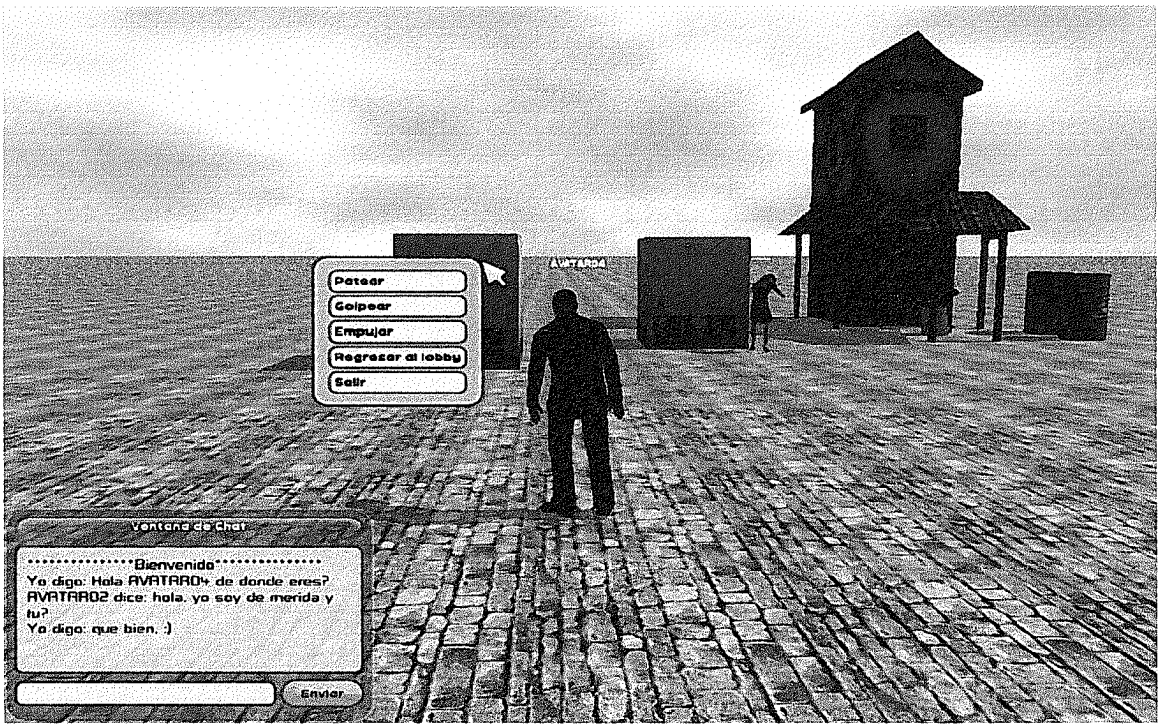

Figura 4.12. Acciones que puede realizar el avatar con los objetos

Fuente: propia

Con respecto al cumplimiento del requisito "j)", es fácil de apreciar en todas las imágenes que se han mostrado. En esta sección, que los avatares que aparecen en ella tienen apariencia de hombres y mujeres humanas.

Para los requisitos enumerados como "k)" y "l)", se observa en la Figura 4.12 que la lista de acciones físicas que están disponibles son las de patear, golpear y empujar. Estas acciones son mostradas cuando el usuario presiona el botón derecho del ratón sobre un objeto dinámico del ambiente. Luego, tendrá que presionar el botón izquierdo del ratón sobre la acción que desea que el usuario realice con el objeto señalado.

Con respecto a la acción de caminar que se aprecia en la Figura 4.14. La misma es iniciada, cuando el usuario presiona el botón izquierdo del mouse en el lugar hacia donde desea que el avatar sea dirigido, a partir de ese momento el avatar comenzará a trasladarse hacia el destino que el usuario que lo controla haya

señalado.

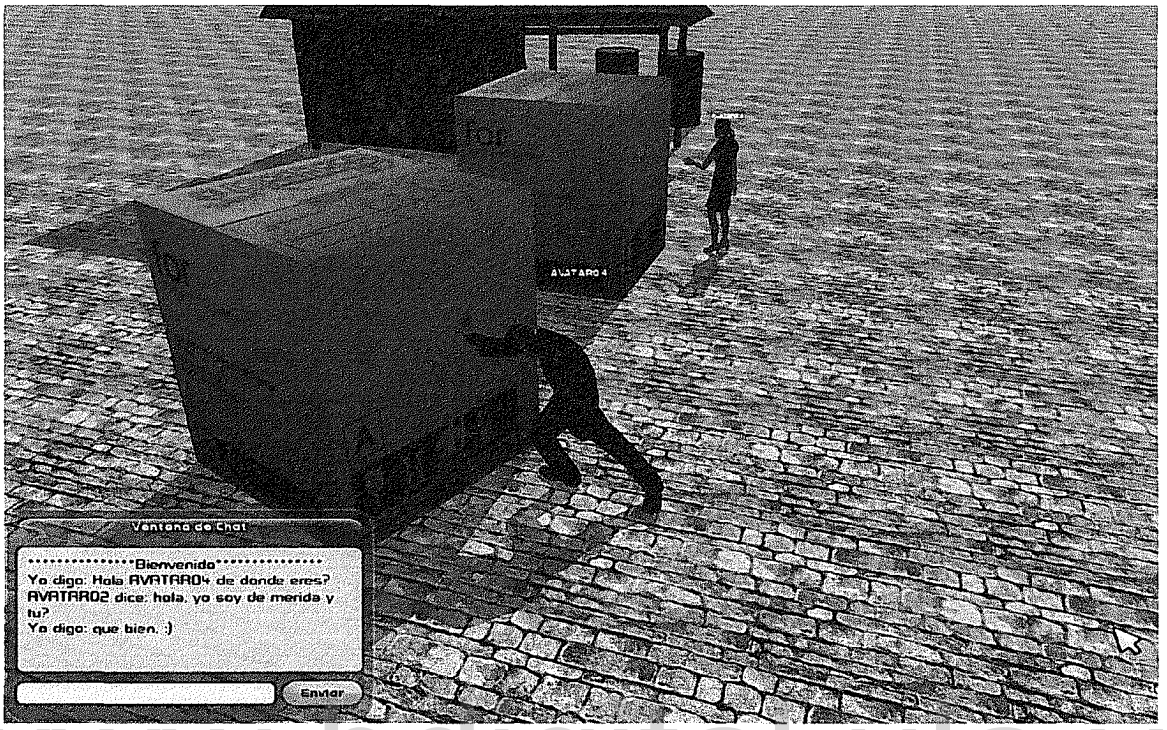

Figura 4.13. Avatar interactuado con un objeto físico del ambiente virtual
Fuente: propia

En la Figura 4.13, la Figura 4.14 y la Figura 4.16, se puede apreciar cómo se ha cumplido con el requisito "m)" del documento de requisitos. En cada una de esas figuras se observa que el avatar se encuentra ejecutando una animación, que depende de la orden que haya sido dada por el usuario que lo controla.

Con respecto al requisito "n)", no es simple apreciar en imágenes el comportamiento físico de los objetos a los cuales se le han asignado diversos tipos de materiales, ya que para ello es necesario ver el movimiento y la trayectoria que describen dichos cuerpos cuando interactúan con el avatar, para ellos. Es necesario ver el sistema en plena ejecución.

El autor puede comentar referente al cumplimiento del requisito "n)" que la reacción del objeto depende de su peso, elasticidad, fricción y dureza.

Figura 4.14. Avatar caminando hacia un destino indicado por el usuario

Fuente: propia

Figura 4.15. Avatar interactuando con un objeto físico en el AVD

Fuente: propia

Finalmente, para el requisito "o)" del documento de requisitos (ver sección 3.2.2.1), se observa su cumplimiento en varias de las figuras de esta sección. Es apreciable en estás figuras que existen objetos estáticos (árboles, rocas, cabañas, etc.) y objetos dinámicos (cajas, barriles, balones, entre otros) los cuales están presentes incluso antes que el usuario ingrese al AVD.

La ubicación de estos objetos en el AVD se mantiene de forma permanente en la base de datos, en una tabla que tiene por nombre "enviroment_item_objects", de esta forma, cada vez que el ambiente es iniciado, se recupera dicha información y se mantiene sincronizada con cada uno de los avatares que se conecten al ambiente.

Ahora, con referencia a la interacción física entre los objetos del AVD y avatares, se observa en la Figura 4.13 y la Figura 4.16 la reacción de los objetos dinámicos, una vez que el usuario decide interactuar con ellos a través del avatar que lo representa en el ambiente.

Figura 4.16. Reacción del objeto físico después de haber interactuado con el avatar

Fuente: propia

En la aplicación que da servicios a los usuarios del AVD, se realizaron varias pruebas para comprobar el funcionamiento del sistema. La primera prueba que se realizó tiene como objetivo probar la puesta en marcha del ambiente virtual dinámico que está siendo distribuido.

En la Figura 4.17 se aprecia el momento en que se pone en ejecución la aplicación servidora, se observar como el agente DataManager establece la conexión con la base de datos y como se inician los preparativos para poner en servicio el ambiente virtual dinámico donde convergerán los avatares de los diferentes usuarios del AVD.

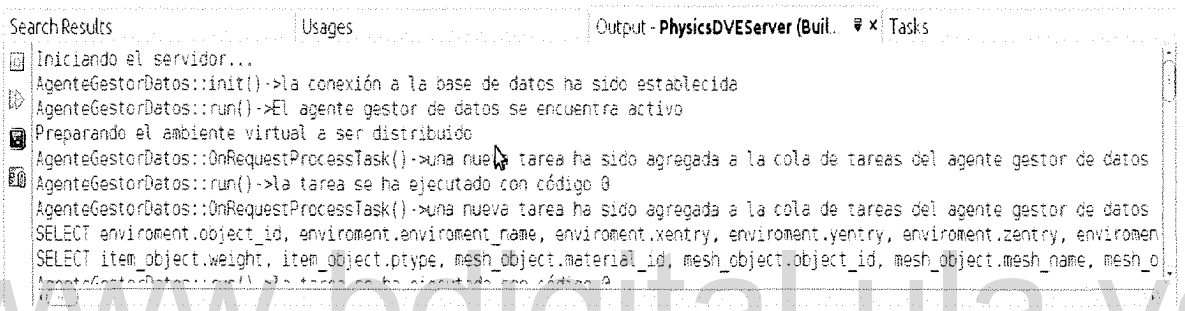

Figura 4.17. Aplicación servidora, Inicio del ambiente virtual
Fuente: propia

Luego, se pasa a probar que los agentes sean despertados y puestos en ejecución. La Figura 4.17 y la Figura 4.18 muestran está actividad, se observa como los agentes son puestos en ejecución por el agente Bootstrapping y finalmente este último se queda a la espera de conexiones de usuario, para posteriormente pasar esas solicitudes a los agentes CustomerService, quienes finalmente son lo que se encargan de la atención de cada cliente.

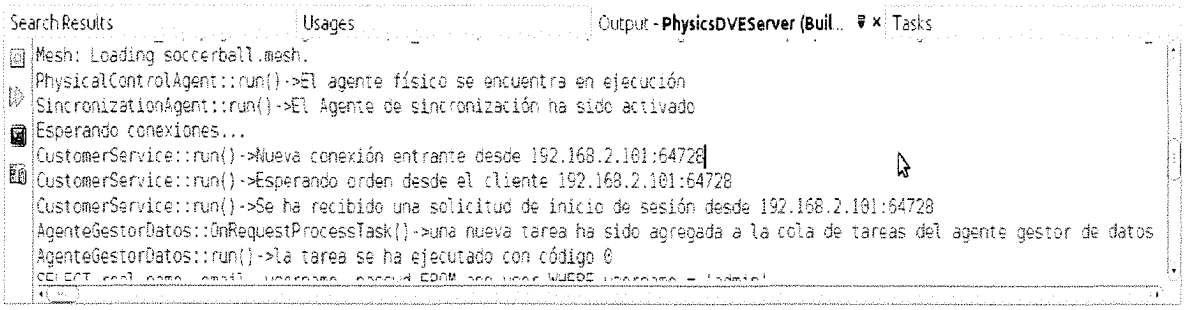

Figura 4.18. Aplicación servidora, despertando los agentes del lado del servidor
Fuente: propia

Otra de las pruebas realizadas a la aplicación servidora, sirve para asegurarse que las peticiones realizadas por el cliente al servidor están siendo procesadas.

En la Figura 4.19 se observa cómo el agente CustomerService recibe las solicitudes de recuperación de los avatares del cliente. Ésta solicitud es pasada al agente DataManager, quien realiza la consulta a la Base de Datos. Luego, estos datos son enviados nuevamente al agente CustomerService quien los remitirá al cliente para que sean presentados al usuario.

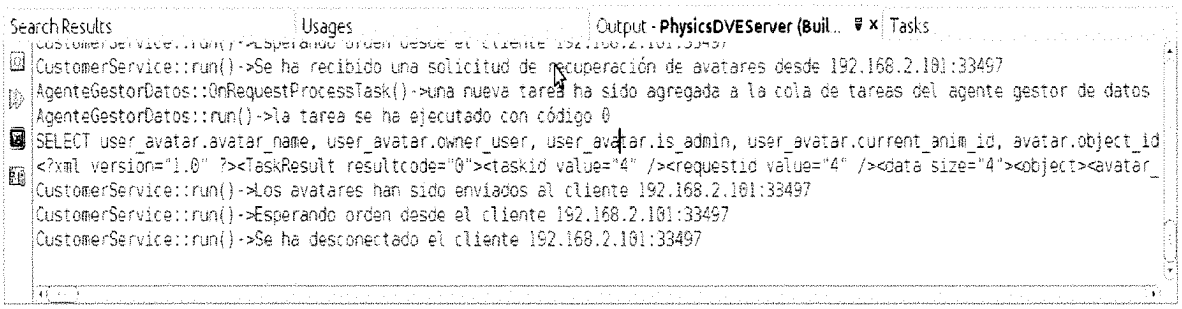

Figura 4.19. Aplicación servidora, procesando acciones solicitadas por el cliente

Fuente: propia

En esta misma figura, se puede apreciar el momento en que el cliente solicita la desconexión del sistema, mientras que en la Figura 4.18 se observa el momento en que el cliente solicita la conexión con la aplicación servidora.

Con respecto al agente físico y al agente de sincronización, no hay manera de observar a través de los mensajes que están cumpliendo con su trabajo, ya que no sería razonable emitir un mensaje cada vez que la simulación física es actualizada o los objetos estén siendo sincronizados, ya que estas tareas son realizadas muchas veces en un segundo.

Las tareas realizadas tanto por el agente PhysicalControl como por el agente Synchronization, se pueden apreciar de mejor manera en las capturas de pantalla de la aplicación cliente, ya que allí se nota como los objetos dinámicos y avatares cambian de posición a medida que la acción se ejecuta a través del tiempo.
Como última fase de este capítulo, se muestra una tabla que resume las pruebas

realizadas a la aplicación cliente y el resultado que se obtuvo de cada una de ellas. Se nota que en las pruebas donde el usuario ingresaba datos incorrectos el sistema respondía con un mensaje de error, informando al usuario que ha realizado una mala operación y que debe corregir los datos.

Tabla 4.6. Pruebas realizadas en la aplicación cliente

Pruebas realizadas en la aplicación cliente	
Prueba	**Resultado del proceso**
Conexión entre la aplicación cliente y la aplicación servidora	Correcto
Registro de un usuario con datos correctos	Correcto
Registro de un usuario con datos incorrectos	Error
Inicio de sesión de un usuario registrado	Correcto
Inicio de sesión de un usuario no registrado	Error
Selección de avatar	Correcto
Creación de un avatar con datos correctos (el usuario posee menos de 5 avatares)	Correcto
Creación de un avatar con datos incorrectos (el usuario posee menos de 5 avatares)	Error
Creación de un avatar con datos correctos (el usuario posee 5 avatares)	Error
Ingreso al ambiente virtual dinámico (al menos un avatar)	Correcto
Ingreso al ambiente virtual dinámico (no ha creado avatar)	Error
Sincronización de la lista de objetos iniciales del AVD	Correcto
Sincronización y ejecución de la acción caminar	Correcto
Sincronización y ejecución de la acción patear objeto	Correcto
Sincronización y ejecución de la acción golpear objeto	Correcto
Sincronización y ejecución de la acción empujar objeto	Correcto
Solicitar lista de objetos que se pueden agregar al ambiente	Correcto
Agregar nuevo objeto al ambiente.	Correcto
Desconexión del avatar del ambiente virtual dinámico	Correcto
Desconexión y cierre de la aplicación cliente	Correcto

Finalmente, en este Capítulo de este capítulo se puede concluir que las pruebas realizadas tanto en la aplicación cliente como en la aplicación servidora, comprueban y validan la arquitectura y las metodologías utilizadas ya que se comprobó el cumplimiento de los requisitos funcionales y no funcionales planteados en el Capítulo 3. La aplicación de las pruebas completa el cumplimiento de los objetivos planteados en el presente trabajo de grado.

Capítulo 5. Conclusiones

Mediante este trabajo se ha logrado diseñar e implementar un SMA, en el cual se definen los tipos de agentes que deben ser parte de un AVD, así como la forma en que estos agentes se interrelacionan, para lograr la sincronización y reproducción de las acciones físicas ejecutadas por cada usuario en los distintos monitores de los clientes conectados al AVD.

Con el desarrollo de este proyecto, se genera un aporte importante al área de los ambientes virtuales dinámicos, al mostrar de forma clara y concisa los elementos que deben formar parte de un ambiente virtual dinámico donde intervengan objetos con propiedades físicas.

Se ha logrado incorporar un modelo físico al ambiente virtual dinámico, lo que permite incrementar el realismo de la interacción entre los avatares y los objetos del AVD, haciendo el ambiente aún más inmersivo.

El estudio de las técnicas usadas actualmente para la construcción de ambientes virtuales, fue fundamental para ayudar a seleccionar la herramienta que se usó en esta investigación como motor de renderizado de los modelos tridimensionales para la aplicación cliente; ya que a través del formato de archivo soportado por la herramienta, se facilitó la reproducción de las animaciones de los avatares al momento en que ejecuta una acción.

Haber realizado un estudio acerca de las técnicas usadas en la actualidad para la construcción de agentes, proporcionó al investigador el conocimiento necesario para llevar a cabo la presente investigación y crear los diferentes modelos de agentes usados en el desarrollo de este trabajo de grado.

Por medio del análisis comparativo realizado sobre los diversos motores físicos usados con mayor frecuencia por los desarrolladores de videojuegos e

investigadores, se logró seleccionar el motor físico más adecuado para la investigación, tomando en cuenta principalmente la facilidad de integración con el motor de renderizado y la privacidad del código fuente. Esto evitó el uso de software privativo en la investigación, así como también, ayudó a reducir el tiempo de integración de las herramientas, ya que se contó con un componente que sirve de interfaz entre el motor de renderizado y el motor físico.

Haber seleccionado un motor físico existente en vez de programar uno propio, ayudó a mejorar el resultado de las interacciones físicas entre los objetos físicos y avatares presentes en el ambiente virtual dinámico, ya que se contó con características probadas por miles de investigadores y desarrolladores de video juegos, que de otra forma habrían tenido que ser probadas por el investigador del presente trabajo, lo que habría incrementado considerablemente el tiempo de la investigación.

El uso de la metodología MASINA, fue de gran importancia para la investigación, ya que ayudó a tener en claro las tareas, servicios y actividades que debían ser llevadas a cabo por cada uno de los agentes, facilitando de esta manera la integración y comunicación de los mismos al momento de resolver un problema determinado.

El desarrollo de la aplicación de pruebas, permitió validar el modelo de agentes desarrollado por medio del uso de la metodología MASINA, demostrando de esta manera que los agentes cumplen con las tareas que les han sido asignadas.

El uso de la metodología MAVD fue de gran importancia, ya que a través de cada una de sus fases se logró crear una aplicación que cumple con los requisitos planteados por el usuario, siguiendo el mismo hilo que se lleva en la grabación de una película. Al mismo tiempo, ayudó a crear una aplicación que iba siendo probada a medida que se iban cumpliendo cada una de las fases de la metodología.

Recomendaciones

Una vez finalizada la investigación, el autor del presente trabajo de grado considera realizar las siguientes recomendaciones, las cuales deben ser tomadas en cuentas en futuras investigaciones sobre este mismo trabajo.

1. Para mejorar el tiempo de respuesta del agente de Synchronization hacia los clientes, se debe agregar una tarea en el agentes CustomerService, de tal forma que sea este quien se encargue de enviar los datos hacia el cliente que atiende. Esto ayudará a distribuir el trabajo de sincronización entre todos los agentes CustomerService presentes en el servidor, reduciendo la carga del agente Synchronization.

2. Flexibilizar los agentes del sistema, incorporando un patrón de diseño de máquinas de estado a los mismos, de tal forma que exista un objeto que se encargue de realizar cada una de las tareas del agente. De esta forma, se podrán incorporar nuevas tareas a los agente, creando una nueva clase que se encargue de dicha tarea, lo que evitará tener que hacer muchas modificaciones a la aplicación, cuando se desee incorporar nuevos requerimientos.

3. Modificar el motor físico en el servidor para evitar el uso de OgreNewt y usar únicamente Newton, ya que debido a que este *wrapper* requiere tener en ejecución el motor de renderizado Ogre3D, el uso del procesador se verá afectado por esta característica, lo que puede disminuir el rendimiento del motor físico al momento de actualizar los objetos que estan siendo simulados físicamente, afectando de esta manera la sincronización de los clientes conectados al ambiente virtual.

4. Incorporar un modelo físico predictivo en los clientes, de tal forma, que los mismos no tengan que esperar por la respuesta del servidor para iniciar la

simulación. Posteriormente la simulación debe ser corregida y sincronizada con la información que llegue desde el servidor. Esta recomendación permitirá mejorar el rendimiento tanto del cliente como del servidor.

5. Otra recomendación que el autor considera bastante importante es la de mejorar la compatibilidad de la aplicación con otros sistemas operativos, con lo cual se evita limitar al usuario al uso estricto del sistema operativo Linux. Sólo habrá que arreglar el módulo de comunicación por socket, así como el módulo de programación multihilos, para hacerlos compatibles con los demás sistemas operativos, ya que tanto el motor de renderizado como el motor físico funcionan sobre diferentes plataformas.

6. La ultima recomendación que el autor realiza sobre la investigación, tiene que ver con la incorporación de sonido a la aplicación cliente, ya que de esta forma, se incrementará el realismo de los ambientes virtuales que serán servidos por la aplicación.

BIBLIOGRAFÍA

Aguilar, J., Bessembel, I., Cerrada, M., Hidrobo, F., & Narciso, F. (2003). Una Metodología para el Modelado de Sistemas de Ingeniería Orientado a Agentes. *Revista Iberoamericana de Inteligencia Artificial*, 34-52.

Aguilar, J., Bessembel, I., Cerrada, M., Hidrobo, F., & Narciso, F. (2008). Una Metodología para el Modelado de Sistemas de Ingeniería Orientado a Agentes. *Revista Iberoamericana de Inteligencia Artificial*, 34-52.

Aguilar, J., Rivas, F., & Cerrada, M. (2010). Sistemas multiagentes para la planificación y manejo de los factores de producción en automatización. *Ciencia e Ingeniería*, 13-24.

Badano, N. (2008). Mundos virtuales, lazos reales Hacia una comprensión de los modos de socialización que habilitan. *Tesina de grado, Universidad Nacional de Rosario*. Rosario, Santa Fe, Argentina.

Bourg, D. M. (2002). *Physics por Game Developers.* United States of America: O'Reilly & Associates.

Briggs, J. R. (2007). *Aprendiendo a Programar con Phyton.* recuperado de: http://www.briggs.net.nz/log/writing/snake-wrangling-for-kids/.

Buss, S. R. (2003). *3-D Computer Graphics. A Mathematical Introduction with OpenGL.* New York: Cambridge University Press.

Caphalor. (2011). *OgrePhysX.* Recuperado el 20 de 02 de 2012, de http://code.google.com/p/ice3d/

Castronova, E. (2001). Virtual Worlds: A First-Hand Account of Market and Society on the Cyberian Frontier. *The Gruter Institute Working Papers on Law, Economics, and Evolutionary Biology*, 66.

Davison, A. (2005). *Killer Game Programming in Java.* O'Reilly.

Deck 13 Interactive. (12 de 11 de 2006). *Ankh.* Recuperado el 20 de 02 de 2012, de http://www.ankh-game.com/

Gahan, A. (2011). *3ds Max Modelling for Games (2ed).* Oxford: Focal Press.

Gallego, I., & Llinás, M. (2000). *Algorítmica y Programación para ingenieros.* Barcelona: Ediciones UPC.

Gónzales, C., & Vallejo, D. (s.f.). *Sistesís de Imagen 3D [Versión eléctronica].* Castilla-La Mancha.

Havok. (2011). *Havok.* Recuperado el 23 de 10 de 2011, de http://www.havok.com/products/physics

Hernandez H., D. (Diciembre de 2001). Arquitectura para ambientes virtuales dinámicos. *V Jornadas Nacionales de Computación Paralela y Distribuida.* Mérida, Venezuela.

Hernández, D., Barrios, J., & Gutierrez, D. (29 de 01 de 2010). Metodología para el desarrollo de un ambiente virtual dinámico. *II Jornadas de Desarrollo de Software*, pág. 20.

Iron Realms Entertainment. (2007). *Earth Eternal.* Recuperado el 20 de 02 de 2012, de http://www.eartheternal.com/

Jerez, J., & Suero, A. (2003). *Newton Game Dynamics.* Recuperado el 11 de 11 de 2011, de Newton Game Dynamics: http://newtondynamics.com/forum/newton.php

Jerez, J., & Suero, A. (2011). *Newton Game Dynamics.* Recuperado el 23 de 10 de 2011, de http://newtondynamics.com/forum/newton.php

La Tumba de Nefertari. (s.f.). Recuperado el 06 de 01 de 2010, de http://www.osirisnet.net/3d-tours/qv66/qv66-cort.wrl

Luca de Tena, J. I. (2007). *3ds Max 9 Essentials.* Madrid: Anaya.

Luna, L., Aguayo, J., & Rossodivita, A. (11 de 2005). *Aquitecturas de los DBMS.* Recuperado el 03 de 11 de 2011, de http://alfa.facyt.uc.edu.ve/computacion/pensum/cs0347/download/exposiciones2005-2006/Oracle POstgre MySQL.pdf

Millington, I. (2007). *Game Physics Engine Development.* New York: ELSEVIER.

Moctezuma, J. (2006). Manipulación Tridimensional de Objetos Deformables Virtuales. *Tesis de Maestría no publicada, Instituto Politécnico Nacional.* México, México.

Neider, J., Davis, T., & Woo, M. (1994). *OpenGL Programming Guide.* Recuperado el 18 de 10 de 2011, de http://fly.cc.fer.hr/~unreal/theredbook/chapter01.html

Nvidia Corporation. (2011). *PhysX Nvidia*. Recuperado el 20 de 10 de 2011, de http://developer.nvidia.com/physx

Ogre-Team. (31 de 07 de 2011). *Ogre Wiki.* Recuperado el 18 de 10 de 2011, de http://www.ogre3d.org/tikiwiki/Getting+Started

OpenFrag. (2011). *OpenFrag*. Recuperado el 03 de 11 de 2011, de http://sourceforge.net/projects/openfrag/

Ramos N., M. d., Larios D., J., Cervantes C., D., & Leriche V., R. (2007). Creación de ambientes virtuales inmersivos con software libre. *Revista Digital Universitaria*, 9.

Rios B., A., Cerrada, M., Narciso, F., Hidrobo, F., & Aguilar, J. (2008). Implantando Sistemas de Control con Agentes Inteligentes. *Revista Ciencia e Ingeniería*, 249-260.

Russell, S., & Norvig, P. (2003). *Inteligencia Artificial, Un Enfoque Moderno.* Mexico: Pearson.

Smith, R. (2007). *Open Dynamics Engine.* Recuperado el 23 de 10 de 2011, de http://ode.org/

Thilo, F. (30 de 10 de 2011). *MD5.* Recuperado el 05 de 11 de 2011, de http://pastebin.com/T2h8KzLM

Thomanson, L. (20 de 09 de 2010). *TinyXML.* Recuperado el 05 de 07 de 2011, de http://sourceforge.net/projects/tinyxml/

Tradky. (2009). *Museo Virtual.* Recuperado el 08 de 10 de 2011, de http://www.tradky.com/museovirtual/3d.php

Walsh, P. (2003). *Advanced 3D Game Programming Using DirectX 9.0* . Plano: Wordware Publishing.

Walton, S. (2001). *Programación de Socket con Linux.* Madrid: Prentice Hall.

Weiss, G. (1999). *Multiagent systems: a modern approach to distributed artificial intelligence.* Cambridge: The MIT Press.

Wikipedia. (27 de 09 de 2011). *Physics engine*. Recuperado el 19 de 10 de 2011, de http://es.wikipedia.org/w/index.php?title=Physics_engine&oldid=50100240

Wikipedia. (29 de 12 de 2011). *Zlib License*. Recuperado el 22 de 04 de 2012, de

http://es.wikipedia.org/w/index.php?title=Zlib_License&oldid=52564150

Wooldridge, M., & Jennings, N. (1995). Intelligent agents: theory and practice. *The Knowledge Engineering Review, Vol 10*, 115-152.

ANEXO A. Instalación de Ogre3D

La instalación del motor de renderizado se llevó a cabo siguiendo los siguientes pasos:

I. Instalar el compilador y herramientas de configuración
   sudo apt-get install build-essential automake libtool cmake-gui

II. Instalar dependencias de Ogre
   sudo apt-get install libois-dev libfreeimage-dev libfreetype6-dev libzzip-dev libxaw7-dev libglew1.5-dev libxrandr-dev libboost-thread-dev freeglut3-dev

III. Instalar otros paquetes usados por ogre (optional)
   sudo apt-get install doxygen graphviz nvidia-cg-toolkit libboost-dev
   sudo apt-get install aplibcppunit-dev

IV. Instalar dependencias para el gestor de ventanas CEGUI (solo si se quiere habilitar CEGUI)
   sudo apt-get install libpcre++-dev libwxgtk2.8-dev libjpeg62-dev
   sudo apt-get install libpng3-dev

V. Otros paquetes usados por CEGUI (optional)
   sudo apt-get install doxygen graphviz

VI. Añadir el repositorio PPA del "team-ogre" al sistema:
   sudo add-apt-repository ppa:ogre-team/ogre

VII. Para actualizar tu lista de paquetes disponibles
   sudo apt-get update

VIII. Instalar ogre

```
sudo apt-get install libogre-dev ogre-doc ogre-samples-bin
sudo apt-get install ogre-samples-source ogre-tools
```

ANEXO B. Instalación de CEGUI

CEGUI es una librería que permite agregar componentes y ventanas a las aplicaciones gráficas que no cuentan con esa funcionalidad. Los pasos para su instalación se enumeran a continuación.

I. Compilar e instalar SILLY (Solo si se quiere instalar el CELayoutEditor para CEGUI)

Descargar la última versión de SILLY desde http://www.cegui.org.uk

Extraer el paquete abrir una consola y entrar al directorio

./configure

make

sudo make install

II. Compilar e instalar CEGUI

Descargar la última versión de cegui desde http://www.cegui.org.uk

Extraer el paquete abrir una consola y entrar al directorio

./bootstrap

./configure

make

sudo make install

III. Agregar al final del archivo /etc/ld.so.conf el texto /usr/local/lib #esto se hace con sudo gedit /etc/ld.so.conf &

IV. Recargar las librerías

sudo ldconfig

V. Instalar CELayoutEditor (Opcional).

Descargar la última versión de CELayoutEditor desde http://www.cegui.org.uk

Extraer el paquete abrir una consola y entrar al directorio

./configure

make

sudo make install

ANEXO C. Instalación de Newton desde repositorio

I. Abrir un terminal y ubicarse en algun directorio donde desee descargar el código.

II. Descargar la última revisión newton usando el svn http://newton-dynamics.googlecode.com/svn/trunk/. Ejecutar en el terminal
svn co -r 1003 http://newton-dynamics.googlecode.com/svn/trunk newton-dynamics

III. dirigase al directorio newton-dynamic/coreLibrady_200/projets/linux32

IV. Abra el makefile con un editor de texto y cambie la línea
CPU_FLAGS = -O2 -fpic -msse -msse2 -ffloat-store -ffast-math -freciprocal-math -funsafe-math-optimizations -fsingle-precision-constant

Por

CPU_FLAGS = -O2 -fpic -msse -msse2 -ffloat-store -freciprocal-math -funsafe-math-optimizations -fsingle-precision-constant

V. En la regla libNewton.a, busque donde aparezca Linux32 y cambielo por linux32 (Hay 3 ocurrencias)

VI. Guarde los cambios y cierre el archivo.

VII. Ejecute el comando make en el terminal
make

VIII. Cree el directorio linux32 en la carpeta packages con el siguiente comando:
mkdir ../../../packages/linux32

IX. Hacer los mismos cambios en los makefiles de los paquetes dContainers, dCustomJoints, dMath, dScene y thirdParty/tinyxml los cuales se encuentran en newton-dynamics/packages/projects/linux32

X. Ingresar a cada directorio y Compilar cada una de las librerias

XI. dirijase al directorio newton-dynamics/coreLibrady_200/packages/

XII. Crear el directorio /usr/include/newtonSDK

```
sudo mkdir /usr/include/newtonSDK
```

XIII. Copiar los directorios dContainers, dCustomJoints, dMath, dScene y thirdParty en /usr/include/newtonSDK

```
sudo cp -r dContainers dCustomJoint dMath dScene thirdParty /usr/include/newtonSDK
```


XIV. Copiar los archivos .a y .so desde la carpeta linux32 hasta /usr/lib

```
sudo cp linux32/lib*.* /usr/lib
```

XV. Copiar Newton.h en /usr/include/newtonSDK

```
sudo cp linux32/Newton.h /usr/include/newtonSDK
```

XVI. Ejecutar ldconfig

```
sudo ldconfig
```

ANEXO D. Instalación de OgreNewt

I. Descargar OgreNewt desde el svn.
   svn co https://svn.ogre3d.org/svnroot/ogreaddons/branches/ogrenewt/newton20

II. Abrir una consola y cambiar al directorio newton20

III. Abrir el archivo CMakeList.txt y asegurarse que en la sección OgreNewt_LIB_SRCS e INSTALL(FILES se encuentren los .cpp y .h y de RayCastVehicle y ConvexCast respectivamente. de no estar se deben agregar manualmente.

IV. Ejecutar los siguientes comandos
   mkdir build
   cd build
   cmake ..
   make
   sudo make install

ANEXO E. Instalación de PostgresSQL 9.0 y libpq

La instalación de estas herramientas se llevó a cabo de la siguiente manera.

I. Agregar el repositorio para postgres y actualizar

sudo add-apt-repository ppa:pitti/postgresql

sudo apt-get update

II. Instalar los paquetes

sudo apt-get install libpq-dev libpqxx3-dev

III. Solo en el servidor instalar postgres9

sudo apt-get install postgresql-9.0

ANEXO F. Archivo de configuración del servidor

```
<?xml version="1.0" encoding="UTF-8" ?>
<PhysicDVEConfiguration>
   <Database>
         <DBName>PhysicsDVE</DBName>
         <host>localhost</host>
         <user>postgres</user>
         <password>123456</password>
         <DBport>5432</DBport>
          <enviroment>Castle</enviroment>
   </Database>
   <Physical>
      <coldata>/home/mbsanchez/Tesis/code/Ogre3D/media/commons/coldata</coldata>
   </Physical>
   <DefaultMaterials>
      <ground>2</ground>
   </DefaultMaterials>
</PhysicDVEConfiguration>
```


En este archivo de configuración se especifican los elementos necesarios para establecer la conexión al servidor de base de datos, la ruta donde se encuentran guardados los archivos de colisión de los objetos estáticos y el material que será usado por defecto por el piso del ambiente virtual.

ANEXO G. Archivo de configuración del cliente

```
<?xml version="1.0" encoding="UTF-8" ?>
<PhysicDVEConfiguration>
   <Server>
      <host>lsiriusblack.local</host>
      <port>6800</port>
   </Server>
</PhysicDVEConfiguration>
```

En el archivo de configuración para el cliente se específica en formato XML los datos necesarios para establecer la conexión con el servidor.

ANEXO H. Archivo de recursos para Ogre3D

```
# Resource locations to be added to the 'boostrap' path
# This also contains the minimum you need to use the Ogre example framework
[Bootstrap]
Zip=/home/mbsanchez/Tesis/code/Ogre3D/media/packs/OgreCore.zip

# Resources required by the sample browser and most samples.
[Essential]
Zip=/usr/share/OGRE/media/packs/SdkTrays.zip
FileSystem=/usr/share/OGRE/media/thumbnails

# Common sample resources needed by many of the samples.
# Rarely used resources should be separately loaded by the
# samples which require them.
[Popular]
FileSystem=/usr/share/OGRE/media/fonts
FileSystem=/usr/share/OGRE/media/materials/programs
FileSystem=/usr/share/OGRE/media/materials/scripts
FileSystem=/usr/share/OGRE/media/materials/textures
FileSystem=/usr/share/OGRE/media/materials/textures/nvidia
FileSystem=/usr/share/OGRE/media/models
FileSystem=/usr/share/OGRE/media/particle
FileSystem=/usr/share/OGRE/media/DeferredShadingMedia
FileSystem=/usr/share/OGRE/media/PCZAppMedia
FileSystem=/usr/share/OGRE/media/RTShaderLib
FileSystem=/usr/share/OGRE/media/RTShaderLib/materials
Zip=/usr/share/OGRE/media/packs/cubemap.zip
Zip=/usr/share/OGRE/media/packs/cubemapsJS.zip
Zip=/usr/share/OGRE/media/packs/dragon.zip
Zip=/usr/share/OGRE/media/packs/fresneldemo.zip
```

```
Zip=/usr/share/OGRE/media/packs/ogretestmap.zip
Zip=/usr/share/OGRE/media/packs/ogredance.zip
Zip=/usr/share/OGRE/media/packs/Sinbad.zip
Zip=/usr/share/OGRE/media/packs/skybox.zip
[General]
FileSystem=/usr/share/OGRE/media
[Imagesets]
FileSystem=/usr/local/share/CEGUI/imagesets
FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/icons
[Fonts]
FileSystem=/usr/local/share/CEGUI/fonts
FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/fonts
[Schemes]
FileSystem=/usr/local/share/CEGUI/schemes
[XMLSchemes]
FileSystem=/usr/local/share/CEGUI/xml_schemas
[LookNFeel]
FileSystem=/usr/local/share/CEGUI/looknfeel
[Layouts]
FileSystem=/usr/local/share/CEGUI/layouts
[Media]
FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/material/textures
FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/material/scripts
FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/models/characters
[MediaCommons]
FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/commons/material/textures
FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/commons/material/scripts
FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/commons/models/
FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/commons/coldata/
FileSystem=/home/mbsanchez/apps/newtonSDK/newton20/demos/media/primitives
[Castle]
FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/material/textures/castle
```

FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/material/scripts/castle

FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/models/enviroments/castle

[Chiropteradm]

FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/material/textures/chiropteradm

FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/material/scripts/chiropteradm

FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/models/enviroments/chiropterad
m

[nefertari]

FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/material/textures/nefertari

FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/material/scripts/nefertari

FileSystem=/home/mbsanchez/Tesis/code/Ogre3D/media/models/enviroments/nefertari

Este archivo de recursos guarda todo lo referente a los distintos recursos que utiliza el sistema de renderizado 3d, así como el gestor de ventanas (texturas, modelos 3d, sonidos, fuentes, entre otros) y que permiten localizar la información fácilmente. Cada ambiente virtual es configurado en un grupo diferente, señalando la ubicación de los scripts, texturas y modelos, de está forma, se cargan solo las texturas que correspondan al ambiente virtual que será renderizado.

ANEXO I. Sentencias SQL de la creación de la base de datos

```
--
-- PostgreSQL database dump
--

-- Dumped from database version 9.0.4
-- Dumped by pg_dump version 9.0.4
-- Started on 2011-12-01 19:03:18 VET

SET statement_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = off;
SET check_function_bodies = false;
SET client_min_messages = warning;
SET escape_string_warning = off;
```



```
--
-- TOC entry 328 (class 2612 OID 11574)
-- Name: plpgsql; Type: PROCEDURAL LANGUAGE; Schema: -; Owner: postgres
--

CREATE OR REPLACE PROCEDURAL LANGUAGE plpgsql;


ALTER PROCEDURAL LANGUAGE plpgsql OWNER TO postgres;

SET search_path = public, pg_catalog;

--
-- TOC entry 18 (class 1255 OID 16393)
```

```
-- Dependencies: 6 328
-- Name: max_avatar_per_user(); Type: FUNCTION; Schema: public; Owner:
postgres
--

CREATE FUNCTION max_avatar_per_user() RETURNS trigger
    LANGUAGE plpgsql
    AS $$
DECLARE
  register RECORD;
  max_avatars INTEGER;
BEGIN
  max_avatars = 5;
  SELECT INTO register COUNT(*) AS current_avatars
  FROM user_avatar
  WHERE user_avatar.owner_user = NEW.owner_user;
  IF register.current_avatars < max_avatars THEN
    RETURN NEW;
  ELSE
    RAISE EXCEPTION 'No pueden existir más de 5 avatares por usuario';
    RETURN NULL;
  END IF;
END;
$$;


ALTER FUNCTION public.max_avatar_per_user() OWNER TO postgres;

SET default_tablespace = '';
```

```
SET default_with_oids = false;

--
-- TOC entry 1520 (class 1259 OID 16394)
-- Dependencies: 1808 6
-- Name: animation; Type: TABLE; Schema: public; Owner: postgres; Tablespace:
--

CREATE TABLE animation (
    animation_id integer NOT NULL,
    denomination character varying(50) NOT NULL,
    can_loop boolean DEFAULT false NOT NULL
);


ALTER TABLE public.animation OWNER TO postgres;
```



```
--
-- TOC entry 1521 (class 1259 OID 16398)
-- Dependencies: 6
-- Name: app_user; Type: TABLE; Schema: public; Owner: postgres; Tablespace:
--

CREATE TABLE app_user (
    username character varying(20) NOT NULL,
    real_name character varying(30) NOT NULL,
    email character varying(30) NOT NULL,
    passwd character varying(50) NOT NULL
);
```

```
ALTER TABLE public.app_user OWNER TO postgres;

--
-- TOC entry 1522 (class 1259 OID 16401)
-- Dependencies: 1809 1810 1811 6
-- Name: avatar; Type: TABLE; Schema: public; Owner: postgres; Tablespace:
--

CREATE TABLE avatar (
    object_id integer NOT NULL,
    xrotation real DEFAULT 0 NOT NULL,
    yrotation real DEFAULT 0 NOT NULL,
    zrotation real DEFAULT 0 NOT NULL
);
```



```
ALTER TABLE public.avatar OWNER TO postgres;

--
-- TOC entry 1523 (class 1259 OID 16407)
-- Dependencies: 1812 1813 1814 1815 1816 1817 6
-- Name: enviroment; Type: TABLE; Schema: public; Owner: postgres;
Tablespace:
--

CREATE TABLE enviroment (
    object_id integer NOT NULL,
    enviroment_name character varying(20) NOT NULL,
    xentry real DEFAULT 0 NOT NULL,
    yentry real DEFAULT 0 NOT NULL,
    zentry real DEFAULT 0 NOT NULL,
```

```
    xlocation real DEFAULT 0,
    ylocation real DEFAULT 0,
    zlocation real DEFAULT 0
);


ALTER TABLE public.enviroment OWNER TO postgres;

--
-- TOC entry 1524 (class 1259 OID 16413)
-- Dependencies: 1818 1819 1820 1821 1822 1823 6
-- Name: enviroment_item_object; Type: TABLE; Schema: public; Owner:
postgres; Tablespace:
--

CREATE TABLE enviroment_item_object (
    item_object_object_id integer NOT NULL,
    enviroment_object_id integer NOT NULL,
    xlocation real DEFAULT 0 NOT NULL,
    ylocation real DEFAULT 0 NOT NULL,
    zlocation real DEFAULT 0 NOT NULL,
    xrotation real DEFAULT 0 NOT NULL,
    yrotation real DEFAULT 0 NOT NULL,
    zrotation real DEFAULT 0 NOT NULL,
    object_number integer NOT NULL
);


ALTER TABLE public.enviroment_item_object OWNER TO postgres;

--
```

```
-- TOC entry 1525 (class 1259 OID 16422)
-- Dependencies: 6
-- Name: item_object; Type: TABLE; Schema: public; Owner: postgres; Tablespace:
--

CREATE TABLE item_object (
    object_id integer NOT NULL,
    weight real NOT NULL,
    ptype integer
);


ALTER TABLE public.item_object OWNER TO postgres;

--
-- TOC entry 1529 (class 1259 OID 16538)
-- Dependencies: 6
-- Name: material; Type: TABLE; Schema: public; Owner: postgres; Tablespace:
--

CREATE TABLE material (
    id integer NOT NULL,
    material_name character varying(20) NOT NULL
);


ALTER TABLE public.material OWNER TO postgres;

--
-- TOC entry 1530 (class 1259 OID 16548)
```

```
-- Dependencies: 1828 1829 1830 1831 1832 1833 1834 1835 1836 1837 1838
1839 6
-- Name: material_pair; Type: TABLE; Schema: public; Owner: postgres;
Tablespace:
--

CREATE TABLE material_pair (
    material_one_id integer DEFAULT 0 NOT NULL,
    material_two_id integer DEFAULT 0 NOT NULL,
    elasticity real DEFAULT 0.4,
    static_friction real DEFAULT 0.9,
    kinetic_friction real DEFAULT 0.5 NOT NULL,
    thickness real DEFAULT 0,
    softness real DEFAULT 0.1,
    CONSTRAINT check_elasticity CHECK (((elasticity >= (0)::double precision)
AND (elasticity <= (1)::double precision))),
    CONSTRAINT check_kinetic_friction CHECK (((kinetic_friction >= (0.01)::double
precision) AND (kinetic_friction <= (2.0)::double precision))),
    CONSTRAINT check_softness CHECK (((softness >= (0.01)::double precision)
AND (softness <= (1.0)::double precision))),
    CONSTRAINT check_static_friction CHECK (((static_friction >= (0.01)::double
precision) AND (static_friction <= (2.0)::double precision))),
    CONSTRAINT check_thickness CHECK (((thickness >= (0)::double precision)
AND (thickness <= (0.125)::double precision)))
);


ALTER TABLE public.material_pair OWNER TO postgres;

--
-- TOC entry 1902 (class 0 OID 0)
```

-- Dependencies: 1530
-- Name: CONSTRAINT check_elasticity ON material_pair; Type: COMMENT; Schema: public; Owner: postgres
--

COMMENT ON CONSTRAINT check_elasticity ON material_pair IS 'La elasticidad debe ser un valor entre 0 y 1';

--
-- TOC entry 1903 (class 0 OID 0)
-- Dependencies: 1530
-- Name: CONSTRAINT check_kinetic_friction ON material_pair; Type: COMMENT; Schema: public; Owner: postgres
--

COMMENT ON CONSTRAINT check_kinetic_friction ON material_pair IS 'La fricción dinámica debe ser un valor entre 0.01 y 2.0';

--
-- TOC entry 1904 (class 0 OID 0)
-- Dependencies: 1530
-- Name: CONSTRAINT check_softness ON material_pair; Type: COMMENT; Schema: public; Owner: postgres
--

COMMENT ON CONSTRAINT check_softness ON material_pair IS 'La dureza debe ser un valor entre 0.01 y 1.0';

```
--
-- TOC entry 1905 (class 0 OID 0)
-- Dependencies: 1530
-- Name: CONSTRAINT check_static_friction ON material_pair; Type: COMMENT;
Schema: public; Owner: postgres
--

COMMENT ON CONSTRAINT check_static_friction ON material_pair IS 'La
fricción estática debe ser un valor entre 0.01 y 2.0';



--
-- TOC entry 1906 (class 0 OID 0)
-- Dependencies: 1530
-- Name: CONSTRAINT check_thickness ON material_pair; Type: COMMENT;
Schema: public; Owner: postgres
--

COMMENT ON CONSTRAINT check_thickness ON material_pair IS 'La
separación debe ser un valor entre 0 y 0.125';



--
-- TOC entry 1526 (class 1259 OID 16425)
-- Dependencies: 1824 1825 1826 6
-- Name: mesh_object; Type: TABLE; Schema: public; Owner: postgres;
Tablespace:
--

CREATE TABLE mesh_object (
    object_id integer NOT NULL,
```

```
    mesh_name character varying(50) NOT NULL,
    xscale real DEFAULT 1 NOT NULL,
    yscale real DEFAULT 1 NOT NULL,
    zscale real DEFAULT 1 NOT NULL,
    material_id integer NOT NULL
);


ALTER TABLE public.mesh_object OWNER TO postgres;

--
-- TOC entry 1528 (class 1259 OID 16528)
-- Dependencies: 6
-- Name: primitive_type; Type: TABLE; Schema: public; Owner: postgres;
Tablespace:
--
```



```
CREATE TABLE primitive_type (
    ptype integer NOT NULL,
    description character varying(50) NOT NULL
);


ALTER TABLE public.primitive_type OWNER TO postgres;

--
-- TOC entry 1527 (class 1259 OID 16431)
-- Dependencies: 1827 6
-- Name: user_avatar; Type: TABLE; Schema: public; Owner: postgres;
Tablespace:
--
```

```
CREATE TABLE user_avatar (
    avatar_name character varying(20) NOT NULL,
    owner_user character varying(20) NOT NULL,
    current_anim_id integer NOT NULL,
    avatar_object_id integer NOT NULL,
    is_admin boolean DEFAULT false NOT NULL
);


ALTER TABLE public.user_avatar OWNER TO postgres;

--
-- TOC entry 1886 (class 0 OID 16394)
-- Dependencies: 1520
-- Data for Name: animation; Type: TABLE DATA; Schema: public; Owner:
postgres
--

INSERT INTO animation VALUES (1, 'Stop2Walk', false);
INSERT INTO animation VALUES (2, 'LWalk', true);
INSERT INTO animation VALUES (3, 'Walk2Stop', false);
INSERT INTO animation VALUES (4, 'LIdle1', true);
INSERT INTO animation VALUES (5, 'KickBall', false);
INSERT INTO animation VALUES (6, 'FrontKick', false);
INSERT INTO animation VALUES (7, 'Punch', false);
INSERT INTO animation VALUES (8, 'Stop2PushOneLeg', false);
INSERT INTO animation VALUES (9, 'LPushOneLeg', true);
INSERT INTO animation VALUES (10, 'PushOneLeg2Stop', false);
INSERT INTO animation VALUES (11, 'Stop2PushWalking', false);
INSERT INTO animation VALUES (12, 'LPushWalking', true);
```

```
INSERT INTO animation VALUES (13, 'PushWalking2Stop', false);
INSERT INTO animation VALUES (14, 'PushNoMove', false);
INSERT INTO animation VALUES (15, 'Stop2BackWalk', false);
INSERT INTO animation VALUES (16, 'LBackWalk', true);
INSERT INTO animation VALUES (17, 'BackWalk2Stop', false);
INSERT INTO animation VALUES (18, 'LIdle2', true);
INSERT INTO animation VALUES (19, 'LIdle3', true);


--
-- TOC entry 1887 (class 0 OID 16398)
-- Dependencies: 1521
-- Data for Name: app_user; Type: TABLE DATA; Schema: public; Owner: postgres
--

INSERT INTO app_user VALUES ('mbsanchez', 'Manuel B. Sanchez', 'mbsanchez@unet.edu.ve', '2a885b51d255fa8779bfc00df0d7a060');
INSERT INTO app_user VALUES ('admin', 'Administrador', 'root@localhost', '21232f297a57a5a743894a0e4a801fc3');
INSERT INTO app_user VALUES ('domingo.hernandez', 'Domingo Hernandez', 'dhh@ula.ve', 'e10adc3949ba59abbe56e057f20f883e');


--
-- TOC entry 1888 (class 0 OID 16401)
-- Dependencies: 1522
-- Data for Name: avatar; Type: TABLE DATA; Schema: public; Owner: postgres
--

INSERT INTO avatar VALUES (1, 0, 0, 0);
```

```
INSERT INTO avatar VALUES (2, 0, 0, 0);
INSERT INTO avatar VALUES (3, 0, 0, 0);
INSERT INTO avatar VALUES (7, 0, 0, 0);
INSERT INTO avatar VALUES (8, 0, 0, 0);
INSERT INTO avatar VALUES (9, 0, 0, 0);
INSERT INTO avatar VALUES (25, 0, 0, 0);


--
-- TOC entry 1889 (class 0 OID 16407)
-- Dependencies: 1523
-- Data for Name: enviroment; Type: TABLE DATA; Schema: public; Owner: postgres
--

INSERT INTO enviroment VALUES (11, 'Castle', 0, 0, 0, 0, 1, 0);
INSERT INTO enviroment VALUES (10, 'Chiropteradm', -9.67000008, 0.0199999996, -8.69999981, 0, 21, 0);
INSERT INTO enviroment VALUES (24, 'nefertari', -5.5, 10.4399996, 45.3600006, 0, 0, 0);


--
-- TOC entry 1890 (class 0 OID 16413)
-- Dependencies: 1524
-- Data for Name: enviroment_item_object; Type: TABLE DATA; Schema: public; Owner: postgres
--

INSERT INTO enviroment_item_object VALUES (19, 11, -1, 1, -4, 0, 1, 0, 1);
INSERT INTO enviroment_item_object VALUES (20, 11, -2, 1, -4, 0, 1, 0, 2);
```

```
INSERT INTO enviroment_item_object VALUES (23, 11, -7, 7, 0, 0, 1, 0, 3);
INSERT INTO enviroment_item_object VALUES (23, 11, 0, 7, -7, 0, 1, 0, 4);
INSERT INTO enviroment_item_object VALUES (23, 11, -2, 5, -7, 0, 1, 0, 5);
INSERT INTO enviroment_item_object VALUES (13, 11, 8, 0, -100, 0, 1, 0, 6);
INSERT INTO enviroment_item_object VALUES (12, 11, 3, 0, -98, 0, 1, 0, 7);
INSERT INTO enviroment_item_object VALUES (12, 11, 3, 0, -101, 0, -178, 0, 8);
INSERT INTO enviroment_item_object VALUES (15, 11, -14, 0, -101, 0, -178, 0,
9);
INSERT INTO enviroment_item_object VALUES (15, 11, -10, 0, -96, 0, -178, 0,
10);
INSERT INTO enviroment_item_object VALUES (15, 11, -16, 0, -96, 0, -178, 0,
11);
INSERT INTO enviroment_item_object VALUES (16, 11, -15, 0, -115, 0, -178, 0,
12);
INSERT INTO enviroment_item_object VALUES (16, 11, -23, 0, -92, 0, -178, 0,
13);
INSERT INTO enviroment_item_object VALUES (16, 11, 35, 0, -97, 0, -178, 0, 14);
INSERT INTO enviroment_item_object VALUES (17, 11, -78, 0, -492, 0, -178, 0,
15);
INSERT INTO enviroment_item_object VALUES (13, 11, -49, 0, -74, 0, 90, 0, 16);
INSERT INTO enviroment_item_object VALUES (22, 11, -45, 0, -78, 0, 90, 0, 17);
INSERT INTO enviroment_item_object VALUES (22, 11, -48, 0, -78, 0, 90, 0, 18);
INSERT INTO enviroment_item_object VALUES (23, 11, 0, 0, -85, 0, 90, 0, 19);
INSERT INTO enviroment_item_object VALUES (19, 11, -10, 0, -98, 0, 90, 0, 20);
INSERT INTO enviroment_item_object VALUES (14, 11, 0, 0, -103, 0, 90, 0, 21);
INSERT INTO enviroment_item_object VALUES (14, 11, -2, 0, -102, 0, 90, 0, 22);
INSERT INTO enviroment_item_object VALUES (14, 11, -5, 0, -104, 0, 90, 0, 23);
INSERT INTO enviroment_item_object VALUES (14, 11, -3, 0, -108, 0, 90, 0, 24);
INSERT INTO enviroment_item_object VALUES (21, 11, -44, 0, -66, 0, 90, 0, 25);
INSERT INTO enviroment_item_object VALUES (21, 11, -40, 0, -60, 0, 90, 0, 26);
INSERT INTO enviroment_item_object VALUES (23, 11, 2.55825996, 1.30654001,
```

-3.69629002, 0, 1, 0, 27);
INSERT INTO enviroment_item_object VALUES (20, 11, 28.9162006, 2.15389999e-08, -84.4889984, 0, 1, 0, 28);
INSERT INTO enviroment_item_object VALUES (12, 11, -6.82091999, 2.92523001e-08, 7.66251993, 0, 136, 0, 29);
INSERT INTO enviroment_item_object VALUES (21, 11, 16.4167995, 5.43062019, -47.074501, 0, 1, 0, 30);
INSERT INTO enviroment_item_object VALUES (22, 10, -4.85427999, 3.37546986e-08, -3.61510992, 0, 1, 0, 1);
INSERT INTO enviroment_item_object VALUES (14, 10, -7.27194023, -1.54120997e-07, -12.7643003, 0, 91, 0, 2);
INSERT INTO enviroment_item_object VALUES (12, 10, -11.1602001, -2.72458998e-08, -6.31282997, 0, 91, 0, 3);
INSERT INTO enviroment_item_object VALUES (21, 10, -6.94750977, 4.55073987e-08, -1.13047004, 0, 91, 0, 4);
INSERT INTO enviroment_item_object VALUES (23, 10, -5.48024988, 2.69062994e-09, -11.1927996, 0, 91, 0, 5);

--
-- TOC entry 1891 (class 0 OID 16422)
-- Dependencies: 1525
-- Data for Name: item_object; Type: TABLE DATA; Schema: public; Owner: postgres
--

INSERT INTO item_object VALUES (13, 0, 10);
INSERT INTO item_object VALUES (14, 8, 7);
INSERT INTO item_object VALUES (15, 0, 10);
INSERT INTO item_object VALUES (16, 0, 10);
INSERT INTO item_object VALUES (17, 0, 10);

```
INSERT INTO item_object VALUES (18, 0, 10);
INSERT INTO item_object VALUES (19, 2, 7);
INSERT INTO item_object VALUES (20, 10, 7);
INSERT INTO item_object VALUES (22, 100, 7);
INSERT INTO item_object VALUES (23, 2, 7);
INSERT INTO item_object VALUES (21, 15, 7);
INSERT INTO item_object VALUES (12, 10, 7);


--
-- TOC entry 1895 (class 0 OID 16538)
-- Dependencies: 1529
-- Data for Name: material; Type: TABLE DATA; Schema: public; Owner: postgres
--

INSERT INTO material VALUES (0, 'MADERA');
INSERT INTO material VALUES (1, 'CUERO');
INSERT INTO material VALUES (2, 'CONCRETO');
INSERT INTO material VALUES (3, 'GOMA');
INSERT INTO material VALUES (4, 'METAL');
INSERT INTO material VALUES (5, 'VIDRIO');
INSERT INTO material VALUES (6, 'PLASTICO');
INSERT INTO material VALUES (7, 'AVATAR_MAT');


--
-- TOC entry 1896 (class 0 OID 16548)
-- Dependencies: 1530
-- Data for Name: material_pair; Type: TABLE DATA; Schema: public; Owner:
postgres
--
```

```
INSERT INTO material_pair VALUES (0, 1, 0.300000012, 0.400000006,
0.300000012, 0, 0.100000001);
INSERT INTO material_pair VALUES (0, 3, 0.680000007, 0.519999981,
0.50999999, 0, 0.100000001);
INSERT INTO material_pair VALUES (0, 4, 0.379999995, 0.400000006, 0.5, 0,
0.100000001);
INSERT INTO material_pair VALUES (0, 5, 0.200000003, 0.400000006,
0.419999987, 0, 0.100000001);
INSERT INTO material_pair VALUES (0, 6, 0.379999995, 0.200000003, 0.25, 0,
0.100000001);
INSERT INTO material_pair VALUES (1, 1, 0.449999988, 0.300000012,
0.400000006, 0, 0.100000001);
INSERT INTO material_pair VALUES (1, 3, 0.680000007, 0.649999976,
0.639999986, 0, 0.100000001);
INSERT INTO material_pair VALUES (1, 2, 0.400000006, 0.600000024,
0.649999976, 0, 0.100000001);
INSERT INTO material_pair VALUES (1, 4, 0.479999989, 0.600000024,
0.589999974, 0, 0.100000001);
INSERT INTO material_pair VALUES (1, 5, 0.200000003, 0.300000012,
0.349999994, 0, 0.100000001);
INSERT INTO material_pair VALUES (1, 6, 0.349999994, 0.649999976,
0.629999995, 0, 0.100000001);
INSERT INTO material_pair VALUES (2, 2, 0.150000006, 0.720000029,
0.709999979, 0, 0.100000001);
INSERT INTO material_pair VALUES (2, 4, 0.150000006, 0.620000005,
0.649999976, 0, 0.100000001);
INSERT INTO material_pair VALUES (2, 5, 0.150000006, 0.569999993,
0.560000002, 0, 0.100000001);
INSERT INTO material_pair VALUES (2, 6, 0.150000006, 0.649999976,
0.629999995, 0, 0.100000001);
```

```
INSERT INTO material_pair VALUES (3, 3, 0.680000007, 0.620000005,
0.610000014, 0, 0.100000001);
INSERT INTO material_pair VALUES (3, 4, 0.680000007, 0.419999987,
0.409999996, 0, 0.100000001);
INSERT INTO material_pair VALUES (3, 5, 0.449999988, 0.449999988,
0.439999998, 0, 0.100000001);
INSERT INTO material_pair VALUES (3, 6, 0.680000007, 0.389999986,
0.370000005, 0, 0.100000001);
INSERT INTO material_pair VALUES (4, 4, 0.150000006, 0.319999993,
0.300000012, 0, 0.100000001);
INSERT INTO material_pair VALUES (4, 5, 0.150000006, 0.5, 0.5, 0,
0.100000001);
INSERT INTO material_pair VALUES (4, 6, 0.150000006, 0.519999981,
0.50999999, 0, 0.100000001);
INSERT INTO material_pair VALUES (5, 5, 0.200000003, 0.899999976,
0.899999976, 0, 0.100000001);
INSERT INTO material_pair VALUES (5, 6, 0.200000003, 0.800000012,
0.819999993, 0, 0.100000001);
INSERT INTO material_pair VALUES (6, 6, 0.200000003, 0.610000014,
0.600000024, 0, 0.100000001);
INSERT INTO material_pair VALUES (0, 0, 0.25, 1.10000002, 0.699999988, 0,
0.100000001);
INSERT INTO material_pair VALUES (2, 3, 0.680000007, 0.819999993,
0.810000002, 0, 0.100000001);
INSERT INTO material_pair VALUES (0, 2, 0.25, 0.620000005, 0.610000014, 0,
0.100000001);
INSERT INTO material_pair VALUES (7, 7, 0, 2, 2, 0, 1);
INSERT INTO material_pair VALUES (0, 7, 0, 0.899999976, 0.5, 0, 0.100000001);
INSERT INTO material_pair VALUES (1, 7, 0, 0.899999976, 0.5, 0, 0.100000001);
INSERT INTO material_pair VALUES (2, 7, 0, 0.899999976, 0.5, 0, 0.100000001);
INSERT INTO material_pair VALUES (3, 7, 0, 0.899999976, 0.5, 0, 0.100000001);
```

INSERT INTO material_pair VALUES (4, 7, 0, 0.899999976, 0.5, 0, 0.100000001);
INSERT INTO material_pair VALUES (5, 7, 0, 0.899999976, 0.5, 0, 0.100000001);
INSERT INTO material_pair VALUES (6, 7, 0, 0.899999976, 0.5, 0, 0.100000001);

--
-- TOC entry 1892 (class 0 OID 16425)
-- Dependencies: 1526
-- Data for Name: mesh_object; Type: TABLE DATA; Schema: public; Owner: postgres
--

INSERT INTO mesh_object VALUES (23, 'soccerball.mesh', 0.0329999998, 0.0329999998, 0.0329999998, 3);
INSERT INTO mesh_object VALUES (21, 'box1.mesh', 0.0599999987, 0.0599999987, 0.0599999987, 0);
INSERT INTO mesh_object VALUES (1, 'people_female_01.mesh', 0.0132999998, 0.0132999998, 0.0132999998, 7);
INSERT INTO mesh_object VALUES (2, 'people_female_02.mesh', 0.0132999998, 0.0132999998, 0.0132999998, 7);
INSERT INTO mesh_object VALUES (3, 'people_female_03.mesh', 0.0132999998, 0.0132999998, 0.0132999998, 7);
INSERT INTO mesh_object VALUES (7, 'people_male_01.mesh', 0.0132999998, 0.0132999998, 0.0132999998, 7);
INSERT INTO mesh_object VALUES (8, 'people_male_02.mesh', 0.0132999998, 0.0132999998, 0.0132999998, 7);
INSERT INTO mesh_object VALUES (9, 'people_male_03.mesh', 0.0132999998, 0.0132999998, 0.0132999998, 7);
INSERT INTO mesh_object VALUES (24, 'qw66.mesh', 2, 2, 2, 2);
INSERT INTO mesh_object VALUES (25, 'people_male_04.mesh', 0.0132999998, 0.0132999998, 0.0132999998, 7);

```
INSERT INTO mesh_object VALUES (10, 'chiropteradm.mesh', 1, 1, 1, 2);
INSERT INTO mesh_object VALUES (11, 'castle.mesh', 1, 1, 1, 2);
INSERT INTO mesh_object VALUES (12, 'barrel.mesh', 1.33000004, 1.33000004, 1.33000004, 0);
INSERT INTO mesh_object VALUES (13, 'Blacksmith.mesh', 1.33000004, 1.33000004, 1.33000004, 2);
INSERT INTO mesh_object VALUES (14, 'rack.mesh', 1.33000004, 1.33000004, 1.33000004, 0);
INSERT INTO mesh_object VALUES (15, 'rock.mesh', 1.33000004, 1.33000004, 1.33000004, 2);
INSERT INTO mesh_object VALUES (16, 'tree1.mesh', 1.33000004, 1.33000004, 1.33000004, 0);
INSERT INTO mesh_object VALUES (17, 'tree2.tga.mesh', 1.33000004, 1.33000004, 1.33000004, 0);
INSERT INTO mesh_object VALUES (18, 'well.mesh', 1.33000004, 1.33000004, 1.33000004, 2);
INSERT INTO mesh_object VALUES (19, 'basketball.mesh', 0.00499999989, 0.00499999989, 0.00499999989, 3);
INSERT INTO mesh_object VALUES (20, 'bowlingball.mesh', 0.00700000022, 0.00700000022, 0.00700000022, 6);
INSERT INTO mesh_object VALUES (22, 'drum.mesh', 0.670000017, 0.670000017, 0.670000017, 4);


--
-- TOC entry 1894 (class 0 OID 16528)
-- Dependencies: 1528
-- Data for Name: primitive_type; Type: TABLE DATA; Schema: public; Owner: postgres
--
```

```
INSERT INTO primitive_type VALUES (1, 'BoxPrimitiveType');
INSERT INTO primitive_type VALUES (2, 'ConePrimitiveType');
INSERT INTO primitive_type VALUES (3, 'EllipsoidPrimitiveType');
INSERT INTO primitive_type VALUES (4, 'CapsulePrimitiveType');
INSERT INTO primitive_type VALUES (5, 'CylinderPrimitiveType');
INSERT INTO primitive_type VALUES (6, 'CompoundCollisionPrimitiveType');
INSERT INTO primitive_type VALUES (7, 'ConvexHullPrimitiveType');
INSERT INTO primitive_type VALUES (8, 'ConvexHullModifierPrimitiveType');
INSERT INTO primitive_type VALUES (9, 'ChamferCylinderPrimitiveType');
INSERT INTO primitive_type VALUES (10, 'TreeCollisionPrimitiveType');
INSERT INTO primitive_type VALUES (11, 'NullPrimitiveType');
INSERT INTO primitive_type VALUES (12, 'HeighFieldPrimitiveType');
INSERT INTO primitive_type VALUES (13, 'ScenePrimitiveType');
```



```
--
-- TOC entry 1893 (class 0 OID 16431)
-- Dependencies: 1527
-- Data for Name: user_avatar; Type: TABLE DATA; Schema: public; Owner: postgres
--

INSERT INTO user_avatar VALUES ('AVATAR02', 'admin', 1, 1, false);
INSERT INTO user_avatar VALUES ('AVATAR001', 'domingo.hernandez', 1, 1, false);
INSERT INTO user_avatar VALUES ('AVATAR002', 'domingo.hernandez', 1, 1, false);
INSERT INTO user_avatar VALUES ('AVATAR01', 'admin', 1, 1, true);


--
```

```
-- TOC entry 1841 (class 2606 OID 16435)
-- Dependencies: 1520 1520
-- Name: pk_animation; Type: CONSTRAINT; Schema: public; Owner: postgres;
Tablespace:
--

ALTER TABLE ONLY animation
    ADD CONSTRAINT pk_animation PRIMARY KEY (animation_id);


--
-- TOC entry 1843 (class 2606 OID 16437)
-- Dependencies: 1521 1521
-- Name: pk_app_user; Type: CONSTRAINT; Schema: public; Owner: postgres;
Tablespace:
--

ALTER TABLE ONLY app_user
    ADD CONSTRAINT pk_app_user PRIMARY KEY (username);


--
-- TOC entry 1846 (class 2606 OID 16439)
-- Dependencies: 1522 1522
-- Name: pk_avatar; Type: CONSTRAINT; Schema: public; Owner: postgres;
Tablespace:
--

ALTER TABLE ONLY avatar
    ADD CONSTRAINT pk_avatar PRIMARY KEY (object_id);
```

```
--
-- TOC entry 1850 (class 2606 OID 16441)
-- Dependencies: 1523 1523
-- Name: pk_enviroment; Type: CONSTRAINT; Schema: public; Owner: postgres;
Tablespace:
--

ALTER TABLE ONLY enviroment
    ADD CONSTRAINT pk_enviroment PRIMARY KEY (object_id);



--
-- TOC entry 1855 (class 2606 OID 16527)
-- Dependencies: 1524 1524 1524 1524
-- Name: pk_enviroment_item_object; Type: CONSTRAINT; Schema: public;
Owner: postgres; Tablespace:
--

ALTER TABLE ONLY enviroment_item_object
    ADD CONSTRAINT pk_enviroment_item_object PRIMARY KEY
(item_object_object_id, enviroment_object_id, object_number);



--
-- TOC entry 1858 (class 2606 OID 16445)
-- Dependencies: 1525 1525
-- Name: pk_item_object; Type: CONSTRAINT; Schema: public; Owner: postgres;
Tablespace:
--
```

```
ALTER TABLE ONLY item_object
    ADD CONSTRAINT pk_item_object PRIMARY KEY (object_id);


--
-- TOC entry 1870 (class 2606 OID 16542)
-- Dependencies: 1529 1529
-- Name: pk_material_id; Type: CONSTRAINT; Schema: public; Owner: postgres;
Tablespace:
--

ALTER TABLE ONLY material
    ADD CONSTRAINT pk_material_id PRIMARY KEY (id);


--
-- TOC entry 1872 (class 2606 OID 16554)
-- Dependencies: 1530 1530 1530
-- Name: pk_material_pair; Type: CONSTRAINT; Schema: public; Owner:
postgres; Tablespace:
--

ALTER TABLE ONLY material_pair
    ADD CONSTRAINT pk_material_pair PRIMARY KEY (material_one_id,
material_two_id);


--
-- TOC entry 1860 (class 2606 OID 16447)
-- Dependencies: 1526 1526
-- Name: pk_mesh_object; Type: CONSTRAINT; Schema: public; Owner:
```

```
postgres; Tablespace:
--

ALTER TABLE ONLY mesh_object
    ADD CONSTRAINT pk_mesh_object PRIMARY KEY (object_id);


--
-- TOC entry 1868 (class 2606 OID 16532)
-- Dependencies: 1528 1528
-- Name: pk_primitive_type; Type: CONSTRAINT; Schema: public; Owner:
postgres; Tablespace:
--

ALTER TABLE ONLY primitive_type
    ADD CONSTRAINT pk_primitive_type PRIMARY KEY (ptype);


--
-- TOC entry 1866 (class 2606 OID 16449)
-- Dependencies: 1527 1527 1527 1527 1527
-- Name: pk_user_avatar; Type: CONSTRAINT; Schema: public; Owner: postgres;
Tablespace:
--

ALTER TABLE ONLY user_avatar
    ADD CONSTRAINT pk_user_avatar PRIMARY KEY (avatar_name, owner_user,
current_anim_id, avatar_object_id);


--
```

```
-- TOC entry 1852 (class 2606 OID 16501)
-- Dependencies: 1523 1523
-- Name: unq_enviroment_name; Type: CONSTRAINT; Schema: public; Owner: postgres; Tablespace:
--

ALTER TABLE ONLY enviroment
    ADD CONSTRAINT unq_enviroment_name UNIQUE (enviroment_name);


--
-- TOC entry 1861 (class 1259 OID 16450)
-- Dependencies: 1527
-- Name: avatar_name_unique; Type: INDEX; Schema: public; Owner: postgres; Tablespace:
--

CREATE UNIQUE INDEX avatar_name_unique ON user_avatar USING btree (avatar_name);


--
-- TOC entry 1847 (class 1259 OID 16451)
-- Dependencies: 1523
-- Name: enviroment_name_unique; Type: INDEX; Schema: public; Owner: postgres; Tablespace:
--

CREATE UNIQUE INDEX enviroment_name_unique ON enviroment USING btree (enviroment_name);
```

```
--
-- TOC entry 1862 (class 1259 OID 16452)
-- Dependencies: 1527
-- Name: idx_avatar_animation; Type: INDEX; Schema: public; Owner: postgres;
Tablespace:
--

CREATE INDEX idx_avatar_animation ON user_avatar USING btree
(current_anim_id);


--
-- TOC entry 1844 (class 1259 OID 16453)
-- Dependencies: 1522
-- Name: idx_avatar_mesh_object; Type: INDEX; Schema: public; Owner:
postgres; Tablespace:
--

CREATE INDEX idx_avatar_mesh_object ON avatar USING btree (object_id);


--
-- TOC entry 1863 (class 1259 OID 16454)
-- Dependencies: 1527
-- Name: idx_avatar_user; Type: INDEX; Schema: public; Owner: postgres;
Tablespace:
--

CREATE INDEX idx_avatar_user ON user_avatar USING btree (owner_user);
```

--

-- TOC entry 1853 (class 1259 OID 16455)

-- Dependencies: 1524

-- Name: idx_enviroment_item_object_enviroment; Type: INDEX; Schema: public; Owner: postgres; Tablespace:

--

CREATE INDEX idx_enviroment_item_object_enviroment ON enviroment_item_object USING btree (enviroment_object_id);

--

-- TOC entry 1848 (class 1259 OID 16456)

-- Dependencies: 1523

-- Name: idx_enviroment_mesh_object; Type: INDEX; Schema: public; Owner: postgres; Tablespace:

--

CREATE INDEX idx_enviroment_mesh_object ON enviroment USING btree (object_id);

--

-- TOC entry 1856 (class 1259 OID 16457)

-- Dependencies: 1525

-- Name: idx_item_object_mesh_object; Type: INDEX; Schema: public; Owner: postgres; Tablespace:

--

CREATE INDEX idx_item_object_mesh_object ON item_object USING btree

(object_id);

--
-- TOC entry 1864 (class 1259 OID 16458)
-- Dependencies: 1527
-- Name: idx_user_avatar_avatar; Type: INDEX; Schema: public; Owner: postgres; Tablespace:
--

CREATE INDEX idx_user_avatar_avatar ON user_avatar USING btree (avatar_object_id);

--
-- TOC entry 1885 (class 2620 OID 16459)
-- Dependencies: 18 1527
-- Name: _trigger_max_avatar_per_user; Type: TRIGGER; Schema: public; Owner: postgres
--

CREATE TRIGGER _trigger_max_avatar_per_user BEFORE INSERT ON user_avatar FOR EACH ROW EXECUTE PROCEDURE max_avatar_per_user();

--
-- TOC entry 1880 (class 2606 OID 16460)
-- Dependencies: 1527 1840 1520
-- Name: fk_avatar_animation; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

```
ALTER TABLE ONLY user_avatar
  ADD CONSTRAINT fk_avatar_animation FOREIGN KEY (current_anim_id)
REFERENCES animation(animation_id) ON UPDATE CASCADE ON DELETE
RESTRICT;


--
-- TOC entry 1873 (class 2606 OID 16465)
-- Dependencies: 1859 1522 1526
-- Name: fk_avatar_mesh_object; Type: FK CONSTRAINT; Schema: public;
Owner: postgres
--

ALTER TABLE ONLY avatar
  ADD CONSTRAINT fk_avatar_mesh_object FOREIGN KEY (object_id)
REFERENCES mesh_object(object_id) ON UPDATE CASCADE ON DELETE
CASCADE;


--
-- TOC entry 1881 (class 2606 OID 16470)
-- Dependencies: 1521 1527 1842
-- Name: fk_avatar_user; Type: FK CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE ONLY user_avatar
  ADD CONSTRAINT fk_avatar_user FOREIGN KEY (owner_user)
REFERENCES app_user(username) ON UPDATE CASCADE ON DELETE
CASCADE;
```

```
--
-- TOC entry 1875 (class 2606 OID 16475)
-- Dependencies: 1524 1849 1523
-- Name: fk_enviroment_item_object_enviroment; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY enviroment_item_object
    ADD CONSTRAINT fk_enviroment_item_object_enviroment FOREIGN KEY (enviroment_object_id) REFERENCES enviroment(object_id) ON UPDATE CASCADE ON DELETE CASCADE;


--
-- TOC entry 1876 (class 2606 OID 16480)
-- Dependencies: 1524 1857 1525
-- Name: fk_enviroment_item_object_item_object; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY enviroment_item_object
    ADD CONSTRAINT fk_enviroment_item_object_item_object FOREIGN KEY (item_object_object_id) REFERENCES item_object(object_id) ON UPDATE CASCADE ON DELETE CASCADE;


--
-- TOC entry 1874 (class 2606 OID 16485)
-- Dependencies: 1523 1526 1859
```

-- Name: fk_enviroment_mesh_object; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY enviroment
  ADD CONSTRAINT fk_enviroment_mesh_object FOREIGN KEY (object_id) REFERENCES mesh_object(object_id) ON UPDATE CASCADE ON DELETE CASCADE;

--
-- TOC entry 1877 (class 2606 OID 16490)
-- Dependencies: 1526 1525 1859
-- Name: fk_item_object_mesh_object; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY item_object
  ADD CONSTRAINT fk_item_object_mesh_object FOREIGN KEY (object_id) REFERENCES mesh_object(object_id) ON UPDATE CASCADE ON DELETE CASCADE;

--
-- TOC entry 1879 (class 2606 OID 16601)
-- Dependencies: 1869 1526 1529
-- Name: fk_material_id; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY mesh_object

```
    ADD CONSTRAINT fk_material_id FOREIGN KEY (material_id) REFERENCES
material(id) ON UPDATE CASCADE ON DELETE RESTRICT;


--
-- TOC entry 1883 (class 2606 OID 16555)
-- Dependencies: 1529 1530 1869
-- Name: fk_material_one; Type: FK CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE ONLY material_pair
    ADD CONSTRAINT fk_material_one FOREIGN KEY (material_one_id)
REFERENCES material(id) ON UPDATE CASCADE ON DELETE CASCADE;
```



```
--
-- TOC entry 1884 (class 2606 OID 16560)
-- Dependencies: 1530 1529 1869
-- Name: fk_material_two; Type: FK CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE ONLY material_pair
    ADD CONSTRAINT fk_material_two FOREIGN KEY (material_two_id)
REFERENCES material(id) ON UPDATE CASCADE ON DELETE CASCADE;


--
-- TOC entry 1878 (class 2606 OID 16533)
-- Dependencies: 1528 1525 1867
```

```
-- Name: fk_ptype; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY item_object
  ADD CONSTRAINT fk_ptype FOREIGN KEY (ptype) REFERENCES
primitive_type(ptype) ON UPDATE CASCADE ON DELETE CASCADE;


--
-- TOC entry 1882 (class 2606 OID 16495)
-- Dependencies: 1527 1522 1845
-- Name: fk_user_avatar_avatar; Type: FK CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE ONLY user_avatar
  ADD CONSTRAINT fk_user_avatar_avatar FOREIGN KEY (avatar_object_id)
REFERENCES avatar(object_id) ON UPDATE CASCADE ON DELETE
CASCADE;


--
-- TOC entry 1901 (class 0 OID 0)
-- Dependencies: 6
-- Name: public; Type: ACL; Schema: -; Owner: postgres
--

REVOKE ALL ON SCHEMA public FROM PUBLIC;
REVOKE ALL ON SCHEMA public FROM postgres;
GRANT ALL ON SCHEMA public TO postgres;
GRANT ALL ON SCHEMA public TO PUBLIC;
```