Universidad Nacional Autónoma de México

Facultad de Estudios Superiores "Aragón"

Ingeniería en Computación

"Metodología para el diseño y creación de entornos virtuales con múltiples participantes empleando plataformas de software para Realidad Virtual Distribuida"

Tesista: Victor Hugo Oropeza Cadena

Agradecimientos

Dios, me has dado todo lo que alguien podría soñar, gracias por tantas bendiciones.

A mi mamá Sonia, por hacerme un hombre de bien, responsable, capaz de hacer cualquier cosa por mi mismo, por amarme y cuidarme, por enseñarme que los sentimientos son tan buenos y que la fe en Dios mueve montañas, sabes que te amo mucho mamita y que eres una de mis razones de vivir.

A mi papá Isi, por enseñarme el valor de la honestidad y de la justicia, cosas que atesoro con mucho fervor, por mostrarme el camino de la integridad y la verdad, eres un ejemplo en mi vida, te amo.

A mis hermanas, Sonia y Ale, por llenar de luz tantos años de mi vida y hacerla divertida y feliz, con enojos y alegrías, compartiendo juegos y momentos, por preocuparse por mi en lapsos de enfermedad y tristeza, las adoro con todo mi corazón.

A mis tías Lupe, Ana, Gloria y Olga, por tantas preocupaciones que pasaron por mi, por cuidarme y darme su apoyo amoroso cuando mis papás estaban lejos, no hubiera podido llegar hasta aquí sin ustedes, las quiero muchísimo.

A mis abuelitos Sol, Juanita, Rey y Luís, que han dado todo para poder disfrutar de lo que con tanto amor y cuidado han formado, por darme el cariño de padres, por quererme y consentirme tanto. Los llevo en mi corazón cada día de mi vida.

A toda mi familia, tíos y primos, que me han ayudado tanto y se han preocupado por lo que me pasa, por tantos bellos momentos que he pasado a su lado y que recuerdo con mucho cariño, por compartir juegos, historias, tristezas y júbilos, gracias por todo el apoyo.

A mi novia Blanca, gracias nena por que me tendiste la mano en el momento en que tuve confusión y perdí la visión de las cosas, por brindarme siempre tu apoyo incondicional sobre todo, por haberme revelado el camino de la perseverancia y la voluntad. Por enseñarme una forma del amor tan linda que no conocía, eres un modelo para mí, te amo mucho mi nena.

A mi amada Facultad de Estudios Superiores Aragón, a mi Alma Mater, la Universidad Nacional Autónoma de México, que me ha entregado las herramientas y el preciado conocimiento para ser profesional, a mis profesores y amigos que me han dado la formación para convertirme en una persona competitiva e integral, en un buen compañero y un estudiante orgulloso de su origen, de tener el corazón azul y la piel dorada.

Por mi raza hablará el espíritu.

Índice

Capítulo 2
Sistemas de Realidad Virtual en Red

Capítulo 3
Plataformas de Software para Realidad Virtual Distribuida

Capítulo 4

Desarrollo de una Aplicación Distribuida utilizando Quanta

Introducción

El conocimiento acerca de la Realidad Virtual en nuestro país no ha sido muy difundido, las personas que saben sobre el particular son investigadores o desarrolladores del área, lo que hace que el tema sea extenso e inalcanzable para muchos. Si personas de múltiples áreas científicas e inclusive sociales se interesaran en este tema, podrían ampliar en gran medida el conocimiento.

Este trabajo trata sobre la historia de la Realidad Virtual, los grandes inventos que han trascendido, así como acontecimientos de suma importancia que envuelven el tema. Con el devenir de los tiempos y con los grandes inventos han ido cambiando a su vez las etapas de la Realidad Virtual hasta nuestros días, con diferentes tipos, aplicaciones y dispositivos que nos permiten llevar a cabo acciones dentro y fuera de un mundo virtual.

El tema de la Realidad Virtual encierra muchos campos de investigación y aplicaciones en sí mismo. Resultaría fascinante adentrase a cada uno de ellos desde una perspectiva analítica y detallada, pero el campo de aplicación y desarrollo es demasiado extenso, por lo que nos enfocaremos a un área específica, una de las que tienen mayor relevancia, retos y futuro dentro la Realidad Virtual, nos referimos al área de la Realidad Virtual Distribuida, que describe la condición en la que pueden trabajar remotamente varios participantes por medio de ambientes virtuales en equipos alejados utilizando la red, por lo que se analizará la manera en que los sistemas se comunican a través de la red y los factores que actúan sobre ellos para darle un mayor rendimiento y efectividad a la comunicación, o por el contrario que ésta muestre deficiencias.

En la búsqueda por descubrir el funcionamiento exacto de un Sistema de Realidad Virtual Distribuido es necesario que se elabore un análisis de las diversas tecnologías de desarrollo e implementación de Realidad Virtual. Todo sistema está conformado por una investigación a fondo de los requerimientos precisos para la creación de un ambiente virtual. Existen varios y diferentes tipos de plataformas, que resultaría complicado decidir por alguna en especial, por lo que revisaremos a detalle cada una de las propuestas a ser empleadas, concluyendo con la mejor elección para construir una aplicación distribuida. Debemos considerar que la elección que tomemos siempre debe hacerse pensando en lo que tenemos planeado realizar y el futuro que le espera para nuestra aplicación.

Por desgracia, conocer la plataforma de desarrollo para realizar una aplicación distribuida no es suficiente. Además de esto, es imprescindible

contar con el manejo de alguna herramienta que nos permita manipular datos sobre la red, ya que trataremos de que se establezca una comunicación entre dos o más participantes, conocer la arquitectura de la red y la forma de proseguir en el desarrollo.

Capítulo 1
¿Qué es la Realidad Virtual?

Cuando escuchamos hablar del término Realidad Virtual es común que imaginemos películas o videojuegos en los que el espectador puede percibir la sensación de realismo e incluso de inmersión. Sin embargo esta visión de la Realidad Virtual es muy limitada puesto que el campo de las aplicaciones es muy extenso, tanto en las ciencias, la tecnología y el arte. Desde sus comienzos con Iván Sutherland[1] hasta nuestros días, la Realidad Virtual ha tenido grandes avances. Hoy en día podemos no solo adentrarnos a parajes virtuales o lejanos, sino que podemos inclusive observar la estructura tridimensional de una molécula y analizarla cuidadosamente desde todos sus ángulos y perspectivas. La interacción dentro de estos mundos virtuales se ha realizado por medio de dispositivos que nos ayudan a "movernos" y en algunos casos a manipular los elementos que lo conforman.

1.1 ¿Qué es Realidad Virtual?

Definitivamente un término muy sonado y controversial. En el nombre en sí hay una gran contradicción: Realidad Virtual. La definición de estos dos conceptos puede iniciar largas discusiones entre los filósofos de hoy en día.

El diccionario de la lengua española define virtual como "que existe o resulta en esencia o efecto, pero no como forma, nombre o hecho real". Realidad es "la verdadera esencia de las cosas que puede ser palpable y percibida por uno o más de los sentidos humanos."

Con estos conceptos podemos aventurarnos a dar una definición de lo que es Realidad Virtual: "La Realidad Virtual es la manipulación de los sentidos humanos (siendo principalmente el tacto, la visión y la audición) por medio de entornos tridimensionales sintetizados por computadora en el que uno o varios participantes acoplados de manera adecuada al sistema de computación, interactúan de manera rápida e intuitiva en el que el entorno real desaparece de la mente del usuario dejando como real el entorno generado por la computadora". Si se tiene en mente la sensación de estar soñando, se puede tener una idea de lo que se supone que es la Realidad Virtual. Del mismo modo que en sueños convive lo que tiene sentido y lo que no lo tiene, en la Realidad Virtual se mezclan libremente lo lógico y lo ilógico.

¿Cómo reconstruiríamos un sueño si pudiéramos conectar nuestra imaginación al motor de una potente computadora? Inmediatamente vendrían imágenes a nuestra mente y podríamos maniobrar con ellas. Con el programa adecuado, la Realidad Virtual nos ofrece la posibilidad

[1] **Ivan Sutherland**. Considerado como el padre de la Realidad Virtual, introdujo los conceptos de modelado por computadora en 3D, simulaciones visuales y Realidad Virtual.

de resolver problemas o de sumergir nuestros sentidos en experiencias nuevas. La tecnología permite crear un entorno y participar en un guión a nuestra elección. Uno se puede ver inmerso de la forma que desee. Imprimir una orden virtualmente posible como "Llévame volando hasta la Luna" y autodefinirse como parte de la escena, asignarse un tamaño tan pequeño como el de Alicia en el país de las maravillas, gravedad cero, o incluso aumentar la energía de forma acelerada.

También se puede asignar a personas u objetos virtuales atributos lógicos y parámetros como el peso, el aspecto físico, la gravedad y la movilidad. La realimentación electrónica basada en estas cualidades refuerza la experiencia llegando a convencer de que algo está ocurriendo realmente. Los entornos o escenarios de la Realidad Virtual pueden ser predefinidos o enfocados de tal manera que el usuario obtenga una destreza específica o una percepción clara, como si estuviera realmente en ellos; las posibilidades que se abren con esto son muy amplias. En Estados Unidos, las inmobiliarias lo utilizan desde hace tiempo para mostrar los pisos en tres dimensiones sin necesidad de visitarlos. Sencillamente se puede representar cualquier entorno físico y moverse por él.

Pero además de “meterse” dentro, uno puede actuar. Es decir, se puede crear una imagen de computadora de sí mismo, tal y como es en la realidad. Esa imagen se puede desplazar por el mundo imaginario, así mismo se pueden conectar más personas y “entrar” desde cualquier parte del mundo. Pero lo mejor de todo es, sin duda, el hecho de que se puede hablar con ellas, tener *chats*[2] como en el Internet pero con una presencia física.

Si se pone todo esto junto, las implicaciones son fantásticas: cada uno podría crear virtualmente el salón de su casa, entrar en él y sentarse en un sillón a charlar con su mamá, que estaría conectada y viéndote también a cientos de kilómetros de distancia; podrías mantener una reunión de trabajo, juntarte en un bar virtual con tus amigos, e incluso charlar con una chica sin salir de casa.

Para conocer más acerca de este interesante tema debemos tener en cuenta algunos hechos muy importantes que marcaron el rumbo de la Realidad Virtual.

1.2 Historia de la Realidad Virtual

- Experiencias precursoras (Ivan Sutherland). En un ambiente acondicionado y utilizando un casco estereoscópico (1956).

- Hacia la mitad de los años 1960, Ivan Sutherland, lanza el concepto de *Ultimate Display*, se trataba de un casco con pantalla que permitía que un piloto viera simultáneamente el paisaje real e imágenes gráficas sobredimensionadas.

- En 1962 es creado el “Sensorama” por Morton Heiling.

[2] **Chats**. En español “charlas”, se refiere a comunicaciones escritas a través de Internet entre dos o más personas en tiempo real. Un *chat* se caracteriza por que las personas que forman parte de el, escriben bajo un pseudónimo llamado *nick*.

Esta máquina simulaba las experiencias sensoriales de un paseo en motocicleta, al combinar imágenes, sonido, viento y aromas. En esta experiencia el usuario subía en el asiento de una motocicleta, tomaba los manubrios y usaba un visor parecido a unos binoculares donde podía pasear por dunas en California o en las calles de Brooklyn, donde unos pequeños aromatizantes cerca de la nariz del usuario emitían aromas auténticos. Sin embargo, este proyecto era sumamente complejo y nunca se lograron materializar versiones más sencillas de éste.

- En 1965 Ivan Sutherland publicó el artículo *The Ultimate Display*.

Ivan Sutherland, quien en 1965, publicó un artículo denominado "*The Ultimate Display*", donde sentó las bases del concepto de Realidad Virtual. Sutherland estipulaba "*La pantalla es una ventana a través de la cual uno ve un mundo virtual. El desafío es hacer que ese mundo se vea real, actúe real, suene real, se sienta real*".

- En 1966 Ivan Sutherland crea un casco visor de Realidad Virtual.

Sutherland creó un casco visor de Realidad Virtual al montar tubos de rayos catódicos en un armazón de alambre, este instrumento fue llamado "Espada de Damocles"[3] debido a que el aparato requería de un sistema de apoyo que pendía del techo.

- En 1972 se desarrolló el primer simulador de vuelo computarizado, el cual fue importante para el despegue de la Realidad Virtual.

- En 1977, fueron creados los primeros guantes utilizados como periféricos de entrada de datos por D. Sandin y R. Sayre en Chicago.

- En 1981, T. Zimerman, investiga la forma de simular un instrumento de música virtual por medio de movimientos simples de la mano. Inventaría después el *Data Glove*[4], que patentó en 1982.

- En 1982 la expresión "mundos virtuales" (*virtual worlds*) fue usada por primera vez por el consultor en administración Donald Schon.

Usó este término para referirse a las imágenes mentales que mantiene un sujeto acerca del entorno con el cual interactúa. Posteriormente, estos términos han sido retomados, discutidos y difundidos en el sentido en que se usan hoy en el ambiente de la computación por el Departamento de Ciencias de la Computación de la Universidad de Carolina del Norte.

- 1982, se recrea la “Telepresencia” en la NASA.

[3] **Espada de Damocles**. Descrita dentro de una anécdota moral de la cultura griega, era una espada que pendía del techo de un pelo de caballo.

[4] **Data Glove**. Guante de lycra cuyos cinco dedos están recubiertos con fibras ópticas que hacen la función de sensores en las flexiones.

El primer sistema que se proponía atravesar el umbral de las dos dimensiones y recrear una verdadera "telepresencia en un espacio de datos" (*Telepresence in Dataspace*) fue concebido en los laboratorios Ames de la NASA en Moffett Field, California, por el equipo dirigido por Michael Mc Greevy.

- En 1984, en el centro NASA-Ames, en California, M. McGreevy iniciaba el proyecto Virtual Workstation para la preparación de misiones en el espacio.
- 1984, se inicia le proyecto "VIVED".

El proyecto Vived (*Virtual Environment Display*), iniciado en 1984, tenía como objetivo la construcción de una estación de trabajo virtual destinada a ser utilizada en misiones espaciales de la NASA.

- En 1985 fue construido el primer sistema práctico de visores estereoscópicos para la NASA.
- En 1986, W. Tobinett escribió el primer programa de modelización dinámica del *Data Glove* en un entorno infográfico tridimensional.
- En 1988 el Laboratorio de Realidad Virtual de la NASA creó el primer modelo virtual de una edificación, precisamente la del propio laboratorio.
- El año de 1989 señaló los inicios industriales de la Realidad Virtual con la aparición de los primeros *Eyephones*[5] de la empresa *California Visual Programming Language Research* (VPL).
- En este mismo año, Rikk Carey y Paul Strauss de *Silicon Graphics Incorporated*[6], iniciaron un nuevo proyecto con el fin de diseñar y construir una infraestructura para aplicaciones interactivas con gráficos tridimensionales.

Los dos objetivos originales eran:

- Construir un ambiente de desarrollo que permitiera la creación de una extensa variedad de aplicaciones interactivas con gráficos tridimensionales distribuidos.
- Utilizar este ambiente de desarrollo para construir una nueva interfaz de usuario tridimensional.

La primera fase del proyecto se concentraba en diseñar y construir la semántica y los

[5] **Eyephones**. Consistía en dos pequeños monitores de color, uno para cada ojo, montados dentro de un casco.

[6] **Silicon Graphics Incorporated**. Empresa dedicada al desarrollo de cómputo de alto desempeño, almacenamiento y tecnología de visualización.

mecanismos para la plataforma de trabajo. El tema de las aplicaciones distribuidas fue tomado en cuenta para el diseño del estándar, aunque estuvo fuera del alcance de la primera implementación.

- En 1989 el Departamento de defensa de los Estados Unidos crea a Simnet

Una red experimental de estaciones de trabajo basadas en microprocesadores que habilitaban al personal a prácticas de operaciones de combate interactivamente en sistema de entrenamiento de tiempo real.

- En 1992 se liberó el Iris *Inventor 3D Toolkit*.

Iris Inventor definía gran parte de la semántica que hoy en día conforma a VRML[7]. Una parte importante del *Iris Inventor* era que el formato del archivo utilizado para guardar los objetos de la aplicación era de poco tamaño y fácil de utilizar.

- 1992, se desarrolla CAVE[8].

CAVE fue diseñada en 1992 para ser una herramienta aprovechable para los científicos en visualización. CAVE fue desarrollada como un teatro de Realidad Virtual con contenido científico. CAVE reúne varios criterios de Realidad Virtual, siendo muy exitosa.

- 1994, se liberó la segunda gran versión de *Iris Inventor* llamada *Open Inventor*. Fue creado VRML.

Ésta era portable para diferentes plataformas y basada en *OpenGL*[9] de Silicon Graphics. El manual de referencia que describe los objetos y el formato de archivo de *Open Inventor* fueron utilizados después por Gavin BEI para escribir la primera propuesta para la especificación de VRML 1.0.

En 1994, Mark Pesce y Brian Dehlendorf crearon el *VRML mailing list* o lista de discusión "WWW-VRML" donde se hizo un llamado abierto a todo el público para dar propuestas para una especificación formal de 3D en el WWW[10]. Dada la magnitud del trabajo se decidió avanzar por etapas y adoptar estándares existentes donde fuera posible. En este mismo año Mark Pesce y Tony Parisi crearon un prototipo de visor de 3D para el WWW.

---

[7] **VRML** (*Virtual Reality Modeling Language*). Formato de archivo normalizado que tiene como objetivo la representación de gráficos interactivos tridimensionales, diseñado particularmente para ser utilizado en la Web.

[8] **CAVE**. Proyecto basado en sistemas de Realidad Virtual que proporciona una perspectiva en función de los movimientos de la cabeza en tiempo real, con un amplio campo de vistas, controles interactivos y pantallas estereoscópicas. Es un cubo de 10x10x10 pies con imágenes proyectadas en tres paredes y en el piso.

[9] **OpenGL**. Entorno para desarrollo de aplicaciones graficas portables e interactivas en 2D y 3D.

[10] **WWW** (World Wide Web). Sistema de hipertexto que funciona sobre Internet, utilizando como visualizador de información una aplicación llamada *Web Browser*.

Después de varias propuestas se escogió la sintaxis de *Open Inventor* de Silicon Graphics como base de un formato de descripción de objetos geométricos texturizados, agregando la posibilidad de combinar objetos guardados remotamente en la red (mediante hiperenlaces como en HTML[11]). De esta manera nació VRML 1.0 que aunque solo era una solución parcial, era una muestra de lo que VRML podría llegar a ser.

- 1995 VRML es reparado y mejorado.

Durante la primer mitad de 1995 la especificación de VRML 1.0 sufrió un gran número de clarificaciones y reparaciones, pero funcionalmente quedó igual. En Agosto de 1995 hubo mucha discusión dentro del grupo de discusión WWW-VRML en cuanto a la creación de VRML 1.1 o VRML 2.0. Algunos pensaban que VRML necesitaba solo de unas cuantas adiciones de contenido, mientras que otros sentían la necesidad de una completa revisión del estándar. El segundo paso comenzó en Siggraph[12] 95 y culminó en Siggraph 96. El nuevo estándar consistió en permitir el movimiento de la geometría estática definida en VRML 1.0.

Se hizo un llamado a presentar propuestas públicamente y se estableció una página de Web para votar. Hubo propuestas de más de 50 compañías como Silicon Graphics, Sony, Netscape, Apple, IBM, Microsoft, entre otras. Ganó la propuesta *Moving Worlds* de Silicon Graphics Incorporated, Sony Corporation y Mitra.

- En el 2000 el diseño tridimensional de *Trispace* permite navegar e interactuar en tiempo real,

El servicio de decoración virtual de la Corte Inglesa fue la primera experiencia en la que *Trispace* aplicaba en Internet su tecnología y diseño. En un entorno virtual, el usuario que accede a este servicio construye la habitación de su hogar que desee amueblar, levantando muros, puertas y ventanas; seleccionando su orientación y su tamaño. Con la estancia compuesta, el cliente puede amueblarla eligiendo entre el amplio abanico de elementos decorativos disponibles, según sus gustos y preferencias. Finalizada esta tarea, puede pasear por la estancia en 3D para comprobar si la distribución del mobiliario es la adecuada. El cliente puede hacerse idea de cómo quedaría el cuarto como si lo tuviera delante de sus ojos.

- Se desarrolla el iGrid en el 2000

El iGrid 2000, la malla internacional, demuestra cómo las características de las redes de investigación permiten el acceso a los recursos de cómputo remotos, a la distribución de medios digitales y a la colaboración con los colegas distantes.

El concepto de conectar recursos de cómputo con gente y las redes de alto rendimiento son un asunto importante de la investigación. El iGrid destaca los logros en el desarrollo de la

[11] **HTML** (Hyper Text Markup Language). Lenguaje de marcas para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas Web.

[12] **Siggraph**. Fundado en 1947, nombre de la conferencia realizada anualmente por el grupo de interés en computación gráfica llamado también SIGGRAPH.

arquitectura de la malla y en los adelantos que permiten la ciencia, la ingeniería, el patrimonio cultural, el arte y la arquitectura, en comunicaciones de los medios y la educación a distancia. El iGrid ofrece 24 usos a partir de 14 regiones: Canadá, Alemania, Grecia, Japón, Corea, México, los Países Bajos, Singapur, España, Suecia, Taiwán, Reino Unido, los Estados Unidos y la unión de países CERN[13], con énfasis sobre la tele-inmersión, grandes bases de datos, cómputo distribuido, la instrumentación remota, la colaboración, interfaces de persona/computadora, medios dinámicos, el vídeo digital y la televisión de alta definición.

La utilización de esta tecnología se presenta en la utilización del teatro de Realidad Virtual de CAVE desarrollado por la Universidad de Illinois en Chicago, el sistema de alta definición desarrollado por el laboratorio de innovación de la red NTT, y el ambiente de presentación de la malla de acceso por el laboratorio de Argonne National.

- En el 2002 se desarrolla un proyecto de vivienda virtual interactiva

Desarrollado durante el año 2002, el proyecto ha alcanzado el objetivo fundamental definido por Telefónica I+D[14]; conectar una vivienda controlada por varios dispositivos hápticos[15] y un software especial de Realidad Virtual, controlado por la aplicación Web de *Trispace*.

Todo ello, además de conseguir la visualización tridimensional de la vivienda donde está instalado el sistema, permite al usuario actuar y visualizar órdenes y acciones en tiempo real sobre los dispositivos o mecanismos implementados en la misma.

Gracias a la nueva interfaz de visión tridimensional en Realidad Virtual y la velocidad de acceso ADSL[16], los propietarios podrán "telecontrolar" las viviendas en tiempo real, actuando sobre los dispositivos en la pantalla. El usuario podrá comprobar, a través de la interfaz de Realidad Virtual, cómo se están ejecutando sus órdenes en la vivienda; subida o bajada de las persianas, activación o regulación de la intensidad de luz, puesta en marcha del riego en el jardín, levantamiento de los toldos, alarmas, visualización de las cámaras, activación de las distintas escenas, etc. El sistema desarrollado por *Trispace* permite ver a través de Internet o directamente desde el interior de la vivienda el estado del hogar, gracias a la fiel reproducción del mismo. Para el desarrollo de esta aplicación, *Trispace* ha utilizado el software de Realidad Virtual y Simulación llamado *virtools*, usado habitualmente en desarrollo de aplicaciones para simuladores de vuelo, ingeniería y automoción, así como

[13] **CERN**. Consejo Europeo para la Investigación Nuclear, formado por varios países europeos teniendo como sede principal Francia.

[14] **Telefónica I+D**. Empresa de telecomunicaciones española dedicada al desarrollo e investigación.

[15] **Dispositivos Hápticos**. Dispositivos que pueden actuar como dispositivos de entrada pero además son capaces de proporcionar al usuario una salida por medio del mismo aplicando fuerzas o vibraciones al usuario, por lo que se dice que un dispositivo háptico es bidireccional.

[16] **ADSL** (Asymmetric Digital Subscriber Line). Es una línea digital de alta velocidad apoyada en par trenzado de cobre que lleva la línea telefónica común.

programación Java[17] para la actuación sobre los dispositivos con base en las especificaciones de Telefónica I+D.

- Se optimiza y mejora el iGrid2000 al iGrid2002

En el 2002 científicos se plantearon desafíos tecnológicos para utilizar redes ópticas experimentales de multi-gigabits. El resultado es un esfuerzo impresionante coordinado por 28 equipos, representando 16 países, que en conjunto desarrollan investigación utilizando innovaciones del middleware[18].

Los científicos computacionales se esfuerzan en entender mejor los sistemas muy complejos como: climas biológicos, ambientales, atmosféricos, geológicos o físicos, del micro nivel al macro nivel, en tiempo y espacio, pero requieren un lugar de almacenamiento masivo para lo que se utiliza un sistema de cómputo petaflop. Un sistema de computo petaflop es una computadora 100 veces más rápida que cualquier otra computadora paralela de las más grandes que hoy existen, la cual procesa diez trillones de operaciones de punto flotante por segundo(10 teraflops). Posee mil millones de gigabytes de almacenamiento y las redes del terabit, transmitirán eventualmente datos a un trillón por segundo, unas 20 millones de veces más rápidamente que una conexión de Internet de 56K.

La investigación se está realizando desde ambientes conectados localmente y procesadores céntricos a los ambientes distribuidos computacionales que confían en conexiones ópticas, donde las redes son más rápidas

El iGrid 2002 demuestra las demandas del uso para el ancho de banda creciente. El iGrid 2002 permite a la comunidad de investigación del mundo trabajar en conjunto brevemente y avanzar velozmente desarrollando nuevas técnicas del control de red.

1.3 Características básicas de la Realidad Virtual

Para que el concepto de Realidad Virtual sea válido es necesaria la conjunción de diferentes aspectos que lo definan, si alguno de estos es deficiente ó inexistente la percepción del concepto se vuelve escasa.

1.3.1 *Interacción*

La interacción se refiere a los rasgos que permiten al usuario manipular el curso de la acción dentro de una aplicación de Realidad Virtual, permitiendo que el sistema responda a los estímulos de la persona que lo utiliza; creando interdependencia entre ellos.

Existen dos aspectos únicos de interacción en un mundo virtual. El primero de ellos es la navegación, que es la habilidad del usuario para moverse independientemente alrededor del

[17] **Java**. Plataforma de software desarrollada por Sun Microsystems de tal forma que los programas desarrollados en ella se pueden ejecutar en diferentes tipos de arquitecturas y dispositivos computacionales.
[18] **Middleware**. Software que comunica dos aplicaciones separadas.

mundo. Las restricciones para este aspecto las coloca el inventor del software, que permite varios grados de libertad, si se puede volar o no, caminar, nadar, etcétera. Otro punto importante de la navegación es el posicionamiento del punto de vistas del usuario. El usuario se puede mirar a sí mismo (a través de los ojos de alguien más), o puede moverse a través de cualquier aplicación observando desde varios puntos de vista.

El otro aspecto de la interacción es la dinámica del ambiente, que no es más que las reglas de cómo los componentes del mundo virtual interactúan con el usuario para intercambiar energía o información.

1.3.2 *Inmersión*

Esta palabra significa bloquear toda distracción y enfocarse selectivamente solo en la información u operación sobre la cual se trabaja. Posee dos atributos importantes, el primero de ellos es su habilidad para enfocar la atención del usuario y el segundo es que convierte una base de datos en experiencias, estimulando de esta manera el sistema natural de aprendizaje humano (las experiencias personales).

1.3.3 *Tridimensionalidad*

Esta es una característica básica para cualquier sistema llamado de Realidad Virtual, tiene que ver directamente con la manipulación de los sentidos del usuario, principalmente la visión, para dar forma a el espacio virtual; los componentes del mundo virtual se muestran al usuario en las tres dimensiones del mundo real, en el sentido del espacio que ocupan y los sonidos tienen efectos envolventes.

1.3.4 *Inducción electrónica de los sentidos*

Relación entre los sentidos, la percepción y las interfaces en un sistema de simulación de Realidad Virtual, cuanto más se perciba con los sentidos, más real será el mundo virtual.

SENTIDO	PERCEPCIÓN	INTERFACES
Vista (provee 80% información)	Luz	Pantallas, sistemas de proyección y ópticas generadoras de imagen 3D, cascos visualización 3D, gafas de obturación rápida
Oído	Onda sonora	Tarjeta de sonido, audio 3D, altavoces, auriculares
Tacto	Percepción táctil y *propioceptiva (*auto percepción)	Dispositivos táctiles (guantes y trajes); sistemas de retorno de fuerzas
Olfato	Química aire	Sistemas odoríferos (experimentales - poco desarrollados -)
Gusto	Química solución	No hay investigación en este campo
Vestibular	Equilibrio	Plataformas móviles alfombras continuas sistemas de rastreo de posición/orientación

Figura 1.1

Figura en la que se muestra la interacción de los sentidos con los dispositivos de Realidad Virtual y la percepción.

1.4 Tipos de Realidad Virtual

Hablar de Realidad Virtual es pensar en un universo de conceptos, objetos y formas de representación, la Realidad Virtual se constituye de formas diversas mediante ambientes de computadora que dan percepciones y efectos distintos sobre el usuario, cada una de estas percepciones y efectos de apreciar los ambientes fueron asignados a una clasificación, las cual se describe a continuación.

1.4.1 *Realidad Virtual Inmersiva*

La Realidad Virtual con frecuencia se liga a un ambiente tridimensional creado por computadora, el cual se manipula a través de cascos, guantes u otros dispositivos que capturan la posición y rotación de diferentes partes del cuerpo humano. Este tipo de sistemas utiliza diferentes dispositivos denominados accesorios, como pueden ser guantes, trajes especiales, visores o cascos, estos últimos le permiten al usuario visualizar los mundos a través de ellos, y precisamente estos son el principal elemento que lo hacen sentirse inmerso dentro de estos mundos. Este tipo de sistemas son ideales para aplicaciones de entrenamiento o capacitación.

1.4.2 *Realidad Virtual No Inmersiva*

La Realidad Virtual No Inmersiva ofrece un nuevo mundo a través de una ventana de escritorio. Este enfoque no inmersivo tiene varias ventajas sobre el enfoque inmersivo como: bajo costo y fácil y rápida aceptación de los usuarios. Los dispositivos inmersivos son de alto

costo y generalmente el usuario prefiere manipular el ambiente virtual por medio de dispositivos familiares como son el teclado y el mouse, que por medio de cascos pesados o guantes. El monitor es la ventana hacia el mundo virtual y la interacción es por medio del teclado, micrófono, mouse o joystick [19], este tipo de sistemas son idóneos para visualizaciones científicas, también son usados como medio de entretenimiento (como son los casos de los juegos arcades[20]) y aunque no ofrecen una total inmersión son una buena alternativa de bajo costo.

Actualmente Internet nos provee de medios para reunirnos con diferentes personas en el mismo espacio virtual. En este sentido Internet tiende a ser un mecanismo de "Telepresencia".

Este medio nos brinda espacios o realidades que físicamente no existen pero que forman parte de nuestras formas de vida. Es a través de Internet como nace VRML, que es un estándar para la creación de mundos virtuales No Inmersivos.

1.4.3 *Sistemas Semi-Inmersivos*

Los sistemas Semi-Inmersivos de proyección se caracterizan por ser 4 pantallas en forma de cubo (tres pantallas forman las paredes y una el piso), las cuales rodean al observador, el usuario usa lentes y un dispositivo de seguimiento de movimientos de la cabeza, de esta manera, al moverse el usuario las proyecciones perspectivas son calculadas por el motor de Realidad Virtual para cada pared y se despliegan en proyectores que están conectados a la computadora. Este tipo de sistemas son usados principalmente para visualizaciones donde se requiere que el usuario se mantenga en contacto con elementos del mundo real.

1.4.4 *Sistemas de ventana al mundo (Window on World systems) WoW*.

Este sistema de Realidad Virtual está basado en el uso de un monitor como medio para apreciar el mundo virtual. En 1965, Ivan Sutherland realizó una investigación sobre graficas de computador (*the ultimate display*), que ha servido como base de trabajo en los últimos 30 años. El propósito de este tipo de Realidad Virtual es llegar a experimentar por medio de nuestros sentidos (especialmente vista, oído, tacto) el mundo virtual que se encuentra en la ventana (monitor) como si fuera real.

1.4.5 *Video Mapping*.

El Video Mapping es una variación del WoW y en está se añaden siluetas en 2D con imágenes computarizadas, con el fin de proporcionar un ambiente más amable al usuario en el momento de la interacción monitor-cuerpo. Este tipo de Realidad Virtual fue creado por Myron Kruger y desde entonces se ha convertido en la base sobre la cual se inspiran los más

[19] **Joystick**. Dispositivo de entrada utilizado, comúnmente en juegos de consola o PC, usado para controlar un objeto del sistema, es literalmente una palanca de juegos.

[20] **Arcades**. Tipo de videojuegos que no tienen una trama o historia realmente definida. Por lo que se dice que solo hay que jugar y divertirse.

variados videojuegos que existen actualmente.

1.4.6 *Telepresencia*.

Esta es una variación de mundos generados por computadora, pues se conectan sensores remotos ubicados en el mundo real. En este sistema de Realidad Virtual se mezcla el mundo real con el virtual, con el fin de que el usuario interactúe en ambos mundos, un claro ejemplo de este tipo de Realidad Virtual son los simuladores de vuelo que utilizan los pilotos.

En este tipo de Realidad Virtual una imagen en movimiento del usuario es proyectada junto con otras imágenes en una extensa pantalla donde el usuario puede verse a sí mismo como si estuviera en la escena. En esencia, los usuarios se miran ellos mismos como proyectados hacia el mundo virtual y pueden pintar diseños de colores en el aire o hacer cualquier movimiento, ya que el sistema reacciona en tiempo real.

A diferencia de los sistemas de inmersión, los sistemas por Telepresencia involucran percepciones y respuestas en tiempo real a las acciones de las personas involucradas, quienes están liberadas, o no están sometidas al uso de cascos, guantes, cascos con pantallas, alambres o cualquier otro tipo de interfaz intrusiva. Los sistemas de inmersión simulan las percepciones del mundo real, el viajero sabe que está ahí porque los sonidos e imágenes del mundo virtual responden de manera similar a como responden los del mundo real a los movimientos de la cabeza. Sin embargo, en los sistemas por Telepresencia, el explorador sabe que está dentro del mundo virtual porque se ve a si mismo dentro de la escena. Es decir es un integrante del mundo virtual. Para lograr esto, el participante es ubicado frente a una pantalla de video, en la cual es proyectada la imagen misma del participante pero *chroma-keyed* (sumada su imagen al video) con otra imagen utilizada como fondo o ambiente, entonces el participante visualiza el mundo virtual completo en la pantalla.

Mediante un software que realiza detección de contornos, es posible realizar manipulaciones dentro de la escena, las cuales son visualizadas en la pantalla. Más que imitar las sensaciones del mundo real, un sistema de Telepresencia cambia las reglas y aplica la vieja noción de "ver para creer" para inducir la sensación de presencia.

1.4.7 *Sistemas Desktop Realidad Virtual*

Engloban aquellas aplicaciones que muestran una imagen 3D en 2D en una pantalla de computadora. Puesto que se representan mundos de 3 dimensiones, los exploradores pueden viajar en cualquier dirección dentro de estos mundos, los ejemplos característicos de estos ambientes son los simuladores de vuelo para computadora y los juegos de alto nivel de realismo para computadora. En resumen, los sistemas Desktop Realidad Virtual muestran mundos tridimensionales a través de pantallas de 2D. Algunos comprenderán interfaces sofisticados, como guantes, controles, cabinas, pero todos tendrán en común la característica antes mencionada, 3D desde 2D.

1.4.8 *Cabina de Simulación*

El ejemplo más común de este tipo de simulador es la cabina para el entrenamiento de pilotos.
Generalmente la cabina recrea el interior del dispositivo o máquina que se desea simular (un carro, un avión, un tanque etc.), las ventanas de la misma se reemplazan por pantallas de computadoras de alta resolución, además existen bocinas estereofónicas que brindan el sonido ambiental y puede estar colocada fija o sobre ejes móviles. El programa está diseñado para responder en tiempo real a los estímulos que el usuario le envía por medio de los controles dentro de las cabinas.

1.4.9 *Realidad Aumentada*

Esta se logra cuando una persona se confía del mundo real como línea de referencia, pero utiliza visores de cristal transparentes u otros medios Inmersivos para aumentar la realidad, superponiendo esquemas, diagramas, textos, referencias, etcétera.

Como ejemplo la Boeing[21] está explorando la posibilidad utilizar este sistema en la ingeniería de los aeroplanos, de tal manera que sus técnicos e ingenieros no tengan que irse a ver un manual para resolver un problema, pues el sistema de realidad aumentada les mostraría los diagramas esquemáticos o las listas de las partes del aeroplano, sin que el operario tenga que moverse de su silla.

1.4.10 *Ventanas Acopladas Visualmente*

Este sistema se basa en colocar las muestras directamente en frente del usuario y conectando los movimientos de la cabeza con la imagen mostrada. Para lograr mayor acople, la inmersión se logra con un casco (HMD) estereofónico, que posee sensores de posición y orientación que informan a la máquina la posición del usuario en todo momento, además de indicarle hacia donde está mirando.

1.5 Dispositivos de Realidad Virtual

La necesidad de realismo y de producción rápida de mundos virtuales ha impulsado el desarrollo de tecnologías de captura y despliegue. Los sistemas de captura permiten digitalizar las geometrías, las texturas y los movimientos de objetos reales permitiendo incorporarlos en forma foto realista en los entornos virtuales o heredarlos a objetos virtuales para que estos tomen algunas propiedades de la realidad; se produzca un híbrido. Mientras que los dispositivos de despliegue buscan reproducir las imágenes de forma foto realista, así como lograr la sensación de inmersión y ampliar el ancho de banda visual.

Estos sistemas cada vez se integran más y son más usados. Las tareas básicas que deben cubrirse son:

[21] **Boeing**. Uno de los principales fabricantes de aviones y equipos aeroespaciales del mundo.

- La creación; que incluye el modelado, la captura, las composiciones.

- El dibujado; que depende en gran medida de los cálculos de iluminación y la integración de objetos visuales y auditivos, o sea, un motor de *render*[22] y de multimedia.

La tecnología de un sistema de Realidad Virtual trata sobre algo más que los dispositivos físicos que se utilizan para la visualización, interacción o retroalimentación de este tipo de sistemas, también incluye el software o programas que se utilizan para la modelación. Ambos elementos se complementan, el hardware se encarga no solo de mandar señales al usuario sino también de recibir señales por parte de éste, el software a su vez, se encarga de procesarlas y transformarlas en un nuevo comportamiento del mundo virtual

1.5.1 *Hardware*

El hardware consiste de dispositivos físicos que forman parte de un sistema de Realidad Virtual y son los que estimulan al usuario en distintas maneras. Estos estímulos son los que le permiten alimentar los sentidos del usuario, y así inducirlo a un mundo creado para él.

Existen diferentes tipos de dispositivos de entrada, por ejemplo los dispositivos hápticos y de salida, los dispositivos visuales ó de sonido.

1.5.1.1 *Máquina de realidad*

La máquina de realidad se refiere al hardware que permite generar modelos virtuales, este hardware puede ser desde una simple PC, hasta estaciones de trabajo diseñadas especialmente para tratar con este tipo de tareas, como por ejemplo la familia de estaciones de trabajo y sistemas Onyx2 como son *Onyx2 Reality Visualization* u *Onyx2 Reality Monster* de Silicon Graphics. Estas máquinas se encargan de realizar todo el proceso de trazado de las imágenes.

1.5.1.2 *Dispositivos visuales*

Cuando se escucha hablar sobre Realidad Virtual, generalmente lo primero que viene a la mente es una persona usando unos lentes o un casco y visualizando algún modelo virtual. Los dispositivos visuales son una de las herramientas más importantes de retroalimentación para el usuario, en la mayoría de los casos es la entrada primaria que éste recibe del sistema de Realidad Virtual.

Una de las principales consideraciones en este tipo de dispositivos es el detalle de las imágenes contra la rapidez en la formación de las imágenes que forman las escenas, además de una visión monoscópica contra una estereoscópica. La formación de escenas en tiempo real le da un sentido de realidad al usuario al eliminar la discontinuidad.

[22] **Render**. Cálculo complejo desarrollado por una computadora para generar una o varias imágenes.

1.5.1.3 *Captura de Movimiento*

Un sistema de este tipo captura los movimientos completos del cuerpo y rasgos faciales con gran exactitud y después los exporta a un software en el que se trabaja la animación.

La captura de movimiento como se conoce a una serie de dispositivos y técnicas que permiten obtener las coordenadas en tres dimensiones durante una secuencia de tiempo, de un conjunto de puntos marcados en un personaje real ó físico (generalmente un actor), para transferírselos a un personaje virtual. Estos dispositivos son ópticos, electromecánicos, infrarrojos y requieren de diversas técnicas y software para lograr la tarea.

1.5.1.4 *Brazos Digitalizadores*

Actualmente se dispone de muchos dispositivos para capturar las geometrías y las texturas de objetos reales para incorporarlos a los mundos virtuales. Se pueden apreciar dispositivos de brazos digitalizadores 3D, que inclusive se acoplan con cascos o lentes estereoscópicos para capturar o interactuar, ahora con objetos virtuales (Figura 1.2, Figura 1.3).

Los brazos digitalizadores 3D, fueron creados específicamente para tomar los movimientos voluntarios de un usuario y emularlos digitalmente en una pantalla de computadora.

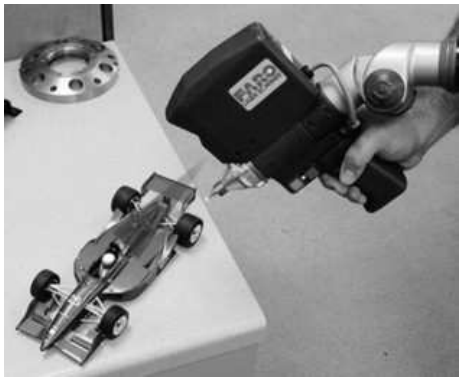
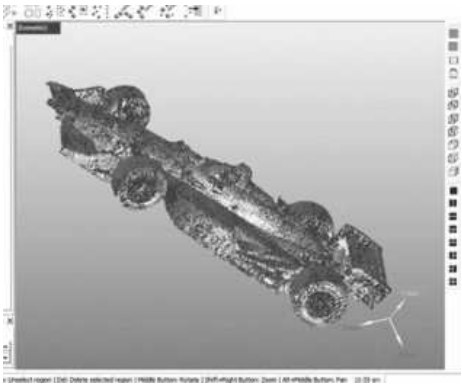


Figura 1.2

Es tomada una muestra física de la textura y geometría mediante el brazo digitalizador, en este caso un pequeña auto de carreras.



Después de ser tomada la muestra física es emulada dentro de la computadora en el mundo virtual.

1.5.1.5 *Omnipantallas*

Por otra parte, las pantallas parabólicas son sistemas de inmersión personales que permiten explorar mundos virtuales calculados en computadoras PC, usando un proyector con un lente especial y las pantallas cóncavas semiesféricas, en las cuales se puede observar un mayor campo de visión e inmersión de un mundo creado virtualmente por computadora.

1.5.1.6 *Vídeo Wall*

Las tarjetas gráficas de última generación, permiten controlar el despliegue de múltiples monitores gracias a una tarjeta gráfica que realiza la escala y multiplicación de una imagen hacia nueve destinatarios, esto permitie ampliar el ancho de banda visual. En la figura siguiente se aprecia el despliegue de un mundo virtual ampliado con nueve monitores.

Figura 1.3

En este Video Wall se muestra el despliegue de un mundo virtual por medio de nueve monitores.

1.5.1.7 *Visionarium*

Las pantallas de Realidad Virtual de proyección (generalmente 160º), permiten la sensación de inmersión tridimensional para grupos amplios de personas. En la imagen siguiente, se ve la visualización de un flujo en un Visionarium.

Figura 1.4

Esta imagen muestra la pantalla de despliegue y las computadoras de manipulación del Visionarium.

1.5.1.8 *Inmersa Desk*

Los escritorios con despliegue por proyección son una herramienta muy útil para el trabajo en grupos pequeños de personas. Estos sistemas van desde simples monitores de retroproyección montados en una mesa, hasta sistemas estereoscópicos con sensores para detectar la posición de la cabeza para corregir los despliegues, ajustándolos a la perspectiva del espectador. Algunos cuentan con monitores interactivos (touch screen) y mouse láser o 3D.

Estos escritorios fueron creados para simular escritorios virtuales de trabajo con características como: despliegue por proyección, pantalla plana ajustable a la vista del espectador, detectores de posición, alta resolución entre otros.

1.5.1.9 *Reality Center*

La presentación de nuevos diseños y de recorridos virtuales de proyectos se consiguen con mucho realismo en los Centros de Realidad Virtual o *Reality Center*. Estos Centros de Realidad Virtual son sistemas de despliegue amplios (generalmente por retrospección) montados en páneles de exhibición, algunos incluyen estereoscopía a través de lentes o de proyectores estereoscópicos (Figura 1.5).

Aunque preferentemente usan un sistema de estereoscopía intermedio que permite que el observador, usando lentes estereoscópicos, vea en 3D; si no cuenta con dicho dispositivo, aún puede apreciar el escenario. Generalmente este despliegue es en escala 1:1 y con una perspectiva natural, para lograr la sensación de inmersión y realismo.

Figura 1.5

El Reality Center formado por una pantalla inmersiva, sonido 3D y computadoras de manejo y procesamiento de imágenes.

1.5.1.10 *Dispositivos para despliegue 3D*

Los dispositivos de despliegue que producen la ilusión óptica de imágenes tridimensionales, requieren que un ojo reciba una imagen ligeramente diferente al otro; simulando el ángulo de diferencia con que ve cada ojo. La perspectiva de estas imágenes depende de la posición de la cabeza con respecto al objeto 3D. Así que es necesario determinar la distancia de la cabeza al objeto y los ángulos o vista que se tiene del objeto.

De tal forma que si se mueve el objeto se recalculan las coordenadas de pantalla (como en cualquier otro dispositivo), pero si se mueve el observador, también será necesario recalcular las dos imágenes que recibe este. Así mismo, si se tienen varios observadores, es necesario dibujar diferentes partes estereoscópicos para cada quien.

1.5.1.11 *Lentes LCD resplandecientes*

Los lentes resplandecientes de despliegue de cristal líquido tienen la apariencia de un par de anteojos, donde un fotosensor es montado en estos lentes para así poder leer una señal de la computadora. Esta señal le comunica a los anteojos si le permite al lente pasar luz del lado izquierdo o derecho del lente. Cuando a la luz se le permite pasar a través del lente izquierdo, la pantalla de la computadora mostrará el lado izquierdo de la escena, lo cual corresponde a lo que el usuario verá a través de su ojo izquierdo. Cuando la luz pasa a través del lente derecho, la escena en la pantalla de la computadora es una versión ligeramente deslizada hacia la derecha. Los lentes de LCD resplandecientes son ligeros y sin cables. Estas dos características los hacen fáciles de usar, sin embargo, el usuario tiene que mirar fijamente y solo a la pantalla de la computadora para ver la escena tridimensional, ya que el campo de vista es limitado al tamaño de la pantalla de la computadora, el medio ambiente real puede también ser visto lo cual no proporciona un efecto de inmersión.

1.5.1.12 *Despliegues montados en la cabeza HMD (Head-Mounted Display)*

Los despliegues montados en la cabeza (HMD por sus siglas en inglés) colocan una pantalla en frente de cada ojo del individuo todo el tiempo (Figura 1.6). El casco que usa el participante tiene unos sensores montados en él, los cuales le permiten reconocer el movimiento de la cabeza por lo que una nueva perspectiva de la escena es generada.

Figura 1.6

Head Mounted Display, cómodamente equipado para ser montado sobre la cabeza como si fueran lentes.

1.5.1.13 *HMD con CRT pequeño*

Este casco utiliza dos tubos de rayos catódicos (CRT) que se posicionan en los lados del casco. Se utilizan espejos para reflejar la escena al ojo del individuo. Una diferencia con el casco proyectado es que en este tipo de casco el fósforo es iluminado por cables de fibras ópticas, el fósforo es iluminado por un rayo de electrones. El casco con CRT es muy similar al casco proyectado, sin embargo, este tipo de casco es más pesado que la mayoría de los otros tipos de casco debido a los componentes electrónicos que le son agregados, lo que también conlleva a la generación de grandes cantidades de calor, lo cual puede hacer que el

individuo que usa este tipo de dispositivo se sienta incómodo.

1.5.1.14 *HMD proyectado*

En este tipo de casco se utiliza la fibra óptica para transmitir la escena a la pantalla. Esta es similar a la CRT con la diferencia de que el fósforo es iluminado por la luz transmitida a través de la fibra óptica, donde cada fibra controla una celda con varios píxeles (Figura 1.7).

El casco proyectado proporciona mejor resolución y contraste que el despliegue de CTR, esto significa que el individuo es capaz de ver una imagen con mucho mayor detalle. La desventaja de este tipo de dispositivos es que es caro y complicado de fabricar.

Figura 1.7

Esta es la forma en la que un HMD es montado en la cabeza de una persona, utilizando fibra óptica la calidad de las imágenes es superior.

1.5.1.15 *Dispositivos sonoros*

La utilización de sonido provee un canal de comunicación muy importante dentro de los sistemas de Realidad Virtual puesto que el sistema auditivo es uno de nuestros componentes preceptúales más importantes. Usar sonido para proporcionar información alternativa o suplementaria a un usuario de computadora puede aumentar en gran medida la cantidad de información que ellos pueden ingerir. El sonido estéreo convencional fácilmente puede poner un sonido en el lado izquierdo y el derecho. Sin embargo, el sonido 3D, puede ser colocado en cualquier lugar, ya sea en el lado izquierdo, derecho, arriba, abajo, cerca o lejos.

1.5.1.16 *Sonido 3D*

Además de una composición visual, un mundo virtual válido de experiencias, se debe incorporar un campo de sonido tridimensional que proporcione al usuario una imagen fidedigna de las condiciones que se desean presentar en el ambiente virtual que se esté experimentando. El tener un campo de sonido que reaccione al ambiente (como el reflejo del sonido en paredes, tener múltiples fuentes de sonidos, ruido de fondo, etc.) requiere de un gran poder de cómputo puesto que para tener una buena simulación de un ambiente virtual, una computadora debe determinar la posición de la fuente relativa al oyente, calcular los efectos del ambiente (como por ejemplo un eco en la pared). El problema de producir un sonido es que no se puede repetir el sonido previamente grabado de manera que el sonido se

escuche detrás del usuario cuando este gira su cabeza.

Para crear un campo de sonido tridimensional se produce sonido que es sintonizado a la cabeza de un individuo. Cuando el sonido alcanza el oído externo, este dobla al frente de la onda del sonido y conduce este al canal del oído. El sonido que realmente alcanza el tambor del oído es diferente para cada persona. Para resolver este problema, la computadora debe crear un sonido que sea diseñado para adecuarse a un usuario en particular. Esto se logra al colocar un micrófono pequeño dentro del canal del oído, para crear sonidos de referencia de varias ubicaciones alrededor del oyente. Entonces se resuelven algunas relaciones matemáticas que describen cómo el sonido cambia dentro del canal del oído.

1.5.1.17 *Dispositivos hápticos*

Hace poco menos de 40 años Ivan Sutherland dijo que el sentido humano kinestésico es como otro canal independiente al cerebro, un canal cuya información es asimilada de una manera bastante subconsciente. Háptica, es el estudio de cómo utilizar el sentido del tacto en un mundo generado por computadora. El estimular el sentido del tacto, como permitir al usuario "tocar" objetos de manera que pueda sentir la forma, textura, temperatura, firmeza y fuerza de éstos, puede agregar un buen nivel de realismo al ambiente virtual.

Existen dos subáreas en las que se divide la háptica para su mejor investigación: la retroalimentación de fuerza (kinestética) y la retroalimentación táctil, la primera trata sobre cómo el usuario aplica fuerzas en músculos y tendones por medio de dispositivos que lo hacen sentir las condiciones correspondientes al ambiente virtual que está experimentando; por ejemplo, el usuario debe chocar con una pared en vez de pasar a través de ella; la segunda subárea se enfoca a los nervios terminales de la piel y cómo perciben el contacto con un objeto al sentir las características de éste, como su temperatura, tamaño, forma, firmeza, textura, entre otras.

1.5.1.18 *Plataformas en movimiento*

La plataforma de movimiento nació junto con los simuladores de vuelo, estas plataformas se mueven hacia los lados o se inclinan hacia enfrente o atrás de acuerdo a las imágenes que el individuo está percibiendo, esto le da la sensación de que realmente se está moviendo.

Figura 1.8

*Plataforma móvil de un simulador de vuelo***.**

1.5.1.19 *Guantes*

El uso de guantes es común como un medio de interacción con objetos en un mundo virtual, estos guantes están diseñados especialmente para proveer al individuo de retroalimentación sobre las características de los objetos, los guantes tienen pistones neumáticos montados sobre la palma del guante, de esta forma cuando un objeto es colocado virtualmente en la palma de la mano, la mano verdadera puede cerrarse alrededor del objeto virtual, cuando ésta se encuentre al objeto, la presión en el guante aumentará dando la sensación de resistencia del objeto virtual(Figura 1.9).

Figura 1.9

El guante es una forma muy natural de interactuar con el mundo virtual. Por medio de sensores se proporciona información que es interpretada como movimientos de la mano.

1.5.1.20 *Mayordomos*

El mayordomo es un robot que se encarga de poner un objeto real donde se supone que el objeto virtual se encuentra, es decir, si el individuo toca virtualmente un escritorio, el mayordomo pondrá un objeto al alcance del individuo, para que se tope con el objeto y se de la sensación de toparse realmente con un escritorio. Un inconveniente de este tipo de robots es que solo puede hacer su trabajo para un objeto a la vez.

1.5.2 *Software*

Existen diferentes API's[23] para la creación de simulaciones de mundos virtuales, las cuales dependiendo de lo que deseamos desarrollar nos facilitan la creación de mundos virtuales.

1.6 API's de Realidad Virtual

Las API's de Realidad Virtual son entornos de programación completos que facilitan la creación de mundos virtuales, cada una de estas cuenta con un conjunto de herramientas de desarrollo enfocadas a diversas tareas útiles, lo que nos permitirá darle forma y movilidad a nuestro entorno virtual. A continuación definiremos algunas de las más importantes.

[23] **API's** (Application Programming Interface). Conjunto de especificaciones de comunicación entre componentes de software.

1.6.1 *DIVE* [1]

DIVE (Distributed Interactive Virtual Environment, Entorno Virtual Interactivo Distribuido), es un sistema de Realidad Virtual basado en sistemas multiusuario por Internet, donde los participantes navegan en espacio de 3D y pueden ver, conocer e interactuar con otros usuarios y aplicaciones. El software DIVE es un prototipo de investigación.

DIVE soporta el desarrollo de entornos virtuales, interfaces de usuarios y aplicaciones basadas en entornos 3D compartidos, es especialmente usado para aplicaciones multiusuario, donde muchos participantes interactúan en red.

En las aplicaciones y actividades de DIVE se incluyen campos de batalla virtuales, modelos espaciales de interacción, agentes virtuales, control robótico en mundo real e interacción multimodal.

El visualizador o graficador (*rendering engine*) usado en DIVE está enfocado básicamente para Realidad Virtual de escritorio. Soporta periféricos de Entrada/Salida tales como HMD´s y guantes.

1.6.2 *DVS*

DVS es un producto enfocado a la simplificación del diseño y desarrollo de ambientes virtuales interactivos y de multi-usuarios. DVS se propone como un sistema operativo de Realidad Virtual que provee servicios tales como despliegues visuales, salida de audio, sistemas de posicionamiento y dispositivos de entrada en un nivel abstracto. Una de las ventajas de DVS es que es independiente de plataformas permitiendo a los desarrolladores de Realidad Virtual la portabilidad de sus aplicaciones entre diversos ambientes.

1.6.3 *Render Ware* por *Criterion Software*

Render Ware es un API portable para un número de plataformas, principalmente Windows, Macintosh y X-Windows. Render Ware está dirigido al desarrollo de juegos y ofrece un mapeo de texturas de gran velocidad con un buen soporte multiplataformas. Este API ha ganado buena popularidad entre los desarrolladores y se ha establecido como un excelente graficador.

1.6.4 *3DR por Intel*

3DR es una biblioteca gráfica para PC producida por Intel que permite aplicaciones gráficas a alta velocidad en tiempo real, para desarrollar en ambientes de Windows (NT, 95). 3DR permite un buen uso del hardware disponible y de las tarjetas gráficas avanzadas. 3DR puede importar objetos de 3D Studio.

1.6.5 *World ToolKit por Sense8*

World ToolKit (WTK) es un API 3D comercial que consiste en un conjunto de rutinas en C++ (Aprox. 400) que permite a los desarrolladores construir simulaciones 3D y

aplicaciones de Realidad Virtual. WTK provee de soporte a una gran variedad de hardware incluyendo varias formas de dispositivos de Entrada/Salida tales como HMD's y dispositivos de seis grados de libertad. WTK está disponible en un número de plataformas, incluyendo la PC convencional, aunque su plataforma primaria está basada en máquinas sobre UNIX que usan OpenGL o SGI Performer[24].

1.6.6 *VRML*

El lenguaje de modelado de Realidad Virtual (VRML por sus siglas en inglés) es un estándar para el manejo de escenas tridimensionales dentro de Internet que permite la interacción con el usuario. Con VRML se puede realizar modelación de objetos, permitiendo darles forma, color, movimiento o comportamiento. Este lenguaje permite la incorporación de pequeños programas en Java o JavaScript, lo que le permite agregarle lógica y sentido a los modelos.

1.6.7 *OpenSG* [2]

OpenSG es un pequeño sistema generador de imágenes para crear programas gráficos en tiempo real para aplicaciones de Realidad Virtual. Es desarrollado siguiendo los principios del Open Source[25] y es un software libre. Corre sobre IRIX, Windows y Linux y está basado en OpenGL.

1.6.8 *Vega* [3]

Esta herramienta fue desarrollada para programadores y no programadores, combina herramientas fáciles de usar que permiten la creación de prototipos y permite la realización de simulaciones complicadas, además de incluir herramientas para la construcción, edición y ejecución de aplicaciones sofisticadas. El API que se incluye puede accesar a las API's gráficas como Performer, IrisGL[26] y OpenGL. Soporta sistemas de un solo procesador, multiprocesadores/multi-canales.

Vega es un software multigenerador de entornos para la creación, desarrollo y simulación de realidad usando audio y video en tiempo real y aplicaciones generales de visualización.

Las industrias espaciales y militares utilizan tecnología avanzada de simulaciones con Vega para operaciones críticas como un piloto o un practicante de tiro, diseños aéreos, planeación de misiones o visualización de misiles.

[24] **SGI Performer**. Es un toolkit para desarrollo de imágenes 3D con alto desempeño de renderización, para desarrollo en tiempo real, multiproceso y aplicaciones de gráficas interactivas

[25] **Open Source**. Iniciativa de programadores a nivel mundial de crear programas y poner a disposición pública y de forma gratuita, los códigos fuentes para que otros programadores puedan modificarlos y/o mejorarlos.

[26] **IrisGL**. API gráfica popular gracias a su simplicidad para ser usada.

1.6.9 *ALICE* [4]

Alice es un sistema para desarrollar gráficos en 3D de forma interactiva. Aunque la programación en computadora ha existido en su forma moderna desde hace cincuenta años, ésta ha sido una actividad exclusiva para una pequeña fracción de la sociedad. Existen barreras sociológicas que hacen difícil el proceso de aprendizaje de la programación. Alice trata de derribar esas barreras proporcionando un excelente entorno para aprender a programar en forma fácil y divertida. En lugar de escribir comandos que siguen reglas de sintaxis desconocidas, los estudiantes pueden manipular los programas por medio de una interfaz gráfica.

1.6.10 *AVANGO* [5]

Avango es un software creado para permitir el desarrollo rápido de aplicaciones de entornos virtuales para visualizaciones inmersivas y no inmersivas.

Avango es un ambiente de programación para aplicaciones distribuidas, interactuando con aplicaciones de Realidad Virtual. Utiliza el lenguaje de programación C++ para definir dos categorías de clases de objetos. Los nodos proveen una API de grafo de escena de un objeto orientado, el cual permite la representación de geometría compleja. Los sensores hacen que Avango sea como una interfaz al mundo real. Tiene una amplia respuesta interactiva, soporte rápido a prototipos de desarrollo de aplicación, habilita el desarrollo de verdaderas aplicaciones distribuidas.

1.6.11 *SYZYGY* [6]

El objetivo principal del software SYZYGY es correr un enorme número de aplicaciones de Realidad Virtual sobre un cluster de PC con la misma o mejor calidad de ejecución que en una máquina SGI, la tradicional plataforma de Realidad Virtual. Las comunicaciones entre los procesadores eran un obstáculo y las herramientas necesitaban ser más poderosas para el buen manejo de aplicaciones de cluster. El sistema SYZYGY permite a los usuarios ejecutar aplicaciones de cluster entre máquinas con diferentes arquitecturas y especificaciones, permitiendo trabajar a máquinas viejas y nuevas juntas. SYZYGY provee herramientas de simple software para construir sistemas heterogéneos distribuidos

1.6.12 *Free VR* [7]

FreeVR es una biblioteca fuente de interfaz/integración de Realidad Virtual. Ha sido diseñada para trabajar con una amplia variedad de entradas y salidas de hardware, con muchos dispositivos de interfaz que ya están implementadas. Una de las contribuciones de FreeVR fue la facilidad de correr aplicaciones de Realidad Virtual. Otro diseño muy bueno es hacer más fácil las aplicaciones de Realidad Virtual ya que es compartida entre sitios de investigación activa de Realidad Virtual utilizando diferente hardware en cada sitio.

1.6.13 *Minimal Reality (MR) Toolkit* [8]

El MR Toolkit (Minimal Reality, Realidad Mínima) es una biblioteca que soporta el desarrollo de aplicaciones de Realidad Virtual y otras formas de interfaces de usuarios en tres dimensiones. El MR Toolkit soporta la distribución de interfaces de usuarios en múltiples estaciones de trabajo, datos distribuidos en un gran número de estas estaciones, numerosas técnicas de interacción, administración de geometría física y virtual así como herramientas de análisis en tiempo real.

1.6.14 dVISE [9]

dVISE es un software de entorno, una plataforma independiente para aplicaciones de Realidad Virtual basada en una arquitectura distribuida de multiusuario. dVISE provee un modelo natural para la simulación de entornos virtuales y desarrollo de aplicaciones a gran velocidad. Los sistemas múltiples dVISE pueden ser unidos en entornos de multiusuarios. dVISE es un mundo virtual que crea y simula herramientas de software que permiten al no programador el fácil desarrollo de mundos virtuales, animaciones con inteligencia y propiedades realistas y experiencias parciales o totales en modos inmersivos.

1.6.15 VR Juggler [10]

VR Juggler son herramientas de Open Source en Realidad Virtual. VR Juggler es un proyecto de investigación activo dirigido por la Dra. Carolina Cruz Neira y un equipo de estudiantes del centro de Realidad Virtual de la Universidad del Estado de Iowa.

VR Juggler es una colección de las tecnologías que proporcionan las herramientas necesarias para el desarrollo de aplicaciones de Realidad Virtual; permite que un usuario corra una aplicación casi en cualquier sistema de Realidad Virtual. Actúa como "pegamento" entre los componentes del sistema además de proporcionar una plataforma virtual para el desarrollo de Realidad Virtual.

VR Juggler puede funcionar con cualquier combinación de tecnologías inmersivas y de hardware de cómputo.

Capítulo 2
Sistemas de Realidad Virtual en Red

La historia nos ilustra la evolución de la Realidad Virtual, ahora podemosLos Sistemas de Realidad Virtual en Red utilizan datos, vistas y procesos que son distribuidos. Estos sistemas demandan gran exactitud y confiabilidad para lograr una sensación de interacción en tiempo real. El punto clave del funcionamiento de estos sistemas es la forma en la que estos se comunican y la manera de intercambiar información, se deben tomar en cuenta varios factores como el ancho de banda, la distribución o la latencia, lo que influye en la forma de transferir y visualizar datos.

2.1 ¿Qué es un sistema de Realidad Virtual en red?

Un sistema de Realidad Virtual en red es un sistema de software en el cual múltiples usuarios interactúan en tiempo real, inclusive si estos usuarios se encuentran localizados en cualquier parte del mundo. De lo que se trata es que cada usuario acceda al ambiente virtual desde su propia computadora o estación de trabajo, utilizando una interfaz.

Un sistema de Realidad Virtual en red se caracteriza por los siguientes puntos:

- ***Una sensación de espacio compartido:*** todos los participantes tienen la sensación de encontrarse en el mismo lugar.
- ***Una sensación de presencia compartida:*** cuando los participantes entran al mismo lugar toman una forma virtual llamada “avatar”, la cual incluye una representación gráfica del participante, que puede ser un cuerpo humano, un modelo entre otras.
- ***Una sensación de tiempo compartido:*** los participantes podrán ver la conducta de los demás en el momento en que ocurra un cambio, en otras palabras podrá interactuar en tiempo real.
- ***Una forma de comunicación:*** el agregar comunicación entre los participantes, como gestos, texto o voz da una mayor sensación de realismo**.**
- ***Una forma de compartir:*** es aquí donde se demuestra el verdadero poder de un sistema de Realidad Virtual en red usando no solo la habilidad de interactuar con otros participantes, sino con el entorno en sí.

Un sistema de Realidad Virtual de Red provee a múltiples usuarios de la habilidad de interactuar con los demás, compartir información y manipular objetos en dicho entorno, todo esto logrado a través de gráficas inmersivas.

2.2 Motores gráficos y pantallas

Los motores gráficos y las pantallas son la base de la interfaz de los sistemas de Realidad Virtual en red. Las pantallas proveen al usuario de una ventana en tres dimensiones que permiten visualizar el entorno virtual y el motor gráfico genera las imágenes para dichas pantallas.

Algunos años atrás solía decirse que este tipo de gráficas estaba solo disponible para estaciones de trabajo de alta resolución gráfica, pero en estos años, las PC se han convertido en poderosas máquinas de alta resolución gráfica.

Aunque las pantallas dan una alta calidad de graficas 3D, tradicionalmente las pantallas ofrecían inmersión limitada y además el usuario podía seguir siendo distraído fácilmente por la luz de la visión periférica. Para una alta experiencia inmersiva, los Sistemas de Realidad Virtual en Red usan dispositivos para bloquear la experiencia con el mundo exterior, de esta manera se tendrá una percepción más completa del mundo virtual.

2.3 Escalabilidad

La Escalabilidad se refiere al tamaño del sistema de Realidad Virtual en red, el cual es medido por el número de entidades que pueden participar simultáneamente en el sistema.

Una entidad en este sistema es un objeto que participa en forma independiente y es modelada por una computadora participante en el entorno. Estas entidades pueden ser vehículos controlados por humanos o computadoras; por ejemplo rocas, árboles, edificios e inclusive objetos abstractos como el clima.

Estos sistemas requieren un amplio rango de escalabilidad. De igual forma una medida de escalabilidad es considerar el número de usuarios que pueden conectarse simultáneamente al sistema y la distancia física existente entre los participantes del sistema de Realidad Virtual en red.

La escalabilidad depende de una variedad de factores, incluyendo la capacidad de la red, la capacidad de procesamiento, velocidad de despliegue de imágenes y la velocidad a través de los servidores compartidos.

La complejidad de un sistema de Realidad Virtual en red se incrementa exponencialmente de acuerdo al número de entidades participantes, esto debido al número posible de interacciones entre esas entidades. Una interacción particular puede involucrar cualquier combinación entre las entidades participantes en el sistema, entonces hay $2^{(\text{número de entidades})}$ posibles interacciones entidad-entidad. Sin embargo, el número de interacciones reales generalmente no se incrementan tan rápido como el número de posibles interacciones. Como regla, una entidad en un sistema de Realidad Virtual en red no interactúa con todas y cada una de las entidades en el sistema.

2.4 Sistemas de Procesamiento

Un sistema de Realidad Virtual en red requiere de una gran cantidad de capacidad de procesamiento. El procesador recibe eventos de los dispositivos de entrada de los usuarios y computadoras, estos eventos representan cambios constantes en la posición de los objetos dentro del mundo virtual, el procesador determina cómo y cuándo notificar a los otros participantes los eventos ocurridos.

De manera similar, el procesador toma la información recibida de los otros participantes, describiendo su ubicación y su conducta dentro del entorno virtual. Finalmente, anima las gráficas para mantener actualizada constantemente la ventana que muestra el entorno virtual.

2.5 Retos en el diseño y desarrollo de sistemas de Realidad Virtual en red

Los Sistemas de Realidad Virtual en Red son muy difíciles de implementar. Son complejos debido a que son una mezcla de diversos tipos de software combinados en una sola aplicación. Los Sistemas de Realidad Virtual en Red son:

- ***Sistemas Distribuidos:*** deben tomar en cuenta la administración de los recursos de una red, la pérdida de datos, las fallas en la red y la concurrencia.

- ***Aplicaciones Gráficas:*** debe mantenerse, al dibujar los cuadros, una velocidad cercana al tiempo real y con cambios suaves, se debe planear cuidadosamente el uso del CPU entre el *rendering*[1] y otras tareas.

- ***Aplicaciones Interactivas:*** deben procesar información de entrada de los usuarios en tiempo real. Los usuarios deben ver el entorno virtual como si existiera localmente, incluso si los participantes están distribuidos en múltiples lugares remotos.

El diseño de los sistemas de Realidad Virtual en red se vuelve más complejo debido a que dichos sistemas deben trabajar con un número determinado de aplicaciones existentes.

Típicamente estos sistemas deben integrar bases de datos que almacenan información de manera persistente. Los sistemas de Realidad Virtual en red necesitan soportar la autenticación de usuarios y poder interactuar con las transacciones del sistema.

Para soportar sistemas de ingeniería de desarrollo, los sistemas de Realidad Virtual en red deben permitir el registro de los eventos en tiempo real, lo que significa el almacenamiento persistente de la información, esta tarea es complicada debido a que el estado completo de un sistema de Realidad Virtual en red no puede ser conocido totalmente por ningún usuario en el sistema.

[1] **Rendering**. Proceso de cálculo complejo desarrollado por una computadora para generar una o varias imágenes.

2.6 Comunicación en la Red

Muchos aspectos de las comunicaciones en red son responsables de definirla como se distribuye el sistema de Realidad Virtual en red.

2.6.1 *Ancho de banda*

El ancho de banda determina el tamaño y la riqueza del entorno virtual, al aumentar el número de participantes en el entorno, aumentarán los requerimientos de ancho de banda.

Si un participante desea recibir información detallada acerca de la actividad de otro participante, la información debe ser enviada a través de la red; si se desean obtener más detalles, entonces debe ser enviada más información. De forma parecida, cuantos más usuarios participen en el entorno virtual, la cantidad de la información agregada aumentará.

Sin embargo, la capacidad de la red es un recurso limitado, por ello el diseño del sistema de Realidad Virtual en red debe ser cuidadoso para determinar que tanta capacidad de almacenamiento es necesaria. Los entornos virtuales distribuidos requieren una gran cantidad de ancho de banda para soportar múltiples usuarios, audio, video y el cambio de graficas 3D primitivas y modelos en tiempo real. Más aún, la mezcla de datos requiere nuevos protocolos y técnicas para manejar los datos apropiadamente sobre la red.

Por ejemplo, en el proyecto computacional de aerociencia de la NASA está planeando utilizar redes de alta velocidad para soportar visualizaciones con mayor fluidez(en el campo de los fluidos), Fluidos Computacionales Dinámicos (CFD).

2.6.2 *Distribución*

La distribución es uno de los puntos que define la calidad en un sistema de Realidad Virtual en red. En algunos sistemas de Realidad Virtual en red el entorno es formado por varios componentes ubicados en diversas máquinas.

Para que esto sea efectivo, los sistemas de Realidad Virtual en red deben presentar a cada usuario la ilusión de que el entorno virtual se encuentra almacenado en su propia máquina y que las acciones que el usuario realice dentro del ambiente virtual tengan un impacto directo e inmediato en el entorno. El sistema necesita atenuar cualquier inconveniente que se presente, esto debido a que la aplicación será distribuida y vista naturalmente.

Algunos esquemas de distribución son mostrados en la Figura 2.1.

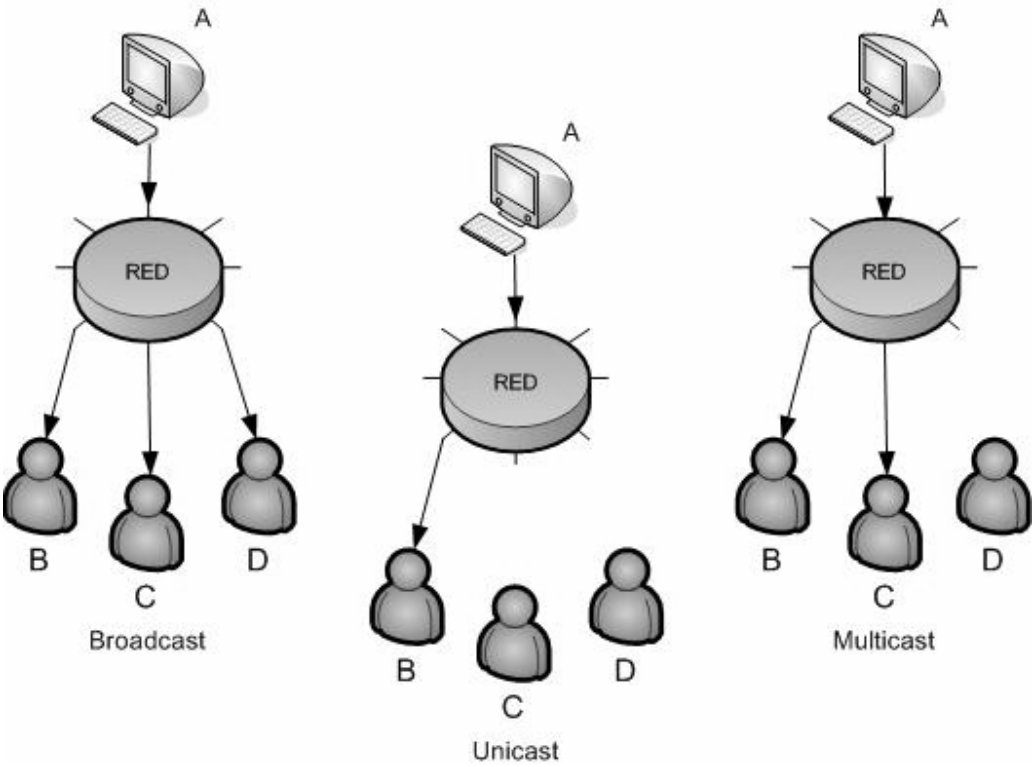


Figura 2.1
Broadcast, Multicast y Unicast
Los tres tipos de distribución.

Los servicios Multicast permiten a grupos de tamaño variable comunicarse sobre una red por medio de una simple transmisión generada por la fuente. Multicast provee servicios de envío "uno a muchos" y "muchos a muchos" para aplicaciones en la que es necesario comunicarse con muchos usuarios, como una teleconferencia o una simulación distribuida

Con "Broadcast", los datos son enviados a todos los usuarios mientras que con "Unicast" o "punto a punto" se establece comunicación solamente entre dos usuarios.

La mayoría de los entornos virtuales distribuidos emplean alguna forma de "Broadcast" o "Unicast"; sin embargo, esos esquemas son ineficientes en cuanto a ancho de banda se refieren para grandes grupos. Más que eso "Broadcast" no es apropiada para trabajos por Internet porque la red se toma lenta con tráfico no deseado y es difícil evitar las rutas cíclicas, además las "IP-Broadcast" requieren que todos los usuarios examinen un paquete aún si la información no es requerida por ese usuario.

"Punto a punto" requiere el establecimiento de una conexión o de un camino para cada nodo hacia otro nodo en la red para un total de N*(N-1) conexiones virtuales en un grupo, cada 1000 usuarios deberá tener direcciones separadas y enviar 999 paquetes idénticos (Figura 2.2).

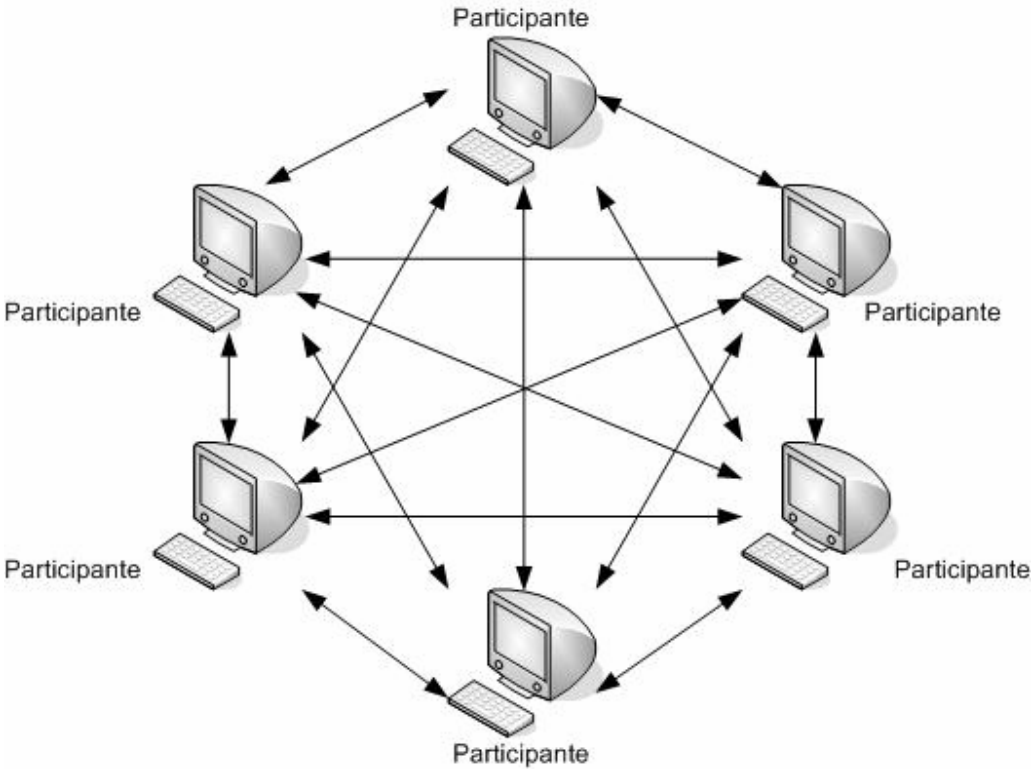


Figura 2.2
Modelo Distribuido punto a punto

Algunos investigadores tienen propuestas, diferentes ideas para usar Multicast para soportar los entornos virtuales. Mantener la ilusión de un solo sistema es difícil. Cada usuario debe, por lo tanto, intentar presentar una visión consistente en tiempo real del sistema de Realidad Virtual en red y considerar el hecho de que toda la información entrante proveniente de los usuarios remotos está totalmente fuera de tiempo cuando llega, debido al tiempo de transmisión de la información.

Esos retardos en la red son particularmente difíciles de manejar cuando múltiples usuarios o componentes interactúan con otros directamente. Un sistema de Realidad Virtual en red debe dar una detección precisa de colisiones, acuerdos y resoluciones entre los participantes. La detección precisa de colisiones es difícil porque en cualquier punto en el tiempo, ningún usuario tiene la información exacta de las posiciones de los otros usuarios, como ya se mencionó. La red se demora, es decir, que toda la información recibida está fuera de tiempo.

En los juegos, una colisión se presenta cuando una bala golpea su objetivo. Incluso un simple apretón de manos envuelve muchas colisiones. Una simulación debe calcular las constantes interacciones entre el vehículo y el camino que es conducido sobre la tierra o en el aire con el flujo del viento. Todas estas interacciones que se dan en un sistema de Realidad Virtual en red incluyen fuerzas de fricción, calor e inclusive información acústica. La distribución

natural de un sistema de Realidad Virtual en red, además, complica la transmisión de la comunicación de audio, ya que el receptor debe determinar cómo atenuar el audio basado en distancia virtual.

2.6.3 *Latencia*

Otro aspecto a contemplar de las comunicaciones es la "Latencia", la cual controla la naturaleza interactiva y dinámica del entorno virtual. Si un entorno distribuido emula el mundo real, debe operar también en términos de tiempo real para la percepción humana, lo cual es algo complicado puesto que envuelve operadores humanos que deben enviar paquetes con un mínimo de tiempo y generar texturas en 3D para garantizar la ilusión de realismo. Eso no es todo, se debe incluir audio realista y video, así como servicios de comunicación entre participantes dentro del entorno virtual.

La latencia de la red puede ser disminuida gracias al uso de ciertas herramientas como los distintos protocolos de comunicación, implementando y mejorando la función de los *routers*[2] y combinando varias tecnologías, haciendo más rápidas las interfaces en las computadoras. Sin embargo, muchos piensan que incrementando el ancho de banda es la solución, pero no es en definitiva lo único que hay que considerar. Operando a velocidades de "Gigabit" se presenta una nueva gama de problemas. Se requieren nuevos métodos para manipular la congestión debido a la alta velocidad de envío de datos.

Los cuellos de botella se dan mucho en este tipo de interfaces de red, problemas con el espacio en memoria, conflictos con los diversos sistemas operativos, entre otros.

Otros métodos están disponibles para aminorar los efectos de la latencia. Existen técnicas de *Dead Reckoning*[3] que reducen las cargas en la comunicación sobre la red y los retardos, ya que pueden predecir el modelo que poseen los diferentes participantes. Así, una entidad local pasa vectores de estado hacia simulaciones remotas. Sin embargo, los retrasos nunca pueden ser totalmente eliminados y menos para entornos virtuales ampliamente distribuidos.

2.6.4 *Confiabilidad*

La confiabilidad en las comunicaciones siempre demanda un compromiso entre el ancho de banda y la latencia. La confiabilidad significa que el sistema puede asumir lógicamente que los datos enviados son siempre recibidos correctamente, para que esto sea cierto es necesario que los datos sean re-enviados periódicamente.

Desafortunadamente para garantizar el envío, la arquitectura de la red debe usar un esquema de reconocimiento de errores y recuperación que pueden generar enormes cantidades de retardos en la red. Eso no es todo, algunos protocolos como el TCP[4], utilizan mecanismos

[2] **Routers**. Dispositivos hardware o software de interconexión de computadoras, los cuales interconectan segmentos de red o redes enteras para comunicación por medio de paquetes de datos.

[3] **Dead Reckoning**. Técnica utilizada en Realidad Virtual para conocer el estado de un objeto dentro de un mundo virtual por medio del cálculo de sus coordenadas y desplazamiento.

[4] **TCP** (*Transmisión Control Protocol*). Protocolo básico de comunicación en Internet que permite la

que causan congestión, esta congestión se refleja en el tráfico de datos en tiempo real.

Los protocolos confiables Multicast no son, actualmente, prácticos para grandes redes porque para garantizar que los paquetes son correctamente recibidos a cada participante en la red, es requerido un reconocimiento y una retransmisión, lo que obviamente causa efecto en la simulación en tiempo real.

Los investigadores intentan desarrollar un protocolo Multicast confiable y escalable. Brian Whetten y Simon Kaplan desarrollaron el confiable protocolo Multicast (RPM, por sus siglas en inglés, Reliable Protocol Multicast), que está basado en el protocolo *Token Ring*[5].

Netrek es un juego de multi jugadores vía Internet. Toma aproximaciones usando diferentes grados de confiabilidad para obtener un buen funcionamiento en tiempo real. Versiones antiguas de este juego utilizaban protocolo TCP, las versiones recientes tienen un protocolo que:

- Garantiza la confiabilidad y de algunos paquetes TCP tales como condiciones de error y el establecimiento de la conexión y para la información que es enviada con poca frecuencia
- Permite el "switcheo" sobre demanda, desde TCP a UDP[6]/TCP y al revés
- No garantiza la confiabilidad por datos frecuentes y datos no críticos
- No terminará la sesión en forma anormal si se pierde un paquete UDP

2.7 Vistas

Las vistas son las ventanas al mundo virtual desde la perspectiva del participante. Existen dos tipos de vistas para los entornos distribuidos.

2.7.1 *Vista en sincronía*

En un simulador de vuelo, donde una máquina controla la imagen principal central y otras dos máquinas procesan las ventanas, izquierda y derecha respectivamente, dan la ilusión al participante de que está en una sola cabina de vuelo.

La sincronía demanda alta confiabilidad y baja latencia. Además de eso, los entornos

transmisión de información en redes de computadoras.

[5] **Token Ring**. Es una arquitectura de red con topología de anillo.

[6] **UDP** (*User Datagram Protocol*). Protocolo de transporte de información basado en el intercambio de fragmentos de paquetes a través de la red sin que se haya establecido previamente una conexión, ya que el propio fragmento de paquete incluye suficiente información de direccionamiento.

virtuales que requieren vistas en sincronía son, por razones obvias, delimitadas a redes locales. Las vistas en sincronía son además importantes para el diseño computarizado.

El instituto Fraunhofer para Gráficos por Computadora ha desarrollado un prototipo virtual que implementa una vista asistida por computadora compartida en 3D.

2.7.2 *Vista asíncrona*

En este tipo de vista, múltiples usuarios tienen el control individual sobre cuando y qué pueden ver en el entorno virtual actual (Figura 2.3).

Los participantes pueden estar físicamente separados dentro de la red, pero tienen el conocimiento de que los demás participantes están dentro del entorno también, los cuales son representados por un objeto dentro del entorno virtual.

Un sistema de Realidad Virtual en red a gran escala utilizará vistas asíncronas porque el costo de sincronización sobre redes grandes es muy elevado.

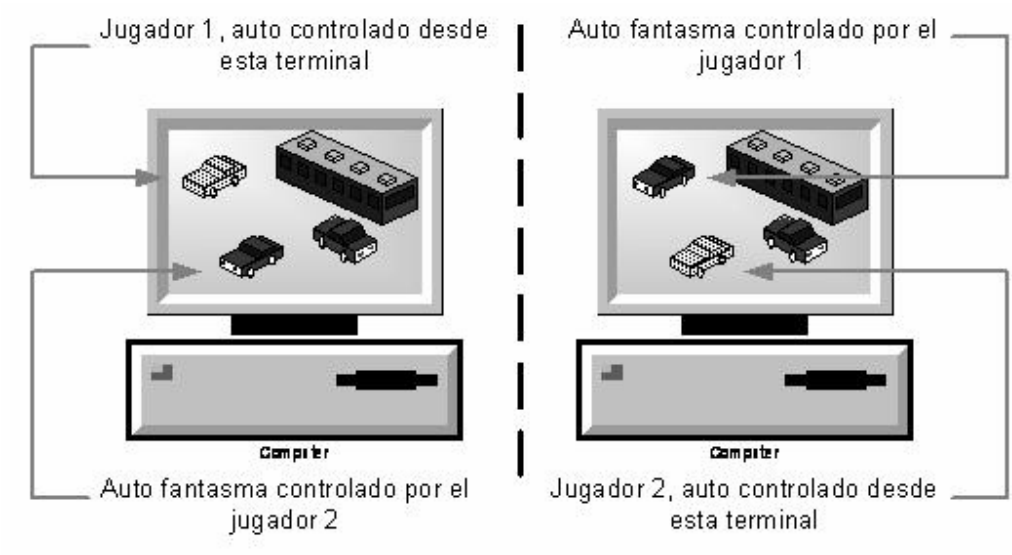


Figura 2.3
Vista asíncrona de simulación de dos distintos Jugadores

2.8 Datos en la red

Los participantes intercambian información dentro del sistema de Realidad Virtual en red. Por ejemplo: un participante se mueve dentro del entorno virtual, este participante debe transmitir actualizaciones sobre la red de manera que los otros usuarios visualicen su posición correcta. Similarmente, si un participante recoge un objeto en el entorno virtual, los demás participantes deben estar enterados de que el objeto es ahora llevado por alguien más.

Posiblemente lo más difícil de la construcción de un entorno distribuido es dónde colocar los datos relevantes del estado del mundo virtual y sus objetos. Estas decisiones afectan la escalabilidad, los requerimientos de comunicación y la confiabilidad de los datos en la red. Existen muchas opciones para la distribución de datos, aquí se mencionan algunas de las más relevantes.

2.8.1 *Sistemas virtuales heterogéneos*

Un método común es inicializar el estado de cada sistema que participa en el entorno virtual distribuido con un mundo homogéneo de datos que contengan información acerca del terreno, modelos geométricos, texturas y la conducta de todo lo que sea representado en el entorno virtual.

En el mundo real, los participantes no tienen acceso al mismo tipo de equipo para explorar mundos virtuales, mientras algunos participantes podrían estar usando computadoras de escritorio conectadas a la línea telefónica, otros podrían estar utilizando un casco de inmersión completa con guantes incorporados a múltiples procesadores conectados a Ethernet.

Esta idea de sistemas de Realidad Virtual en red heterogéneos es deseable, de esta forma ninguno de los participantes estarían en desventaja sin tomar en cuenta el realismo que se puede conseguir con sistemas especializados de inmersión.

2.8.2 *Base de datos compartida y centralizada*

Por otro lado, los sistemas Virtuales de Telecomunicaciones Espaciales (VISTEL), utilizan una base de datos que contiene el mundo compartido. Como su nombre lo implica, VISTEL es un sistema de teleconferencias que muestra modelos en 3D para cada uno de los que forman parte de la teleconferencia. Los cambios que se hacen en la forma del modelo son enviados por medio de mensajes al servidor central para ser después redistribuidos. Cabe aclarar que solamente un usuario a la vez puede modificar la base de datos (Figura 2.4).

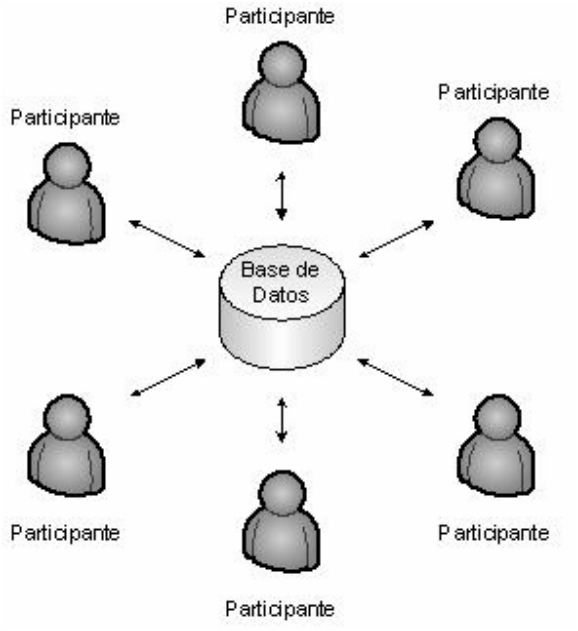


Figura 2.4
Modelo Centralizado

Para comunicaciones por texto, típicamente soporta cincuenta usuarios a la vez, los cuales se "mueven" entre el cuarto, creando y borrando nuevas acciones u objetos y comunicándose con otros participantes dentro del entorno. El problema de este esquema es que toda la información es enviada al servidor central, el cual tiene que retransmitir la información a los demás participantes, incrementando el ancho de banda, ya que el mensaje que es enviado por un participante debe ser distribuido a todos los demás. Esto podría causar cuellos de botella y diferentes problemas en la red.

2.8.3 *Base de datos compartida y distribuida peer-to-peer*[7]

Muchos sistemas distribuidos estriban en simular arquitecturas de memorias compartidas, pero la desventaja que presentan éstas es que son difíciles de escalar, ya que los costos de las comunicaciones asociados con el mantenimiento de la confiabilidad y la consistencia de datos a través de la red son muy elevados.

Este tipo de distribución de datos utiliza el lenguaje de programación paralelo llamado Linda, además del funcionamiento para un modo de programación que es relativamente simple.

Linda es un lenguaje de programación paralela basado en C (C-Linda) y Fortran (Fortran-Linda); el diseño de este leguaje está coordinado de tal forma que puede combinarse

[7] **PEER-TO-PEER** (Par-a -Par). Esta palabra hacer referencia a un tipo de proceso en paralelo.

con lenguaje C y Fortran.

Linda es capaz de crear programas paralelos que funcionan en un amplio espectro de plataformas. Está basado en un modelo de memoria virtual compartida, llamado *tuple space*, que provee una comunicación y sincronización interproceso que es independiente de la plataforma. En el modelo *tuple space* diferentes datos pueden vivir sobre diferentes procesadores, pero los componentes del proceso se ven como una memoria global, dicho de otra forma, el uso de diferentes procesadores para un trabajo que corre bajo Linda crea una máquina virtual con memoria compartida.

Linda implementa el paralelismo con un pequeño número de operaciones sencillas sobre la memoria virtual compartida, para crear y coordinar procesos paralelos asignando la distribución maestro/esclavo para el algoritmo.

El funcionamiento de un sistema multiusuario está limitado a tres participantes. La simplicidad e ilusión de la memoria compartida presentada por el lenguaje de programación paralelo es, además, la razón del porqué este tipo de sistema sufre por la pobre demostración. Los datos deben residir en algún lugar. En este caso, es en el servidor central.

2.8.4 *Base de datos compartida y distribuida cliente-servidor*

Una técnica diferente es utilizar una variante del modelo cliente-servidor en el cual la base de datos es particionada entre todos los participantes y la comunicación es mediada por el servidor central. Esto significa que a cada participante se le dará una parte del mundo virtual y cuando otro participante requiera esa parte hará la petición a través del servidor central.

Algunos de los sistemas cliente-servidor, utilizan un Procedimiento Remoto de Llamadas (RPC) que no es del todo escalable por un variado número de razones.

Una característica negativa del RPC se presenta desafortunadamente en redes de alta velocidad ya que al enviar un mensaje necesariamente se requiere esperar la respuesta. Como se vayan incrementando los retrasos en la red, el RPC se vuelve más caro.

2.8.5 *Desarrollo y configuración*

Si el software de un sistema de Realidad Virtual en red es amplio y consistente, sería inapropiado para descargar. Al contrario, el software debe ser diseñado como una biblioteca compacta y que los componentes sean dinámicamente descargados dependiendo de los cambios necesarios de ejecución.

La estructuración será compleja si el sistema de Realidad Virtual en red es ejecutado dentro de un *Web Browser*[8] sobre Internet. En este caso, el diseñador del sistema debe tomar en cuenta ciertos puntos:

[8] **Web Browser**. Aplicación software que permite recuperar y visualizar documentos de hipertexto, gráficos, secuencias de videos, sonido, animaciones, vínculos y programas diversos desde servidores Web de todo el mundo a través de Internet.

- Que el entorno sea fácil de descargar.
- Que la implementación del sistema sea de forma segura y disponible para diferentes entornos.
- Que el software se ejecute correctamente en las diferentes plataformas existentes.

Los participantes deben tener acceso a la información de configuración, seguridad (claves de acceso y códigos), imágenes y modelos computacionales de los diferentes participantes y más. El desarrollador de sistemas de Realidad Virtual en red debe colectar esta información y hacerla disponible para todos los participantes.

2.9 Procesos

Los procesos distribuidos a múltiples usuarios incrementan la necesidad de poder de cómputo para la simulación. Estos procesos no solo se pueden utilizar para proveer la capacidad de distribuir vistas, sino para manipular una gran variedad de dispositivos de entrada.

Prácticamente todos los entornos distribuidos asumen que el mismo tipo de procesos están corriendo en cada uno de los participantes que tengan las mismas funciones. La ventaja de este acercamiento es la consistencia. La desventaja es que es muy inflexible.

Los lenguajes *script*[9] son herramientas de propósitos generales, ya que pueden proveer la capacidad de migrar procesos y objetos a través de diferentes plataformas. JAVA[10] de SUN[11], es el primer ejemplo de esta clase. Un byte[12] compilado, lenguaje interpretativo, similar a C++[13], JAVA es el mejor en arquitecturas cliente-servidor en el *World Wide Web*[14]. Además de eso, JAVA soporta el protocolo Multicast de comunicación.

[9] **Lenguajes Script**. Lenguajes interpretados que forman un subconjunto de los lenguajes de programación. A diferencia de los lenguajes compilados, los lenguajes script no necesitan ser preprocesado mediante un compilador, esto significa que la computadora es capaz de ejecutar las instrucciones dadas sin tener que leer y traducir todo el código.

[10] **JAVA**. Plataforma de software desarrollada por Sun Microsystems de tal forma que los programas desarrollados en ella se pueden ejecutar en diferentes tipos de arquitecturas y dispositivos computacionales.

[11] **SUN** (Sun Microsystems). Empresa productora de software, semiconductores y equipos informáticos de Estados Unidos de América.

[12] **Byte**. Unidad básica de almacenamiento de información generalmente equivalente a 8 bits de datos.

[13] **C++**. Lenguaje de programación orientado a la implementación de sistemas operativos que proporciona orientación a objetos. Es la versión de C mas aceptada, difundida y funcional.

[14] **World Wide Web** (Telaraña Mundial). Sistema de hipertexto que funciona sobre Internet, utilizando como visualizador de información una aplicación llamada *Web Browser*.

Gavyn Bell y Tony Parisi de SGI[15] han desarrollado el Lenguaje de Modelado de Realidad Virtual (VRML), que es un lenguaje que describe múltiples participantes simulando interacciones, mundos virtuales en red por medio de Internet global con la *World Wide Web*.

[15] **SGI** (Silicon Graphics Incorporated). Empresa dedicada al desarrollo de cómputo de alto desempeño, almacenamiento y tecnología de visualización.

Capítulo 3
Plataformas de software para Realidad Virtual distribuida

Teniendo el conocimiento del funcionamiento y los requerimientos de un sistema de Realidad Virtual en Red podemos preparar el desarrollo de una aplicación, para lo cual es necesario determinar ahora que tipo de plataforma se utilizará para la implementación. Para esto es imprescindible el análisis de algunas de las diversas tecnologías de Realidad Virtual Distribuida, mediante el cual podremos conocer y seleccionar la plataforma que más se adapte a nuestras necesidades y posibilidades.

3.1 CAVERN: Una arquitectura distribuida para soportar entornos virtuales colaborativos, escalables, persistentes y con interoperabilidad

Cavernsoft es un software colaborativo que utiliza mecanismos de almacenamiento de datos distribuidos ligeros para manipular un amplio rango de volúmenes de datos que son necesarios, típicamente para Entornos Virtuales persistentes.

La Realidad Virtual Colaborativa (CVR) es actualmente un área de amplios retos en la investigación de Realidad Virtual, la parte colaborativa agrega una nueva dimensión de factores humanos complejos, redes y bases de datos. Esto requiere un uso diverso de la red, diferentes tamaños de bases de datos y de gráficos. Cavernsoft es una arquitectura que soporta Realidad Virtual Colaborativa.

3.1.1 *Escenarios representativos de la Realidad Virtual Colaborativa*

Existen varios contextos en los que se puede hablar Realidad Virtual Colaborativa debido a su relación con los objetos y a su participación en los entornos virtuales, a continuación se describirán algunas características representativas.

3.1.1.1 *Diseño e ingeniería colaborativa*

El trabajo de diseño colaborativo típicamente relaciona a un pequeño grupo de usuarios, en sincronía o no, involucrados con la construcción y manipulación de objetos en el mundo virtual.

La Alianza de Cómputo Nacional de la Ciencia (NCSA), ha estado trabajando en unión con Caterpillar[1] Bélgica, para desarrollar un sistema que permita localizar a los ingenieros

[1] **Caterpillar**. Empresa dedicada a desarrollar equipo para construcción y minería, además de varios tipos de motores para vehículos.

remotamente para trabajar en el diseño de nuevos vehículos de construcción. Esta es una forma práctica de mantener cerca a ingenieros de América y Europa.

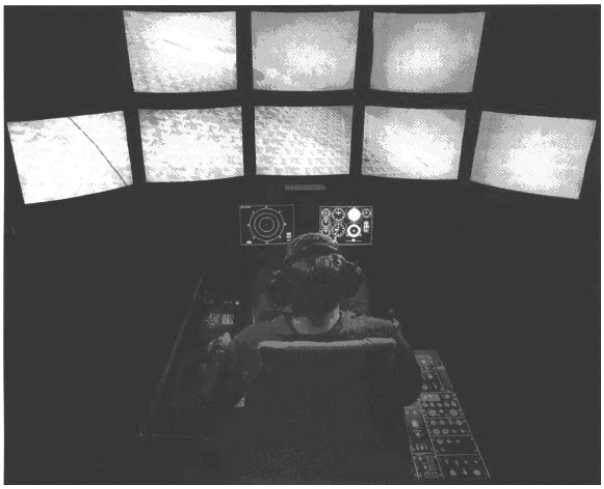
Para soportar comunicaciones usuario-usuario se utilizan herramientas de audio y video para teleconferencias que fueron modificadas para trabajar con CAVE[2], video e imagen de cada participante es mapeado y texturizado para establecer la presencia de este en el entorno.

3.1.1.2 *Entrenamiento colaborativo*

Para entrenar ejércitos en el campo de batalla fue utilizado SIMNET[3] y NPSNET[4], en estos simuladores el participante podía utilizar un casco HMD, el cual entra en el entorno como un soldado junto con otros participantes (utiliza el *Dead Reckoning*[5] para conocer la ubicación de los demás). SIMNET permite gran complejidad y realismo.

Figura 3.1

El simulador SIMNET, controlado por un soldado durante su entrenamiento.



3.1.1.3 *Visualización científica colaborativa*

La visualización científica colaborativa consiste en que un grupo de científicos remotamente localizados, entren en un entorno virtual para discutir acerca de los datos que están visualizando, lo más importante es que permita a los diferentes científicos expresar

[2] **CAVE**. Proyecto basado en sistemas de Realidad Virtual que proporciona una perspectiva en función de los movimientos de la cabeza en tiempo real, con un amplio campo de vistas, controles interactivos y pantallas estereoscópicas. Es un cubo de 10x10x10 pies con imágenes proyectadas en tres paredes y en el piso.

[3] **SIMNET** (Simulator Networking). Proyecto de simuladores vía WWW que permitía a soldados practicar habilidades de batalla participando en una guerra virtual.

[4] **NPSNET**. Entorno virtual para múltiples participantes sobre Internet.

[5] **Dead Reckoning**. Técnica utilizada en Realidad Virtual para conocer el estado de un objeto dentro de un mundo virtual por medio del cálculo de sus coordenadas y desplazamiento.

sus opiniones sobre lo que están trabajando. Estas visualizaciones no son homogéneas puesto que cada uno ve un ángulo diferente de la imagen, dependiendo de su posición en el entorno de la visualización.

3.1.2 CALVIN & NICE, la prehistoria de Cavernsoft

Antes de que Cavernsoft fuera una plataforma de Realidad Virtual sólida y confiable fueron desarrollados algunos entornos virtuales de los cuales sobresalieron dos de ellos que poseen características peculiares, éstos se describen a continuación.

3.1.2.1 *CALVIN, (Arquitectura colaborativa en capas por Navegación Inmersiva)*

CALVIN es un entorno virtual colaborativo que permite a múltiples usuarios, en sincronía o asincronía, experimentar con una arquitectura de habitaciones en capas.

En CALVIN, los participantes pueden diseñar lo que quieran y entrar en el entorno como “mortales” o ”deidades” (mortales refiriéndose a entrar como una persona normal con visión limitada y deidades refiriéndose a que al entrar al mundo virtual, estos pueden verlo todo).

CALVIN tiene una interfaz bilingüe, Japonés-Inglés, además, los participantes pueden guardar su progreso dentro del entorno, así que la siguiente ocasión que entren, se carga el entorno tal y como estaba cuando lo abandonaron la última vez, no importando que cada participante tenga su propia versión, ya que CALVIN es como un árbol con muchas ramas, y en cada rama es posible que exista un árbol.

CALVIN utiliza un modelo compartido variable basado en un sistema de memoria compartida distribuida (DMS) para eliminar la necesidad de que los programadores desarrollen protocolos específicos para comunicaciones en la red. La DMS utiliza un protocolo confiable además de clases programadas en C++, aunque es confiable introduce “latencia” que es aceptable en pequeños grupos.

El problema es cuando dos participantes a la vez tratan de tomar el control sobre un mismo objeto, ya que no existe una regla que bloquee el objeto al tratar de ser modificado por un participante.

Figura 3.2
Una pequeña representación virtual (avatar) dentro de CALVIN.

3.1.2.2 *NICE Entornos Inmersivos, Narrativos Colaborativo/Constructivos.*

NICE es un entorno virtual para niños, una isla donde crecen plantas y flores, es un jardín virtual con diferentes niños como representación virtual, los cuales pueden cortar flores y recoger vegetales, lo que hace diferente a NICE de otros es que, aunque todos los participantes hayan salido del entorno, este sigue creciendo y modificándose. NICE utiliza un protocolo no confiable y no tiene un límite específico de usuarios.

Figura 3.3

Dos pequeñas representaciones virtuales (avatares) dentro de NICE.

Un niño controlando su representación virtual (avatar) dentro de NICE.

3.1.3 Requerimientos particulares de la parte colaborativa

Para que la Realidad Virtual pueda llegar a ser Realidad Virtual Colaborativa es necesario que cumpla con ciertas características, que como su nombre lo indica, permita la manipulación del entorno a más de un participante en colaboración mutua.

3.1.3.1 *Avatares*

En todo entorno virtual es necesaria una representación de cada participante, a esto se le llama Avatar.

3.1.3.2 *Interfaces para la manipulación y visualización colaborativa*

Esto se refiere a la existencia de candados, cuando un usuario tome un objeto para los demás usuarios este objeto estará bloqueado, también debe existir una actualización rápida del estado de estos objetos para poder ser visualizados en su estado actual.

3.1.3.3 *Teleconferencias de audio y video*

Este tipo de comunicaciones es una característica muy importante para proveer una experiencia colaborativa, debe existir baja "latencia" para poder ver cara a cara, en tiempo real a otro participante y entablar una charla con él.

Figura 3.4
Teleconferencia que ofrece un mandatario en un seminario.

3.1.4 Soporte flexible para varias características de datos

Existen cuatro atributos que caracterizan la Realidad Virtual Colaborativa.

- ***Calidad de Servicio (QoS).*** Para poder trabajar bien con la Realidad Virtual Colaborativa se requieren siempre los mínimos niveles de latencia y ruido, un buen ancho de banda además de incluir el uso de protocolos de transmisión confiables y no confiables, aunque muy pocos utilizan ambos, ya que la mayoría de los entornos virtuales colaborativos aún son experimentales, por lo general solo utilizan uno. La calidad de servicio entonces, se refiere a la mayor eficacia que se puede lograr en un entorno virtual colaborativo.

- ***Tamaño de Datos.*** Se dividen en tres tipos:

  - *Pequeños eventos de datos*. Son aquellos que se transmiten con prioridad y no presentan latencia.

  - *Datos medios atómicos*. Son aquellos suficientemente pequeños para poder ser almacenados en la memoria física de un cliente.

  - *Amplios segmentos de datos*. Estos son muy grandes y solo se puede accesar a ellos por segmentos.

- ***Enlistado/ desenlistado.*** Los datos que son enviados a los clientes o servidores, ya sea que estén almacenados en una base de datos o no, necesitarán ser de cualquier forma enlistados o desenlistados. Los datos enlistados, a diferencia de los desenlistados, son datos que deben llegar completos al cliente o al servidor ordenadamente. Esto implica el uso de un protocolo confiable aunque hay algunas veces donde un protocolo no confiable puede ser de mucha ayuda especialmente para conferencias de audio.

- ***Datos persistentes y transitorios.*** La persistencia de datos se refiere a que los datos serán almacenados en una base de datos para su uso posterior, aunque el usuario abandone el entorno virtual, al regresar estarán como se encontraban antes de salir. Por el contrario los datos transitorios, como el audio y video, no son almacenados en una base de datos; al ser transmitidos solo existirán esa sola ocasión.

3.1.5 Construcción topológica escalable y flexible

Existen cuatro clases importantes de topologías distribuidas utilizadas en la Realidad Virtual Colaborativa de CAVERN:

- ***Homogénea replicada.*** Es la clásica simulación militar en la Realidad Virtual, en la cual el cliente retiene una réplica completa de la base de datos.

- ***Compartido centralizado.*** Aquí todos los datos son almacenados en el servidor central, la ventaja de esto es que simplifica el control de muchos clientes.

- ***Compartido distribuido con actualizaciones peer-to-peer.*** Esta simula una extensa área estructural de memoria compartida en la que los objetos que son instanciados en un sitio son automáticamente replicados a todos los sitios remotos.

- ***Compartido distribuido usando subgrupos Cliente-Servidor.*** Esta topología distribuye las bases de datos entre múltiples servidores, los clientes son conectados al servidor apropiado según lo requiera.

3.1.6 Colaboración síncrona y asíncrona

La colaboración síncrona se refiere a que todos los participantes estén trabajando en el ambiente al mismo tiempo, sin embargo hay situaciones donde es deseable poder trabajar en forma asíncrona, por ello es importante proporcionar medios para distribuir el trabajo en grupo.

3.1.7 Persistencia en la Realidad Virtual Colaborativa

La persistencia en la Realidad Virtual Colaborativa consiste en que los Entornos Virtuales existan después de que todos los participantes han abandonado el entorno.

3.1.7.1 *Persistencia participatoria*

Es cuando el entorno virtual colaborativo existe por un tiempo mientras los participantes estén allí. Si los usuarios abandonan el entorno, la siguiente vez que entren será desde el principio.

3.1.7.2 *Estado persistente*

Es donde el estado del Entorno Virtual puede ser guardado para ser utilizado posteriormente.

3.1.7.3 *Persistencia continua*

Es donde el Entorno Virtual existe aunque todos los participantes lo hayan abandonado. Si los participantes acceden a este entorno posteriormente este habrá cambiado.

3.1.8 Interoperabilidad con sistemas heterogéneos

La variedad de computadoras y sistemas operativos en la que el entorno virtual colaborativo es aplicado, requiere conectividad entre recursos heterogéneos como bases de datos externas, supercomputadoras o cualquier sistema de Realidad Virtual.

3.1.9 Servidores de Aplicación específica

Estos no son simples almacenes de datos, poseen capacidades gráficas para representar el espacio virtual, los cuales, por lo tanto, son necesarios.

3.1.10 CAVERN (Cave Research Network)

CAVERN es una agrupación de participantes industriales e instituciones de investigación equipadas con CAVE Inmersa Desk y recursos de alto desempeño de cómputo con el propósito de soportar entornos virtuales colaborativos.

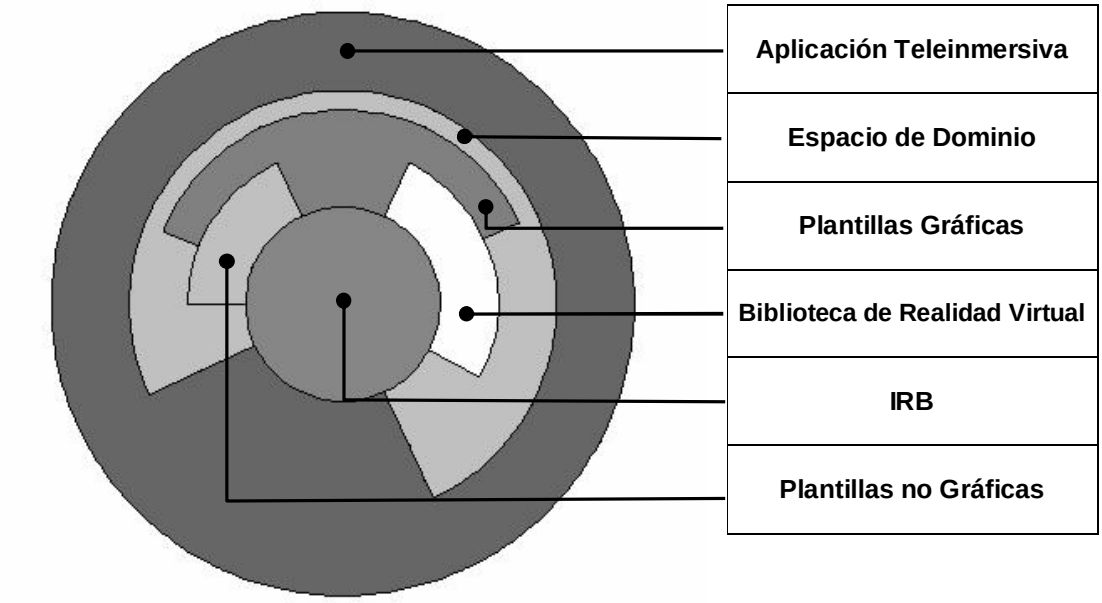


Figura 3.5
Esquema de CAVERNsoft.

3.1.11 Information Request Broker (IRB)

El Information Request Broker es el núcleo de todas las aplicaciones cliente-servidor en CAVERN. Un IRB es un depósito autónomo de persistencia de datos al que se puede acceder con una variedad de interfaces de red.

La meta es desarrollar un sistema híbrido que combine la idea del modelo de memoria compartida distribuida en CALVIN, con tecnología de base de datos y tecnología de red en tiempo real bajo una interfaz indefinida para proporcionar de forma flexible soporte a la distribución de datos en Realidad Virtual Colaborativa.

Una aplicación cliente es construida usando una interfaz IRB (IRBi). La IRBi comunica la petición a los clientes personales IRB que se comunican con otros clientes IRB remotos.

Una aplicación cliente es construida usando una interfaz IRB (IRBi) la cual, siendo llamada, fragmentará el IRB personal del cliente. Este IRB es usado para recuperar datos escondidos de otros IRB's durante la operación del cliente. Un servidor de aplicación específica es similarmente construido usando el IRBi. Por lo tanto existe una pequeña diferencia entre el cliente y el servidor (Figura 3.6).

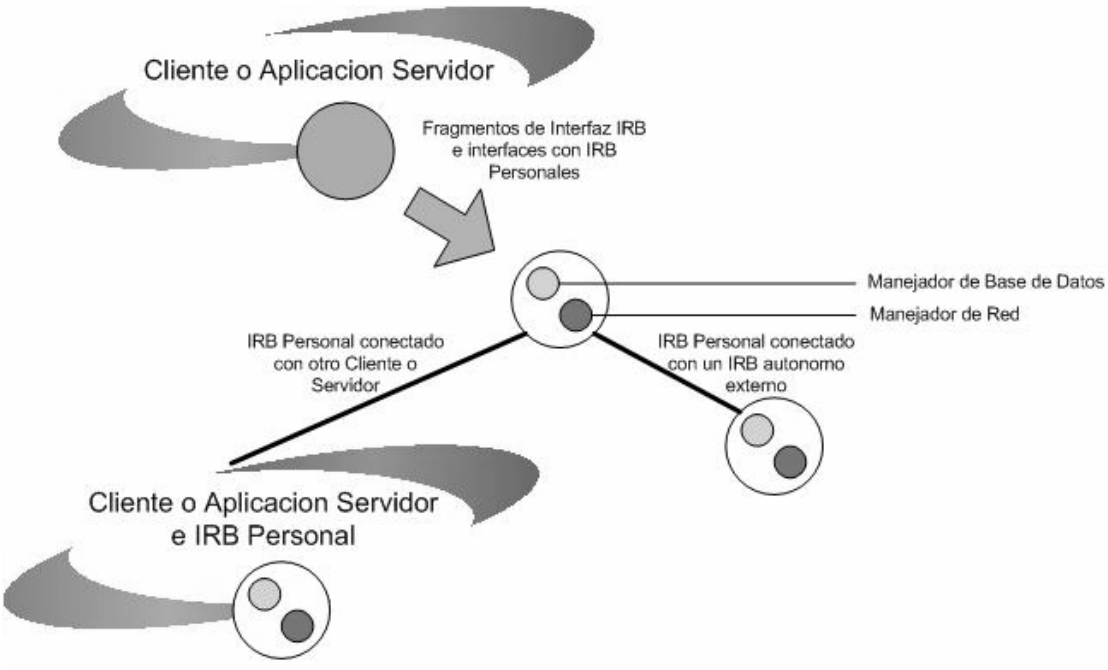


Figura 3.6
Los Clientes y Servidores utilizan la interfaz IRB para comunicarse con otros clientes, servidores o IRB autónomos externos.

Utilizando el IRBi un cliente puede arbitrariamente formar una conexión, después de haber adquirido los permisos apropiados con cualquier otro cliente o servidor para acceder a sus recursos. El IRBi comunicará la petición al IRB personal del cliente, el cual a su vez se comunicará con los clientes o servidores remotos IRB.

Es responsabilidad del IRB negociar la red y los servicios de base de datos requeridos por las aplicaciones cliente servidor. Este tipo de flexibilidad y simetría permitirá que todas las topologías de Realidad Virtual Colaborativa sean construidas rápidamente.

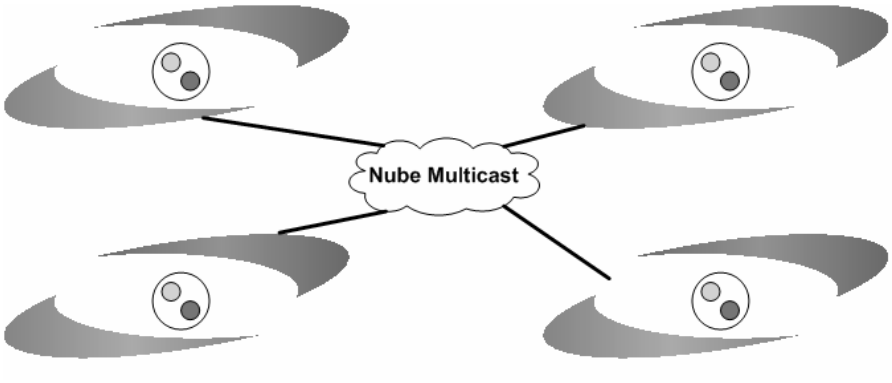


Figura 3.7
Muestra un IRB basado en clientes con todas las posibilidades de replegar sus bases de datos compartiendo actualizaciones vía "Multicast".

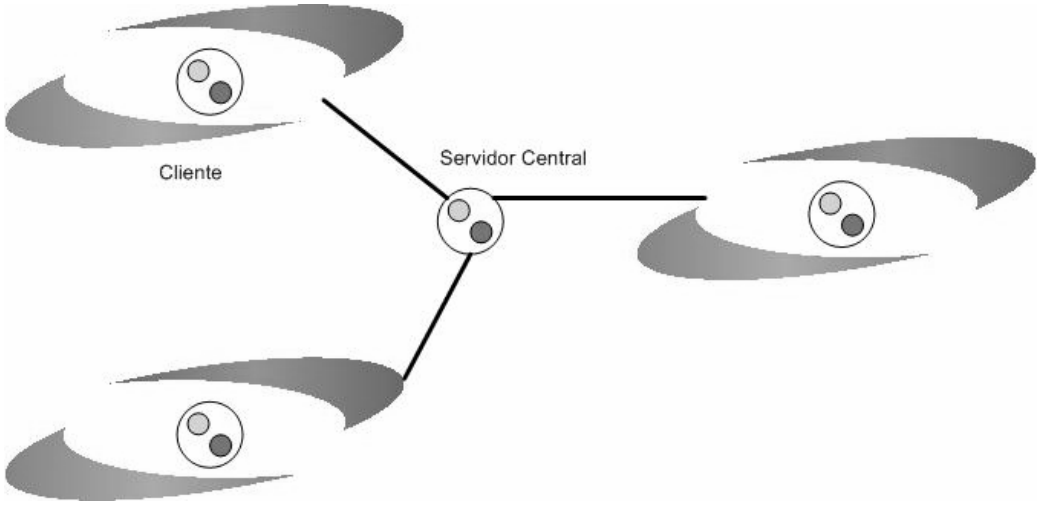


Figura 3.8
Muestra el uso de IRB's en una base de datos compartida y centralizada.

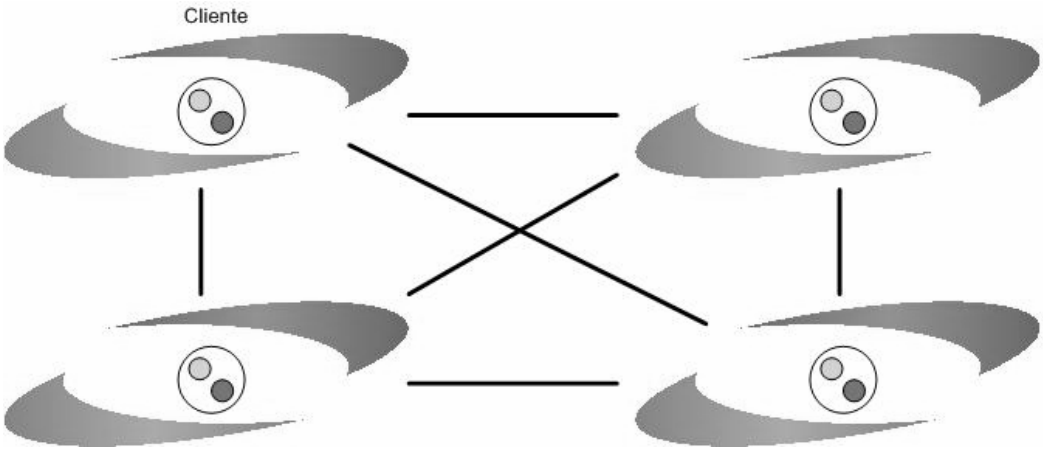


Figura 3.9
Muestra un IRB basado en clientes con una configuración de conexión completa para soportar una base de datos compartida y distribuida con actualizaciones peer-to-peer.

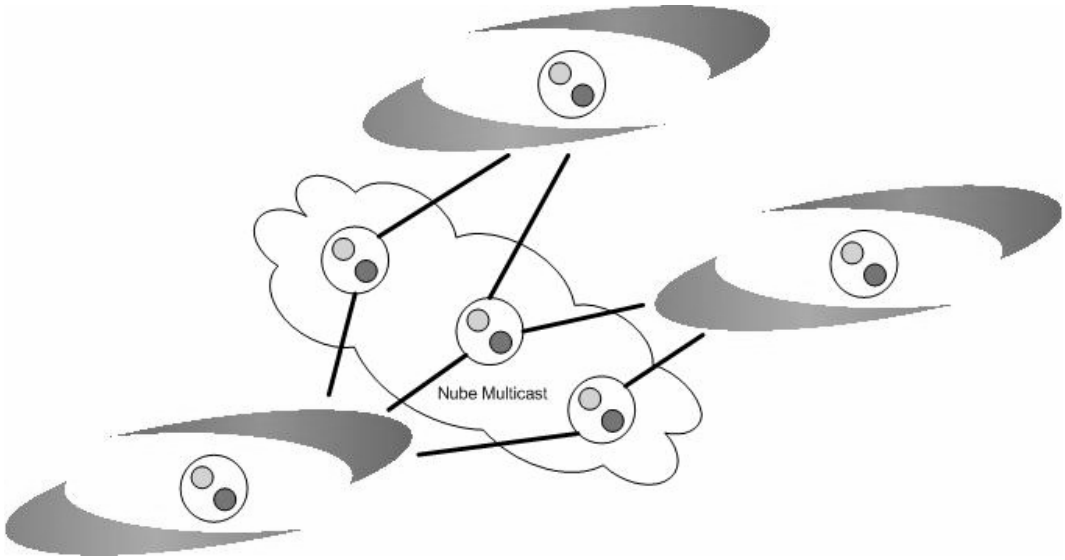


Figura 3.10
Muestra un IRB basado en clientes y servidores que están conectados en forma de base de datos compartida y distribuida cliente-servidor.

Como se muestra en la Figura 3.10, los clientes pueden conectarse arbitrariamente a cualquiera de los servidores utilizando cualquier protocolo de comunicación deseado para recuperar información. Gracias a que no hay distinción entre un cliente y un servidor, un IRB basado en un programa podría ser un cliente ejecutándose desde una supercomputadora, o un servidor interconectado con una amplia base de datos de información científica.

3.1.12 La Interfaz del Information Request Broker (IRBi)

Como se menciono anteriormente la IRBi es la interfaz del cliente y del servidor al IRB. El IRBi provee una plantilla de interfaz de red, de bases de datos y de alto nivel.

Es responsabilidad del IRB negociar los servicios de red y de bases de datos requeridos por las aplicaciones cliente/servidor. Esta forma de flexibilidad y simetría permitirá que se construyan de manera rápida todas las topologías de Realidad Virtual Colaborativas. Los clientes pueden estar arbitrariamente conectados en cualquiera de los servidores usando cualquier protocolo deseable de comunicación para recuperar información. Debido a que no se hace una distinción entre un cliente y un servidor, un programa basado en el IRB puede estar corriendo sobre una supercomputadora o sobre una interfaz de servidor con una amplia base de datos científicos.

3.1.12.1 *Canal de propiedades*

Este canal permite a los clientes especificar el servicio de red deseado para el envío de datos. Los clientes pueden especificar un protocolo confiable TCP o un protocolo no confiable UDP y multicast. Paquetes grandes son enviados sobre canales no confiables que automáticamente fragmentarán los datos desde la fuente y los reconstruirán en el destino. Si un fragmento se pierde, el paquete es rechazado. Además se especificarán los requerimientos de la Calidad del Servicio (QoS), por lo que se vuelven deseables ciertas condiciones en ancho de banda, latencia, jitter[6] y flujo de datos.

3.1.12.2 *Características de acoplamiento*

El acoplamiento permite a los clientes especificar las acciones que tomarán cuando llaves remotas o locales sean acopladas, esto incluye el poder escoger entre actualizaciones pasivas o activas y poder seleccionar el comportamiento inicial y subsecuente de sincronización.

3.1.12.3 *Propiedades de las llaves*

Las propiedades de las llaves deben ser definidas por el IRB personal del cliente o proporcionar un IRB remoto si es necesario, las llaves son además transitorias o persistentes.

[6] **Jitter**. Variación en el tiempo de llegada de paquetes, causado por congestionamientos de la red, cambio de rutas o pérdida de sincronización.

Las llaves persistentes son llaves que serán guardadas en el repositorio de datos del IRB, entonces cuando un cliente o servidor se vuelve a ejecutar, los datos seguirán siendo recuperables especificando el mismo identificador de llave. Los clientes determinan si una llave es persistente preguntando al IRB por la ejecución de un "commit"[7] sobre los datos. Además, se proporcionan funciones simples de bloqueo para permitir a los clientes bloquear llaves locales o remotas.

Las funciones de bloqueo son "no bloqueables" para prevenir en tiempo real que las aplicaciones no se queden trabadas cuando intenten adquirir bloqueos sobre llaves.

3.1.12.4 *Accionar eventos de forma asíncrona*

Muchos eventos pueden ocurrir durante la distribución de datos entre clientes y servidores como:

- ***Llegada de datos nuevos.*** Es cuando una llave recibe una pieza de datos.
- ***Conexión IRB de evento rota.*** Cuando una conexión de este tipo es perdida, los clientes continuarán su función accesando a versiones locales de datos suscritos.
- ***Eventos desviados de QoS.*** Es cuando una negociación QoS falla.

3.1.13 Registro de llaves

EL registro de llaves proporciona la facilidad de mantener un estado persistente en el mundo virtual. Los registros guardan cada variación de valor cuando ocurre un cambio, así como el estado de todas las llaves en un amplio intervalo de tiempo. El registro de las llaves es necesario para marcar los cambios graduales en el entorno virtual en un cierto plazo mientras que guardar la variación de cambios en las llaves es necesario para establecer puntos de chequeo, de tal forma que las grabaciones se hagan de forma rápida hacia delante o hacia atrás sin tener que calcular cada estado sucesivo que conduzca a la locación hacia delante o hacia atrás.

3.1.14 Interfaz de conexión directa

Además de las capacidades automáticas de red proporcionadas por la IRBi, esta debe soportar accesos directos a sockets[8] de bajo nivel como TCP, UDP o Multicast, para hacer una buena conectividad.

[7] **Commit**. Es el último paso de un exitoso término de una modificación previamente iniciada en una base de datos, como parte del manejo de una transacción en un sistema de cómputo.

[8] **Sockets**. Puntos de comunicación por los cuales los procesos pueden emitir o recibir información.

3.1.15 Facilidad suplementaria de procesamiento concurrente

Es necesario proveer un control de concurrencia básico primitivo de exclusión mutua y señalización, estos son implementados como macro definiciones sobre las bibliotecas subyacentes usadas por el IRB.

3.1.16 Teleinmersión

Fue usada por primera vez en octubre de 1996 como un taller organizado por el Laboratorio de Visualización Electrónica en la Universidad de Chicago en Illinois (UIC). El Laboratorio de Visualización Electrónica está interesado en investigar aspectos técnicos relacionados con la creación de entornos, así como los aspectos sociales y de comunicación al colaborar usando estos medios, así como la evaluación inicial de su efectividad. La meta básica es hacer este tipo de colaboración posible y además conveniente.

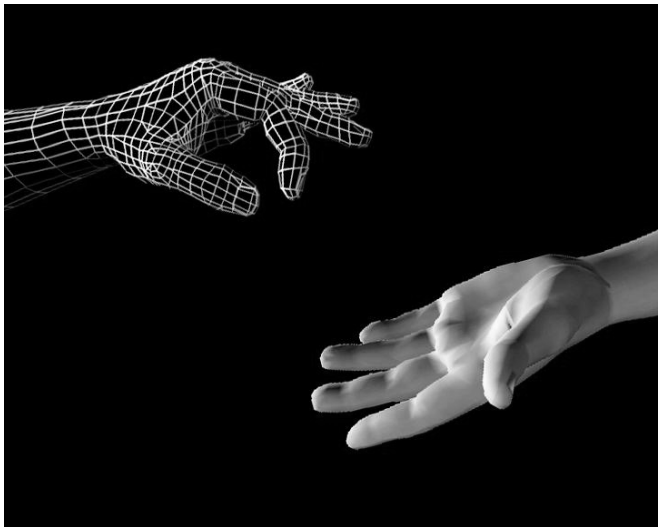


Figura 3.11

La Teleinmersión permitirá la interacción entre humanos y simulaciones generadas por la computadora.

Los requerimientos para soportar la teleinmersión son divididos en tres niveles, el nivel de usuario final, de desarrollo de aplicación y el nivel de desarrollo tecnológico del núcleo.

3.1.16.1 *Nivel de usuario final*

Le concierne el más alto nivel para el soporte de comunicación, colaboración, interacción y manipulación colaborativa, persistencia, grabación y conectividad.

- ***Comunicación en entornos teleinmersivos.*** Típicamente involucra el uso de video y audio conferencias, además incluye el uso de avatares para dar una representación del participante en el entorno.

- ***Colaboración síncrona y asíncrona.*** Los entornos persistentes asumen la habilidad de trabajar en un entorno al mismo tiempo que otros participantes (Síncrona),

además de permitir trabajar en el mismo entorno en tiempos diferentes (Asíncrona), cualquier cambio en el entorno debe persistir cuando otros participantes regresen.

- ***Capacidad de grabación.*** Denota la habilidad de grabar experiencias virtuales para continuaciones futuras. Además permite a los participantes hacer anotaciones en el entorno para que otros participantes puedan verlas en otro momento. Este es un aspecto muy importante en los entornos virtuales colaborativos.

- ***Manipulación e interacción colaborativa.*** Son las herramientas que permiten a los participantes interactuar y manipular objetos compartidos así como conjuntos de datos. Estas herramientas deben garantizar la consistencia en el entorno, especialmente cuando varios participantes tratan de modificar el mismo objeto.

- ***Conectividad con recursos internos.*** Los datos a los que acceden los usuarios están en una base de datos externa. Los datos pueden consistir de modelos 3D, colecciones de campos de datos, datos calculados, grabaciones persistentes de Entornos Virtuales, flujos de audio y video. Aunque algunas simulaciones no son iniciadas en el entorno virtual sino en una estación de trabajo, las simulaciones seguirán necesitando direccionar el resultado de los cálculos hacia una base de datos que el Entorno Virtual pueda accesar. Además, los usuarios finales necesitarán herramientas que permitan que las aplicaciones y herramientas que no sean de Realidad Virtual puedan fácilmente interactuar con el Entorno Virtual.

3.1.16.2 *Nivel de desarrollo de aplicación*

Este nivel involucra principalmente la forma de integrar las herramientas de software en aplicaciones que los usuarios necesiten.

3.1.16.3 *Nivel de desarrollo del núcleo*

Los programadores de aplicaciones no deberán preocuparse por el desarrollo del núcleo, el hardware y software tendrán la infraestructura para soportar los patrones de acceso a recursos.

3.1.17 Conclusiones del análisis de CAVERN

La forma de comunicación vía IRB y el registro de actividades hacen de CAVERN una herramienta muy poderosa y completa. Para poder utilizar CAVERN es necesario conocerla a fondo, por lo que nos llevaría mucho tiempo el poder manipularla en la práctica a un nivel óptimo para el desarrollo.

3.2 AVANGO: un marco de Realidad Virtual Distribuida

Avango provee a los programadores el concepto de grafo de escena compartido, accesible en todos los procesos que forman una aplicación distribuida. Cada proceso tiene su propia copia local del grafo de escena y el estado de información contenida en él, el cual permanece en sincronía.

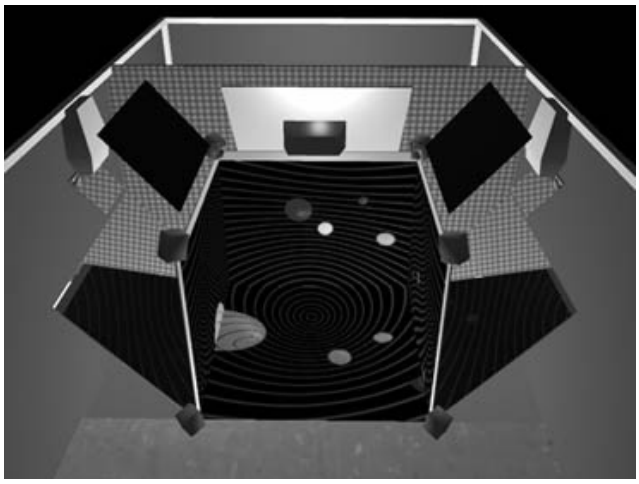
Se provee además, de un marco que combina el modelo de programación común del toolkits existente con soporte para la distribución de datos que es casi invisible al desarrollo de la aplicación.

Avango es un marco de programación para construir aplicaciones de Realidad Virtual interactivas y distribuidas, Utiliza C++ para definir dos categorías de clase de objetos, nodos y sensores. Todos los objetos de Avango son contenedores de campo que representan información del estado del objeto como colección de campos. Ellos soportan una interfaz genérica de flujos. Avango utiliza conexiones entre campos para construir un flujo de datos gráfico ortogonal[9] al grafo de escena, el cual es usado para especificar comportamientos en aplicaciones interactivas.

Avango es usado sobre SGI Performer[10] para obtener los mejores resultados.

Figura 3.12

Stage cibernético inmersivo desarrollado con Avango.



3.2.1 Campos y contenedores de campo

Avango utiliza campos como contenedores para los atributos del estado de los objetos. Los campos encapsulan tipos de datos básicos y proveen una interfaz genérica de flujos, estos son implementados como miembros de clases públicas. Los campos pueden ser de uno o dos tipos.

- ***Simples.*** Contienen valores básicos de datos
- ***Múltiples.*** Contienen un vector con un número variado de valores.

Todos los campos son implementados en C++.

[9] **Ortogonal**. Objeto que está en posición perpendicular o que forma un ángulo de 90° con respecto a otro.

[10] **SGI Performer**. Es un toolkit para desarrollo de imágenes 3D con alto desempeño de renderización, para desarrollo en tiempo real, multiproceso y aplicaciones de gráficas interactivas.

Los objetos en Avango son contenedores de campo que representan el estado de un objeto como una colección de campos a los que se les puede preguntar por el número de campos contenidos y sus referencias.

Los adaptadores de campo son usados para unir el método basado en la API[11] de Performer al campo orientado a la API de Avango. Esto asegura que el desempeño relacionado con el estado de la información es correctamente actualizado de acuerdo al cambio de valor en el campo.

La interfaz de contenedor de campo provee una base sólida para las subsecuentes implementaciones genéricas de características avanzadas como flujo de datos de cómputo, scripts[12] y distribución.

3.2.2 Conexiones de campo

Los campos compatibles pueden ser conectados de tal forma que cuando el valor del campo fuente cambie sea inmediatamente enviado dicho valor al campo destino. Usando estos conductos, un grafo de flujo de datos puede construirse siendo conceptualmente ortogonal al grafo de escena. Avango utiliza este mecanismo para especificar relaciones adicionales entre los nodos, las cuales no podrían ser expresadas en términos del grafo de escena estándar. Esto facilita la implementación de comportamiento interactivo y la importación de datos del mundo real dentro del grafo de escena. El contenedor de campos puede hacer lo que sea para alcanzar los efectos deseados incluyendo la manipulación de otros campos.

Las conexiones de campo no son distribuidas, éstas solo existen en procesos que conectan los campos involucrados.

3.2.3 Nodos

Las adaptaciones del contenedor de campo existen para todo nodo y objeto de Performer, las cuales en conjunto representan el API de grafo de escena de Performer. Por convención los nodos de Avango remplazan a los de Performer. La habilidad de mezclar nodos de Avango con nodos de Performer para construir grafos de escena puede ser convenientemente usada para definir nuevos nodos con funciones adicionales.

[11] **API** (Application Programming Interface). Es un conjunto de especificaciones de comunicación entre componentes de software.

[12] **Scripts**. Lenguajes interpretados que forman un subconjunto de los lenguajes de programación. A diferencia de los lenguajes compilados, los lenguajes script no necesitan ser preprocesados mediante un compilador, esto significa que la computadora es capaz de ejecutar las instrucciones dadas sin tener que leer y traducir todo el código.

3.2.4 Sensores

Representan la interfaz de Avango con el mundo real. Se derivan de clases de contenedores de campo pero no de cualquier nodo de Performer. Los Sensores encapsulan el código necesario para accesar a varios tipos de dispositivos de entrada, los datos generados por estos dispositivos son mapeados a los campos del sensor. Siempre que un dispositivo genere nuevos datos, los campos del sensor correspondiente son actualizados. Las conexiones de campos desde los campos del sensor a los campos del nodo en el grafo de escena son usadas para incorporar datos provenientes de dispositivos de aplicación.

3.2.5 Distribución

El principal objetivo del diseño de Avango con respecto a la distribución fue hacer un desarrollo de aplicaciones distribuidas tan simple como el desarrollo de aplicaciones *stand-alone* (en una sola máquina).

3.2.6 Modelo distribuido de memoria compartida

La memoria compartida denota un área local de memoria a la que se puede acceder simultáneamente por más de un proceso en una sola máquina. Cada proceso incluye un segmento de memoria compartida que verá todos los cambios que otro proceso incluya al segmento de aplicación del proceso. Es aquí donde se extiende el concepto de Memoria Distribuida Compartida (DSM). Localmente los procesos negocian con sus segmentos de memoria y podrían no necesitar saber cual de ellos es distribuido. Los objetos alojados en un segmento DSM podrían visualizar todos los procesos de los participantes.

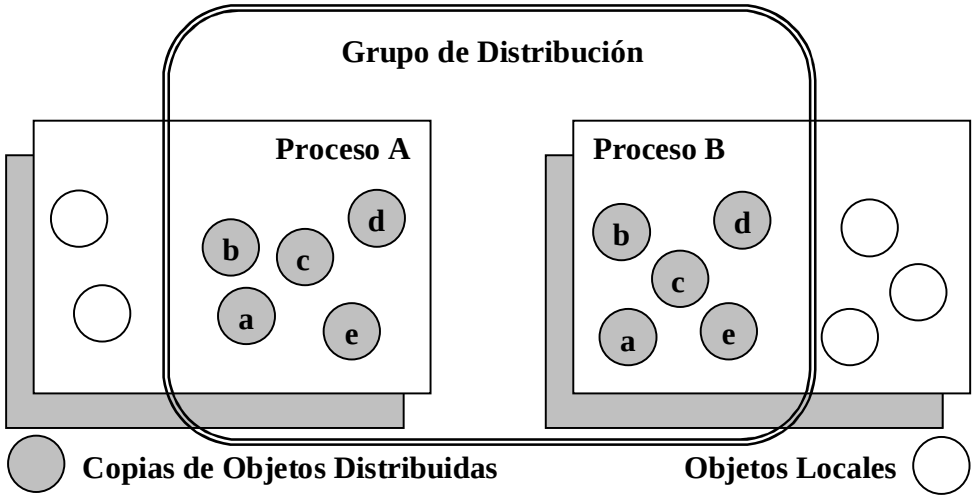


Figura 3.13
Los objetos de Avango pueden ser creados localmente al proceso o pueden ser migrados a un grupo de distribución y replegarse a su vez para que todos los procesos estén unidos.

3.2.7 Grafos de escena

Los objetos de Avango son contenedores de campos que encapsulan el estado del objeto en un conjunto de campos, a esto se le llama grafo de escena. La interfaz de flujos de campo y las clases del contenedor de campo permiten una implementación muy elegante de la semántica de objetos distribuidos.

La creación de objetos distribuidos en Avango es un proceso de dos partes. Primero, es creado un objeto local, entonces un segundo paso es migrar este objeto al grupo de distribución deseado. La migración envuelve el anuncio del nuevo objeto dentro del grupo de distribución y la diseminación del estado de los objetos actuales a todos los miembros del grupo. La interfaz que fluye al contenedor de campo es usada para colocar en una serie el estado del objeto dentro del buffer de datos de la red, el cual entonces es enviado a todos los miembros del grupo. Los miembros del grupo revertirán este proceso y crearán una copia del objeto de la información del estado en la serie. El nuevo objeto creado distribuido, existirá como una copia local en cada uno de los procesos participantes.

En Avango se puede acceder al estado de los objetos a través de sus valores de campo. Siempre que un valor de campo sobre un objeto distribuido sea localmente cambiado, el nuevo valor fluye hacia el buffer de la red y envía a todos los miembros del grupo de distribución para mantener las copias distribuidas del objeto sincronizado.

3.2.8 Comunicación en grupo confiable

Avango utiliza el sistema de ensamble de la Universidad de Cornell, que es un modulo grupal de procesos que provee una comunicación multicast confiable entre procesos distribuidos. Parte del ensamble es gracias al toolkit "Maestro"[13], que provee la infraestructura necesaria para la diseminación consistente del estado de aplicación de Avango.

3.2.9 Ordenamiento total de mensajes

"Maestro" permite el envío de mensajes totalmente ordenados, de tal forma que cada miembro del grupo recibirá todos los mensajes enviados al grupo exactamente en el mismo orden. Este envío introduce algo de latencia pero es una forma muy consistente.

3.2.10 Servicio de membresía al grupo

"Maestro" maneja los procesos que comunican a un grupo, como una lista de miembros de grupo llamada "vista". Cuando un nuevo miembro se une al grupo o un antiguo miembro se va, la vista es actualizada. Cuando una vista cambia es anunciada a todos los miembros,

[13] **Maestro toolkit**. Grupo de herramientas que utilizan una combinación de intercepción e integración de aproximaciones, lo que proporciona un esquema orientado a objetos para procesos basados en comunicación de grupo.

entonces cada miembro tiene una lista actualizada de todos los participantes del grupo.

3.2.11 Transferencia de estado atómica (por unidades completas)

Inmediatamente después de unirse al grupo, nuevos miembros no conocen la historia del grupo, no poseen una copia del estado compartido de la aplicación, entonces podemos iniciar un estado de transferencia atómica de uno de los antiguos miembros a uno de los nuevos. Durante este proceso las demás comunicaciones en el grupo son suspendidas. Después del proceso, el nuevo miembro tiene exactamente el mismo estado de información que el antiguo, entonces podemos continuar con las operaciones normales. De esta manera se pueden unir nuevos miembros al grupo sin destruir la consistencia, la desventaja que se tiene es que si se lleva mucho tiempo este proceso, será reflejado e informado a los otros usuarios congelando las aplicaciones.

3.2.12 Conclusiones del análisis de AVANGO

Avango introduce el concepto del grafo de escena, que es una manera muy práctica de almacenar y tratar la información del estado de los objetos. Una buena característica es que los mensajes que se generan son mandados a cada uno de los objetos en forma estrictamente ordenada, lo que permite la confiabilidad e integridad de la información. La contra que observo es que al utilizar estos procesos de ordenamiento y copia idéntica de datos a cada uno de los objetos del grupo, todas las demás actividades dentro del sistema son detenidas, por lo que si el proceso tarda mucho introduce cierto nivel de latencia que se refleja en el desempeño del programa.

3.3 OCTOPUS: un API de plataforma cruzada para permitir aplicaciones de Realidad Virtual Distribuida

Octopus es una aplicación multiplataforma con un API de objetos para construir entornos virtuales dirigidos a resolver algunos problemas. Es una biblioteca de software distribuida, es Realidad Virtual distribuida desarrollada por el Centro de Aplicaciones de Realidad Virtual de la Universidad del Estado de Iowa.

3.3.1 Diseño de Octopus

Octopus está separado en tres partes: la biblioteca núcleo, la biblioteca general de avatar y las bibliotecas específicas de API’s gráficas de avatares.

La biblioteca núcleo maneja la red y la manipulación de objetos, la biblioteca general de avatar provee la interfaz básica usada en la implementación de avatares y la de API´s gráficas provee las implementaciones para que los avatares puedan ser escritos o modelados usando varias API’s gráficas.

Octopus está escrito en C++ utilizando ACE (Comunicación de Entorno Adaptativa). La idea principal de Octopus es usar la neutralidad de un sistema operativo, para darle

portabilidad entre plataformas. El uso de ACE fue motivado por el diseño Orientado a Objetos (O.O.) que se acopló muy bien a Octopus. Un diseño Orientado a Objetos provee beneficios a los programadores de Octopus, ya que hace más fácil las tareas de colaboración de aplicaciones existentes. Un requerimiento natural hace a Octopus capaz de ser agregado a aplicaciones existentes por lo que es totalmente independiente de cualquier biblioteca como VR Juggler o CAVE.

3.3.2 Comunicación en red

Para alcanzar el mejor desempeño de Octopus en la red, utiliza un protocolo UDP para transmitir paquetes entre nodos. De hecho, el envío de paquetes entre nodos puede ser "punto a punto", donde cada nodo sabe de cada otro nodo y envía actualizaciones individuales. El envío puede ser por un simple canal IP Multicast para reducir el número de envíos. Este método muestra dinamismo permitiendo agregar o sacar usuarios.

Todos los nodos son iguales en la red, no hay un servidor central, un nodo es escogido para actuar como mediador para el control de objetos compartidos (se hace por medio de peticiones).

3.3.3 Objetos compartidos

El Procedimiento de Llamada Remota (RPC) maneja el ordenamiento y desordenamiento de datos entre diferentes tipos de nodos a través de su representación externa. El procedimiento de llamada remota, sin embargo, no ajusta bien en el diseño de marcos orientados a objetos en Octopus.

Octopus provee un marco especial para objetos compartidos entre nodos, estos objetos pueden ser agregados al entorno estática o dinámicamente y múltiples usuarios pueden interactuar con un único objeto simultáneamente. Las interacciones con los objetos son enviadas a los otros usuarios en el entorno, entonces ellos ven las actualizaciones cuando suceden eventos.

Un simple nodo en el grupo de usuarios debe ser elegido para actuar como un mediador de petición de pertenencia de objetos. Este nodo no necesita ser el primero en unirse a la sesión y trabaja en curso para permitir la recuperación si es que el nodo mediador falla.

3.3.4 Avatares generales

Octopus provee avatares y un marco para agregar nuevos avatares definidos por el usuario. Un avatar en Octopus es visto como un objeto compartido que es propiamente la representación del usuario. Los programadores pueden construir sus propios avatares e incorporarlos a clases jerárquicas definidas en la biblioteca general de Octopus. La implementación actual de los avatares está definida en la biblioteca de gráficos específica.

3.3.5 Bibliotecas específicas de API's graficas de avatares

En esta parte es donde están definidos los avatares, es aquí donde se utilizan bibliotecas como OpenGL[14] y Performer. Está separado para permitir un mecanismo flexible para que los usuarios definan sus avatares y además correspondan a las API's gráficas usadas para la aplicación.

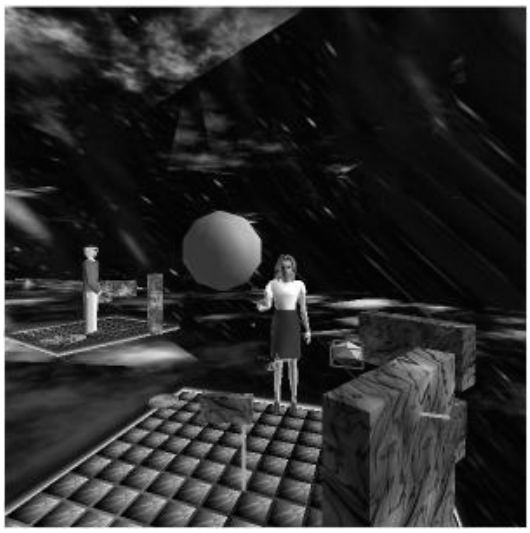


Figura 3.14

Modelo de avatar, parte de la Biblioteca de Octopus.

3.3.6 Conclusiones del análisis de Octopus

Octopus nos provee una gran ventaja que es la tomar las bases del sistema local para poder ser ejecutado, de esta manera puede correr sobre múltiples plataformas de manera independiente. La característica de tomar nodos mediadores y no tomar un nodo central hace a Octopus dinámico, ya que pueden interactuar entre varios nodos permitiendo la recuperación de la información si es que algún nodo mediador falla. Octopus algunas múltiples características importantes que nos permiten una buena interacción entre varios nodos, el enfoque de nuestra aplicación a desarrollar no es tan amplia por lo que la utilización de múltiples nodos no será necesaria.

3.4 BAMBOO: soportando protocolos dinámicos para Entornos Virtuales

Bamboo es el resultado del desarrollo de un toolkit adecuado para la investigación y desarrollo de entornos virtuales. Este objetivo se logra entendiendo aspectos claves y proporcionando un soporte directo en ellos, aplicando lecciones aprendidas de previos esfuerzos. Estas soluciones son proporcionadas en forma de mecanismos prácticos implementados usando diseños orientados a objetos y técnicas de programación genérica. Bamboo es un toolkit para entornos virtuales enfocado en la habilidad del sistema para configurarse por sí mismo dinámicamente sin la interacción explícita del usuario

[14] **OpenGL**. Entorno para desarrollo de aplicaciones gráficas portables e interactivas en 2D y 3D.

permitiendo al sistema tomar nueva funcionalidad después de la ejecución.

3.4.1 Extensibilidad Dinámica

Todos los esfuerzos de programación, incluyendo a los entornos virtuales distribuidos, se benefician de los buenos diseños de ingeniería de software y de las metodologías de desarrollo, con estas técnicas se facilitan las cualidades como baja dependencia de código, alta cohesión y la reutilización del código. La habilidad de retrasar mucho las decisiones de programación en el ciclo de ingeniería de diseño, provee además excelente flexibilidad. Estos beneficios son especialmente notables cuando cada decisión puede ser tomada después de la ejecución.

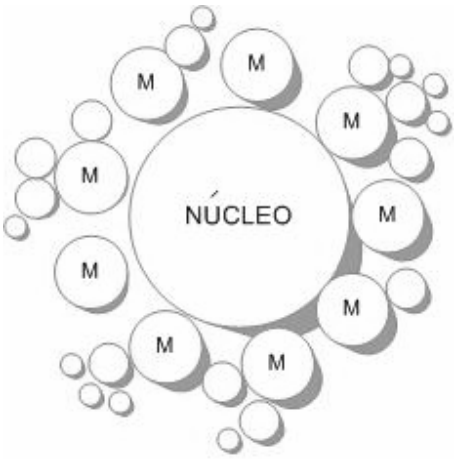


Figura 3.15

Esquema de Bamboo, el núcleo y sus módulos.

La configuración dinámica no había sido previamente aplicada a Entornos Virtuales. La extensibilidad dinámica es la decisión de diseño más importante de Bamboo, el cual está conformado por varios módulos; el núcleo (Figura 3.15) tiene solamente la lógica necesaria para cargar y descargar módulos y proporciona el marco inicial para el acoplamiento de *plug-in's*[15]. De esta forma, no se presupone qué características son necesitadas por el núcleo, sino que son determinadas en tiempo de ejecución por la aplicación que se cargue.

Por lo tanto, si una aplicación en particular no necesita un driver[16] para un dispositivo específico, el sistema no cargará ese módulo ahorrando tiempo de procesamiento y

[15] **Plug-In's**. Programas de cómputo que interactúan con otros programas para aportarles una utilidad o función específica. Se utilizan como forma de expansión a los programas, de tal forma que se pueda añadir nueva funcionalidad sin afectar el funcionamiento existente ni complicar el desarrollo del programa principal.

[16] **Driver**, Programa encargado de actuar como interfaz entre un sistema software y los dispositivos físicos, de esta forma todos los componentes se entienden y trabajan conjuntamente.

memoria, que de otra forma sería consumido por el mecanismo.

Cada módulo cuando está siendo cargado únicamente necesita verificar que sus dependencias inmediatas estén ya en la memoria, cargándolas si aún no están. Si el módulo no tiene una firma confiable, el sistema trabajará para darle una.

Por ejemplo, en la figura siguiente asumimos que el módulo M4 es cargado en el sistema. Primero el sistema debe verificar que el módulo M3 esté totalmente cargado en la memoria. Asumiendo que M3 no ha sido cargado, el sistema debe verificar que ambos, M1 y M2, han sido cargados en la memoria. Suponiendo que tampoco hayan sido cargados M1 y M2, el sistema continúa y hace lo que tiene que hacer para no tener alguna dependencia. Una vez que M1 y M2 han sido cargados, M3 podrá ser cargado y finalmente M4.

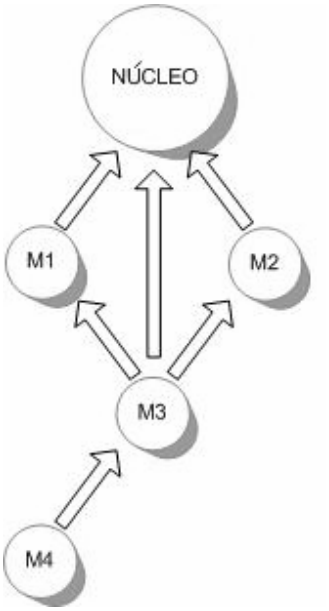


Figura 3.16
Esta figura ilustra cómo un módulo solo necesita especificar su dependencia inmediata.

Para permitir la extensibilidad dinámica se requiere simplemente, colocar nuevo código en el mismo espacio de dirección como un simple ejecutable. Bamboo provee un marco que permite que nuevo código pueda adjuntarse explícitamente por sí mismo.

La clase “Callback” implementa uno de los mecanismos más fundamentales en todo Bamboo. La abstracción del “callback” pasa una referencia para invocar objetos. La “callback” es la columna vertebral de Bamboo.

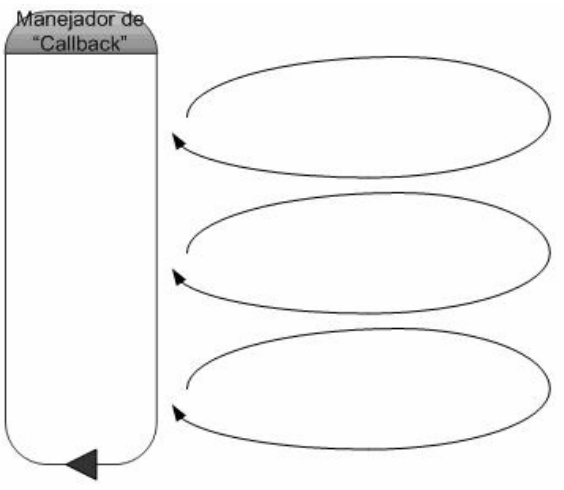


Figura 3.17
Manejador de Callback.

La clase del manejador de "callback" colabora con la clase "callback", esto provee una interfaz que puede añadir o quitar "callback's", tiene la propiedad de ejecución secuencial, esto es que se pueden ejecutar series de "callback's" en orden. El manejador de "callback" puede ser ejecutado explícitamente cada vez que una subrutina es ejecutada.

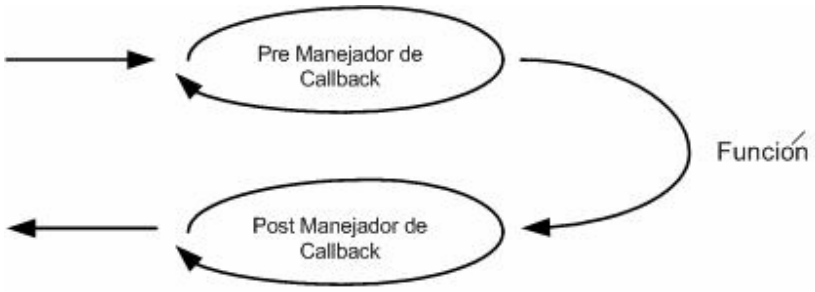


Figura 3.18
Todos los Callback's son recursivos.

Cada callback es recursivo de modo que involucra los manejadores de callback's, de esta forma el ejecutable puede ser como un árbol de callback's.

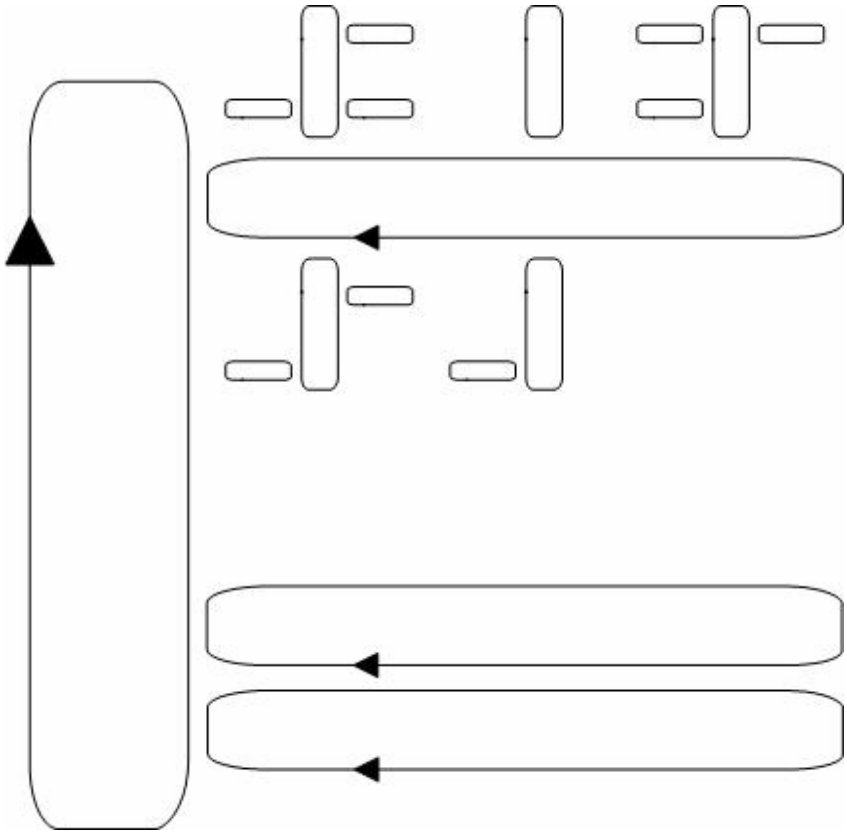


Figura 3.19
Árbol de Callback's.

3.4.2 Protocolos dinámicos

El uso inicial por módulos permite a varios grupos compartir sus propios resultados con los demás entornos. Esos resultados típicamente representan la geometría, textura, sonido y conductas en el entorno virtual. Todos los usuarios pueden estar de acuerdo en usar un solo protocolo, pero eso no ayudará mucho al escenario existente. En lugar de eso, por otra parte, imaginemos que cada usuario puede definir su propio protocolo, esto es que la única forma en que se representa un usuario en el entorno, es si el sistema carga primero el módulo del usuario que incluye sus características y además inicia y decodifica el tráfico de la red con el usuario.

La mayor interrogante aquí, es como o cuando hacer que el módulo se instale a sí mismo. Hay dos soluciones básicas a esta cuestión:

- ***Aproximación proactiva al módulo de instalación.*** Este escenario describe un módulo cliente soportado por un servidor. El modulo dinámicamente cargado debe escuchar los paquetes que son enviados a él para mantener un estado, es interesante saber que algunos módulos no necesitan servicio de soporte ya que mantienen su propio estado internamente.

- ***Aproximación reactiva al módulo de instalación.*** Requiere que el usuario esté activamente informado de la existencia de otros usuarios. El usuario no sabe acerca de los demás hasta el tiempo de ejecución. Entonces cada sistema debe activamente abrir y escuchar en un puerto sobre el que se espera recibir la respuesta del cliente. Una solución simple, implementada por DIS, es tener cada objeto periódicamente monitoreado con un mensaje de *heartbeat*[17] que identifique la existencia del resto del mundo. No recibir un *heartbeat* de un usuario en un tiempo definido indicará que el usuario ya no es representado y puede ser removido sin problemas del sistema.

3.4.3 Aproximaciones reactivas

La aproximación inicial es proporcionada ya que Bamboo tiene definidos archivos propios de si mismo, por lo tanto el Web Browser[18] sabe como lanzar a Bamboo y pasar el archivo como un comando, aquí Bamboo probará una firma válida. Este módulo puede tener cualquier número de dependencias pero él mismo define las especificaciones de la aplicación del entorno en el que el usuario se encuentra, entonces el módulo define el entorno completo. Existen algunas ventajas de retrasar las decisiones, las cuales pertenecen a las relaciones del entorno de un Manejador de Áreas de Interés (AOIM). Un AOIM es usado para reducir la complejidad de una simulación. Inmediatamente que es cargado todo el entorno, el propósito del AOIM termina, si es que éste estaba siendo utilizado.

El siguiente paso de la exacta naturaleza del sistema es determinado por la carga del módulo, existen muchas secuencias que son viables.

3.4.4 Tres niveles para el manejo de la red

La experiencia con la implementación de los protocolos dinámicos ha dirigido a la observación de que existen al menos 3 niveles para el manejo de la red, estos son: global, por entorno y por objeto, los cuales describen la abstracción de la capa del entorno en la que existe el tráfico sobre la ejecución de ciertas operaciones.

- ***Global.*** Esta capa es compartida en el World Wide[19] y consiste en un nivel de tráfico muy bajo utilizado para sincronizar entornos.

- ***Por entorno.*** Esta capa es compartida por entorno y consiste en un bajo nivel de tráfico para sincronizar objetos, permite el descubrimiento de objetos y manejar el entorno por sí mismo.

[17] **Heartbeat.** Latido de corazón, nombre asignado a una señal para identificar existencia.

[18] **Web Browser**. Aplicación software que permite recuperar y visualizar documentos de hipertexto, gráficos, secuencias de videos, sonido, animaciones, vínculos y programas diversos desde servidores Web de todo el mundo a través de Internet.

[19] **World Wide** (de World Wide Web). Sistema de hipertexto que funciona sobre Internet, utilizando como visualizador de información una aplicación llamada *Web Browser*.

- ***Por objeto.*** Esta capa es usada sobre bases por objetos y consiste en tráfico para sincronizar objetos. Pueden existir múltiples subcapas.

3.4.5 Tres protocolos dinámicos con Bamboo

El siguiente paso para cargar el entorno depende de que el módulo haya sido cargado. El módulo debe conectar la red de tráfico del entorno. Este proceso se debe considerar como parte del proceso de implementación de la red. De aquí nacen tres importantes intentos de implementaciones:

- ***Basado en DIS aproximado.*** Esta prueba inserta una nueva y previa enumeración por entidad dentro de un ejercicio que está corriendo en DIS. En términos de los 3 niveles del manejador de la red, DIS no implementa el concepto de capa global.

- ***Basado en HLA (High Level Architecture)[20] aproximado.*** Similar a DIS, incorpora dinámicamente un nuevo objeto dentro de una simulación HLA que esta corriendo. En términos de los 3 niveles de manejador de red, no implementa el concepto de la capa global. HLA utiliza un canal multicast con un único puerto por conjunto de participantes.

- ***Basado en CN (Communications Network)[21] aproximado.*** Ésta prueba la habilidad de insertar un objeto completamente nuevo dentro del entorno CN que esta corriendo. En términos de los 3 niveles del manejador de red, este utiliza un único direccionamiento multicast para la capa global, por entorno y por capas de objetos.

3.4.6 Conclusiones del análisis de Bamboo

Bamboo permite manejar los datos y la información una gran velocidad, ya que cada módulo contiene únicamente la información básica de formación, si este necesita algún driver o programa extra lo ira solicitando durante su creación, si no se necesita algo no se cargará. Cada módulo se preocupará únicamente de la implementación de sus módulos dependientes inmediatos. La utilización de los “callbacks” para agregar funcionalidad al sistema permite reducir latencia cuando un cambio o modificación de código es requerido. Una característica que se me hizo buena y a la vez lenta es que cada usuario puede elegir su propio protocolo de comunicación, estoes bueno en cuanto a la parte de multiplataformas pero en el momento de hacer las adaptaciones necesarias estas requieren de tiempo, que se traduce como latencia.

[20] **HLA** (High Level Architecture). Protocolo de Internet que permite soportar simulaciones interactivas y distribuidas.

[21] **CN** (Communications Network). Protocolo de comunicación en la red.

3.5 MASSIVE: Un sistema de Realidad Virtual Distribuida que incorpora intercambio espacial

MASSIVE es un modelo de arquitectura y sistemas para interacción espacial en entornos virtuales. Es un sistema experimental distribuido de Realidad Virtual que soporta actividad colaborativa. Los factores espaciales como la posición, orientación y la dirección fija son muy importantes en un manejo de conversaciones. El énfasis particular de MASSIVE es sobre la amplia escala y entornos multiusuario, entornos que podrán eventualmente soportar cientos o miles de usuarios simultáneamente. Las facilidades que provee MASSIVE están basadas en el modelo de interacción espacial, MASSIVE podría proveer formas ricas de interacción que pueden influir en las conductas sobre el mundo real para que convierta los entornos en algo utilizable y controlable en mundos virtuales con alta carga de usuarios.

Las características que distinguen a MASSIVE de otros es que incluye una completa implementación del módulo de interacción espacial, el uso estricto de intercambio espacial con el arreglo individual de conexiones peer-to-peer para todas las interacciones entre objetos en el mundo virtual y facilita la interacción en el mundo real.

3.5.1 El modelo espacial de interacción

El modelo puede ser dividido en dos componentes mayores con diferentes objetos, el primero que facilita la escalabilidad y el segundo que controla la interacción espacial.

3.5.1.1 *Escalabilidad*

La escalabilidad de MASSIVE está basada el en concepto de “Aura”. Cada objeto en el mundo virtual tiene un “Aura” para cada medio en el que interactúa. Esta “Aura” define la magnitud de interacción que es posible con los otros objetos. La interacción entre dos objetos llega a ser permitida solo cuando sus “Auras” están juntas e indican la posibilidad de interacción, esto se refiere a una colisión de “Auras”.

El uso de las “Auras” facilita el escalamiento de muchos usuarios por medio de la limitación del número de interacciones de los objetos que deben ser manejados. Para mundos virtuales suficientemente amplios, el número de interacciones debe ser manejado dependiendo únicamente de la magnitud del “Aura” de los objetos y de la densidad de participantes en el espacio.

3.5.1.2 *Control de la interacción*

El segundo componente de la interacción espacial negocia con el control de la interacción o comunicación entre dos objetos, una vez que ellos llegan a estar al tanto uno del otro a través de una colisión de “Auras”.

El concepto principal involucrado en controlar esta interacción es llamado “Awareness”. Un “Awareness” de objeto hacia otro objeto cuantifica la importancia subjetiva o la relevancia del objeto en un medio dado. Para soportar los trabajos de comunicación social, los niveles de “Awareness” son negociados entre objetos, en general ambos objetos en una

interacción desearán afectar sus niveles de “Awareness” mutuo, el objeto observado deseará enfocar su atención en áreas particulares mientras el objeto observado necesitará su visibilidad o accesibilidad. El concepto de “Focus” describe el nivel de atención del observador, mientras que el concepto de “Nimbus” describe el objeto que se manifiesta al observador. El “Awareness” del observador de observación es una combinación de “Focus” y ”Nimbus” del observador. “Aura”, ”Focus” y ”Nimbus” pueden ser funciones multivaluadas que pueden tener formas arbitrarias, por lo tanto, estos tres y el “Awareness” pueden ser manipuladas en tres formas:

- ***Forma implícita.*** A través de acciones espaciales como el movimiento.
- ***Forma explicita.*** Escogiendo de entre diferentes formas y tamaños.
- ***Objetos adaptados.*** Que transforman combinaciones de “Aura”, ”Focus” y ”Nimbus”.

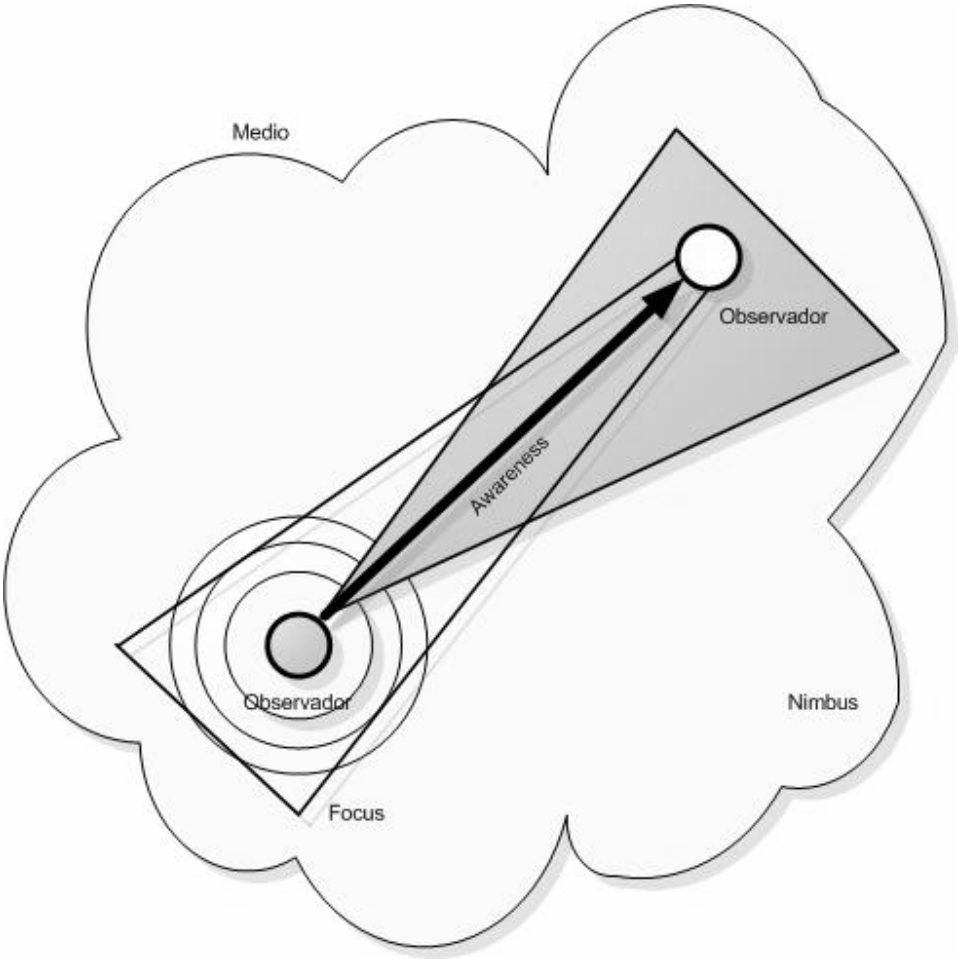


Figura 3.20
Una sola dirección, Focus. Nimbus y Awareness entre dos observadores.

3.5.2 Funcionalidad de MASSIVE

El sistema ha sido dividido en dos requerimientos principales:

- ***Escala.*** Soportará tantos usuarios como sea posible
- ***Heterogéneo.*** Soportará interacciones entre usuarios que tienen diferentes quipos y capacidades.

MASSIVE soporta mundos virtuales múltiples conectados mediante portales, cada mundo puede ser habilitado por muchos usuarios, los cuales pueden relacionarse con buenas interacciones gráficas, audio e interfaces de texto. Otra característica principal de MASSIVE es que sus interfaces pueden ser combinadas arbitrariamente de acuerdo a las capacidades de la terminal de usuario. MASSIVE relaciona interfaces compatibles entre objetos siempre que ellos se reúnan en el espacio. El efecto RED consiste en que los usuarios con equipos radicalmente diferentes puedan interactuar. En la implementación actual, los usuarios pueden explícitamente manipular su "Awareness" por la elección entre tres opciones para "Focus" y "Nimbus", normal, estrecho y amplio, además dos adaptadores de objeto provistos:

- ***Un podium*** que extiende el "Aura" y "Nimbus" del usuario.
- ***Una tabla de frecuencias*** que reemplaza las "Auras" normales de los usuarios, "Focus" y "Nimbus" con nuevos conjuntos.

MASSIVE funciona actualmente sobre SUN[22] y SGI[23].

3.5.3 Arquitectura de MASSIVE

3.5.3.1 *Arquitectura de comunicación*

En contraste con los sistemas de Realidad Virtual Distribuida basados en una base de datos compartida, el modelo de distribución usado en MASSIVE es de procesos independientes computacionales de comunicación sobre un tipo de conexión "peer-to-peer". Cada proceso computacional puede tener cualquier número de interfaces por las que interactúa con el resto del sistema. Cada interfaz es caracterizada por una combinación de procesos de llamadas remotas (RPC's).

3.5.3.2 *Aura e intercambio espacial*

La interacción entre objetos en espacios virtuales solo es posible cuando dos posiciones se reúnen, primero debe estar establecido que el objeto en cuestión posee algunas interfaces

[22] **SUN** (Sun Microsystems). Empresa productora de software, semiconductores y equipos informáticos de Estados Unidos

[23] **SGI** (Silicon Graphics Incorporated). Empresa dedicada al desarrollo de cómputo de alto desempeño, almacenamiento y tecnología de visualización.

compatibles y segundo, los objetos deben estar lo suficientemente cercanos para ser determinados por sus "Auras". Ambas condiciones son reflejadas en el concepto de intercambio espacial, el cual combina la técnica de Realidad Virtual de detección de colisiones con el concepto de sistemas distribuidos. El término "intercambio" engloba lo que son servicios de nombres basados en atributos *trader*.

3.5.3.3 *Implementación de "Focus" y "Nimbus"*

Una vez conectado el cálculo de los niveles de "Awareness" mutuo, la utilización de "Focus" y "Nimbus" es responsabilidad del par de objetos. "Focus" y "Nimbus" pueden ser descritos por funciones matemáticas que lidian sus atributos de los objetos que se están comunicando en los valores internos de "Focus" y "Nimbus" que son combinados en niveles mutuos de "Awareness". "Focus" y "Nimbus" son típicamente atributos espaciales y pueden adicionalmente tomar en cuenta otros atributos no espaciales.

3.5.3.4 *Interacción de programas de cliente*

MASSIVE soporta actualmente tres interfaces de cliente, una por audio, otra por graficas y por texto, que pueden ser combinadas arbitrariamente por el usuario. Cada protocolo de cliente es compatible con el protocolo "peer-to-peer" y pueden definirse atributos adicionales RPC's y flujos requeridos. En el caso donde los usuarios corran múltiples clientes simultáneamente, únicamente uno de ellos actuará como cliente maestro controlando la navegación y los otros actuarán como clientes esclavos, marcando la posición desde el maestro y actualizando su posición y actualizando sus vistas.

3.5.3.5 *Portales*

Los portales son puertas de enlace (*gateways*) entre mundos diferentes o entre diferentes locaciones dentro del mismo mundo. Los portales son implementados como interfaces en un medio de portal especial que incluye una destinación del atributo de locación.

3.5.4 Comunicación en MASSIVE

En MASSIVE las comunicaciones se hacen por medio de conexiones entre pares de diferentes tipos de interfaces. La conexión provee un contexto en el que los mensajes pueden ser interpretados. Las interacciones en MASSIVE son una mezcla de cliente-servidor y peer-to-peer. Una conexión punto a punto provee un contexto único en el que notificaciones asíncronas y respuestas pueden ser interpretadas.

3.5.5 Intercambio espacial

El acceso al modelo construido en torno a una interfaz de intercambio, es un servicio que provee la declaración de interfaces para el intercambio y asistencia a usuarios.

Las cuatro características principales del intercambio espacial son: las peticiones exhaustivas, las peticiones y ofertas de larga duración, las peticiones y ofertas sujetas a

alteraciones incrementales; y un simple dominio en el que las peticiones y las ofertas pueden ser organizadas y manejadas para facilitar la escalabilidad del sistema. Las aplicaciones que comparten todas esas características son incluidas en muchas situaciones cooperativas, otras comparten menos características pero seguirán siendo benéficas para extender o especializarse en el modelo de intercambio.

3.5.6 Conclusiones del análisis de MASSIVE

Con la introducción del modelo espacial MASSIVE logra crear la forma de interacción, no solo entre usuarios, sino también entre mundos virtuales, lo que lo hace diferente a las demás plataformas de software de Realidad Virtual. Una de las características más importantes de MASSIVE, y como tiene que serlo en las distintas plataformas, es que provee un ambiente heterogéneo por lo que no importa que tipo de equipo posean los participantes, el sistema creará una similitud entre ellos para la interacción.

MASSIVE es una plataforma muy completa y a mi punto de vista muy interesante y funcional, pero es demasiado amplia y compleja para la aplicación que deseamos desarrollar.

3.6 VR Juggler: una plataforma virtual para desarrollo y aplicaciones de Realidad Virtual.

Hoy en día se espera que los desarrolladores de aplicaciones sean expertos en el dominio de un problema, además deben ser especialistas en el desarrollo de sistemas de software sofisticado que tiene como características realizar múltiples procesos, el paso de mensajes, memoria compartida, manejo de memoria dinámica, y una variedad de procesos y técnicas de sincronización. Además, deben comprender el manejo de los “drivers” para controlar dispositivos de entrada y salida o técnicas para generar múltiples vistas estereoscópicas.

Para permitir el crecimiento del uso de la tecnología de Realidad Virtual, una aplicación común necesita ser especificada. Esta interfaz de desarrollo debe esconder los detalles específicos de las tecnologías subyacentes, proporcionando una plataforma virtual al diseñador de la aplicación. Una plataforma virtual permite a los investigadores concentrar sus esfuerzos en el desarrollo del contenido de la aplicación, esto es, centrarse en los detalles relacionados a la visualización y manipulación de los datos de un problema, y no a los detalles de la programación compleja. Además, gracias a que la tecnología de Realidad Virtual continúa evolucionando, una plataforma virtual facilita la adaptabilidad y mejoramiento de las aplicaciones existentes sobre nuevos sistemas sin afectar el *core*[24] de la aplicación especifica y debe proporcionar un entorno funcional a los desarrolladores para que puedan crear y ejecutar una aplicación independientemente de los recursos disponibles.

[24] **Core**. Se refiere a la base del sistema.

3.6.1 Una plataforma virtual para Realidad Virtual

Una plataforma virtual provee un entorno de desarrollo y ejecución que es independiente de la configuración de la arquitectura de hardware del sistema operativo y las configuraciones de hardware disponibles para Realidad Virtual. Esto provee un entorno operativo unificado en el que los desarrolladores pueden escribir y probar aplicaciones usando los recursos disponibles y garantizando la portabilidad de la aplicación a otros recursos.

3.6.1.1 *Independencia del sistema operativo*

La independencia de las tecnologías subyacentes es uno de los principales requerimientos para soportar el desarrollo de aplicaciones en plataformas cruzadas. Es también crítico liberar a los desarrolladores de aplicaciones de la responsabilidad de equilibrar la interpretación de sus aplicaciones para especificar las plataformas que están siendo usadas.

Para permitir la independencia del sistema, VR Juggler aísla las dependencias del sistema específico detrás de una simple interfaz de plataforma virtual implementada como kernel[25]. En VR Juggler la interfaz del objeto del kernel define la plataforma virtual para el desarrollo y agrupamiento de los detalles del sistema específico de Realidad Virtual.

3.6.1.2 *Dispositivos de abstracción*

Una plataforma virtual provee una abstracción para los dispositivos de Realidad Virtual basados en su funcionalidad. Los dispositivos de Realidad Virtual pueden ser abstractos dentro de muchas clases básicas de entrada y salida como de posición, de orientación, digital, análoga, guantes y gestos. VR Juggler define la interfaz de la clase base de cada clase de dispositivos para hacer que todos los dispositivos de una clase dada se vean igual a la desarrollada. Por ejemplo, en cualquier aplicación, un driver de un dispositivo de una simulación posicional tiene la misma interfaz que un *tracker*[26] magnético. Como VR Juggler únicamente interactúa con dispositivos a través de su interfaz, los desarrolladores de aplicaciones no están atados a las especificaciones de hardware.

3.6.1.3 *Entorno operativo*

La plataforma virtual proporciona un entorno de operación simple para las aplicaciones de Realidad Virtual. Este entorno operativo permite ejecuciones múltiples de aplicaciones y componentes.

VR Juggler permite entornos que tienen alta especialización en aplicaciones que se ejecuten simultáneamente, considerando a cada aplicación una instancia de aplicación objeto. Esto es similar al conflicto del escritorio: en una sesión típica de escritorio, los usuarios tienen muchos programas individuales como Web-browsers, lectores de e-mail, lanzadores de programas, sesiones de Chat y editores de texto activos simultáneamente, entonces cambian entre estos para completar una tarea. VR Juggler permite este mismo tipo de productividad multiprogramas con aplicaciones de Realidad Virtual. En una actividad de trabajo inmersiva, podría haber una herramienta que permita lanzar nuevas aplicaciones, otra

[25] **Kernel**. Es el núcleo de cualquier sistema operativo.
[26] **Tracker**. Dispositivo de control de posición, similar a un mouse.

podría permitir al usuario recibir comunicaciones de personas en otro entorno virtual, mientras otra toma notas verbales acerca de los datos de las aplicaciones que están corriendo. Hay un amplio rango de aplicaciones útiles que pueden ser de gran ayuda para los usuarios en un entorno virtual.

3.6.1.4 *API´s gráficas en VR Juggler*

Otro componente clave de cualquier aplicación de Realidad Virtual son las API's gráficas usadas para crear elementos visuales. Una plataforma virtual debe permitir a los desarrolladores utilizar cualquier API gráfica que ellos elijan.

VR Juggler soporta múltiples API's gráficas gracias al agrupamiento de todas las conductas de las API's específicas en manejadores de dibujo. Ya que el kernel representa solamente la parte que esconde los de talles del sistema en una plataforma virtual, se puede agregar una interfaz adicional que es específica para cada API gráfica soportada. La interfaz del manejador de dibujo puede ser considerada una pequeña plataforma virtual que representa a la aplicación con una abstracción específica de la API.

3.6.1.5 *Descripción de la plataforma virtual de VR Juggler*

La interfaz de aplicación de la plataforma virtual (Figura 3.21) esta formada por la interfaz del kernel que provee la abstracción de hardware para la plataforma virtual y el manejador de dibujo que proporciona a su vez la abstracción de las API's gráficas.

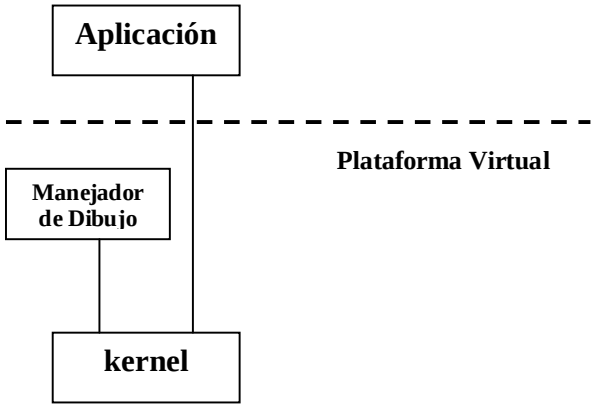


Figura 3.21
Interfaz de aplicación de la plataforma virtual.

El kernel controla todos los componentes de VR Juggler excepto las API's gráficas, su interfaz proporciona la plataforma virtual para los detalles específicos del entorno de hardware. La interfaz del kernel es la única forma en que la aplicación puede acceder al hardware, esto es posible extendiendo o cambiando los detalles de implementación de cualquier componente de VR Juggler, siempre y cuando la interfaz del kernel permanezca igual.

Esta combinación, el kernel y el manejador de dibujo, brindan a las aplicaciones de VR Juggler independencia del sistema. Una vez que una aplicación de VR Juggler es programada puede ser ejecutada en cualquier otro sistema que soporte VR Juggler.

3.6.2 Desarrollo de aplicaciones

Para todo desarrollo de una aplicación se tiene que definir una metodología, VR Juggler mantiene una propia muy especial y funcional.

3.6.2.1 *Aplicación objeto*

En VR Juggler, las aplicaciones de los usuarios son objetos (Figura 3.22). VR Juggler utiliza la aplicación objeto para crear el entorno de Realidad Virtual con el que el usuario interactúa.

La aplicación objeto implementa las interfaces necesarias por la plataforma virtual para crear el entorno virtual.

El kernel mantiene control sobre el entorno y llama a los métodos definidos en la interfaz de la aplicación. Cuando el kernel llama a estos métodos, temporalmente cede el control a la aplicación objeto, entonces la aplicación puede actuar.

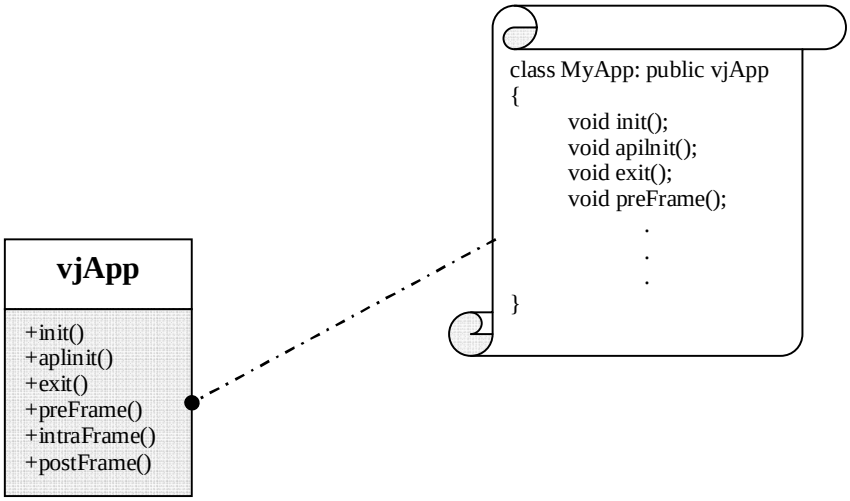


Figura 3.22
Aplicación Objeto

No hay una función “main( )”. Las aplicaciones de VR Juggler son objetos, los desarrolladores no tienen que escribir una función principal. En lugar de eso, los desarrolladores crean una aplicación objeto que implementa un conjunto de interfaces predefinidas.

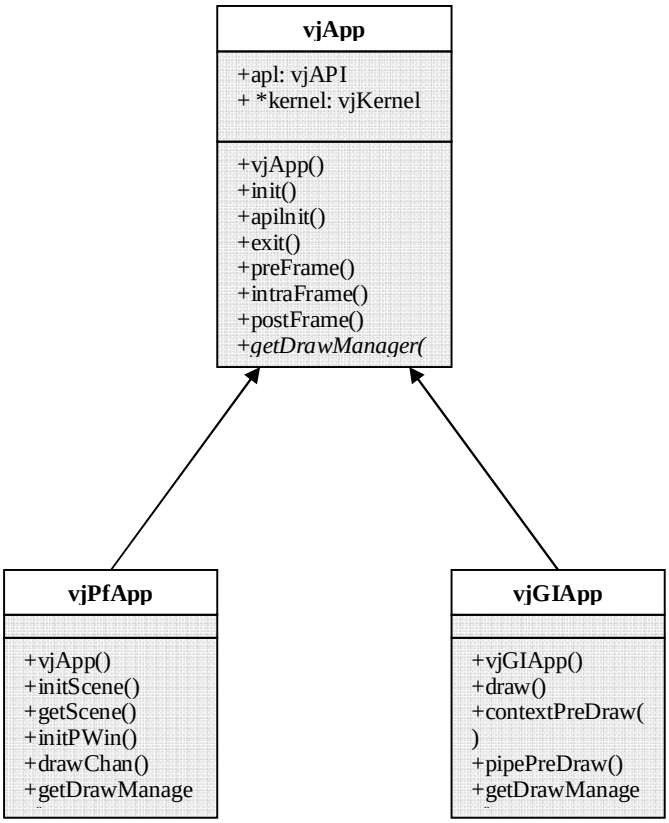


Figura 3.23
Jerarquía de clases de la aplicación.

El kernel de VR Juggler y el manejador de dibujo, se comunican con una aplicación a través de interfaces predefinidas que deben implementar todas las aplicaciones (Figura 3.23).

Cuando el kernel y el manejador de dibujo necesitan que la aplicación procese o retorne información, llaman a la función miembro de la interfaz. VR Juggler incluye clases base que proporcionan interfaces necesarias para que el kernel y el manejador de dibujo interactúen con la aplicación. Una aplicación hereda y extiende estas clases base para crear las interfaces requeridas. El kernel llama a cada una de las funciones miembro basadas en un estricto *frame*[27] de ejecución. Durante el *frame* de ejecución, el kernel llama a los métodos de la aplicación y ejecuta actualizaciones internas (Figura 3.24).

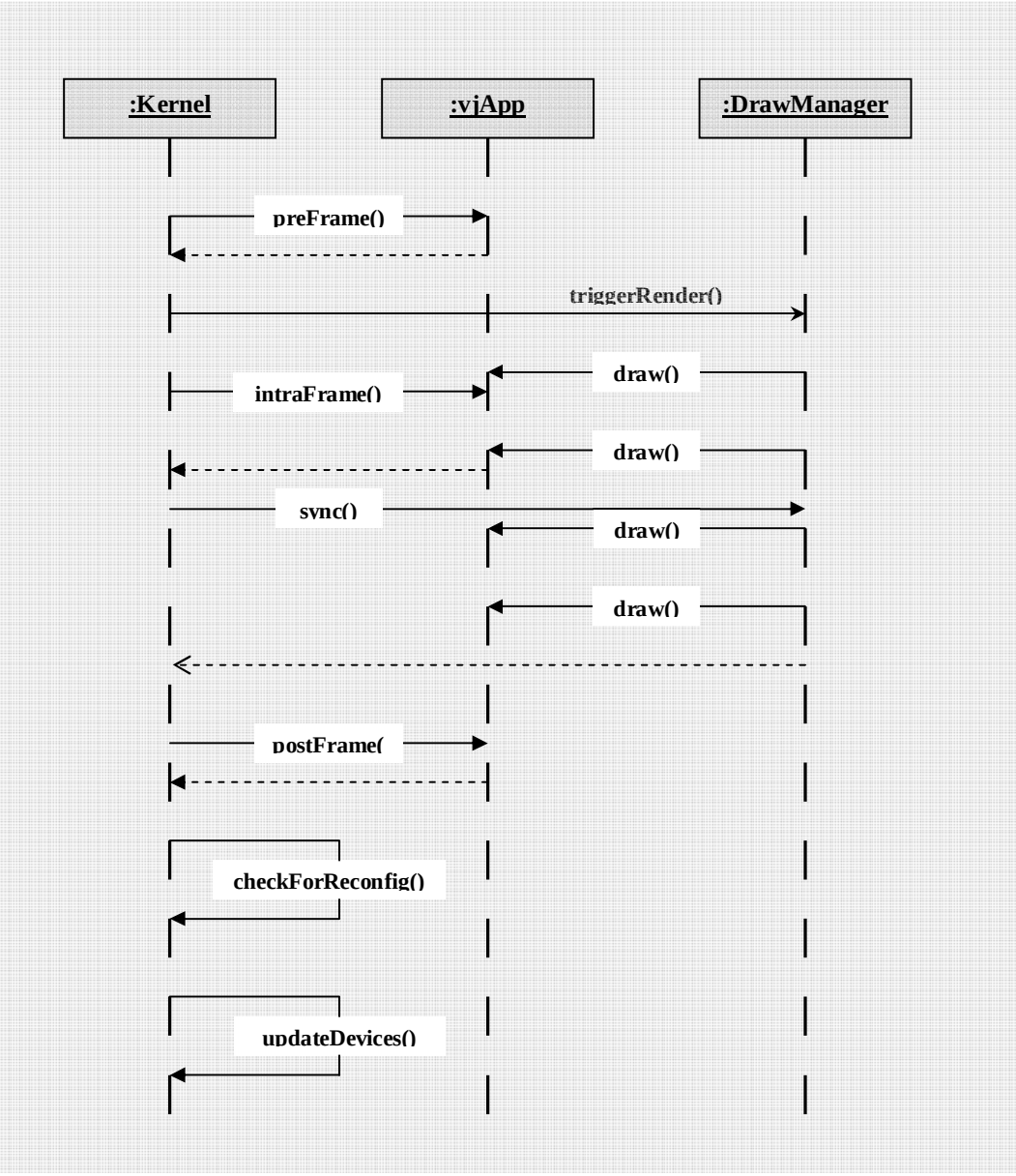


Figura 3.24
Frame del kernel.

[27] **Frame**. Ventana de un determinado sistema.

Ya que el kernel tiene completo control sobre el *frame*, puede hacer cambios predefinidos como *safe times*[28] cuando la aplicación no está ejecutando ningún proceso (*chekForReconfig( )*). Durante este *safe time*, el kernel puede cambiar la configuración de la plataforma virtual siempre y cuando la interfaz de la aplicación se vea que permanece igual.

El *frame* de ejecución además sirve como una ventana de trabajo para la aplicación.

3.6.2.2 *Beneficio de las aplicaciones objeto*

Lo más común para el desarrollo de aplicaciones de Realidad Virtual es tener definida la aplicación en la función principal y poder llamar las bibliotecas cuando la aplicación las necesite. Las bibliotecas en este modelo únicamente ejecutan el código cuando es requerido por la aplicación puesto que la aplicación tiene el control principal de ejecución. Este modelo hace al desarrollador de la aplicación responsable de coordinar la ejecución de los diferentes componentes de los sistemas de Realidad Virtual y más aún, responsable de crear aplicaciones complejas y eficientes.

VR Juggler siendo una plataforma virtual, no utiliza este modelo porque necesita mantener el control sobre los diferentes componentes para brindar la flexibilidad necesaria y hacer cambios a la Plataforma Virtual sobre el tiempo de ejecución de la aplicación.

Gracias a que el kernel controla la ejecución, siempre sabe el estado actual de las aplicaciones, por lo tanto, puede manejar con seguridad reconfiguraciones en el entorno virtual sobre la ejecución. Bajo este modelo, casi cada parámetro que se esté ejecutando en el entorno, puede ser modificado o reconfigurado sobre el tiempo en que se ejecuta la aplicación. Es también posible cambiar entre aplicaciones (*switchear*), reconfigurar dispositivos e iniciar nuevos, así como enviar información de las reconfiguraciones a la aplicación objeto.

Las aplicaciones objeto conducen a una arquitectura robusta como resultado de la baja duplicación y la buena definición de las dependencias entre objetos. La interfaz de la aplicación define el único camino de comunicación entre la aplicación y la plataforma virtual la cual restringe las interacciones con las interfaces del kernel, el manejador de dibujo y las aplicaciones. Esto reduce la duplicación permitiendo que los cambios en el sistema sean locales. Los cambios en un objeto no afectarán a otros a menos que el cambio involucre la interfaz de uno de los objetos en cuestión, lo que nos lleva a un código más robusto y extenso.

Ya que la aplicación es simplemente un objeto, es posible cargar y descargar dinámicamente aplicaciones sobre el tiempo de ejecución de la aplicación. Cuando la plataforma virtual es levantada, espera hasta que una aplicación le es asignada. Cuando la aplicación es dada por el kernel de VR Juggler sobre el tiempo de ejecución, este ejecuta algunos pasos de inicialización, y entonces activa la aplicación. Puesto que las aplicaciones utilizan diferentes interfaces para comunicarse con la plataforma virtual, los cambios en la implementación de la Plataforma Virtual no afectarán a la aplicación. Esto lo hace simple: hace cambios significativos en la implementación de la plataforma virtual sin afectar

[28] **Safe Times**. Momentos en los que la información es guardada.

ninguna aplicación que esté actualmente siendo ejecutada sobre la plataforma. Estos cambios podrían incluir la corrección de *bugs*[29], un desempeño equilibrado, soporte a nuevos dispositivos, o cualquier número de diferentes cambios.

Esta habilidad de modificar la conducta del sistema sobre el tiempo de ejecución es una de las mayores ventajas que tiene VR Juggler.

3.6.3 Diseño de VR Juggler

3.6.3.1 Microkernel

La Plataforma Virtual de VR Juggler esta basada sobre una arquitectura patrón de microkernel. El microkernel controla todo el sistema sobre el tiempo de ejecución y maneja toda la comunicación dentro del sistema. La arquitectura de microkernel de VR Juggler tiene un objeto del *core* del kernel que implementa los servicios centrales necesarios para el desarrollo de aplicaciones de Realidad Virtual.

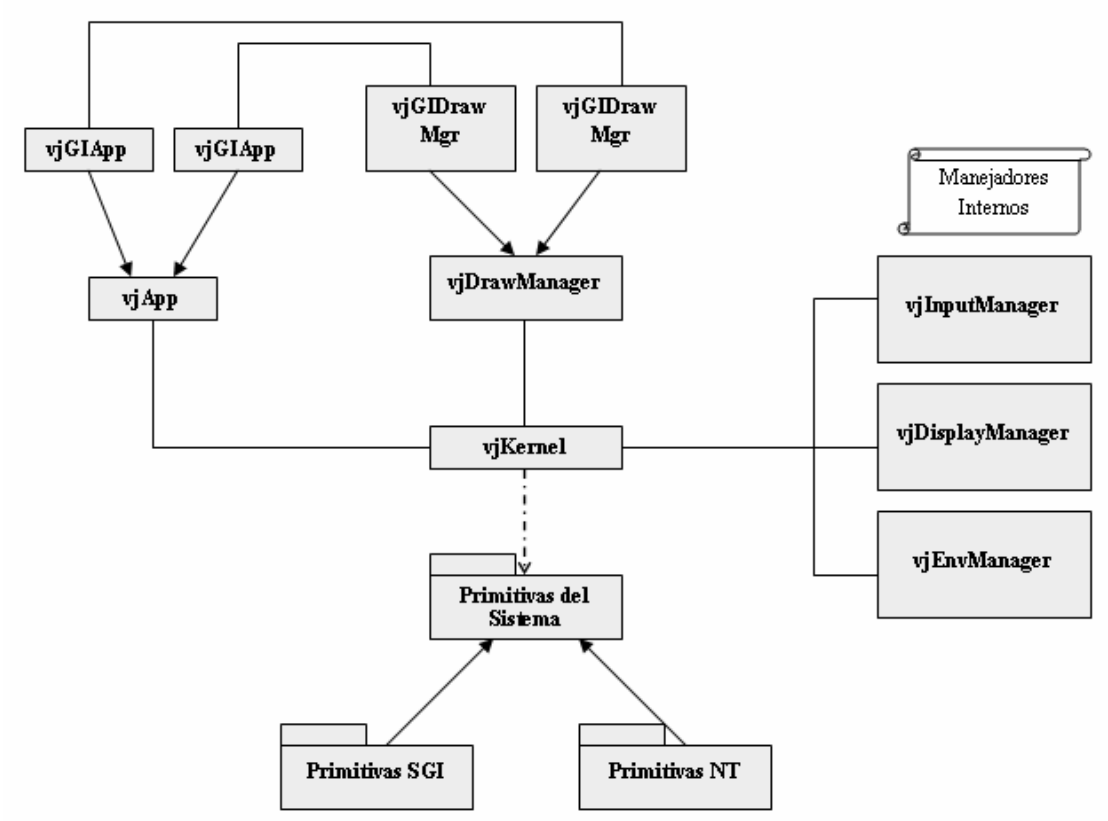


Figura 3.25
Arquitectura de Microkernel.

[29] **Bugs**. Nombre que se da a los errores que pueden existir en el Software, es el resultado de fallas de programación que se introducen en el proceso de creación de un programa.

Los manejadores internos implementan funcionalidad al *core* que no es fácilmente manejable en el objeto del kernel. Si un servicio indebidamente incrementa su tamaño o complejidad, el kernel utiliza un manejador interno para proveerse de lo que necesite para soportar ese servicio. Hay manejadores internos que controlan dispositivos de entrada, opciones de pantalla, información de configuración y comunicación con aplicaciones externas.

Los manejadores externos proporcionan una interfaz con la plataforma virtual que especifica el tipo de aplicación. Las aplicaciones cliente se comunican con el sistema VR Juggler a través de las interfaces de los manejadores externos y del kernel. Actualmente los únicos manejadores externos son los manejadores de dibujo de API's gráficas específicas.

3.6.3.1.1 *Kernel como mediador*

Muchos de los manejadores son objetos activos que se mantienen sincronizados por el kernel. El kernel mantiene el control ya que todos los manejadores requieren al kernel para dar señales durante los distintos niveles de sus procesos. Dentro del *frame* de ejecución, el kernel controla el tiempo de todos los otros objetos activos en el sistema. Los manejadores y las aplicaciones adquieren tiempo de procesamiento cuando el kernel se los proporciona, por la llamada al método de la clase, o por dar la señal de que el objeto activo puede continuar con su proceso.

El kernel en VR Juggler actúa como mediador, uniendo a todos los demás manejadores que interactúan en el sistema. No existen las dependencias directas entre los manejadores. El kernel puede cambiar la forma en que se ejecutan los *frames* del sistema sin cambiar la forma en que los manejadores se comportan o se relacionan con los demás.

3.6.3.1.2 *Portabilidad del kernel*

El kernel de VR Juggler esta definido en la punta de un grupo de primitivas de bajo nivel que permiten fácilmente el desempeño equilibrado sobre el hardware y su portabilidad. El manejador de procesos de control de primitivas y la sincronización, dependen del hardware. Ya que esas clases primitivas son la mayoría de las diferencias en la implementación de hardware específico, pueden fácilmente llevarse a otras arquitecturas de VR Juggler. Mientras es transportado, cada primitiva es extendida a bajo nivel y optimizada para alcanzar un alto desempeño sobre cada sistema.

3.6.3.2 *Información de configuración*

Toda la información de la configuración está contenida dentro de unas pequeñas unidades llamadas *chunks*, los cuales están divididos en diferentes propiedades.

Cada propiedad tiene un tipo de uno o más valores. Un *chunk* contiene toda la información de configuración para una parte en particular de la aplicación o el sistema de Realidad Virtual.

En la figura 3.26 se define la información de la configuración para una ventana.

Chunk: Ventana		
Nombre	Cadena	
Tamaño	int	int
Origen	int	int

Figura 3.26
Chunk de ventana que contiene tres propiedades: nombre, tamaño y origen.

VR Juggler utiliza *chunks* para colocar opciones de configuración para los componentes del sistema. Existen *chunks* para especificar configuraciones para todos los tipos de información necesaria para instalar un entorno de Realidad Virtual: *chunks* de pantalla, *chunks* de *trackers*, *chunks* de HMD, y más. La información de configuración es editada con un GUI[30] basado en Java llamado VjControl. Este permite a los usuarios crear y editar archivos de configuración, cambiar la configuración sobre el tiempo de ejecución, iniciar y detener dispositivos y ver la interpretación de datos.

3.6.3.3 *Manejadores internos*

Los manejadores internos se encargan de interpretar datos obtenidos de dispositivos externos o del propio sistema para realizar ciertas acciones que se reflejarán en la ejecución del sistema de Realidad Virtual.

3.6.3.3.1 *Manejador de entrada*

El manejador de entrada controla todas las conductas de los dispositivos de entrada para el kernel.

Los dispositivos de entrada están divididos en distintas categorías, incluyendo dispositivos de posición (*trackers*), dispositivos digitales (mouse), dispositivos análogos (palancas), y dispositivos de guante (*Data Gloves*).

[30] **GUI** (Graphical User Interface). Interface gráfica que permite al usuario utilizar sus programas a través de menús e iconos representativos. Existen varios tipos de GUI's, los cuales están claramente definidos en cada Sistema Operativo, y se caracterizan por utilizar apuntadores tales como Mouse, Trackballs, etc.

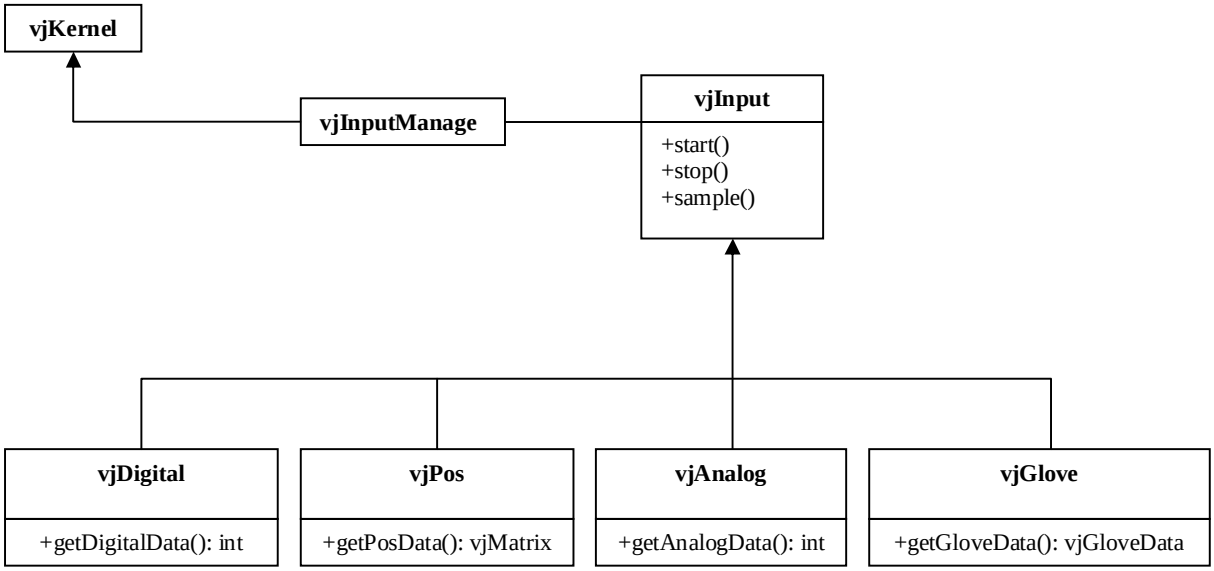


Figura 3.27
Jerarquía de clases de los dispositivos de entrada

VR Juggler define una clase jerárquica para cada uno de los tipos de dispositivos de entrada que especifica los métodos que deben ser implementados para iniciar, detener y actualizar el dispositivo (Figura 3.27).

Existe además una clase base definida para cada distinta categoría que especifica la interfaz genérica que cualquier dispositivo de cualquier categoría debe soportar. Para agregar soporte a nuevos dispositivos, una nueva clase es creada para la especificación del dispositivo. La nueva clase se deriva de las clases bases de las categorías de entrada que el nuevo dispositivo puede regresar. (Figura 3.28). Esta clase tiene que definir funciones miembro que implementen la interfaz del dispositivo de entrada base así como definir los métodos para regresar el tipo de entrada de cada clase padre.

La aplicación utiliza dispositivos *proxies*[31] como interfaz con todos los demás dispositivos. Antes de que una aplicación obtenga los datos de algún dispositivo, primero debe conseguir un proxy hacia el dispositivo. El manejador de entrada busca el dispositivo requerido y regresa un proxy al dispositivo físico.

Los dispositivos proxies permiten a la aplicación estar dispareja de los dispositivos en uso.

VR Juggler utiliza un almacenamiento de dispositivos para permitir al sistema mantener todos los drivers de los dispositivos separados de la biblioteca principal y cargar estos drivers dinámicamente.

[31] **Proxies**. Dispositivos de red que permiten el traslado de datos de una red a otra.

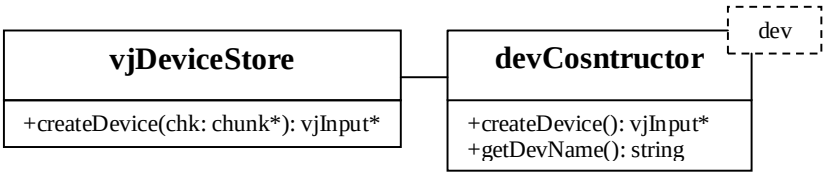


Figura 3.28
Almacenamiento de Dispositivos.

El almacenamiento de dispositivos es un objeto que mantiene el rastro de los dispositivos constructores y la asociación de estos con sus respectivos drivers que están registrados en el sistema.

3.6.3.3.2 *Manejador de entorno*

El manejador de entorno retiene la información acerca del estado del sistema y permite la comunicación del estado a programas externos. El control de la interfaz sobre tiempo de ejecución de VR Juggler, llamado VjControl, se comunica con el manejador de entorno vía conexión de red.

El manejador de imagen encapsula toda la información acerca de la configuración de las ventanas mostradas (Figura 3.29). Esto incluye la información como la medida, ubicación, gráficas, y los parámetros de vista que están siendo usados.

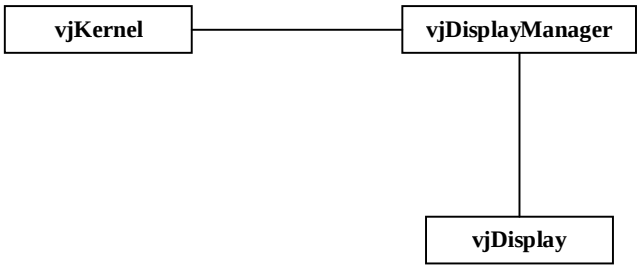


Figura 3.29
El manejador de Imagen.

El manejador de imagen es además responsable del desempeño de todos los cálculos visuales de las ventanas y es utilizado para configurar el manejador de dibujo.

3.6.3.4 *Manejadores externos*

Por otra parte los manejadores externos se encargan de interpretar las salidas de datos del sistema de tal forma que podamos identificar el resultado del proceso.

3.6.3.4.1 *Manejador de dibujo*

El manejador de dibujo ejecuta las aplicaciones del cliente que necesitan acceder a la API específica. Esto define una interfaz de la clase base de la aplicación que es modificada por una API gráfica específica. Modificar los manejadores de dibujo para las API's especificas permite un máximo desempeño de la aplicación ya que la aplicación puede hacer uso de cualquier API que tenga características avanzadas.

Los manejadores de dibujo controlan todos los detalles específicos de cada API, independientemente de la aplicación. Estos pueden manejar los detalles de configuración de las opciones de las API's, parámetros de instalación de vista de cada *frame*, y el *rendering* de las vistas. El manejador de dibujo además controla la API específica de ventanas y la creación de contexto gráfico. Ya que todos los detalles de las API's gráficas específicas son capturados en el manejador de dibujo, VR Juggler mantiene la portabilidad de la mayoría de las API's gráficas.

3.6.3.4.2 *Aplicación*

Solo una aplicación de VR Juggler y el manejador de dibujo externo asociado a una API gráfica de aplicación, tienen acceso al kernel. La aplicación pregunta al kernel por el estado del sistema y también si puede tener acceso a cualquier *proxy* de entrada que será necesario para la aplicación. VR Juggler trata la aplicación como a cualquier otro componente conectado al kernel. El kernel mantiene la aplicación sincronizada con el resto del sistema llamando aplicaciones “callback” en tiempos predefinidos durante la ejecución del *frame*. Ya que las aplicaciones están conectadas al kernel, éstas pueden ser removidas o reemplazadas en cualquier momento.

Esto permite a las aplicaciones ser modificadas o cambiadas mientras VR Juggler este activo.

La interfaz base esta extendida a través de sub clases para crear interfaces de aplicación que se especializan en API's gráficas específicas.

3.6.3.5 *Reconfiguración en tiempo de ejecución*

Todos los manejadores de VR Juggler soportan ser reconfigurados sobre el tiempo de ejecución. Cada manejador soporta una interfaz de configuración que permite añadir y remover *chunks* de configuraciones dentro de la configuración actual. Cuando el kernel recibe una petición de configuración, este manda la petición a cada manejador y a la aplicación actual. El manejador de recepción o la aplicación, procesan la petición o utilizan el mismo método de pasar la petición a sus dependencias.

Los dispositivos de entrada pueden ser reconfigurados sobre el tiempo de ejecución sin

afectar la aplicación actual. Una petición de reconfiguración puede ser enviada al manejador de entrada solicitando a un *proxy* específico que esté apuntando a un dispositivo diferente. Las pantallas pueden también ser agregadas o removidas en tiempo de ejecución. Esta característica es particularmente de mucha ayuda para sistemas de Realidad Virtual multi-pantalla ya que las pantallas pueden ser activadas y desactivadas mientras las aplicación esté en ejecución. Las aplicaciones pueden además recibir información de configuración. Esto permite a las aplicaciones responder a los cambios sobre tiempo de ejecución en la misma forma que el resto de la Plataforma Virtual lo hace.

3.6.3.6 *Soporte local multiusuario*

VR Juggler soporta a múltiples usuarios trabajar en la misma maquina. Ya que la arquitectura de VR Juggler no tiene lazos directos entre los manejadores, no hay requerimientos de que las vistas sean solo para un usuario.

Todo lo que se necesita para el *render* de la vista de múltiples usuarios, es asociar la posición del usuario con la ventana *rendering* de su vista. VR Juggler no limita el numero de *trackers* que pueden ser activados durante la ejecución de una aplicación, entonces, múltiples vistas con *trackers* pueden ser generadas dentro de la aplicación.

3.6.4 Elementos y conceptos necesarios

Para poder entender el tema de una forma clara citamos a continuación algunos conceptos constantemente utilizados dentro del tema de VR Juggler, que si bien ya fueron definidos es necesario ser claro

3.6.4.1 *Plataforma virtual*

El concepto de Plataforma Virtual facilita y simplifica los esfuerzos del desarrollo de aplicaciones en sistemas de Realidad Virtual complejos. Esto provee de un entorno de trabajo unificado que soporta el desarrollo y ejecución de aplicaciones independientemente de la tecnología subyacente. Una plataforma virtual garantiza la durabilidad de aplicaciones y permite a los desarrolladores mantenerse actualizados con los avances de la tecnología, sin tener que invertir tiempo ni modificar recursos de aplicaciones para soportar las nuevas tecnologías.

Aunque existe una creencia popular de que la abstracción orientada a objetos introduce problemas severos en la programación, el acercamiento en el diseño orientado a objetos para VR Juggler como plataforma virtual, provee la mejor forma de alcanzar las metas deseadas. Los desarrolladores de VR Juggler han hecho un gran esfuerzo sobre la optimización de los niveles de abstracción para minimizar el impacto de la ejecución.

Además han realizado pruebas, evaluando el desempeño de las aplicaciones de VR Juggler, obteniendo como resultado que el funcionamiento total es afectado en baja proporción por la abstracción orientada a objetos.

3.6.4.2 *Abstracción de hardware*

Las pantallas son abstraídas para permitir cualquier dispositivo de Realidad Virtual. El manejador de pantallas utiliza una descripción de superficie genérica que permite cualquier número de superficies de proyección. Actualmente, se utiliza VR Juggler basado en sistemas de proyección como CAVE. Además, estos sistemas soportan HMD's y sistemas Desktop de Realidad Virtual. Ya que utilizan una descripción de superficie genérica, se puede configurar una aplicación para que sea ejecutada en cualquier dispositivo de pantalla de Realidad Virtual.
Para algunos tipos de pantallas, es necesario cambiar el modelo de interacción utilizado en la aplicación. Actualmente se investigan formas para permitir a las aplicaciones modificar sus métodos de interacción basados en un tipo de dispositivo.

La entrada es abstraída a través de *proxies* que proveen a la aplicación de una interfaz uniforme para todos los dispositivos. El desarrollador de aplicaciones nunca interactúa de manera directa con los dispositivos físicos o las clases de entradas específicas que los controlan. Nuevos dispositivos pueden ser agregados por la derivación de nuevas clases que manejen los dispositivos. Toda vez que estas clases existen, las aplicaciones pueden utilizar inmediatamente el nuevo dispositivo.

3.6.4.3 *Flexibilidad en tiempo de ejecución*

El sistema basado en *proxies* proporciona a VR Juggler mucha de su flexibilidad sobre el tiempo de ejecución. Las clases de dispositivos físicos pueden ser transportadas alrededor, removidas, reiniciadas o reemplazadas sin afectar la aplicación. Con los *proxies* pasa lo mismo, ellos pueden permanecer igual, inclusive si algún dispositivo subyacente es cambiado.

3.6.4.4 *Funcionamiento en equilibrio*

VR Juggler incluye la capacidad de monitorear su funcionamiento. Esto incluye la habilidad de acumular datos acerca del tiempo que se tardan varios procesos, el funcionamiento de datos para el hardware de gráficos subyacentes, y la medida de latencia.

El manejador de entornos permite a los usuarios reconfigurar el sistema sobre el tiempo de ejecución como intento de optimizar el funcionamiento. Cuando los usuarios cambian la configuración del sistema de Realidad Virtual, los efectos en el funcionamiento serán visibles inmediatamente en el monitoreo del funcionamiento.

3.6.4.5 *Plataforma cruzada*

VR Juggler es portable para la mayoría de las plataformas usadas para desarrollo de Realidad Virtual. Para mantener la portabilidad, todos los sistemas específicos necesitan ser agrupados por la abstracción de clases. La biblioteca únicamente utiliza una interfaz uniforme. Esto permite la fácil portabilidad de la biblioteca a otras plataformas reemplazando las clases específicas del sistema derivadas de las bases abstractas.

3.6.4.6 *Extensibilidad*

La biblioteca permite la extensión sin causar impacto en el resto del sistema o de las aplicaciones existentes. Agregar nuevos dispositivos genéricos soportados es simple y transparente a las aplicaciones. Esto es gracias a que la biblioteca utiliza interfaces de clase base genéricas a una interfaz con todos los objetos en el sistema.

3.6.5 *Estado actual*

Actualmente VR Juggler es soportado en plataformas IRIX, LINUX y Windows NT/XP. Es soportado para OpenGL, OpenGL-Performer, y VTK. Es de distribución libre bajo licencia *Open Source*[32].

3.6.6 Conclusiones de VR Juggler

VR Juggler provee una plataforma de desarrollo versátil, completa y sencilla, en el que los desarrolladores no tienen que preocuparse por la programación avanzada, al contrario de esto tienen mas tiempo para preocuparse por las funciones de su aplicación, esto se logra gracias a que VR Juggler emula su propio kernel en el que se encuentran lo detalles específicos y la programación compleja. La facilidad de comunicación con el hardware es de igual manera bien lograda y sencilla, ya que el mismo kernel es quien interactúa con los dispositivos mediante su interfaz.

Una de las mayores ventajas de VR Juggler es que nos permite la ejecución de aplicaciones simultáneas, además de que una vez que la aplicación ha sido programada exitosamente puede ser ejecutada en cualquier sistema que soporte VR Juggler.

Pero sin duda que la mayor ventaja de esta plataforma es que puede ser modificada y reconfigurada al mismo tiempo que esta siendo ejecutada, por lo que no hay que detener el proceso si es que surge algún problema, ya que puede ser solucionado.

El kernel de VR Juggler esta creado de tal forma que puede soportar algún malfuncionamiento de las aplicaciones, si una aplicación demanda más recursos este puede proveerlos de alguna manera para que la ejecución del programa no sea detenida por alguna falla.

VR Juggler es sin duda la mejor opción para realizar nuestra aplicación, ya que además de las ventajas anteriormente mencionadas es una plataforma totalmente portable por haber sido realizada con métodos básicos que la mayoría de los sistemas identifican.

[32] **Open Source**. Iniciativa de programadores a nivel mundial de crear programas y poner a disposición pública y de forma gratuita, los códigos fuentes para que otros programadores puedan modificarlos y/o mejorarlos.

Capítulo 4
Desarrollo y prueba de una Aplicación Distribuida utilizando Quanta

Una vez analizadas de manera detallada las características y el funcionamiento de las diferentes plataformas de desarrollo de Realidad Virtual, podemos elegir una de ellas, que combinada con una herramienta que nos proporcione el envío de datos sobre redes, podemos desarrollar una aplicación distribuida que nos permita experimentar, comprobar y conocer como es que en realidad funciona una Aplicación Distribuida.

4.1 Definición de la mejor elección

Tomando en cuenta el análisis de las diferentes plataformas de Realidad Virtual, hemos elegido VR Juggler, ya que su arquitectura esta diseñada para ser utilizada en multiplataforma. Todas estas características son de gran utilidad, porque la aplicación podría ser implementada y mejorada en el futuro.

Su diseño orientado a objetos es de fácil portabilidad y presenta una interfaz sencilla de aprender. VR Juggler proporciona muchos niveles de abstracción con muy pequeñas o casi invisibles fallas de funcionamiento. Se espera que VR Juggler proporcione una plataforma a nuevas generaciones con alta portabilidad y escalabilidad para aplicaciones de Realidad Virtual.

Ya definimos la plataforma que nos permitirá desarrollar una aplicación de Realidad Virtual, ahora necesitamos la herramienta que nos permita la transmisión de datos a través de la red, para que la Realidad Virtual simple se convierta en Distribuida.

La herramienta más accesible y disponible para manejar datos a través de la red para trabajar con VR Juggler fue Quanta, esta fue probada por elementos del departamento de Realidad Virtual de la Dirección General de Servicios de Cómputo Académico de la UNAM, permite una excelente adaptabilidad a plataformas cruzadas, como lo es VR Juggler, además de hacer el envío y recepción de datos de manera sencilla, ya que sus funciones y parámetros están definidos y nombrados casi tal y cual es la acción que ejecutan.

A continuación se muestra lo que es Quanta y como funciona.

4.2 Quanta: Un toolkit para alto desempeño en Envío de Datos sobre Redes

Quanta es un toolkit adaptante de redes para plataformas cruzadas que soporta requerimientos de envíos de datos interactivos y de aplicaciones intensivas de banda ancha, como entornos colaborativos amplificados.

Uno de los objetivos principales de Quanta es proveer un fácil uso del sistema que permita a los programadores especificar las características de transferencia de datos de sus aplicaciones en un alto nivel, y dejar que transparentemente Quanta traduzca esos requerimientos en decisiones apropiadas sobre la red. Estas decisiones incluirán reservaciones necesarias para calidad de servicio (QoS) utilizando protocolos de transporte que satisfagan los requerimientos de transmisión de datos del usuario.

4.2.1 *Diseño de Quanta*

Quanta emergió después de casi una década de experiencia en conexiones inmersivas de sistemas CAVE a supercomputadoras, a lo que se le llamo teleinmersion.

El predecesor de Quanta es CAVERNsoft, el cual ha sido ampliamente utilizado por la comunidad de CAVE para desarrollar aplicaciones avanzadas de teleinmersión.

Consecuentemente, Quanta heredó todo acerca de abstracción de datos compartidos que han sido encontrados muy útiles en el desarrollo de estas aplicaciones, así como las aplicaciones de redes en general. Quanta provee un Control Adaptante de Red (Adaptive Network Controller) y soporta tres servicios:

- Monitoreo de Recursos
- Calidad de Servicio(QoS)
- Colección de mecanismos de transporte de datos y abstracciones de datos compartidos

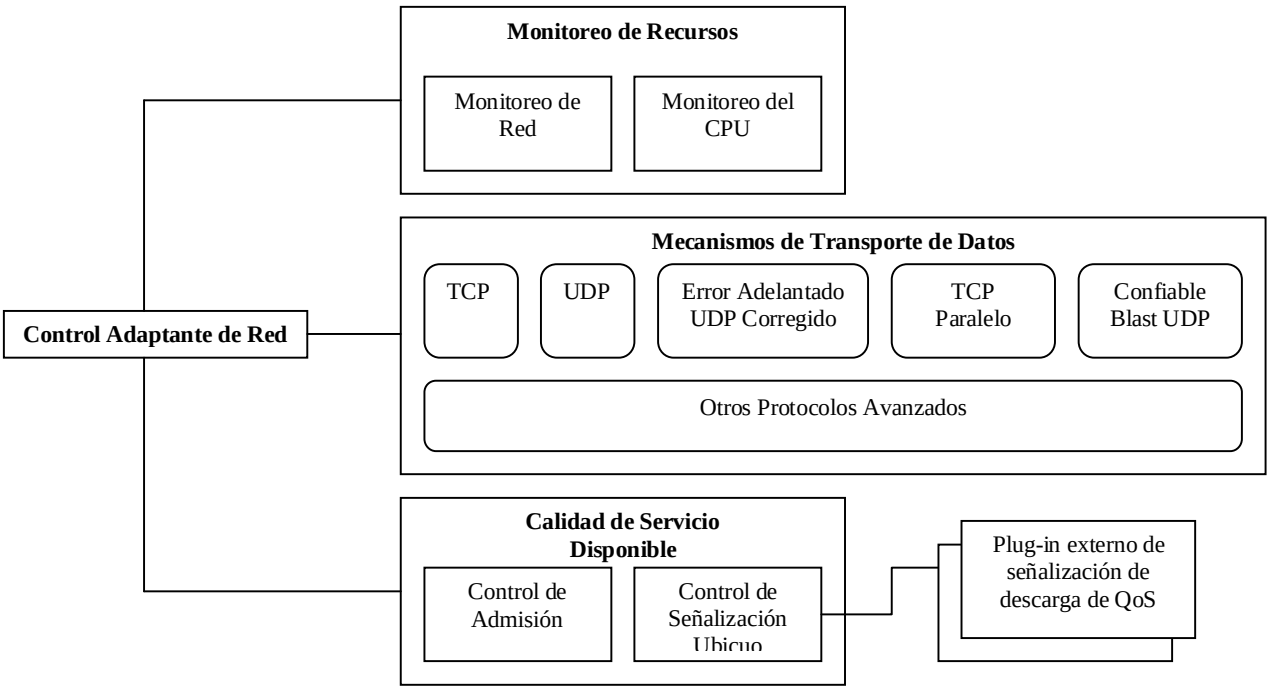


Figura 4.1
Sistema de Control Adaptante de Red de Quanta.

El primer rol del control adaptante de red es tomar los requerimientos del envío de datos específicos de la aplicación (ancho de banda, latencia, confiabilidad, etc.), para trasladarlos dentro de la red y asignar los recursos de cómputo necesarios para reunir las demandas de las aplicaciones.

El control adaptante de red monitoreará el estado actual de la red o la capacidad de QoS, seleccionando un óptimo protocolo de transmisión y haciendo las peticiones de QoS. Si éstas están disponibles, además contacta al control de admisión del sistema para determinar si el ancho de banda deseado esta disponible, y entonces hace la reservación utilizando un control de señalización. Una vez que la estrategia ha sido activada, el control adaptante de red revisará el progreso de la transmisión de datos y ajustará la red y los parámetros de cómputo para sostener el funcionamiento deseado.

Para acomodar simultáneamente múltiples y heterogéneos flujos de datos, el control adaptante de red podría alertar los parámetros del protocolo de transporte a bajo nivel o ajustar dinámicamente las reservaciones de QoS. El control de señalización mantiene independencia entre los dispositivos vía arquitectura *plug-in*[1], que dinámicamente carga las bibliotecas que proveen servicios específicos a la señal para QoS.

[1] **Plug-In**. Programa de cómputo que interactúa con otros programas para aportarles una utilidad o función específica. Se utiliza como forma de expansión a los programas, de tal forma que se pueda añadir nueva funcionalidad sin afectar el funcionamiento existente ni complicar el desarrollo del programa principal.

4.2.1.1 *Stargate: Trayectoria ligera para Quanta*

El trabajo está actualmente enfocado al desarrollo de infraestructura de software para trayectorias ligeras sobre redes.

Mientras que Quanta puede asegurarse que los datos son entregados óptimamente sobre estas trayectorias ligeras, no tiene del todo la habilidad de asignar estas trayectorias. Es aquí donde entra el término *Stargate*.

Una aplicación pretendiendo asignar una trayectoria ligera entre dos puntos finales contactará su *Stargate* local, el cual atenderá genéricamente los mensajes de señalamiento de la trayectoria ligera a los *Stargates* vecinos hasta que el destino es alcanzado.

Cada *Stargate* transportará el mensaje genérico de señalamiento de la trayectoria ligera dentro del mensaje de señalamiento nativo, que es entendido por la facilidad del señalamiento de la trayectoria ligera del domino interno local.

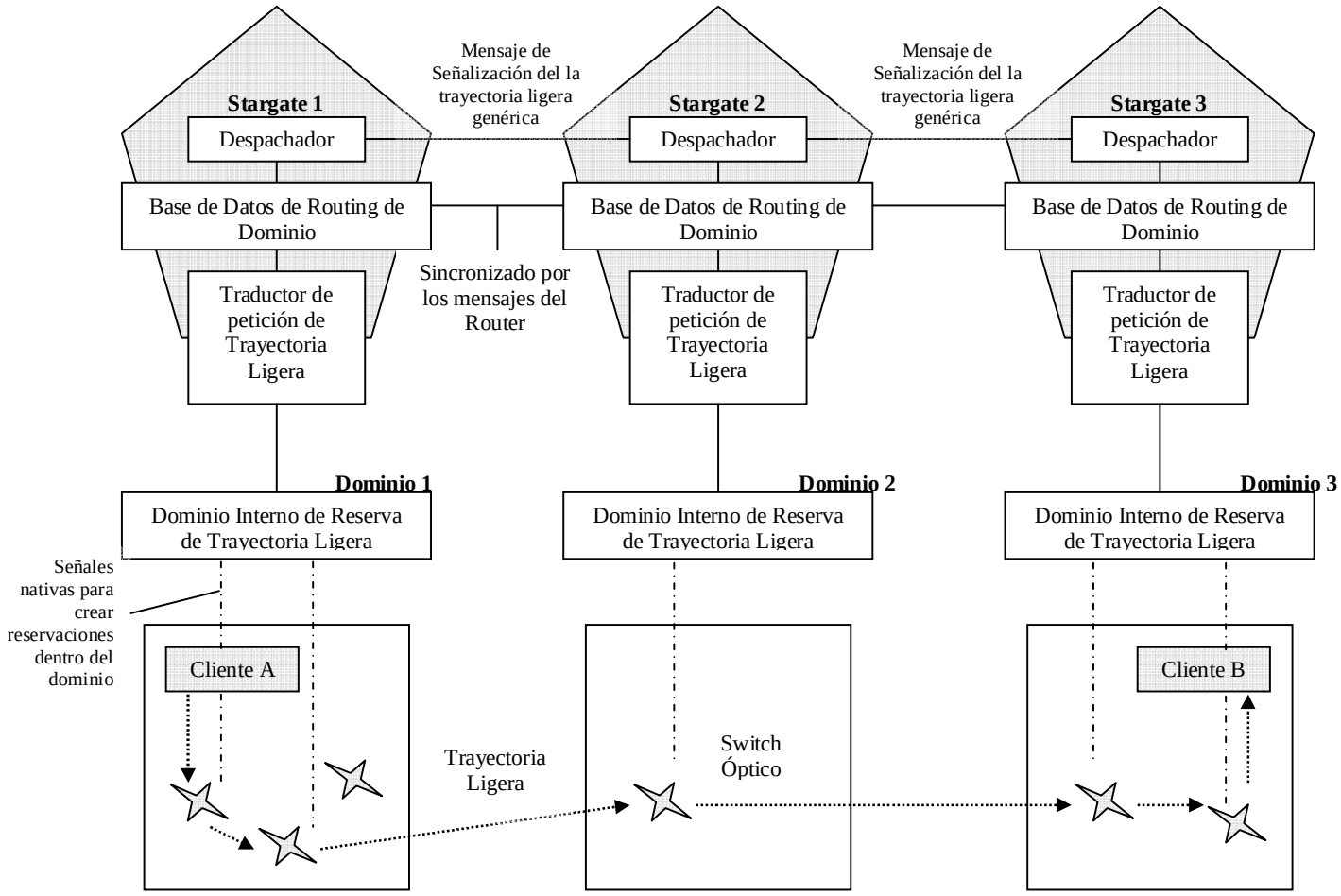


Figura 4.2
Arquitectura del Interdominio que proporciona la trayectoria ligera del Stargate.

4.2.1.2 *Capacidad de transporte y compartición de datos de Quanta*

La capacidad de transportar datos de Quanta incluye clases de C++[2] que simplifican el nivel de programación de sockets[3] de comunicaciones TCP[4], UDP[5] y multicast[6], las cuales

[2] **C++**. Lenguaje de programación orientado a la implementación de sistemas operativos que proporciona orientación a objetos. Es la versión de C mas aceptada, difundida y funcional.

[3] **Sockets**. Puntos de comunicación por los cuales los procesos pueden emitir o recibir información.

[4] **TCP** (Transmission Control Protocol). Protocolo básico de comunicación en Internet que permite la transmisión de información en redes de computadoras.

[5] **UDP** (User Datagram Protocol). Protocolo de transporte de información basado en el intercambio de fragmentos de paquetes a través de la red sin que se haya establecido previamente una conexión, ya que el propio fragmento de paquete incluye suficiente información de direccionamiento.

están agrupadas en clases de C++ como: **QUANTAnet_tcp_c, QUANTAnet_udp_c y QUANTAnet_mcast_c,** respectivamente. Todas las clases de transporte de datos tienen funciones de monitoreo interno, de tal forma que la aplicación puede fácilmente determinar qué proporción de ancho de banda se está usando y qué tanta latencia se está experimentando. Como Quanta es un toolkit de plataforma cruzada, proporciona un paquete de API's[7] que permiten a las aplicaciones asegurarse de que sus transmisiones son correctamente realizadas en el formato especificado dentro del sistema especificado.

4.2.2 Protocolo confiable blast UDP

El confiable blast UDP (Reliable Blast UDP, RBUDP) tiene dos objetivos principales:

- Mantener los canales de transmisión de red llenos tanto como sea posible durante la transferencia de los datos.

- Evitar la interacción por paquete TCP, lo que significa que estos paquetes no son enviados por una ventana de datos trasmitidos, si no que los datos son agregados y entregados al final de la fase de transmisión.

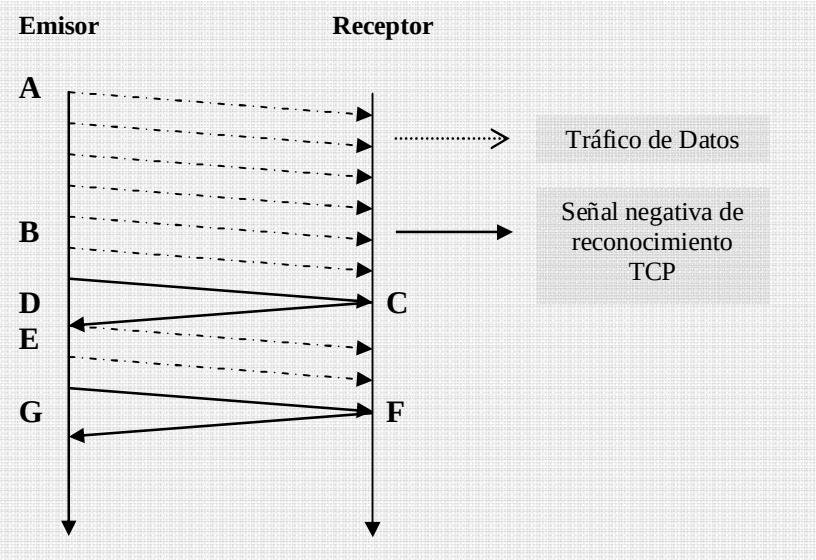


Figura 4.3
Diagrama de secuencia de tiempo del RBUDP.

[6] **Multicast**. Esquema de red en el cual existe un único flujo de datos proveniente de una sola fuente.

[7] **API´s** (Application Programming Interface). Conjunto de especificaciones de comunicación entre componentes de software.

En la primera fase de transmisión de datos (de A a B en la figura), RBUDP envía toda la carga útil al usuario especificado mandando el índice utilizando paquetes UDP. Ya que UDP es un protocolo no confiable algunos paquetes podrían llegar a perderse debido a la congestión o imposibilidad del receptor para leer paquetes lo suficientemente rápido. El receptor por lo tanto debe guardar la cuenta de los paquetes que son recibidos para determinar cuales paquetes deben ser retransmitidos. Al final de la fase de transmisión de datos, el emisor envía una señal de *Done* vía TCP (C en la figura) por lo que el receptor se entera que ya no hay más paquetes UDP por llegar. El receptor responde enviando un reconocimiento, este consiste de una cuenta de mapas de bits de los paquetes recibidos (D en la figura). El emisor responde re-enviando el paquete perdido, y el proceso se repite así mismo hasta que no se necesite retransmitir más paquetes.

En RBUDP el parámetro más importante de entrada es el índice de envío del Blast UDP. Para minimizar las pérdidas, el índice de envío no debe ser más extenso que el ancho de banda.

Existen tres versiones desarrolladas de RBUDP:

- **RBUDP sin optimización de unión/dispersión**. Esta es una mínima implementación de RBUDP donde cada paquete entrante es examinado para determinar a que parte del buffer de memoria de la aplicación debe ir, y después éste se mueve ahí.

- **RBUDP con optimización de unión/dispersión.** Esta implementación toma ventaja del factor entrada de paquetes conforme van llegando, y si el índice de transmisión está bajo el máximo rendimiento de procesamiento de la red, los paquetes son difícilmente perdidos.

- **RBUPD artificial.** Esta implementación es la misma que el esquema sin optimización de unión/dispersión excepto por los datos entrantes; nunca son movidos a la memoria de la aplicación.

4.2.3 *Conclusión del análisis Quanta*

Se ha descrito la arquitectura y capacidades de Quanta, una plataforma cruzada de C++, para el desarrollo de aplicaciones altamente funcionales en red. Se describió en particular el esquema de transferencia de datos llamado RBUPD. RBUPD elimina el lento comienzo y congestión de los mecanismos de control TCP, y agrega reconocimientos de tal forma que todo el ancho de banda de un canal es usado solamente para entrega de datos. Para amplias transmisiones, RBUPD puede proporcionar una entrega precisa en índices de envío a usuarios específicos. RBUPD funciona mejor en grandes cargas útiles que en pequeñas.

Por estas razones se ha escogido Quanta para la implementación distribuida de nuestra aplicación.

4.3 Requerimientos: software y hardware utilizado

La aplicación será realizada en la Dirección General de Servicios de Cómputo Académico, dentro del área de Realidad Virtual, ya que ésta área cuenta con personal altamente capacitado en materia de Realidad Virtual y tienen el equipo y recursos para proporcionar las herramientas necesarias para el desarrollo de esta aplicación.

La consola física donde residirá nuestro programa es sobre dos PC´s, con sistema operativo Linux GNU/Debian[8]. Utilizaremos la versión 2-Alpha 4 de VR Juggler así como la versión 0.1 de Quanta. Recurriremos a un modelo existente en 3D de OpenSG[9], "OsgNav". Por último, la conjunción de todos estos elementos estará hecha por el lenguaje de programación C++, que cuenta con una excelente flexibilidad para trabajar con múltiples toolkit's y plataformas.

El ambiente de programación será proporcionado mediante el editor de desarrollo "emacs", ya que permite una manera limpia y eficiente de programación.

4.4 Desarrollo de la aplicación

Para demostrar esta aplicación se utilizarán los dos tipos de documentación para evidenciar programas: la documentación externa y la documentación interna. La documentación externa se representará explícitamente en cada uno de los siguientes pasos, mientras que la documentación interna se encuentra descrita dentro del código fuente que se encuentra en los Anexos.

4.4.1 *Objetivo y descripción de la aplicación*

Se pretende realizar una aplicación que demuestre el uso de VR Juggler y Quanta como herramientas para el desarrollo de una Aplicación Distribuida.

Se generará un programa cliente y un programa servidor. El programa servidor será ejecutado en la PC servidor, este programa esperará hasta recibir la confirmación de que el programa cliente, alojado en la PC cliente, esté conectado, creando entre estas dos PC's una sesión de comunicación mediante la cual persistirá el envío de datos a través de la red.

El programa cliente estará integrado con una función de VR Juggler que permitirá interpretar datos desde el mouse. Cada botón enviará una señal diferente al programa servidor, el cual tomará esos datos para enviar la señal de recepción y reconocimiento, entonces ejecutará los datos recibidos sobre su propio programa servidor que estará corriendo durante toda la sesión. La sesión terminará cuando el cliente se ausente de la sesión.

[8] **Linux GNU/Debian**. Distribución de Linux que basa su principio y fin en el software libre.

[9] **OpenSG** (Open Scene Graph). Toolkit para alto desempeño en el desarrollo de gráficas 3D.

Gráficamente se tendrá que ver la misma imagen en las dos PC's, pero el cliente controlará la imagen que se verá reflejada en sí mismo y en el servidor.

El desarrollo de esta aplicación será de manera clara y concreta, ya que se pretende que ésta aplicación sea retomada en el futuro para ser ampliada y mejorada.

4.4.2 *Planteamiento*

La aplicación será realizada utilizando básicamente el compilador de C++ dentro de un ambiente Linux GNU/Debian. El lenguaje C++ será la herramienta que nos permita fusionar la Plataforma Virtual de VR Juggler y la herramienta de transmisión de datos que es Quanta.

Se desarrollarán dos programas por separado: el programa servidor y el programa cliente. El programa servidor tendrá como función básica levantar el programa que fungirá como servidor, esto significa que abrirá un puerto y esperará la respuesta del programa cliente para comunicarse con el. A su vez mantendrá una conexión activa de envío y recepción de datos.

Este programa, además, será el que reciba la información que enviará el cliente y la interpretará para posteriormente ejecutarla sobre el modelo de OpenSG que utilizaremos (llamado OsgNav). Los datos que el servidor reciba servirán para permitir que nuestro modelo "OsgNav", se traslade hacia el fondo de la pantalla, hacia el frente, o que se inmovilice. Para esto nos basamos en la utilización de las funciones de Quanta, éstas nos permitirán detectar la conexión del cliente y revisar la conexión para determinar si el envío o recepción de datos es factible, además de utilizar un buffer para almacenamiento y empaquetado de los datos a ser enviados e interpretados.

Por su parte, el programa cliente mantendrá el esquema básico de una aplicación de Realidad Virtual, éste es el de recibir datos de un dispositivo externo, en este caso los tres botones de el mouse (botón derecho, botón izquierdo y central) con una función, e interpretarlos en el programa local, asimismo gracias a las funciones de Quanta, estos mismos datos son enviados en tiempo real al programa servidor para ser interpretados como anteriormente se explicó. Esto hará que los dos programas, siendo ejecutados en diferentes máquinas, establezcan una comunicación (utilizando la dirección IP del servidor), la cual se verá plasmada en los monitores de los respectivos equipos, en los dos se tendrá que ver exactamente lo mismo.

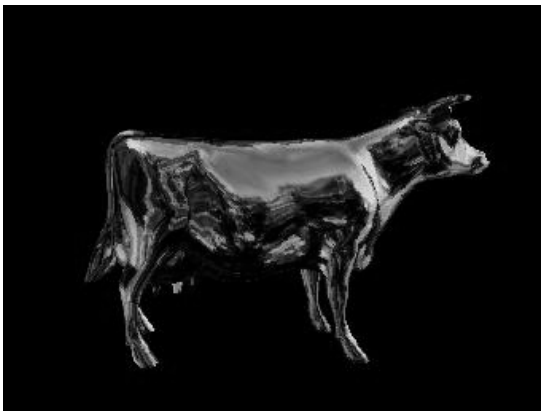


Figura 4.4

Modelo OsgNav de OpenSG que se utilizará para el desarrollo de la aplicación.

Tanto el cliente como el servidor estarán montados con VR Juggler. No existirá una función principal que controle toda la ejecución del programa como se explicó anteriormente. La aplicación es montada usando el esquema de "preFrame" de VR Juggler. El "preFrame" se puede definir como una función recursiva que realiza el despliegue de imágenes actualizadas, lo que nos permite percibir la sensación de movimiento. Esta función fue descrita a detalle en el anterior capitulo.

En todo sistema de Realidad Virtual Distribuida la latencia es algo de lo que no se puede salir intacto. La forma de reducir latencia en este sistema será por la optimización de código, ya que no es una aplicación tan robusta, por lo que se tratará de programar de la manera mas apropiada y reduciendo código. Esto se puede hacer gracias a la ayuda de Quanta, ya que existen funciones de ésta que pueden utilizarse para evitar algunas partes de código de programación con C++.

El tipo de distribución de base de datos de Realidad Virtual será homogénea (en cierta forma esto también ayudará a reducir latencia), lo que significa que los dos participantes del sistema, tanto el servidor como el cliente tienen exactamente los mismos datos: el modelo "OsgNav", la plataforma VR Juggler, el compilador de C++ y el toolkit Quanta, por lo que solo serán enviados y recibidos los datos para la manipulación de la aplicación. Cabe mencionar que el modelo "OsgNav" llevará un desplazamiento constante, por lo que si se envía una orden de trasladarse al fondo, el modelo seguirá este patrón con una cierta velocidad, no importando que ya no se esté presionando algún botón del mouse.

4.4.3 *Definición de módulos e interfaces, entrada y salida de datos*

4.4.3.1 *El editor EMACS*

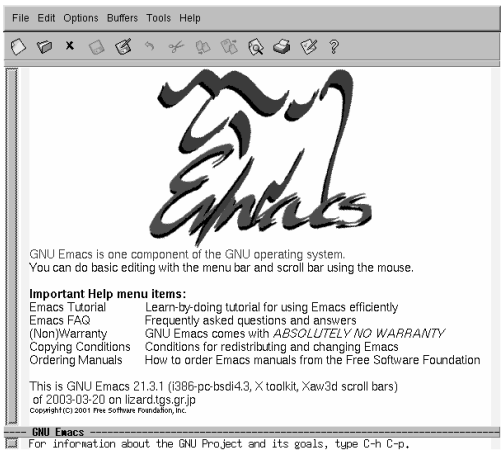
Hasta ahora hemos hablado de la funcionalidad del programa, de lo que realizará y como *grosso modo* se enlazarán todos los elementos que conformarán la aplicación, pero algo que es importante mencionar es la interfaz de desarrollo. El sistema operativo Linux nos permite utilizar por *default* el editor de texto "vi", con el que podemos crear programas o textos, pero para fines prácticos utilizaremos el editor "emacs", éste nos permite manejar bibliotecas, funciones y etiquetas dependiendo del programa que queramos desarrollar.

En este caso manejaremos los programas con extensión ".cpp", programas de C++, por lo que el editor "emacs" asignará colores y sangrías dependiendo de lo que estemos manejando, ya sea una función, una declaración de variable, un comentario, etc.

No solo por el formato se decidió utilizar este editor, sino también porque permite compilar el programa desde adentro, a diferencia del editor "vi", que para compilar un programa se tiene que hacer desde la línea de comandos. Además de esta ventaja, permite el manejo, desplazamiento y búsqueda a través del código de una manera óptima y sencilla, así como proporcionar la posibilidad de abrir varias ventanas dentro del mismo "emacs" que fungen como terminales o editores.

Figura 4.5

Ventana del editor Emacs.



4.4.3.2 *El programa servidor*

El programa servidor, realizado en C++, esta formado por una serie de funciones tanto del propio C++, como de Quanta y VR Juggler, a continuación se presentará el esquema del programa y algunos fragmentos de código necesarios para la explicación del mismo. El código fuente completo con la documentación interna se encuentra en el Anexo A.

Anteriormente se explicó que VR Juggler no maneja una función principal "main( )", como en la mayoría de las otras plataformas, en este caso se maneja un programa ".cpp" que tiene el esquema como se muestra en la siguiente figura.

```
void InitClientData()
{
        /*Ciclo de inicialización de clientes*/
}

void OsgNav::preFrame()
{
        /*Programa general, conexión con cliente,
        recepción de datos, validaciones, navegación*/
}

void OsgNav::bufferPreDraw()
{
        /*Datos que cargan el buffer antes del dibujado de
        frame*/
}

void OsgNav::myInit()
{
        /*Elementos que se requiere sean cargados antes
        de iniciar la ejecución*/
}
```

Figura 4.6
Esquema básico del programa Servidor

La función **InitClientData( )** nos sirve para inicializar el cliente. Lo que hace esta función es llevar un control del tamaño de datos que son transmitidos dependiendo el número de clientes que definimos puedan ser conectados, esta función se dejó indicada básicamente para que la aplicación sea retomada en el futuro para ser ampliada. En general, dentro de esta aplicación no es muy importante, ya que solo se maneja la conexión con un cliente a la vez.

La función **OsgNav::myInit( )**, es descrita antes que la función **OsgNav::preFrame( )** ya que dentro de ésta van todas las inicializaciones que necesitaremos para el funcionamiento de la aplicación. A continuación se explica en orden el contenido de esta función.

- Se inicializa Quanta con la función **QUANTAinit()**, si no hacemos esto, las funciones de Quanta simplemente no funcionarán.
- Llamamos a la función para inicializar los clientes **InitClientData( )**.
- Declaramos nuestro programa como servidor con la función **QUANTAnet_tcpServer_c**.
- Asignamos el buffer y su tamaño, tanto de lectura como de escritura.
- Damos de alta el puerto por el que trabajaremos, en este caso, para el departamento

de Realidad Virtual utilizaremos el puerto 4500. En caso de que el puerto no esté disponible se mandará a la pantalla un mensaje de error, en caso contrario se levantará el servidor.

- El modelo "OsgNav" es cargado con sus respectivas transformaciones, además de tener parámetros que pueden ser modificados, como la orientación del modelo.

- Inicializamos la velocidad de comienzo para la navegación así como el incremento de ésta entre cada frame. La velocidad inicio será 0 y el incremento entre cada frame será de 0.0005.

En la función **OsgNav::preFrame()** se define el funcionamiento en tiempo real de la aplicación. Es en esta parte donde se unen los elementos de VR Juggler con los de Quanta por medio de algunas funciones que se describen adelante. A continuación se explicará paso a paso las acciones que se realizan dentro de la función **OsgNav::preFrame()** para el programa servidor.

- Se detecta que se haya podido levantarse a sí mismo exitosamente como servidor, en caso contrario no podrá acceder a las demás funciones. La forma de levantar el servidor está explicada dentro de la función **OsgNav::myInit()**.

- Después continúa con la detección del cliente, en caso de no detectar alguno, el programa seguirá corriendo hasta detectar una conexión. En caso de hallar un cliente, manda un mensaje de conexión establecida y accede al siguiente paso.

- Se inicializa el número del cliente para poder ser manejado.

- Entra a un ciclo de envío de datos, siempre que exista un dato, éste será leído hasta que no quede alguno.

- En caso de haber datos, éstos son leídos del buffer al que el cliente mandó el dato para ser almacenado, para esto nos valemos también de la función **QUANTAnet_tcpClient_c::NON_BLOCKING**. El "**NON_BLOCKING**" nos sirve para que el programa siga corriendo si en determinado momento se ha dejado de enviar datos, esto es así ya que el modelo "OsgNav" en su movimiento lleva un desplazamiento constante.

- Quanta revisa que la operación de la recepción del dato se haya llevado a buen término con la función **QUANTAnet_tcpClient_c::OK**, este chequeo sirve para determinar la continuación del programa.

- Después de recibir el dato, este es sacado del buffer y es almacenado en una variable que nos permite la interpretación del valor sobre el desplazamiento de la imagen.

- El dato es pasado como parámetro a una función de navegación, éste es interpretado y reflejado como desplazamiento sobre el modelo que se mostrará en pantalla.

Para finalizar con la estructura del programa se realiza la llamada a la función **OsgNav::bufferPreDraw()**, la cual hace la función de asignar color al buffer de almacenamiento de la imagen, que es el fondo que se mostrará detrás del modelo OsgNav.

4.4.3.2 *El programa cliente*

El programa cliente, al igual que el servidor y como complemento, está realizado en C++, formado por funciones de C++, Quanta y VR Juggler. El esquema del programa cliente es similar al del servidor. El código completo con la documentación interna se encuentra en el Anexo B.

```
void OsgNav::preFrame()
{
        /*Programa general, comunicación con servidor,
        envío de datos, validaciones, navegación*/
}

void OsgNav::bufferPreDraw()
{
        /*Datos que cargan el buffer antes del dibujado de
        frame*/
}

void OsgNav::myInit()
{
        /*Elementos que se requiere sean cargados antes
        de iniciar la ejecución*/
}
```

Figura 4.7
Esquema básico del programa cliente.

La función **OsgNav::myInit( )**, al igual que el programa servidor, es descrita antes que la función **OsgNav::preFrame( )**, de la misma manera en esta función son cargados los elementos necesarios para el funcionamiento del programa.

- Se inicializa Quanta con la función **QUANTAinit()**.
- Declaramos nuestro programa como cliente con la función **QUANTAnet_tcpClient_c**.
- Asignamos el buffer y su tamaño, tanto de lectura como de escritura.
- Damos de alta el puerto por el que trabajaremos, que es, como anteriormente lo describimos, el puerto 4500.
- El modelo “OsgNav” es cargado con sus respectivas transformaciones.

- Inicializamos la velocidad del comienzo con valor 0 para la navegación así como el incremento de 0.0005 por cada frame.

Al igual que en el servidor, en la función **OsgNav::preFrame()** se define el funcionamiento en tiempo real de la aplicación y se fusionan los distintos elementos que interactuarán para formar la aplicación. A continuación se explicará paso a paso las acciones que se realizan dentro de la función **OsgNav::preFrame()** para cliente.

- Se obtiene la información del dispositivo externo, en este caso el mouse, por el que se obtendrán los datos a ser interpretados por la aplicación.

- Se asigna un tipo de valor que representará a cada uno de los botones del mouse que sean presionados, siendo que el desplazamiento hacia adelante se representa con valores positivos presionando el botón izquierdo del mouse, el desplazamiento hacia atrás se representa con valores negativos presionando el botón derecho del mouse y el central de no desplazamiento con valores de cero.

- Si el dato es positivo se suma el incremento al valor anterior de desplazamiento, manda el dato al buffer que será recibido por el servidor y la función de navegación local para ser interpretado y reflejado como desplazamiento hacia adelante sobre el modelo que se mostrará en pantalla en ambos programas.

- Si el dato es cero la variable de desplazamiento se pone en cero, manda el dato al buffer que será recibido por el servidor y a la función de navegación local para ser interpretado y reflejado como desplazamiento nulo sobre el modelo que se mostrará en pantalla en ambos programas.

- Si el dato es negativo se resta el incremento al valor anterior de desplazamiento, manda el dato al buffer que será recibido por el servidor y a la función de navegación local para ser interpretado y reflejado como desplazamiento hacia atrás sobre el modelo que se mostrará en pantalla en ambos programas.

Para finalizar con la estructura del programa se realiza la llamada a la función **OsgNav::bufferPreDraw()**, la cual hace la función de asignar color al buffer de almacenamiento de la imagen, que es el fondo que se mostrará detrás del modelo OsgNav.

4.4.4 *Requerimientos y uso futuro*

El programa utiliza los datos que recibe del dispositivo externo que en este caso es el mouse, el programa no está limitado a utilizar solo estas entradas, podría estar conectado a cualquier otro dispositivo y solo se tendrían que hacer algunos ajustes para obtener el buen funcionamiento de la aplicación. Recordemos que esto es posible gracias a VR Juggler.

4.4.5 *Uso de la aplicación*

A continuación se mostrará como funciona la aplicación, se revela paso a paso como es que debe ser ejecutada y algunos de los errores que se podrían generar.

En la siguiente figura se muestra el modelo OsgNav que fue utilizado para la realización de ésta aplicación, en esta imagen se puede ver la ejecución simple del modelo en el espacio.

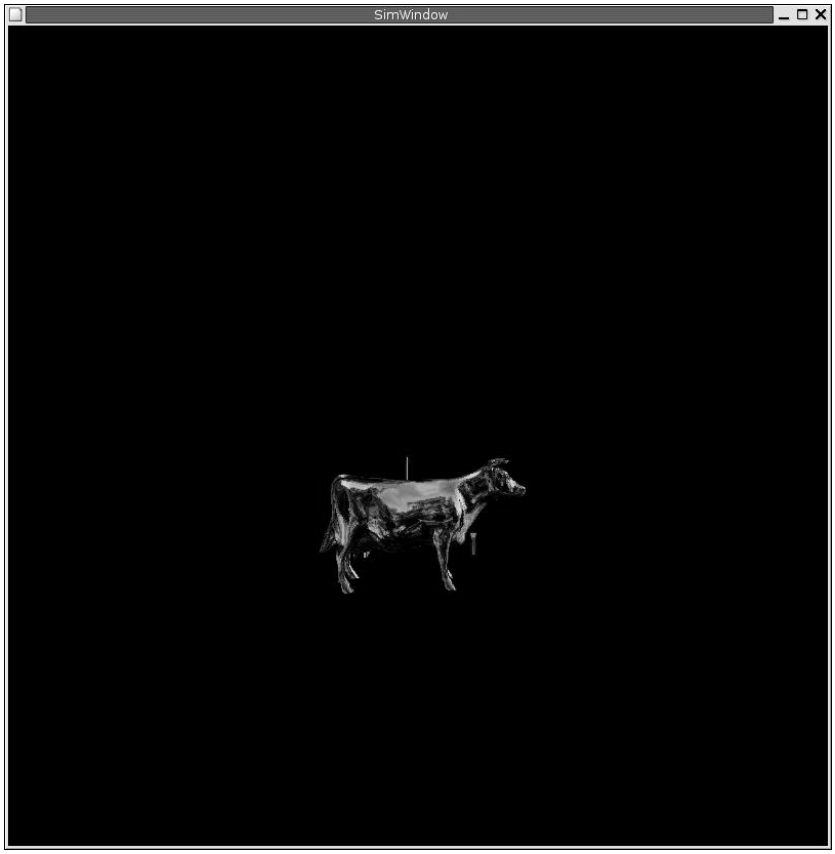


Figura 4.8
Modelo OsgNav.

El primer paso para el buen funcionamiento de esta aplicación es ejecutar el servidor. El servidor tiene que estar funcionando antes de que el cliente se conecte para brindarle el servicio, cualquiera que este sea. En caso contrario, si el cliente fuera ejecutado antes de que el servidor estuviera activo, un error ocurriría y nos enviaría en pantalla lo que se muestra en la siguiente imagen.

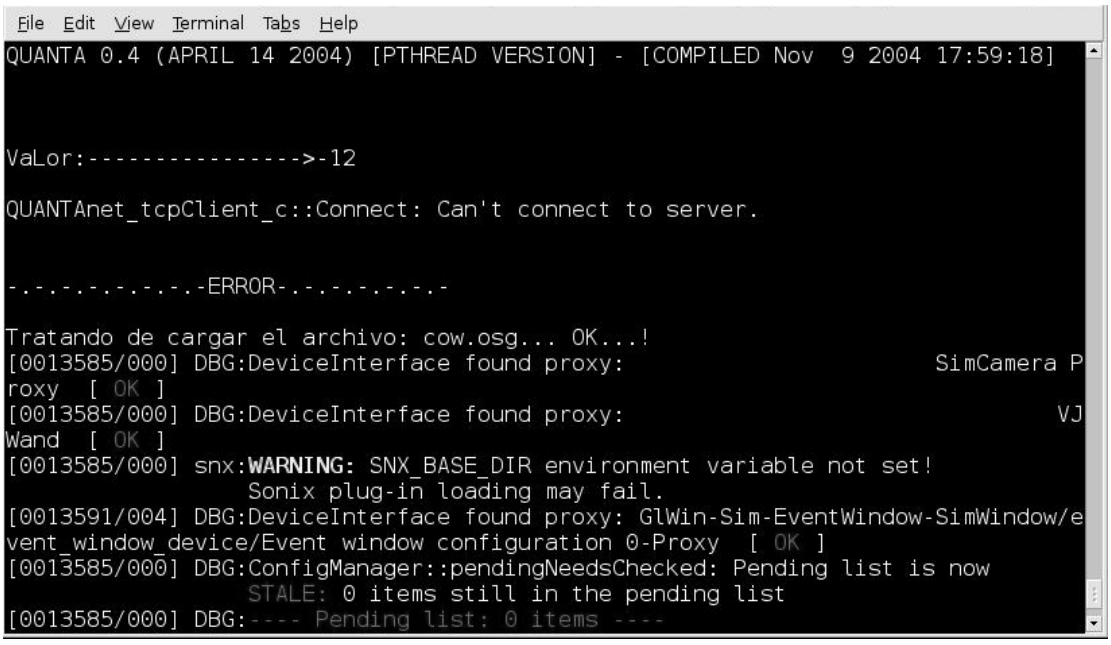


Figura 4.9
Error de conexión con el servidor.

Por lo que se mostró en la anterior figura, lo que se debe hacer es ejecutar el programa servidor antes que el cliente. Si este programa es ejecutado con éxito, lo que significa, que el puerto solicitado este disponible, mandará un mensaje en pantalla de que el servidor se ha levantado exitosamente.

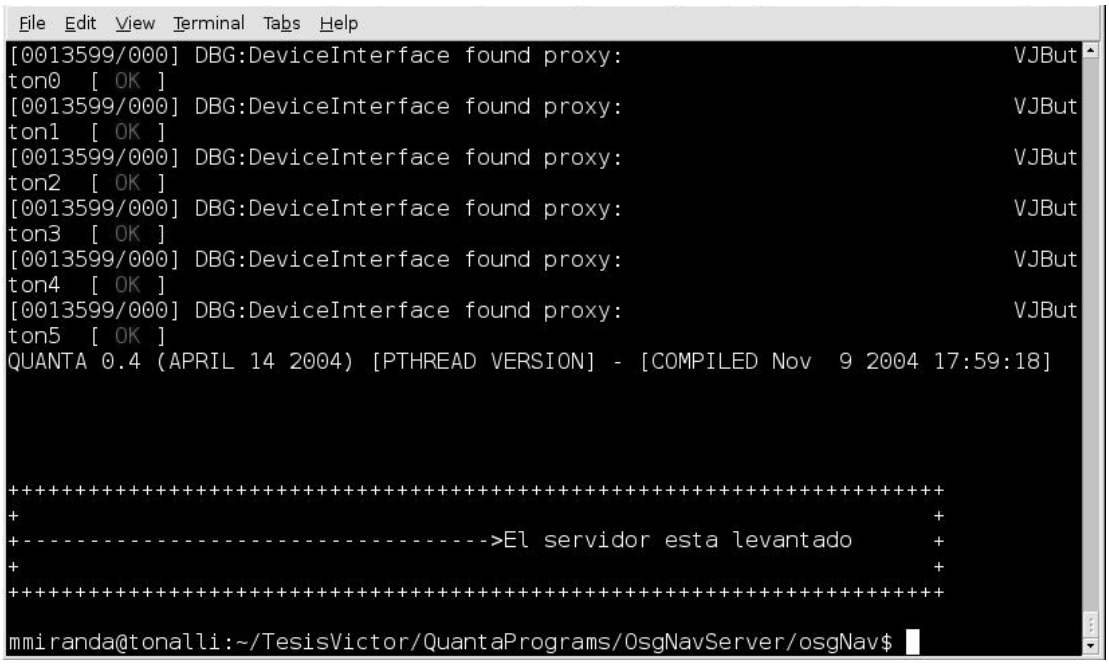


Figura 4.10
El programa servidor ha sido exitosamente ejecutado.

El servidor está corriendo, pero aún no hay algún cliente conectado por lo que se mostrará en pantalla iteradamente un mensaje de no conexión de un cliente.

Este menaje es persistente, ya que está dentro de la función “preFrame” de VR Juggler, por lo que es un ciclo hasta que encuentre que la conexión con el cliente se ha establecido.

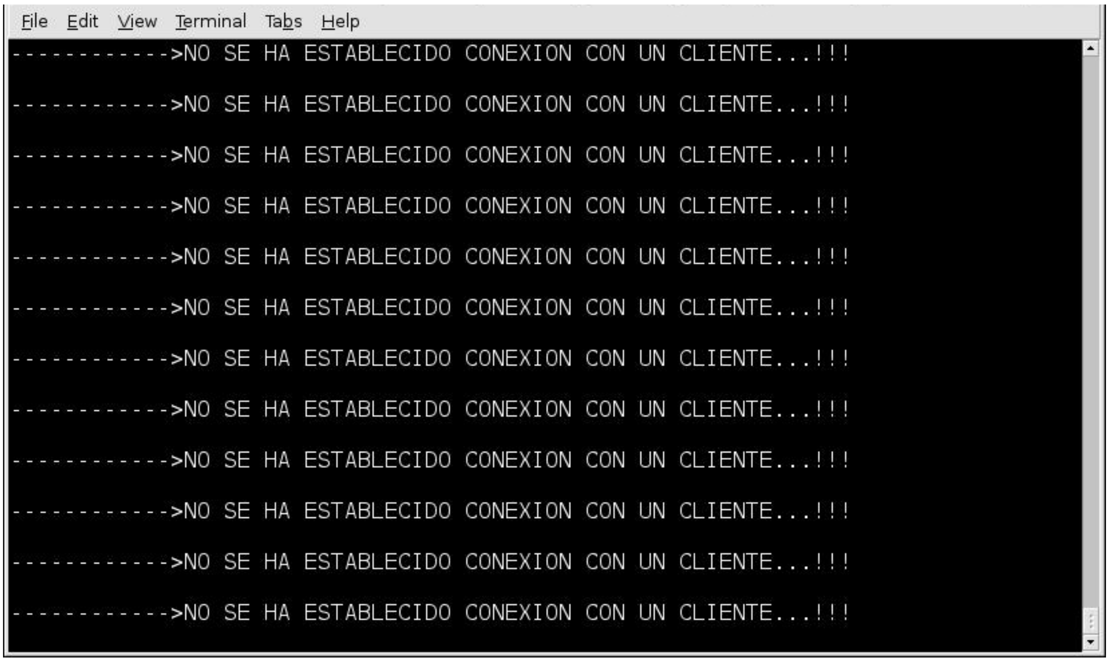


Figura 4.11
No se ha detectado aún algún cliente.

Ya teniendo al servidor en ejecución en una PC, lo siguiente es ejecutar el programa cliente con la dirección IP del servidor respectivo. Si el cliente es levantado exitosamente se desplegará el siguiente mensaje en pantalla:

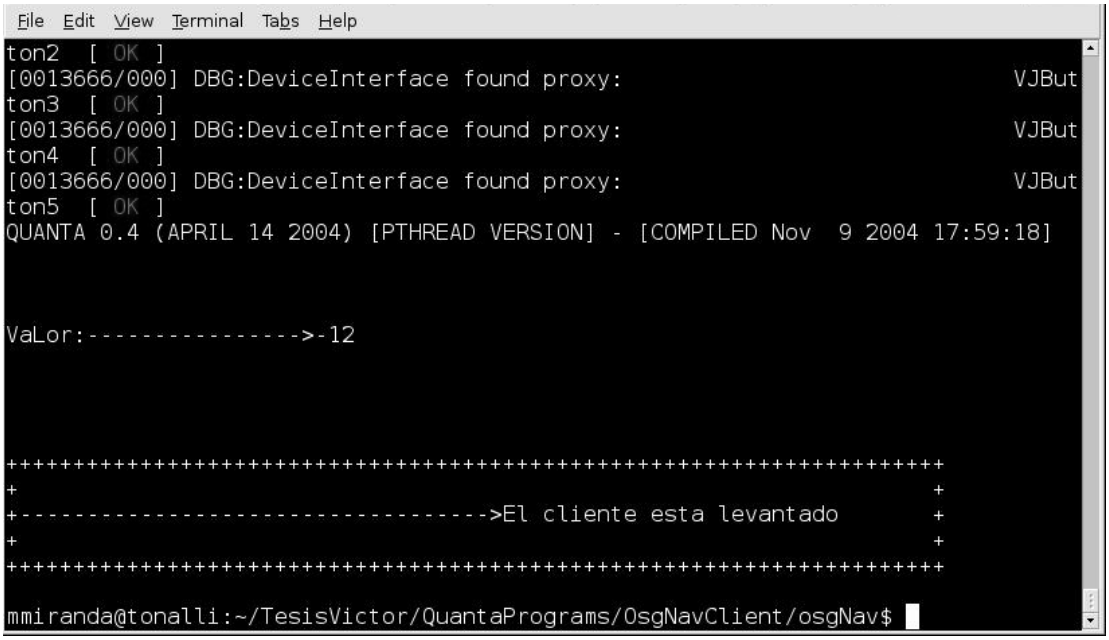


Figura 4.12
El programa cliente fue ejecutado exitosamente.

El cliente esta ahora arriba, en este momento es cuando el servidor debe detectar la conexión que el cliente le está solicitando.

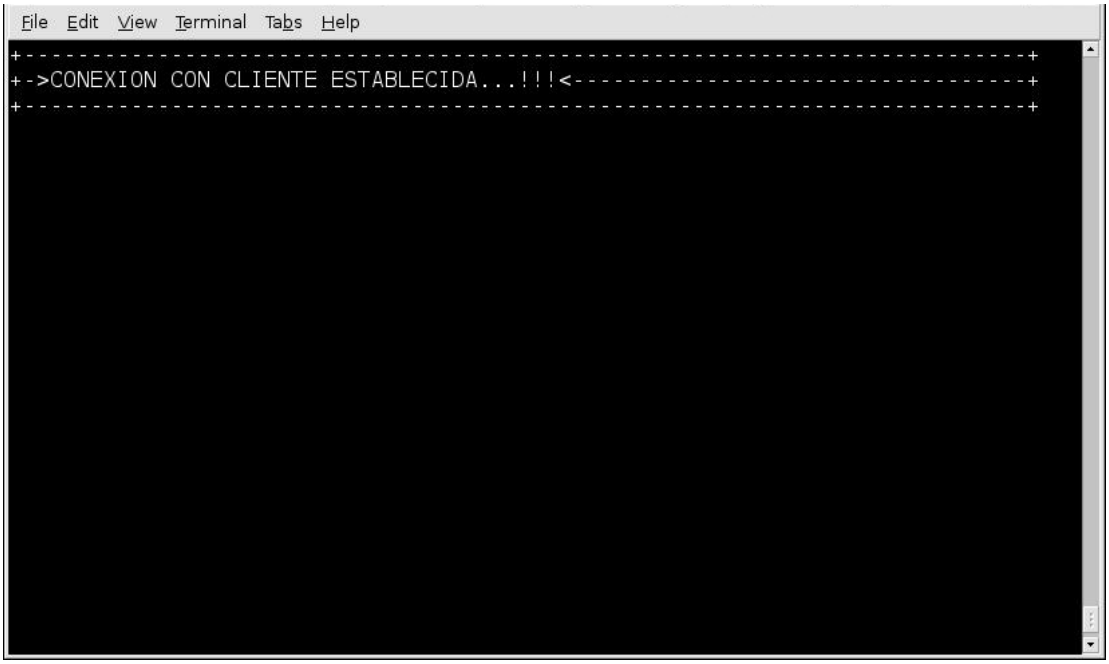


Figura 4.13
El servidor ha detectado que el cliente esta siendo ejecutado y esta listo para la comunicación.

Cuando el cliente es ejecutado se muestra un marco de control, esta es la interfaz de la función que recibe los datos de un dispositivo externo, en este caso del mouse. El puntero debe estar sobre este marco para poder comenzar a presionar los botones y enviar los datos al programa cliente.

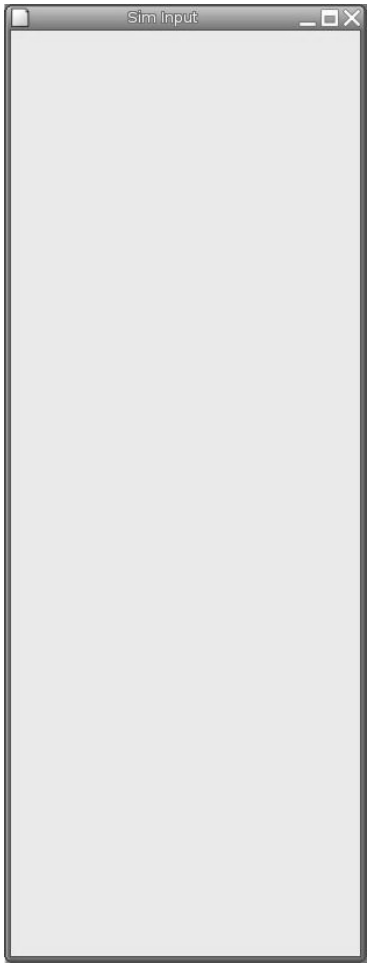


Figura 4.14

Marco de control que sirve como interfaz para que puedan ser obtenidos los datos provenientes del mouse.

Con una imagen simultánea de ambas PC's, podemos mostrar como es que se ejecutan los programas en ambos casos, el cliente y el servidor.

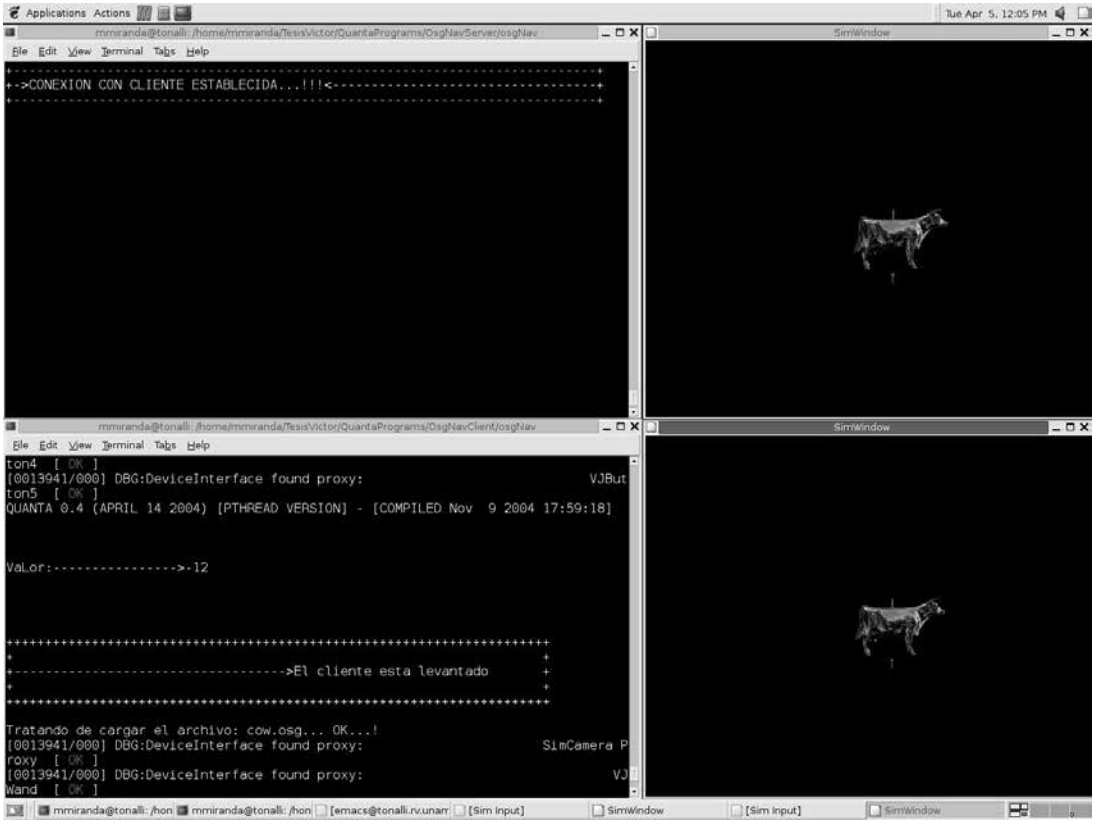


Figura 4.15
Vista simultánea del programa cliente y el programa servidor en ejecución.

Utilizando una conexión remota podemos emplear una misma pantalla para monitorear ambos programas al mismo tiempo sin perder un solo detalle de la ejecución.

Para utilizar la conexión remota hacemos uso del comando “ssh – X”, este comando nos permitirá conectarnos remotamente con el equipo y nos permitirá hacer uso de las aplicaciones gráficas que posea, por lo que podremos observar en una sola pantalla las dos aplicaciones aunque estén situadas en diferentes PC’s.

Para hacer más fácil la manipulación, acomodamos las ventanas, en la parte superior mostraremos el programa servidor en ejecución, en la parte inferior el programa cliente y al centro el marco de control de obtención de datos.

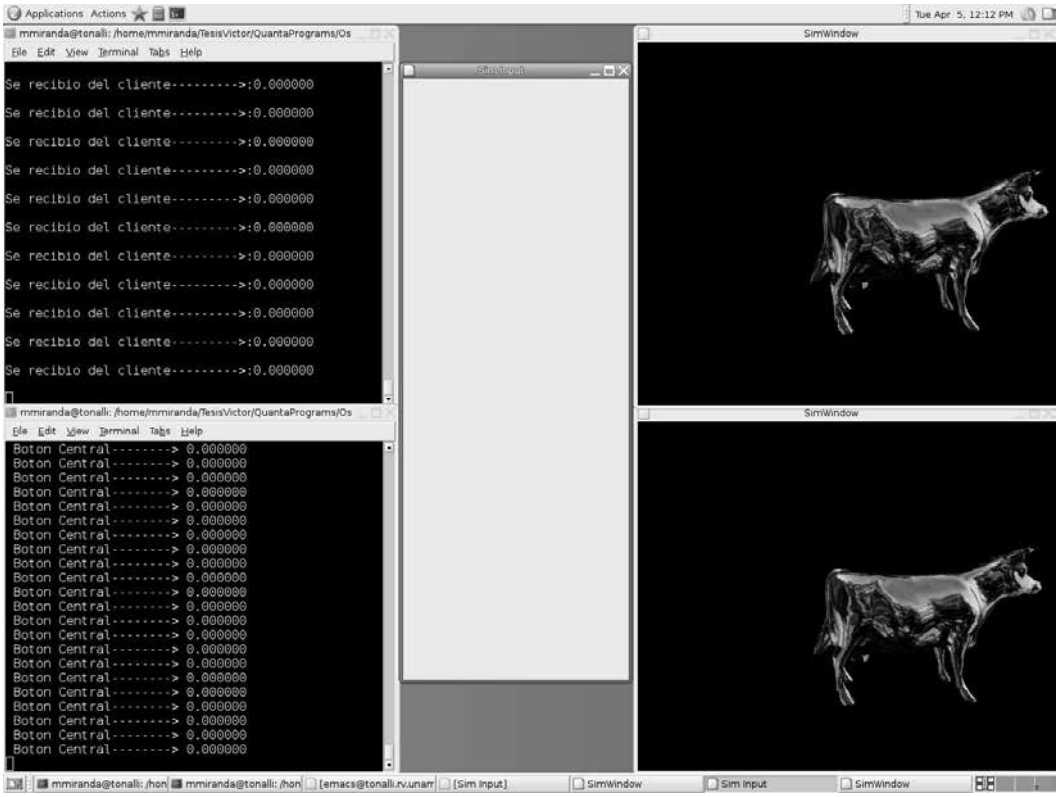


Figura 4.16
Vista de ambos programas desde la misma pantalla.

En las siguientes imágenes nos enfocaremos al funcionamiento de la aplicación, cómo es que ésta responde al presionar sobre el marco de control cada uno de los botones que se utilizan del mouse al mandarse datos del cliente, siendo recibidos por el servidor y señalizando que han sido recibidos correctamente.

El programa fue desarrollado para que además de ver el funcionamiento por el movimiento aparente de la imagen, pudieran verse en pantalla los datos que son enviados desde el cliente hasta el servidor, simultáneamente de que en el cliente se mostrará un mensaje del botón que ha sido presionado.

Así, cada uno de los datos que obtiene el programa cliente del mouse, es mostrado en la pantalla local, este dato es enviado en tiempo real, por lo que al recibir este dato el programa servidor, también lo mostrará en pantalla. Cabe mencionar que hasta que el dato enviado por el cliente haya llegado a su destino, no se enviará otro dato.

Existe la posibilidad de Latencia, esto puede ocurrir ya que si el cliente envía excesivos datos al servidor, este requiere de una fracción de segundo de mas tiempo para interpretarlos y desplegarlos, al ser manipulada la aplicación desde el programa cliente, el que sufre obviamente el retraso es el servidor.

En la siguiente imagen se muestra la acción de presionar el botón central del mouse, este botón manda el valor de cero, por lo que el modelo permanece estático.

Figura 4.17
Del cliente (terminal inferior) se envía el valor del botón central que es cero, este valor es enviado al servidor (terminal superior) y desplegado en pantalla.

Después se muestran los valores negativos resultantes de presionar el botón derecho del mouse. Con esta acción el modelo se trasladará a la parte trasera del espacio.

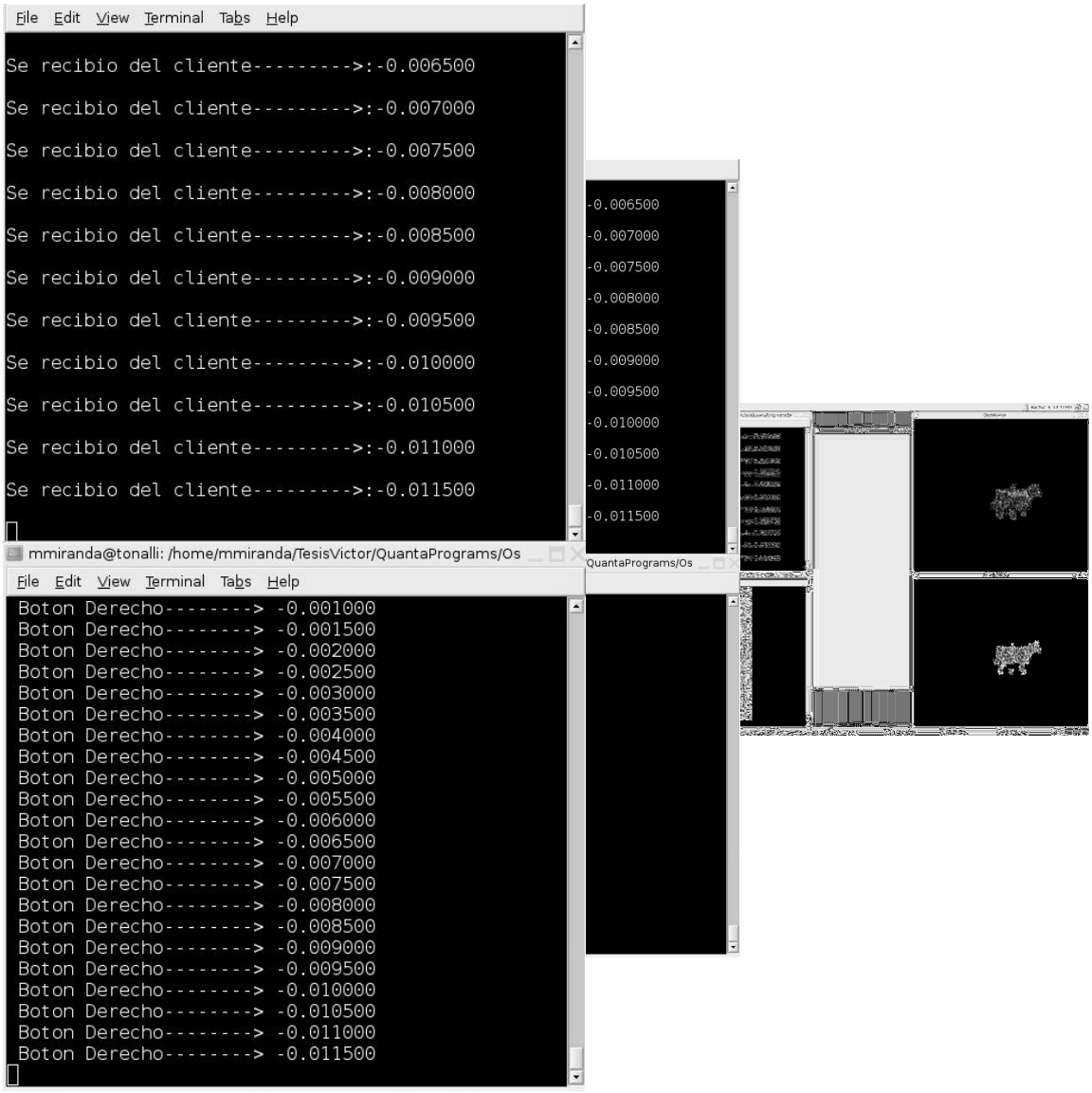


Figura 4.18

Del cliente se envía el valor del botón derecho, que son valores negativos, estos valores son enviados al servidor, mostrados en pantalla e interpretados sobre el modelo.

El último movimiento que realiza ésta aplicación es el de desplazarse hacia adelante, esto se logra enviando valores positivos desde el cliente, presionando el botón izquierdo del mouse.

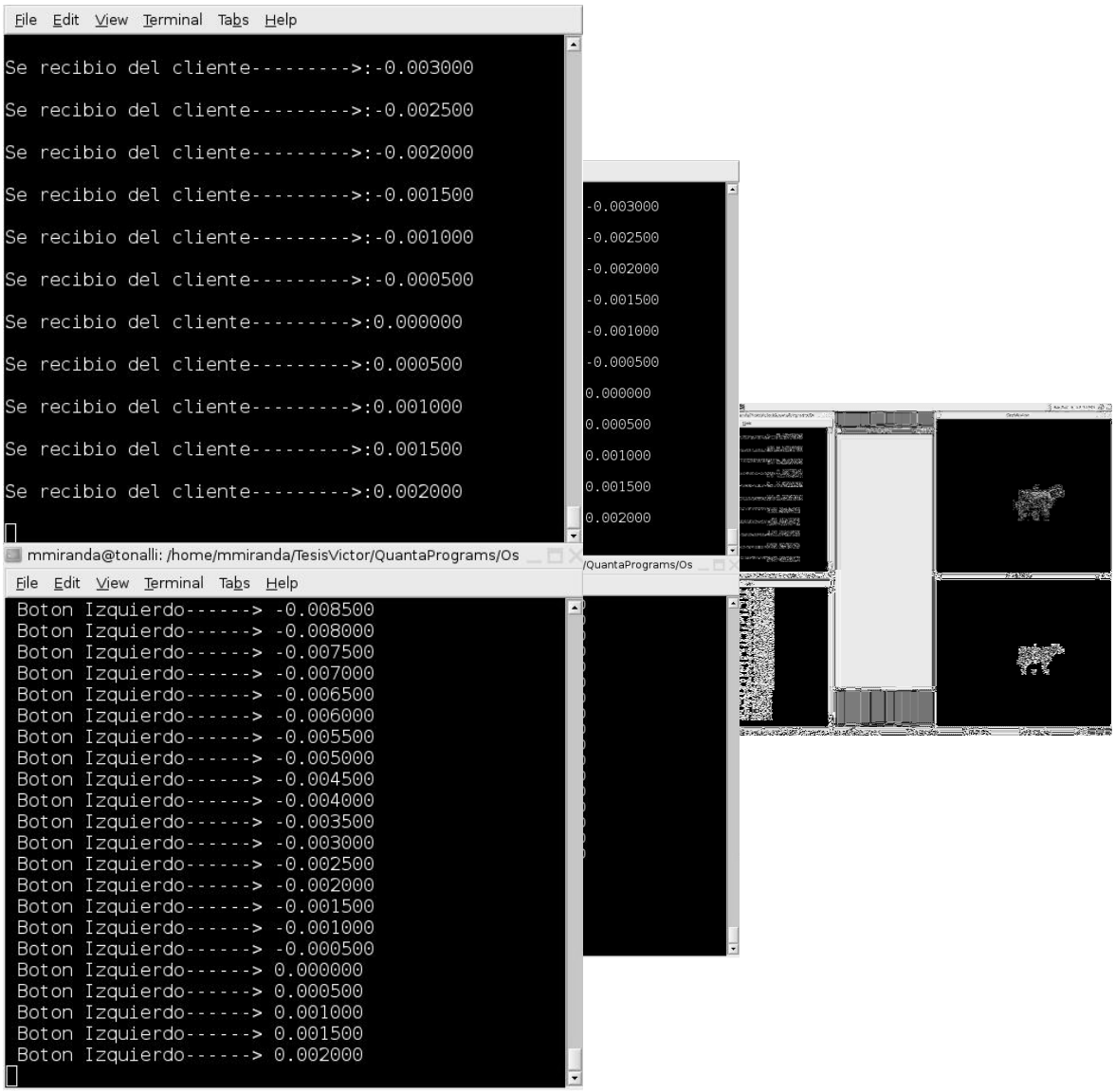


Figura 4.19
Ahora es enviado el valor del botón izquierdo que es positivo, la imagen se desplazará hacia adelante.

En los tres movimientos del modelo en el espacio, podemos darnos cuenta de que es exactamente el mismo valor que se envía que el que se recibe, se puede apreciar que los valores son los mismos por lo que podemos decir que la aplicación se apega de una manera estricta a alcanzar lo que se busca en toda aplicación de Realidad Virtual Distribuida, actuar en tiempo real.

4.4.6 *Descripción de datos de prueba*

Se realizaron diferentes pruebas para comprobar la funcionalidad de la aplicación.

Al principio el código era más extenso, lo que producía una gran latencia ya que se introdujeron algunas funciones de C++ que hacían la transferencia y recepción de datos lenta.

Entonces se tomó la decisión de indagar más a fondo acerca de la funcionalidad de algunos parámetros de las funciones de Quanta, las cuales utilizadas. Esto nos llevó a optimizar aún más el código por la supresión de código de C++.

La latencia fue reducida en gran parte, pero no por esto desapareció totalmente. Al realizar varias pruebas se descubrió que efectivamente si hay Latencia.

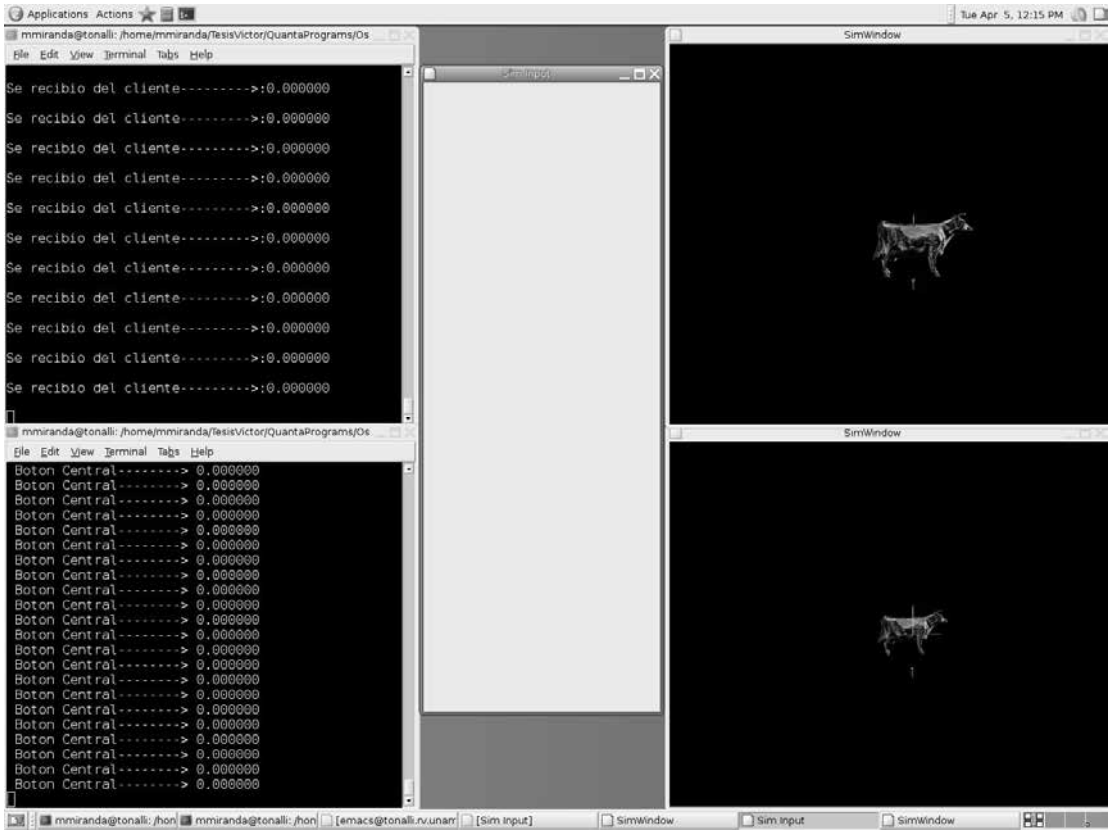


Figura 4.20
En esta figura se puede apreciar que las imágenes están desfasadas después del envío de una cantidad considerable de datos a una gran velocidad.

Aunque la Latencia es mínima y casi invisible, ésta se percibió cuando se realizaron pruebas de envío de datos a gran velocidad y sin dejar de presionar algún botón del mouse.

Conclusiones

No obstante de que la Realidad Virtual es un tema aún inalcanzable para muchos, en algunos países como Japón se realizan grandes inversiones de dinero para el desarrollo y mejoramiento de sistemas de Realidad Virtual Distribuida.

Técnicamente ya es posible crear simulaciones virtuales distribuidas muy realistas de cualquier entorno físico, pero el costo de esta tecnología es muy grande, por lo que se limita solamente a ser poseída por grandes industrias, como lo son la milicia, el área médica, la investigación y en una parte aún pequeña, al entretenimiento. Es por esta razón que tendremos que esperar un poco más para que esta tecnología sea de uso común entre cualquier persona.

Los esfuerzos por crear Realidad Virtual no son, como alguna vez lo fueron, basados totalmente en simulaciones locales, en la actualidad eso es solo una parte, ya que un objetivo muy importante es crear Realidad Virtual a través de redes.

La escalabilidad de estos sistemas, sigue siendo un área principal de exploración, por lo que cada nuevo esfuerzo por desarrollar una innovación no se queda allí, los trabajos son retomados y rediseñados para agregar mejoras que pueden revolucionar paso a paso el futuro de la Realidad Virtual Distribuida.

Basándonos en la investigación realizada, la aplicación fue desarrollada con el fin de demostrar el uso de una de las herramientas que fueron analizadas, combinando los resultados obtenidos de la investigación y la utilización del toolkit Quanta. Ésta fue desarrollada de manera ordenada y legible, por lo que se estableció que la aplicación está en disposición de ser ampliada, modificada o mejorada.

Estamos en el inicio de una nueva era en la que experimentaremos la Realidad Virtual Distribuida, lo que alguna vez parecía sacado de libros de ciencia ficción es ahora una realidad, y hay más por dessarrollar. Muchas veces lo único que esperan los científicos, investigadores y desarrolladores es que sea creada la siguiente generación de hardware para gráficos y redes, o el siguiente milenio de dispositivos hápticos para comenzar a buscar la manera de crear nuevos y mejores sistemas de Realidad Virtual.

El programa realizado no solamente se queda como una simple aplicación demostrativa ya que puede además ser considerada como una herramienta, con la que se lograron combinar diferentes elementos y se generó un

programa estándar que puede trabajar distribuidamente de manera sencilla y descifrable, por lo que es posible que sea utilizado posteriormente para comunicar aplicaciones mucho más robustas y funcionales.

Personalmente me siento muy satisfecho del esfuerzo hecho en la elaboración de este proyecto de tesis. Cualquiera podría afirmar que lo más complicado fue la programación, y lo fué en su momento, pero la investigación fue el punto crucial en el desarrollo de este trabajo, ya que casi todas las fuentes fidedignas de información con respecto a los temas aquí tratados estaban en otros idiomas, principalmente en Inglés.

Con este trabajo deseo hacer una aportación útil para todas las personas que se interesen en tratar estos temas de Realidad Virtual, pero sobre todo, espero que este documento contribuya al conocimiento general de estudiosos de mi escuela, la Universidad Nacional Autónoma de México y de mi país.

Anexo A
Código fuente del Programa Servidor

Código fuente Principal del Servidor, programa ".cpp"

```
/*****************************************/
/*              OsgNav Servidor ".cpp"              */
/*****************************************/

#include <vrj/vrjConfig.h>
#include "OsgNav.h"
#include <gmtl/Vec.h>
#include <gmtl/Coord.h>
#include <gmtl/Xforms.h>
#include <gmtl/Math.h>

/*****************Declaracion de variables**********************/
int argc;
int status, dataSize;
int iCiclo = 1;
int iContaPara = 0;
int iExit = 1, h;
int numberOfClients = 0;
float fParametro, f;
char clientData[MAX_NUM_CLIENTS][TRANSMITTED_DATA_SIZE];
char sendBuffer[TRANSMITTED_DATA_SIZE];
char receiveBuffer[BROADCAST_DATA_SIZE];
char *argv[100];
bool bandera1;
bool bandera2;

/**************Funcion de Inicializacion de clientes***************/
void InitClientData()
{
        int i, j;
        for (i = 0; i < MAX_NUM_CLIENTS; i++)
        {
                for (j = 0; j < TRANSMITTED_DATA_SIZE; j++)
                {
                        clientData[i][j] = '\0';
                }
        }
}
```

```
/**********Funcion representativa del programa en general**********/

void OsgNav::preFrame()
{
   int i = 0;

/*---Con este codigo se obtiene la informacion del dipositivo externo, del WAND---*/

  gmtl::Matrix44f wandMatrix = mWand->getData();
  osg::Matrix osgWandMat;
  osgWandMat.set(wandMatrix.getData());

/*-------------------------------------------------------------------------------------------------*/

// Se da entrada a este if si el servidor se pudo levantar de manera existosa

if(server != 0)
{
     // Ya estando arriba el servidor revisa si existe alguna nueva conexion
     aClient = server->checkForNewConnections();

     // Esta es la inicializacion del Cliente, si detecta un cliente entrara al ciclo if
     if(aClient)
     {
          if(bandera2 == false)
          {
             printf("\n\n\n\n\n\n\n\n");
            printf("\n+------------------------------------------------------------------------+");
             printf("\n+->CONEXION CON CLIENTE ESTABLECIDA...!!!<-------+");
             printf("\n+------------------------------------------------------------------------+");
             printf("\n\n\n\n\n\n\n\n");
             bandera2 = true;
          }

        // Asignamos a la bandera1 el valor de true para poder entrar al ciclo subsecuente
          bandera1 = true;
          client[numberOfClients] = aClient;
          numberOfClients++;
     }

     // Ya que existe el cliente y la bandera es true entra a este if que sera el que se
     //encargue de realizar la recepcion de los datos

     if(numberOfClients && bandera1)
     {
        // Este ciclo for nos sirve para poder seguir enviando los datos siempre y cuando
        //siga existiendo informacion que enviar
```

```
            for (i = 0; i < numberOfClients; i++)
            {
               dataSize = TRANSMITTED_DATA_SIZE;
               status = client[i]->read(dataBuffers[i], &dataSize,
              QUANTAnet_tcpClient_c::NON_BLOCKING);

               // Si se da la condicion de que se recibe un dato, el valor de status es OK,
               //por lo que entra al if
               if(status == QUANTAnet_tcpClient_c::OK)
               {
                     printf("Se recibio del cliente--------->:%s\n", dataBuffers[i]);

                     // Detecta el dato enviado y continua con el programa
                     strcpy(clientData[i], dataBuffers[i]);
                     dataSize = BROADCAST_DATA_SIZE;
                     fParametro = atof(clientData[i]);

               }

               // Si el status fuera diferente a OK lo unico que haria es seguir mandado el
               //mismo dato que se almaceno en fParametro si es que alguna vez se recibio tal,
               //de no ser asi se queda por default el valor de 0
               else
               {
 /*---------------------------------Navegacion-----------------------------------------------*/
                     gmtl::Vec3f direction;
                     gmtl::Vec3f Zdir = gmtl::Vec3f(0.0f, 0.0f, fParametro);
                     gmtl::xform(direction, wandMatrix, Zdir);
                     mNavTrans->preMult(osg::Matrix::translate(direction[0], direction[1],
                     direction[2]));
/*-------------------------------------------------------------------------------------------------*/
               }
            }
      }
      else
      {
            printf("\n---------->NO SE HA ESTABLECIDO CONEXION CON UN
            CLIENTE...!!!\n");
      }
   }
}
```

/*Asigna color al buffer de almacenamiento, al espacio en el cual esta contenido el

```
modelo*/

void OsgNav::bufferPreDraw()
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
}

// En esta funcion se cargan todos los elementos de inicio para que el programa
//funcione solo se cargan al principio para toda la ejecucion del programa.
void OsgNav::myInit()
{
   QUANTAinit();
   InitClientData();
   server = new QUANTAnet_tcpServer_c;

   // Ponemos el tamanio de los sockets para escritura y lectura
        server->setSockOptions(QUANTAnet_tcpServer_c::READ_BUFFER_SIZE,
        SOCK_BUF_SIZE);
        server->setSockOptions(QUANTAnet_tcpServer_c::WRITE_BUFFER_SIZE,
        SOCK_BUF_SIZE);

   // VR UNAM Vamos a usar este puerto en Realidad Virtual.
   if (server->init(4500) == 0)
   {
        printf(".-.-.-.-.-.-.-.-.-Error-.-.-.-.-.-.-.-.-.-\n");
   }
   else
   {
        printf("\n\n");
        printf("\n++++++++++++++++++++++++++++++++++++++++++++++++++");
        printf("\n+                                                +");
        printf("\n+---------------------------------->El servidor esta levantado  +");
        printf("\n+                                                +");
        printf("\n++++++++++++++++++++++++++++++++++++++++++++++++++");
        printf("\n\n");
        QUANTAsleep(1);
   }
   numberOfClients=0;

// El nivel mas alto de niveles del arbol de donde naceran las imágenes posteriores

  mRootNode = new osg::Group();
  mNoNav    = new osg::Group();
  mNavTrans = new osg::MatrixTransform();
  mRootNode->addChild( mNoNav );
  mRootNode->addChild( mNavTrans );
```

```
//Carga el modelo
  std::cout << "Tratando de cargar el archivo: " << mFileToLoad << "... ";
  mModel = osgDB::readNodeFile(mFileToLoad);
  std::cout << "OK...!" << std::endl;

// Transforma el nodo para el modelo
  mModelTrans  = new osg::MatrixTransform();


//Aqui podemos cambiar la orientacion del modelo
  mModelTrans->preMult( osg::Matrix::rotate( gmtl::Math::deg2Rad(-90.0f ), 1.0f, 0.0f,
  0.0f) );

// Agrega el modelo a transformar
  mModelTrans->addChild(mModel);

// Agrega la transformacion al arbol
  mNavTrans->addChild( mModelTrans );

// Corre la optimizacion sobre el grafo de escena
  osgUtil::Optimizer optimzer;
  optimzer.optimize(mRootNode);

// La velocidad inicial para la navegacion es CERO
  speed = 0.0f;

// Que tanto se acelera cada frame en cuanto al movimiento se refiere
  inc = 0.0005f;
}
```

Código fuente del Servidor, programa de cabecera ".h"

```
/*****************************************/
/*              OsgNav Servidor ".h"             */
/*****************************************/

#ifndef _OSG_NAV_
#define _OSG_NAV_
#include <QUANTA/QUANTA.hxx>
#include <stdlib.h>
#ifdef WIN32
#include <string.h>
#else
#include <strings.h>
#endif
#include <vrj/vrjConfig.h>
#include <iostream>
#include <iomanip>
#include <vrj/Draw/OGL/GlApp.h>
#include <gadget/Type/PositionInterface.h>
#include <gadget/Type/AnalogInterface.h>
#include <gadget/Type/DigitalInterface.h>

//OSG  includes
#include <osg/Math>
#include <osg/Geode>
#include <osg/Material>
#include <osg/Vec3>
#include <osg/Matrix>
#include <osg/Transform>
#include <osg/MatrixTransform>
#include <osgUtil/SceneView>
#include <osgUtil/Optimizer>
#include <osgDB/ReadFile>
#include <math.h>
#include <vrj/Draw/OSG/OsgApp.h>
#include <stdio.h>

#define BROADCAST_DATA_SIZE 1024
#define TRANSMITTED_DATA_SIZE 512
#define MAX_NUM_CLIENTS 100
#define SOCK_BUF_SIZE ((10024*10024*10024))      //1024*1024*32

/* Demostración de la aplicación clase de Open Scene Graph */
class OsgNav : public vrj::OsgApp
{
public:
```

```
/*Aquí es donde se incluyen las clases de Quanta*/
  QUANTAnet_tcpServer_c *server;
  QUANTAnet_tcpClient_c *aClient;
  QUANTAnet_tcpClient_c *client[MAX_NUM_CLIENTS];

  char dataBuffers[MAX_NUM_CLIENTS][TRANSMITTED_DATA_SIZE];
  int dataSize;
  int status;
  char sendBuffer[TRANSMITTED_DATA_SIZE];
  char receiveBuffer[BROADCAST_DATA_SIZE];

OsgNav(vrj::Kernel* kern) : vrj::OsgApp(kern)        // Initialize base class
{
    mFileToLoad = std::string("");
}

virtual void initScene()
{
// Inicializacion de dispositivos

    std::string wand("VJWand");
    std::string vjhead("VJHead");
    std::string but0("VJButton0");
    std::string but1("VJButton1");
    std::string but2("VJButton2");
    std::string but3("VJButton3");
    std::string but4("VJButton4");
    std::string but5("VJButton5");

    mWand.init(wand);
    mHead.init(vjhead);
    mButton0.init(but0);
    mButton1.init(but1);
    mButton2.init(but2);
    mButton3.init(but3);
    mButton4.init(but4);
    mButton5.init(but5);
    myInit();
}
void myInit();

virtual osg::Group* getScene()
{
        return mRootNode;
}
```

```
virtual void configSceneView(osgUtil::SceneView* newSceneViewer)
{
    newSceneViewer->setDefaults();
    newSceneViewer->setBackgroundColor( osg::Vec4(0.0f, 0.0f, 0.0f, 0.0f) );
    newSceneViewer->setDrawBufferValue(GL_NONE);
    newSceneViewer->getLight()->setAmbient(osg::Vec4(0.3f,0.3f,0.3f,1.0f));
    newSceneViewer->getLight()->setDiffuse(osg::Vec4(0.9f,0.9f,0.9f,1.0f));
    newSceneViewer->getLight()->setSpecular(osg::Vec4(1.0f,1.0f,1.0f,1.0f));
}

void bufferPreDraw();

completes<BR><BR>

void setModelFileName(std::string filename)
{
    mFileToLoad = filename;
}

private:
  osg::Vec3 mPos;
  double posInc;
  osg::Group* mRootNode;
  osg::Group* mNoNav;
  osg::MatrixTransform* mNavTrans;
  osg::MatrixTransform* mModelTrans;
  osg::Node* mModel;
  float speed;
  float inc;
  std::string mFileToLoad;

public:
  gadget::PositionInterface  mWand;
  gadget::PositionInterface  mHead;
  gadget::DigitalInterface   mButton0;
  gadget::DigitalInterface   mButton1;
  gadget::DigitalInterface   mButton2;
  gadget::DigitalInterface   mButton3;
  gadget::DigitalInterface   mButton4;
  gadget::DigitalInterface   mButton5;
};

#endif
```

Anexo B
Código fuente del Programa Cliente

Código fuente Principal del Cliente, programa ".cpp"

```
/*****************************************/
/*                OsgNav Cliente ".cpp"                */
/*****************************************/

#include <vrj/vrjConfig.h>
#include "OsgNav.h"
#include <gmtl/Vec.h>
#include <gmtl/Coord.h>
#include <gmtl/Xforms.h>
#include <gmtl/Math.h>

/**********************Declaracion de variables*************************/
int status, dataSize;
int iCiclo = 1;
int iContaPara = 0;
int iExit = 1;
float fParametro;

/***************Funcion representative del programa en general*************/

void OsgNav::preFrame()
{

/*---Con este codigo se obtiene la informacion del dipositivo externo, del WAND---*/

  gmtl::Matrix44f wandMatrix = mWand->getData();
  osg::Matrix osgWandMat;
  osgWandMat.set(wandMatrix.getData());

/*-------------------------------------------------------------------------------------------------*/

// Entra a este if si el boton Izquierdo del mouse es presionado y escribe el valor en el
//buffer
  if ( mButton0->getData() == gadget::Digital::ON )
  {
     // Hacia adelante
     speed = speed + inc;
```

```
      fParametro = speed;
      printf("Boton Izquierdo--------> %1.6f\n ", fParametro);
      sprintf(sendBuffer,"%1.6f\n\n", fParametro);
      dataSize = DATOS_A_TRANSMITIR;
      client->write(sendBuffer, &dataSize, QUANTAnet_tcpClient_c::BLOCKING);
   }

// Entra a este if si el boton Central del mouse es presionado y escribe el valor en el
//buffer
    if ( mButton1->getData() == gadget::Digital::ON )
    {
      // Alto
         speed = 0.0;
         fParametro = speed;
        printf("Boton Central--------> %1.6f\n ", fParametro);
        sprintf(sendBuffer,"%1.6f\n\n", fParametro);
        dataSize = DATOS_A_TRANSMITIR;
        client->write(sendBuffer, &dataSize, QUANTAnet_tcpClient_c::BLOCKING);
    }

// Entra a este if si el boton Derecho del mouse es presionado y escribe el valor en el
//buffer
    if ( mButton2->getData() == gadget::Digital::ON )
    {
      // Hacia atras
      speed = speed - inc;
      fParametro = speed;
      printf("Boton Derecho--------> %1.6f\n ", fParametro);
      sprintf(sendBuffer,"%1.6f\n\n", fParametro);
      dataSize = DATOS_A_TRANSMITIR;
      client->write(sendBuffer, &dataSize, QUANTAnet_tcpClient_c::BLOCKING);
    }

/*-----------------------------------Navegacion-----------------------------------------------*/
   gmtl::Vec3f direction;
   gmtl::Vec3f Zdir = gmtl::Vec3f(0.0f, 0.0f, fParametro/*speed*/);
   gmtl::xform(direction, wandMatrix, Zdir);
   mNavTrans->preMult(osg::Matrix::translate(direction[0], direction[1], direction[2]));
/*--------------------------------------------------------------------------------------------*/
}

/*Asigna color al buffer de almacenamiento, al espacio en el cual esta contenido el
modelo*/
void OsgNav::bufferPreDraw()
{
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
}
// En esta funcion se cargan todos los elementos de inicio para que el programa
//funcione, solo se cargan al principio para toda la ejecucion del programa.
void OsgNav::myInit()
{
   QUANTAinit();
   client = new QUANTAnet_tcpClient_c;

   // Ponemos el tamanio de los sockets para escritura y lectura
        client->setSockOptions(QUANTAnet_tcpClient_c::READ_BUFFER_SIZE,
        SOCK_BUF_SIZE);
        client->setSockOptions(QUANTAnet_tcpClient_c::WRITE_BUFFER_SIZE,
        SOCK_BUF_SIZE);

   // VR UNAM Vamos a usar el puerto 4500 en Realidad Virtual.
   printf("\n\nVaLor:---------------->%d\n\n");
   if (client->connectToServer("localhost", 4500) < 0)
   {
        printf("\n\n-.-.-.-.-.-.-.-ERROR-.-.-.-.-.-.-.-\n\n");
   }

   else
   {
        printf("\n\n");
        printf("\n++++++++++++++++++++++++++++++++++++++++++++++++++");
        printf("\n+                                                +");
        printf("\n+---------------------------------->El cliente esta levantado  +");
        printf("\n+                                                +");
        printf("\n++++++++++++++++++++++++++++++++++++++++++++++++++");
        printf("\n\n");
        QUANTAsleep(1);
   }

// El nivel mas alto de niveles del arbol de donde naceran  las imágenes posteriores

  mRootNode = new osg::Group();
  mNoNav    = new osg::Group();
  mNavTrans = new osg::MatrixTransform();

  mRootNode->addChild( mNoNav );
  mRootNode->addChild( mNavTrans );

//Carga el modelo
  std::cout << "Tratando de cargar el archivo: " << mFileToLoad << "... ";
  mModel = osgDB::readNodeFile(mFileToLoad);
  std::cout << "OK...!" << std::endl;
```

```
// Transforma el nodo para el modelo
  mModelTrans  = new osg::MatrixTransform();

//Aqui podemos cambiar la orientacion del modelo
  mModelTrans->preMult( osg::Matrix::rotate( gmtl::Math::deg2Rad(-90.0f ), 1.0f, 0.0f,
0.0f) );

// Agrega el modelo a transformar
  mModelTrans->addChild(mModel);

// Agrega la transformacion al arbol
  mNavTrans->addChild( mModelTrans );

// Corre la optimizacion sobre el grafo de escena
  osgUtil::Optimizer optimzer;
  optimzer.optimize(mRootNode);

// La velocidad inicial para la navegacion es CERO
  speed = 0.0f;

// Que tanto se acelera cada frame mientras el boton este presionado
  inc = 0.0005f;
}
```

Código fuente del Cliente, programa de cabecera ".h"

```
/*****************************************/
/*              OsgNav Cliente ".h"      */
/*****************************************/

#ifndef _OSG_NAV_
#define _OSG_NAV_
#include <QUANTA/QUANTA.hxx>
#include <stdlib.h>
#ifdef WIN32
#include <string.h>
#else
#include <strings.h>
#endif
#include <vrj/vrjConfig.h>
#include <iostream>
#include <iomanip>
#include <vrj/Draw/OGL/GlApp.h>
#include <gadget/Type/PositionInterface.h>
#include <gadget/Type/AnalogInterface.h>
#include <gadget/Type/DigitalInterface.h>

//OSG  includes
#include <osg/Math>
#include <osg/Geode>
//#include <osg/GeoSet>
#include <osg/Material>
#include <osg/Vec3>
#include <osg/Matrix>
#include <osg/Transform>
#include <osg/MatrixTransform>
#include <osgUtil/SceneView>
#include <osgUtil/Optimizer>
#include <osgDB/ReadFile>
#include <math.h>

#include <vrj/Draw/OSG/OsgApp.h>

#define DATOS_A_TRANSMITIR 512
#define BROADCAST_DATA_SIZE 1024
#define SOCK_BUF_SIZE ((10024*10024*10024))      //1024*1024*32
```

```
/* Demostración de la aplicación clase de Open Scene Graph */
class OsgNav : public vrj::OsgApp
{
public:

/*Aquí es donde se incluyen las clases de Quanta*/
     QUANTAnet_tcpClient_c *client;

   char sendBuffer[DATOS_A_TRANSMITIR];
   char receiveBuffer[BROADCAST_DATA_SIZE];

OsgNav(vrj::Kernel* kern) : vrj::OsgApp(kern)        // Initialize base class
{
     mFileToLoad = std::string("");
}

virtual void initScene()
{

// Inicialización de dispositivos
     std::string wand("VJWand");
     std::string vjhead("VJHead");
     std::string but0("VJButton0");
     std::string but1("VJButton1");
     std::string but2("VJButton2");
     std::string but3("VJButton3");
     std::string but4("VJButton4");
     std::string but5("VJButton5");

     mWand.init(wand);
     mHead.init(vjhead);
     mButton0.init(but0);
     mButton1.init(but1);
     mButton2.init(but2);
     mButton3.init(but3);
     mButton4.init(but4);
     mButton5.init(but5);
     myInit();
   }
void myInit();


virtual osg::Group* getScene()
{
     return mRootNode;
}
```

```
virtual void configSceneView(osgUtil::SceneView* newSceneViewer)
{
    newSceneViewer->setDefaults();
    newSceneViewer->setBackgroundColor( osg::Vec4(0.0f, 0.0f, 0.0f, 0.0f) );
    newSceneViewer->setDrawBufferValue(GL_NONE);
    newSceneViewer->getLight()->setAmbient(osg::Vec4(0.3f,0.3f,0.3f,1.0f));
    newSceneViewer->getLight()->setDiffuse(osg::Vec4(0.9f,0.9f,0.9f,1.0f));
    newSceneViewer->getLight()->setSpecular(osg::Vec4(1.0f,1.0f,1.0f,1.0f));
}

void bufferPreDraw();

// Llamada una vez antes de cada frame.
virtual void preFrame();

completes<BR><BR>

void setModelFileName(std::string filename)
{
    mFileToLoad = filename;
}

private:
  osg::Vec3 mPos;
  double posInc;
  osg::Group* mRootNode;
  osg::Group* mNoNav;
  osg::MatrixTransform* mNavTrans;
  osg::MatrixTransform* mModelTrans;
  osg::Node* mModel;
  float speed;
  float inc;
  std::string mFileToLoad;

public:
  gadget::PositionInterface  mWand;
  gadget::PositionInterface  mHead;
  gadget::DigitalInterface   mButton0;
  gadget::DigitalInterface   mButton1;
  gadget::DigitalInterface   mButton2;
  gadget::DigitalInterface   mButton3;
  gadget::DigitalInterface   mButton4;
  gadget::DigitalInterface   mButton5;
};

#endif
```

Literatura Citada

- William R. Sherman & Alan B. Craig "*Understanding Virtual Reality",* Morgan Kaufmann, 2003.

-Sandeep Singhal & Michael Zyda "*Networked Virtual Enviroments*", Addison-Wesley, 1999.

-Michael R. Macedonia, Michael. J. Zyda "*A Taxonomy for Networked Virtual Enviroments* ", NPSNET Research Group, CRCG VR Center, Octubre 1995.

-Andrew E. Johnson "*CAVERN: The CAVE Research Network*", Electronic Visualization Laboratory, University of Illinois Chicago, Marzo 1998.

-Jason Leigh, Andrew E. Johnson, Thomas A. DeFanti "*CAVERN: A Distributed Architecture for Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environments*", Electronic Visualization Laboratory, University of Illinois Chicago, Julio 1997.

-Henrik Tamberend "*Avango: A Distributed Virtual Reality Framework*", German National Research Center for Information Technology.

-Chris Greenhalgh, Steve Benford "*MASSIVE: Distributed Virtual Reality System Incorporating Spatial Trading*", Department of Computer Science, The University of Notthingham.

-Patrick Hartling, Carolina Cruz-Neira "*Octopus: A Cross-Platform API for Enabling Distributed Virtual reality Applications*", Virtual Reality Application Center (VRAC), Iowa State University.

- Ken Watsen, Mike Zyda "*BAMBOO: Supporting Dynamic Protocols for Virtual Enviroments*", NPSET research Group, Naval Postgraduate School, Monterrey CA.

Mesografía

- Laboratorio de Visualización, UNAM DGSCA 2002 home page:
http://www.labvis.unam.mx/visualizacion

- Laboratorio de Realidad Virtual de EAFIT home page:
http://arcadia.eafit.edu.co/historia.html

- Pontificia Universidad Católica de Chile home page:
http://www.puc.cl/infsecic/boltec7/rvhist.html

- Dirección de Telemática home page:
http://telematica.cicese.mx/computo/super/cicese2000/realvirtual/
- Centro Nacional de Cálculo Científico home page:
http://www.cecalc.ula.ve/documentacion/tutoriales/LINDA/node1.html

- The DIVE home page:
[1]- http://www.sics.se/dive/

- OpenSG home page:
[2]- http://www.opensg.org

- MiltiGen-Paradigm home page:
[3]- http://www.multigen-paradigm.com/products/runtime/vega/index.shtml

- Alice home page:
[4]- http://www.alice.org

- Avango home page:
[5] - http://www.avango.org

- Integrated Systems Laboratory home page:
[6]- http://www.isl.uiuc.edu/Syzygy/documents/Introduction.html

- FreeVR home page:
[7]- http://www.freevr.org

- Department of Computing Science home page:
[8]- http://web.cs.ualberta.ca/~graphics/MRToolkit/MRannouncement.html

- Imperial College London web site:
[9]- http://www.doc.ic.ac.uk/~np2/dve/systems.html

- VR Juggler home page:
[10]- http://www.vrjuggler.org/