Universidad de las Ciencias Informáticas

Facultad 5

Arquitectura de red para la confección de videojuegos multijugadores

TRABAJO DE DIPLOMA EN OPCIÓN AL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

Autores:

Carlos Arcenio Rodríguez Noa Yordanis Gómez Finalé

Tutor:

Ing. Igr Alexánder Fernández Saúco

Ciudad de La Habana

Junio 2008

Declaración de Autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los __ días del mes de junio del año 2008.

_____ _____

Carlos Arcenio Rodríguez Noa

Autor

Yordanis Gómez Finalé

Autor

_____

Ing. Igr Alexánder Fernández Saúco

Tutor

Datos de Contacto

Nombre y Apellidos: Igr Alexánder Fernández Saúco
Edad: 28
Ciudadanía: Cubana
Institución: Universidad de las Ciencias Informáticas (UCI)
Título: Ingeniero Informático
Categoría Docente: Profesor Instructor
E-mail: alexanderfs@uci.cu
Jefe del grupo de Arquitectura y Tecnologías, Dirección Técnica, IP – UCI.

Opinión del Tutor del Trabajo de Diploma

Título: Arquitectura de red para la confección de videojuegos multijugadores.

Autores: Carlos Arcenio Rodríguez Noa
Yordanis Gómez Finalé

El tutor del presente Trabajo de Diploma considera que durante su ejecución los estudiantes mostraron las cualidades que a continuación se detallan.

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____.

Por todo lo anteriormente expresado considero que los estudiantes están aptos para ejercer como Ingenieros Informáticos; y propongo que se le otorgue al Trabajo de Diploma la calificación de _____.

_____ _____

Firma del Tutor Fecha

Agradecimientos

A mis padres y a mi hermana, para los cuales no tengo palabras capaces de expresar el sentimiento que en mí despiertan y por quienes daría mi vida por verlos felices, porque supieron darme todo el amor y educación del mundo, porque me dieron su apoyo sin el que tampoco lo hubiese logrado.

A Eylen por su paciencia y comprensión, por su cariño y por todo el amor y felicidad que ha traído a mi vida.

Al profesor y tutor Ing. Igr Alexánder Fernández Saúco.

A toda mi familia por seguir mis pasos y por ayudarme durante mis estudios.

A todos los miembros del proyecto Juegos Consola, que de una forma u otra contribuyeron al desarrollo de este trabajo y a mi formación como profesional.

A todos mis amigos que mucho me han enseñado y ayudado en los momentos difíciles.

Carlos.

A todas las personas que de una forma u otra han contribuido en mi formación como profesional.

A mis padres por su esfuerzo incondicional.

A todos mis compañeros por el tiempo compartido.

A Igr Alexánder Fernández Saúco por la tutoría.

A Morgan McGuire por el proyecto G3D.

Yordanis.

Dedicatoria

A la memoria de mis abuelos.

A mis padres. A mi hermana.

A mi abuela Bella.

A toda mi familia.

Carlos.

A la memoria de mi querida abuela Melba.

A mi abuelo, mis padres, mi hermano, y a mi primita Aimée.

A toda mi familia que siempre me ha apoyado en mis decisiones.

Yordanis.

"Busca la profundidad de las cosas; hasta allí nunca logra descender la ironía."

Rainer Maria Rilke

Resumen

En el mundo actual el desarrollo de videojuegos es una de las ramas más rentables de la Informática y posiblemente de toda la industria del entretenimiento. La magnitud de este fenómeno es tal, que ha provocado incluso demandas del sector cinematográfico a desarrolladores de videojuegos por pérdidas millonarias dada la ausencia de público a los cines aludiendo que éste prefiere quedarse en casa a disfrutar de las emociones que los videojuegos pueden proporcionar. En los últimos años se ha incrementado en el área de Entornos Virtuales el desarrollo de videojuegos multijugadores, este hecho ha aumentado la atracción de los usuarios, pero al mismo tiempo, la complejidad del desarrollo de este tipo de software. En cada videojuego a desarrollar es necesario que las máquinas envíen y reciban datos por la red, y en dependencia del valor de estos datos y según la lógica correspondiente al videojuego en cuestión, se ejecutarán acciones que modificarán el estado del mismo.

En el presente trabajo se hace un análisis del funcionamiento de la red en los videojuegos así como de las arquitecturas que se emplean para el desarrollo de los mismos. Además se muestra el desarrollo de un modelo de objetos flexible y modular que brinda los componentes necesarios para elaborar un videojuego con una arquitectura cliente servidor, facilitando y agilizando el trabajo de los desarrolladores, permitiéndoles de esta forma acelerar su trabajo y liberar un producto en menor tiempo.

Palabras claves

Arquitectura de red y videojuego multijugador.

Índice

Índice de Figuras

Introducción

Mucho ha sucedido desde 1958 cuando William A. Hinginbotham trabajando para el "Brookhaven National Laboratory" lograra simular con un osciloscopio un juego virtual de tenis, llamado "Tennis For Two", reconocido como el primer videojuego (*del inglés* video game) de la historia de la Informática. Desde entonces han sido múltiples los videojuegos desarrollados, así como las empresas dedicadas a esta rama. En un principio no eran muy populares, la industria del videojuego permaneció por muchos años en la sombra debido a la industria cinematográfica, las películas y la televisión, alcanzando bajos ingresos.

En los primeros años de la década de los 90 todo comienza a cambiar. El rápido crecimiento del poder de procesamiento de las computadoras comienza a permitir a los desarrolladores de videojuegos incorporar gráficos más reales, así como efectos de sonido. Ya en años anteriores, debido a que era más complicado hacer que la computadora por sí diera un nivel de juego aceptable se habían creado los llamados videojuegos multijugador (del inglés mutiplayer), que no son más que aquellos juegos en los que participan diversos jugadores, ya sea en una misma computadora o consola, o a través de Internet u otro tipo de red. Todos estos factores, unidos con una Internet en sus inicios, hacen posible el juego en red conectando a personas en diversos países, convirtiendo a la industria del juego en una de las más rentables dentro del mercado del entretenimiento. Juegos como Doom3, Half Life2, Grand Theft Auto: San Andreas, y Halo, fueron exitazos en el año 2004, alcanzando ingresos para la industria del juego de $9.9 billones de dólares [1].

Cuba, centrando su esfuerzo en el desarrollo de la industria del software, como una de las principales tareas de la Batalla de Ideas ha comenzado a dar sus primeros pasos en este campo. Con el objetivo de insertar a Cuba en el mercado del software a nivel mundial y para la informatización del país se crea en el 2002 la Universidad de las Ciencias Informáticas (UCI), la cual desde sus inicios tiene como objetivo revolucionar la industria del software en Cuba, alcanzando hoy día notables logros en el ámbito internacional.

La UCI, en aras de convertir la Informática como una de las ramas más productivas de la nación cubana y asumir con orgullo y placer el reto de informatización de la sociedad cubana, se ha convertido en un centro científico donde se vincula la docencia con el proceso productivo en el desarrollo de varios perfiles de la Informática. Enmarcado en estas ramas de producción surge el perfil

de Entornos Virtuales, dentro del cual se encuentra el proyecto Juegos Consola, con el objetivo de desarrollar videojuegos en tres dimensiones (3D) para Computadoras Personales (*del inglés* Personal Computer) y videoconsolas.

Situación problemática

Como ya se ha comentado en los últimos años se ha incrementado en el área de Entornos Virtuales el desarrollo de videojuegos multijugadores, este hecho ha aumentado la atracción de los usuarios, pero al mismo tiempo, la complejidad del desarrollo de este tipo de software. En cada videojuego a desarrollar es necesario que las máquinas envíen y reciban datos por la red y en dependencia del valor de estos datos y según la lógica correspondiente al videojuego en cuestión, se ejecutarán acciones que modificarán el estado del mismo. En el proyecto Juegos Consola de la facultad 5 de la UCI no existen componentes para crear videojuegos con arquitectura de red que permitan la comunicación entre nodos, independientemente de la localización de los mismos. Componentes que puedan ser reutilizados en el desarrollo de otros videojuegos facilitando la elaboración de los mismos y por lo tanto la disminución del tiempo de desarrollo y del costo de producción para satisfacer al cliente en tiempo y con un producto de calidad.

Problema científico

¿Cómo lograr la comunicación y sincronización entre los nodos de las aplicaciones videojuegos multijugadores?

Objeto de estudio

Transmisión de datos en arquitecturas red.

Campo de Acción

Transmisión de datos en arquitecturas cliente servidor para el desarrollo de videojuegos.

Objetivo

Implementar una arquitectura de red cliente servidor para la transmisión de datos en videojuegos multijugadores.

Hipótesis de la investigación

Con la creación y uso de un módulo que represente una arquitectura genérica cliente servidor se logra la comunicación y sincronización entre los nodos de las aplicaciones videojuegos multijugador.

Tareas

- Analizar las bases teóricas de la arquitectura cliente servidor.
- Analizar diferentes algoritmos que implementen protocolos de transmisión de datos en video juegos multijugadores.
- Diseñar el modelo de objetos.
- Codificar el modelo anterior.
- Probar mediante una aplicación demostrativa el modelo de objetos creado.

Métodos científicos

- Teóricos

Analítico sintético: Se analizan las teorías, documentos entre otros, permitiendo la extracción de los elementos más importantes que se relacionan con el objeto de estudio.

Inductivo deductivo: Formas de razonamiento que permiten llegar a un grupo de conocimientos generalizadores, tanto desde el análisis de lo particular a lo general, como desde el análisis de los elementos generalizadores a uno de menor nivel de generalización.

- Empíricos

Observación: Registro visual de lo que ocurre en una situación real, en un fenómeno determinado, clasificando y consignando los hechos y acontecimientos pertinentes de acuerdo con algún esquema previsto.

Organización del documento: El documento se encuentra organizado del siguiente modo:

El capítulo 1. Fundamentación teórica: Realiza una introducción a los videojuegos y las arquitecturas de comunicación con que son desarrollados. Además se analizan las técnicas de transmisión, protocolos y la tecnología empleada en el desarrollo de este trabajo.

El capítulo 2. Características de la Arquitectura: Define la arquitectura de comunicación a desarrollar y se analiza el funcionamiento que ésta debe tener a través del modelo de dominio, requerimientos y casos de uso.

El capítulo 3. Diseño e Implementación de la Arquitectura: Muestra el diseño del módulo desarrollado, a través de los diagramas de clases por paquetes y la interacción entre las clases. Además se especifican los patrones aplicados en la solución y se muestran los diagramas de colaboración, despliegue y componentes.

El capítulo 4. Resultados: Se presentan los resultados que se obtuvieron con el desarrollo de la arquitectura al elaborar una aplicación demostrativa que prueba todas las funcionalidades de la misma. Además se recogen las estadísticas de prueba para evaluar el funcionamiento del módulo.

Capítulo 1: Fundamentación Teórica

En el presente capítulo se tratan temas respecto a los videojuegos y las arquitecturas de comunicación con que son desarrollados. Detallando la estructura que debe tener el funcionamiento de la red en los videojuegos y cómo se comportan en estos las características como latencia o ancho de banda. Además se analizan las técnicas de transmisión, los protocolos y la tecnología empleada en el desarrollo de este trabajo.

1.1 Videojuegos

Un videojuego es un programa informático, creado en un principio para el entretenimiento, basado en la interacción entre una o varias personas y un aparato electrónico que ejecuta el videojuego. Los videojuegos recrean entornos y situaciones virtuales en los cuales el jugador puede controlar a uno o varios personajes (o cualquier otro elemento de dicho entorno), para conseguir uno o varios objetivos por medio de unas reglas determinadas. Pueden ser ejecutados en equipos electrónicos de uso general, como las computadoras o los teléfonos celulares, en equipos de uso específico para este fin (videoconsolas) o en equipos no diseñados o pensados para jugar pero que disponen de un display y controles para interactuar con el videojuego.

Con el desarrollo tecnológico actual y la evolución de los juegos, se han hecho muy populares y demandados los juegos en red, por lo que cada vez es más complejo el diseño de los mismos. Muchos son los elementos que hay que tener en cuenta antes de desarrollar un juego en red, es por eso que se deben tener claras las características de la red como latencia o ancho de banda, así como las arquitecturas empleadas a la hora de desarrollar el juego, entre otros factores. Un dominio total de estos elementos será reflejado en el producto final, al lograr un juego coherente y realista.

1.2 Arquitectura de red

Las computadoras se comunican por medio de redes. La red más sencilla es una conexión directa entre dos computadoras. Sin embargo, también pueden conectarse a través de grandes redes que permiten a los usuarios intercambiar datos, comunicarse mediante correo electrónico y compartir recursos.

En general la arquitectura o topología de red no es más que la disposición física en la que se conectan los nodos de una red de ordenadores o servidores, mediante la combinación de estándares y protocolos. Sin embargo en el contexto del desarrollo de videojuegos, el término arquitectura de red (o de comunicación) se refiere a la forma en que se implementa, a nivel lógico y de aplicación, el software que gestiona la comunicación entre los nodos (las computadoras en la red). Ésta es muy importante, pues la arquitectura de comunicación define en parte el ancho de banda a consumir, así como también la complejidad que va a tener el desarrollo del juego para lograr una coherencia entre los nodos terminales conectados y el tiempo de reacción, factores importantes para lograr resultados satisfactorios.

1.3 Capas de comunicación

Todos los videojuegos en red deben ser capaces de administrar bien los recursos de red, reponerse a fallas en la conexión y mantener una coincidencia simultánea entre todos los nodos, de manera que no existan diferencias entre los jugadores. Para esto se estructura el funcionamiento de la red del juego en 3 capas de comunicación lo que permite la abstracción de los componentes por funcionalidad.

1-Capa Física.

2-Capa Lógica.

3-Capa de Aplicación.

Cada uno de estos niveles está relacionado con las operaciones de los datos, ya sea transmisión, procesamiento o almacenamiento (Tabla 1.1). En la Capa Física normalmente no se cambia nada, es la que introduce las limitaciones de recursos como ancho de banda o latencia. La Capa Lógica es propuesta para los diseñadores del sistema ya que ésta provee las abstracciones necesarias de los lenguajes de programación para el trabajo con las entidades de datos y los canales de comunicación. La Capa de Aplicación añade la interpretación de los datos y junto con esto el control de los estados y el flujo de datos de los nodos, ya sea controlando la sincronización o solucionando situaciones de bloqueo mutuo (*del inglés* deadlock) [8].

Tabla 1.1 Capas de Comunicación con respecto a los datos

	Operaciones Sobre los Datos		
Capa	**Almacenar**	**Procesar**	**Transferencia**
Física	Memoria	Procesador	Red
Lógica	Datos de las Entidades	Procesos de Control	Canales de Comunicación
Aplicación	Estados	Control de la Integridad	Soporte Multi-Source

1.3.1 Capa Física

Esta capa es la de más bajo nivel, pues está relacionada con el cableado y el hardware. En la misma están presentes las limitaciones de los recursos físicos, que influyen en el diseño de la aplicación. Además una vez establecida la conexión entre los nodos se necesitan técnicas para la transmisión de información de un nodo a otro, así como los protocolos de comunicación que definen la forma de

transmisión para que los nodos puedan comunicarse y la información sea entregada de forma correcta [8].

1.3.1.1 Limitaciones de Recursos

El ancho de banda es la capacidad de transmisión de la comunicación, esto es, el promedio de la cantidad de datos transmitidos o recibidos por unidad de tiempo. Este factor depende de la longitud de los mensajes que son transmitidos, del número y distribución de los nodos, así como también, de la técnica de transmisión usada. Desde luego es conveniente tener un elevado ancho de banda como recurso, pero es conveniente también que las aplicaciones consuman la menor cantidad de ancho de banda posible.

La latencia (o retardo) es el intervalo de tiempo que demora la transmisión de un mensaje desde un nodo a otro, a la variación de este parámetro en el tiempo se le conoce como jitter (inestabilidad), el cual es otro factor que afecta a las aplicaciones de redes. Es favorable una latencia baja y estable (el ser estable implica un jitter bajo), pero ésta no puede ser eliminada totalmente.

Aunque la latencia y el ancho de banda no están necesariamente relacionados, en general las redes de área amplia (WAN) suelen tener bajo ancho de banda y alta latencia, sin embargo en las redes de área local (LAN) esto se comporta a la inversa, pues las mismas poseen un elevado ancho de banda y baja latencia. Para sistemas interactivos en tiempo real como los videojuegos es recomendable una latencia menor de 100 ms [2]. Según lo mencionado anteriormente la latencia afecta el rendimiento, un control continúo y fluido es posible cuando la latencia no excede los 200 ms, pero el umbral de cuándo la latencia es inconveniente para el usuario depende del tipo de videojuego. En los juegos de estrategia en tiempo real puede ser aceptable una alta latencia, incluso hasta 500 ms mientras permanezca estable la imagen, lo cual significa tener un valor de jitter bajo, por otro lado los juegos que requieren un ajustado control de la imagen y del tiempo de reacción como los juegos en primera persona demandan una latencia cercana a los 100ms [7].

1.3.1.2 Técnicas de Transmisión y Protocolos.

Las técnicas para la transmisión de mensajes pueden ser divididas en tres tipos:

- Unicasting: La comunicación se establece entre dos nodos (uno emisor y el otro receptor); esta técnica permite controlar y direccionar el tráfico de punto a punto. Si el mismo mensaje es deseado por varios receptores, la transmisión unicasting derrocha ancho de banda enviando mensajes repetitivos.

- Multicasting: La comunicación se establece entre un emisor y múltiples receptores subscriptos a grupos específicos. Cuando el emisor envía un mensaje, es recibido sólo por los receptores interesados (los que pertenezcan al grupo al cual se envía el mensaje), sin necesidad de duplicarlo, por lo que ésta es una técnica eficiente para transmitir información a un gran número de nodos.

- Broadcasting: La comunicación se establece entre un emisor y todos los restantes nodos de la red, por lo que cada nodo tiene que recibir y procesar cada mensaje emitido. Esto provoca que en redes con grandes números de nodos no se garantice el broadcasting (como es el caso de las redes WAN) [8].

Unido a estas técnicas de transmisión se encuentran los protocolos**.** Un protocolo no es más que el conjunto de reglas que dos aplicaciones pueden seguir para poder comunicarse. El Protocolo de Internet (*del inglés* Internet Protocol (IP)), comprende los protocolos de bajo nivel que guían los mensajes desde su origen hasta su destino ocultando la actual dirección de transmisión. Las aplicaciones de red raramente usan el protocolo IP directamente, sino que usan los protocolos que están por encima de IP. Los protocolos más comunes usados por las aplicaciones son Transmission Control Protocol (TCP/IP) y User Datagram Protocol (UDP/IP).

- TCP/IP provee una conexión confiable punto a punto dividiendo la información a transmitir en paquetes de red. Para extraer los datos el receptor ordena los paquetes en el orden correcto para verificar que todos los datos son recibidos en el mismo orden en que fueron enviados, descarta paquetes duplicados y demanda al emisor reenviar los paquetes perdidos o dañados,

pero en consecuencia, requiere de un mayor esfuerzo computacional y de paquetes más grandes.

- UDP/IP es menos confiable, pues no garantiza que todos los paquetes de red en que fue dividida la información lleguen a su destino ni en el orden correcto, pero esto tiene la ventaja de que la transmisión sea más rápida y además los datos son más fáciles de procesar, pues no llevan información adicional como en el caso de TCP/IP que incorpora cierta información para el chequeo de errores, lo cual permite al protocolo UDP/IP ser más apropiado en multicasting y en broadcasting [8].

1.3.2 Capa Lógica

La Capa Lógica es construida en un nivel superior a la Capa Física, la misma provee los componentes necesarios para la creación de arquitecturas para la comunicación, transmisión y el control de datos a través de canales de comunicación. Mientras que la Capa Física observa la red como nodos que están conectados físicamente, la Capa Lógica define cómo fluyen los mensajes en la red. A continuación son definidas cada una de las principales arquitecturas involucradas en esta capa.

1.3.2.1 Arquitecturas de Comunicación

Las arquitecturas de comunicación pueden ser elegidas a partir de diferentes modelos las cuales pueden ser organizadas de acuerdo a su grado de despliegue (Figura 1.1). En un grafo de comunicación los nodos representan los procesos ejecutados en computadoras remotas y las aristas indican que los nodos pueden intercambiar mensajes [8].

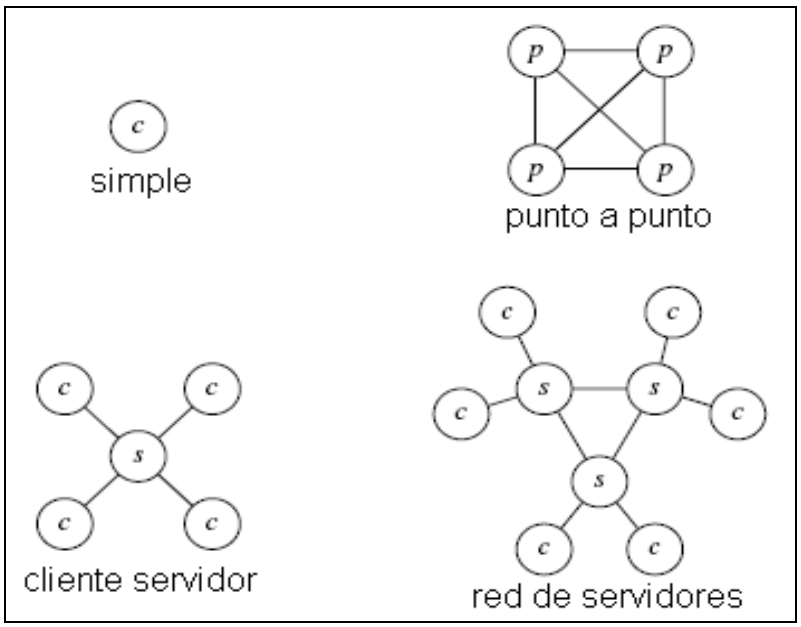


Figura 1.1 Arquitecturas de Comunicación según el Grado de Despliegue

El modelo más sencillo es conocido como simple (*del inglés* single node), en el cual realmente no es usada la red, pues la aplicación se desarrolla en un solo nodo (por ejemplo en un juego en el cual dos jugadores juegan en la misma computadora donde la pantalla se divide en dos partes lógicas).

La arquitectura punto a punto (*del inglés* peer to peer) está compuesta por un conjunto de nodos, de los cuales cada uno está conectado a los demás, no hay nodos intermediarios y cada nodo puede transmitir mensajes a todos los demás (esto provoca menos latencia). Sin embargo no es escalable pues carece de una estructura jerárquica, por lo que sólo es útil cuando el número de participantes es pequeño o se comunican a través de una red LAN.

En la arquitectura cliente servidor (*del inglés* client server) uno de los nodos es designado como servidor y el resto como clientes. Toda la comunicación es administrada por el servidor, ningún cliente puede enviar un mensaje directamente a otro cliente, sino que lo hace a través del servidor. Aunque el servidor retarda el envío de paquetes, se obtienen beneficios pues es posible controlar el flujo de paquetes y de esta forma no se tiene que enviar necesariamente todos los paquetes a todos los nodos.

Una arquitectura red de servidores (*del inglés* server network) está compuesta por un conjunto de servidores interconectados, donde cada servidor está conectado además a un conjunto de nodos clientes. Esto se podría entender como un conjunto de subredes cliente servidor de las cuales sus servidores están conectados punto a punto entre ellos. La arquitectura red de servidores reduce los requerimientos de capacidad impuestos a un servidor. En consecuencia provee mejor escalabilidad pero aumenta la complejidad de manipular el tráfico de la red [8].

1.3.2.2 Arquitecturas de Control y Datos

La arquitectura de control y datos se refiere a la distribución física de los datos, donde éstos son controlados. Existen tres modelos para definir estas arquitecturas:

- La arquitectura centralizada donde sólo un nodo controla toda la información.
- La arquitectura de réplica donde existe una copia de los datos en cada nodo.
- La arquitectura distribuida donde cada nodo manipula un subconjunto de los datos.

Dos atributos importantes definen a estos modelos de arquitectura de control y datos: la consistencia (*del inglés* consistency) y la sensibilidad o grado de reacción (*del inglés* responsiveness). La consistencia es el grado de exactitud que existe en los procesos que son ejecutados en cada nodo y la sensibilidad (o grado de reacción) es cuán rápido son respondidas las peticiones de cada nodo. Por lo tanto es conveniente tener el mayor valor posible para cada uno de estos atributos, pero ninguna arquitectura es capaz de poseer el mayor valor para ambos, por lo que se trata de balancear estos valores.

Para garantizar una alta consistencia la arquitectura debe permitir que los procesos que corren en los nodos remotos estén estrechamente acoplados, usualmente se requiere gran ancho de banda, baja latencia, y un número limitado de nodos conectados. Para alcanzar un alto grado de reacción las peticiones de datos deben ser respondidas rápidamente lo que conduce a nodos holgadamente acoplados, en este caso los nodos incluyen mayor cómputo para reducir los requerimientos de ancho de banda y latencia.

La arquitectura centralizada puede ser vista como una base de datos compartida que mantiene la consistencia del sistema todo el tiempo. Obviamente ésta puede carecer de sensibilidad, lo cual es elemental para las aplicaciones en tiempo real como los videojuegos. Las arquitecturas de réplica y distribuida sin embargo permiten alto grado de sensibilidad, pero aumentan la complejidad del manejo de los datos [8].

1.3.3 Capa de Aplicaciones

La Capa de Aplicación es construida en un nivel superior a la Capa Lógica. Las aplicaciones interactivas de red y de tiempo real han sido objeto de estudio en el área del desarrollo de simuladores, entornos virtuales y videojuegos. Los factores fundamentales que influyen en el diseño de estos sistemas son: la escalabilidad (*del inglés* scalability), la persistencia (*del inglés* persistence) y la colaboración (*del inglés* collaboration) entre jugadores.

La escalabilidad es la capacidad de adaptar el poder de cómputo en dependencia de la cantidad de recursos disponibles en todo momento. Por lo tanto la aplicación debe ser capaz de adaptarse dinámicamente y distribuir de forma adecuada el cómputo en cada nodo ante la variación en el número de jugadores. Para ello la implementación de la aplicación tiene que basarse en software concurrente, usando el paralelismo en hardware a través de la red de nodos. La escalabilidad cumple con el siguiente principio: Cada participante nuevo sobrecarga la comunicación pero al mismo tiempo, ofrece un mayor poder de cómputo a la aplicación.

La persistencia es la manera en que un nodo coexiste con la aplicación, esto se refiere a la entrada y salida del nodo y cómo esto afecta al estado de la aplicación. Por ejemplo cuando un nodo abandona la aplicación ésta debe actualizar el estado del juego redistribuyendo las responsabilidades de este nodo y por otra parte si el nodo se desconecta bruscamente, la aplicación pierde los objetos existentes en ese nodo. Por lo tanto la persistencia tiene que garantizar la administración de la configuración, así como también la detección y recuperación de errores.

La colaboración se refiere a la existencia de equipos en los cuales los miembros actúan unidos con el propósito de lograr un objetivo común (por ejemplo eliminar a otro equipo). Para lograr la colaboración la aplicación debe proveer a un jugador información abundante y exacta acerca de los otros jugadores. Esta información puede ser conocimiento acerca del estado de cada jugador o pueden

compartir un canal de comunicación dedicado. Técnicamente la colaboración requiere que la comunicación entre dos jugadores sea priorizada y está claro que el concepto de equipo es a nivel de aplicación, pues el concepto de colaboración a distancia puede ser complejo y requiere más recursos.

En la capa de aplicación se interpretan los datos según el contexto y la lógica (por ejemplo un valor entero representa una posición), por lo tanto esta capa está estrechamente relacionada con los usuarios finales y las aplicaciones específicas, ésta se deriva de las especificaciones del diseño de la aplicación ya sea un juego o cualquier otro sistema [8].

1.4 Arquitectura Cliente Servidor

El modelo cliente servidor (CS) es la arquitectura más usada actualmente en el desarrollo de videojuegos. En esta arquitectura hay un nodo especial (el servidor) al cual se conectan los demás nodos (los clientes), es por ello que en general la arquitectura sea centralizada. Son varias las razones por las cuales la arquitectura cliente servidor centralizada es tan popular. En primer lugar, CS es simple de implementar en relación con las demás arquitecturas, pues sólo se tiene en cuenta un tipo de conexión: la existente entre el cliente y el servidor. Por otra parte el servidor centralizado garantiza una alta consistencia y además los proveedores de juegos (para el caso de los juegos por Internet) pueden crear y controlar una estructura de subscripción para usarla con objetivos de mercado. Hay que destacar que también existen algunos problemas al usar esta arquitectura. El modelo implica una alta dependencia con el servidor, por lo que si este nodo falla, lo hará también toda la aplicación. El requerimiento de ancho de banda para el servidor siempre supera significativamente al del cliente. Y por último, el envío de mensajes hacia los clientes a través del servidor provoca una latencia adicional a la comunicación.

1.4.1 Ancho de Banda y Latencia

1.4.1.1 Requerimiento de Ancho de Banda.

En todo videojuego multijugador cada jugador ejecuta un ciclo durante el juego que incluye: leer la entrada local (provocada por los dispositivos de entrada como el mouse y el teclado), recibir las actualizaciones de los jugadores remotos, calcular el nuevo estado local y dibujar la vista gráfica del mundo virtual. El tiempo que demora en ejecutarse este ciclo es conocido como 'período de actualización del jugador', denominado T. En cada iteración del ciclo, el jugador envía al servidor un mensaje de actualización, el cual posee un tamaño en bytes, representado por L.

En la arquitectura CS un cliente envía la actualización del jugador en cada período, lo que representa una capacidad de L/T. Por otra parte el servidor envía hacia este cliente la actualización de cada jugador (sea N el número de jugadores del juego), esto es una capacidad de N*L/T. Luego la suma de estos valores es (N+1)*L/T, la cual es el requerimiento de ancho de banda del cliente. Como L y T son aproximadamente constantes, se llega a la conclusión de que el requerimiento de ancho de banda del cliente es una función lineal que depende de N.

De manera similar, el servidor recibe las actualizaciones de cada jugador, acorde con ello actualiza el estado global del juego y responde a los clientes con la actualización del nuevo estado. El tiempo que demora el servidor en recibir el estado de un jugador y responderle a éste con el nuevo estado global del juego, es conocido como 'latencia del servidor' (Ts). De modo que el servidor recibe un mensaje de actualización de cada cliente en un período de actualización T, esto requiere N*L/T y a su vez es enviado un mensaje de actualización del estado global del juego (el estado de cada cliente) a todos los clientes, lo que representa N*(N*L)/T. La suma de estos resultados es N*(N+1)*L/T, lo que significa que el requerimiento de ancho de banda del servidor es una función cuadrática dependiente del número de jugadores N. Aunque es posible disminuir significativamente el tráfico de red en el servidor, con el uso de técnicas de filtrado [9].

1.4.1.2 Latencia e Inconsistencia

El estado Si(t) de un juego para el jugador i en el instante de tiempo t es la colección de todos los atributos que describen la vista del juego para ese jugador en ese tiempo. El estado del juego debe incluir elementos como posición, dirección, velocidad y otros datos de cada jugador. Lo ideal sería que todos los jugadores tuviesen el mismo estado del juego, o sea Si(t) = Sj(t) para cada pareja de jugadores i,j en cualquier instante de tiempo t. Debido a que la comunicación entre los jugadores ocurre a través de la red, existe siempre un tiempo de retardo. Sea Di,j el tiempo de retardo entre el jugador i y el j, una acción del jugador i en el instante t_0, será conocida por el jugador j en el instante t_0 + Di,j. Durante ese periodo de tiempo existe un estado de inconsistencia entre jugadores y aunque es imposible eliminarlo totalmente, existen mecanismos para disminuirlo. Aunque este período de inconsistencia puede ser diferente para cada jugador, es de especial interés el máximo de estos períodos, lo que se conoce como Máximo Período de Inconsistencia (MPI).

En la arquitectura CS, para la estimación del MPI hay que tener en cuenta los siguientes factores: el período de actualización del jugador Tu, la latencia del servidor Ts, y el tiempo que demora el envío de un mensaje desde el cliente i hacia el servidor en adición al tiempo que demora recibir la respuesta del servidor a este cliente (tiempo de ida y vuelta) Ri,s. De esta forma la estimación de MPI sería:

MPI = MAXi(Tu + Ts + Ri,s) [9]

1.4.2 El Servidor

El servidor es una sesión que inicia y finaliza juegos, y acepta nuevas conexiones de clientes. Este programa es responsable de recibir todas las entradas de los clientes, calcular todo el estado del juego y mantener a los jugadores actualizados. El servidor realiza además los cálculos físicos que gobiernan el movimiento de las entidades y ejecuta la Inteligencia Artificial de los personajes.

Para optimizar, el servidor puede filtrar sólo información concerniente a cada cliente en particular, por ejemplo, enviarle al jugador sólo la información de las entidades que están en su campo de visión y de manera similar con los sonidos, actualizar sólo los que el jugador puede escuchar. Para ello existen

técnicas de filtrado, con las cuales se puede lograr disminuir los requerimientos de ancho de banda a un orden inclusive inferior al lineal, dependiendo de la aplicación específica que se está diseñando.

Es posible que el programa servidor esté corriendo solo en un proceso (por ejemplo cuando se dedica una PC exclusivamente para servidor), a este tipo de servidor se le conoce como servidor dedicado (*del inglés* dedicated server), por el contrario si el servidor y un cliente corren ambos en un mismo proceso (cuando por ejemplo existe una PC servidor pero en la cual también existe un jugador) se le conoce como servidor de escucha (*del inglés* listen server).

En resumen, la principal tarea del servidor es realizar toda la simulación del estado del juego.

1.4.3 El Cliente

El cliente es el programa con el cual el jugador interactúa, ejecuta los gráficos y los sonidos además de manipular la entrada de los dispositivos como teclado, mouse y joystick. Por lo tanto el cliente es responsable de enviar la entrada del jugador al servidor y de esperar por la respuesta de éste último para actualizar la vista del juego.

Para lograr un mayor realismo y una mejor sensibilidad, algunas de las simulaciones del juego pueden realizarse en el cliente, sobre todo para el caso en que éstas no influyan en el estado del juego para el resto de los jugadores. Por ejemplo si el jugador se voltea, esta acción puede ser ejecutada al instante, sin necesidad de esperar por la respuesta del servidor. También se pueden realizar en el cliente detección de colisiones.

1.5 Técnicas para compensar la Latencia

Un juego basado en la arquitectura cliente servidor básicamente ejecuta el siguiente algoritmo: Los clientes reciben la entrada de los usuarios y la envían al servidor el cual actualiza el estado del juego ejecutando la lógica y la simulación. El servidor envía mensajes de actualización a los clientes los cuales dibujan la escena actualizada con el nuevo estado del juego. Si el tiempo que transcurre entre el envío de los controles de un cliente y el recibo del mensaje de respuesta por parte del servidor es

considerable, entonces se puede provocar un efecto indeseable para el jugador, el cual observa que sus acciones se reflejan muy tarde en el juego.

1.5.1 Predicción

Esta técnica está basada en la idea de que el cliente no espere por la respuesta del servidor y para ello predice esta respuesta, así el cliente puede responder a la entrada del usuario inmediatamente. Existen dos modalidades de esta técnica: Predicción del Jugador (*del inglés* Player Prediction) que es la predicción de la respuesta relacionada con las acciones del jugador y Predicción del Oponente (*del inglés* Opponent Prediction) que es la predicción de la posición de las entidades que no son controladas por el jugador, sino por otro jugador o por el mismo servidor. Cuando el cliente recibe la respuesta del servidor es actualizada la vista del juego para mantener la consistencia.

Básicamente en la Predicción del Oponente el cliente calcula los parámetros de posición, velocidad o aceleración basándose en los últimos valores registrados y dependiendo del tiempo que ha transcurrido desde la última actualización.

Estas técnicas pueden disminuir drásticamente los efectos de la latencia: Pueden aumentar la sensibilidad o grado de reacción de la aplicación, pero al mismo tiempo pueden provocar inconsistencia, pues los clientes pueden tener estados de juego diferentes al del servidor [7].

1.5.2 Manipulación del Tiempo.

Otro de los problemas provocado por la latencia es cuando varios clientes son actualizados por el servidor en tiempos diferentes, debido a la diferencia entre la latencia que puede existir para cada cliente. Esto puede afectar considerablemente algunos juegos, en los cuales los jugadores necesitan reaccionar en tiempos coherentes. Dos técnicas usadas para compensar este problema son: Demora del Tiempo (*del inglés* Time Delay) para limitar a jugadores que tengan baja latencia y Distorsión del Tiempo (*del inglés* Time Warp) para aventajar a jugadores que posean alta latencia.

Demora del Tiempo se basa en demorar por parte del servidor el procesamiento de los clientes con baja latencia, al igual que el envío del estado del juego, el cual puede ser enviado primero a los clientes que posean una elevada latencia y luego a los de menor, esto provocaría que los clientes sean actualizados en aproximadamente iguales instantes de tiempos. Por otro lado en esta técnica también se puede retardar el envío de mensajes del cliente hacia el servidor para lograr el mismo objetivo. Como es lógico esto puede provocar que disminuya el grado de reacción para algunos clientes.

Distorsión del Tiempo se basa en la manipulación del tiempo por parte del servidor de manera tal que sea posible retroceder el estado del juego en un intervalo de tiempo deseado. Un cliente envía una acción en el tiempo t_1 la cual llega al servidor en el tiempo t_2 diferente a t_1 provocado por la latencia, en este caso el servidor retrocede el estado del juego en un tiempo $t_2 - t_1$ y vuelve a actualizar ejecutando la acción del cliente, si esta acción provoca un cambio en el juego entonces son actualizados el resto de los clientes afectados. En general el servidor ejecuta las acciones de cada cliente retrocediendo el estado del juego en un intervalo de tiempo resultante de restarle al tiempo en el servidor el valor de la latencia correspondiente al cliente que está procesando y luego actualiza el juego y a los clientes afectados para mantener la consistencia [7].

1.6 Quake

Quake es uno de los pocos juegos comerciales que está disponible como código abierto, el mismo ha tenido un gran impacto en el desarrollo de los videojuegos en los últimos años y existe actualmente una comunidad que contribuye al desarrollo de éste. Es un juego tridimensional de combate con armas de fuego, sus creadores (“id Software”) han hecho aportes revolucionarios a los gráficos y sobre todo a la arquitectura de red para este tipo de juegos, aportes que comenzaron a asombrar desde su antecesor Doom, cuando por primera vez se vio pasar por al lado al “otro”, recibiendo de él un cohete disparado directamente al rostro. Se puede jugar a través de Internet y todas las interacciones con el jugador ocurren en tiempo real. Además existe un proyecto de documentación que ha compilado información útil para el estudio de Quake [10].

Por todas estas razones es Quake un caso de estudio ideal como guía para el desarrollo de juegos y en particular de una arquitectura cliente servidor (que es precisamente la que usa Quake).

1.6.1 La arquitectura de Quake

Quake posee una arquitectura cliente servidor centralizada, en la cual el servidor es responsable de calcular todo el estado del juego y distribuirlo a los clientes. Por otra parte la entrada y salida que provoca el jugador ocurre en el cliente, el cual envía estos datos al servidor. Por lo tanto en el juego cada cliente envía sus acciones al servidor, éste las procesa y devuelve el resultado hacia los clientes [10].

Existen dos modos de inicializar el juego: En el modo simple existe un solo jugador que juega contra la computadora, a diferencia del modo multijugador en el cual participan varios jugadores. Aunque en modo simple no se hace uso de la red, la arquitectura mantiene la misma estructura, es decir, existe un servidor y un cliente, pero en este caso ambos programas corren en el mismo proceso y la comunicación ocurre a través de de buffers de memoria. En el modo multijugador el cliente y el servidor son procesos separados, y corren en máquinas diferentes (excepto en el caso especial de que un cliente y el servidor estén en la misma máquina y corran en el mismo proceso) [5].

Como se ha podido analizar son varios los beneficios que tiene el modelo cliente servidor en el desarrollo de videojuegos multijugadores, pues resulta sencillo mantener la consistencia o la sincronización entre los jugadores. Para el desarrollador, es útil el hecho de poder usar este modelo en los juegos tanto en modo multijugador como simple, lo cual es muy conveniente pues el diseño es forzado a ser modular y a tener un único canal entre el cliente y el servidor simplificando el debugging y además ayudando a tener código idéntico en ambos modos de juego.

1.7 Fundamentación de la Tecnología

1.7.1 Biblioteca gráfica OpenGL

OpenGl es la biblioteca gráfica por excelencia, desarrollada por Silicon Graphics Incorporated (SGI) en 1992. Ésta no es más que una especificación estándar que define una Application Programming Interface (API) multilenguaje y multiplataforma por lo que puede ser utilizada en varios Sistemas Operativos (SO) como Linux, Unix, Mac OS, Windows e incluso en móviles. Se emplea para crear

aplicaciones gráficas en 2D y 3D, ya sea aplicaciones de realidad virtual, simulación de vuelos, Diseño Asistido por Computadora (del inglés Computer Aided Design) o desarrollo de videojuegos. Su nombre viene del inglés **Open G**raphics **L**ibrary, cuya traducción puede ser *Biblioteca de Gráficos Abierta* (o mejor *libre*, teniendo en cuenta su política de licencias).

OpenGL tiene dos propósitos principales:

- Ocultar la complejidad de la interfaz con las diferentes tarjetas gráficas, presentando al programador una API única y uniforme.
- Ocultar las diferentes capacidades de las diversas plataformas hardware, requiriendo que todas las implementaciones soporten el conjunto completo de características de OpenGL (utilizando emulación software si fuese necesario).

La operación básica de OpenGL es aceptar primitivas tales como puntos, líneas y polígonos y convertirlas en píxeles. Este proceso es realizado por una pipeline gráfica conocida como la Máquina de Estados de OpenGL. La mayor parte de los comandos de OpenGL emiten primitivas a la pipeline gráfica o configuran cómo la pipeline procesa dichas primitivas. Hasta la aparición de la versión 2.0 cada etapa de la pipeline ejecutaba una función establecida, resultando poco configurable. A partir de la versión 2.0 varias etapas son completamente programables usando GL Shading Language (GLSL).

OpenGL cuenta con un enorme prestigio entre la comunidad de programadores de aplicaciones gráficas debido a su excelente rendimiento, calidad en el render y compatibilidad. Aunque existen otras bibliotecas de gran popularidad como DirectX, OpenGL ha sido la preferida de programadores gráficos como John Carmack, con la que ha desarrollado los motores gráficos del Quake y del Doom.

1.7.2 Biblioteca gráfica multiplataforma G3D Engine

Graphics Three Dimensional (G3D) es un Engine 3D disponible como código abierto (del inglés Open Source) y libre (con licencia BSD), desarrollado en C++. Esta biblioteca es usada en el desarrollo de juegos, artículos investigativos, simuladores militares, y en cursos universitarios. La misma provee un conjunto de funcionalidades que son útiles para el desarrollo de cualquier aplicación gráfica. Además

facilita el uso de bibliotecas de bajo nivel como OpenGL y sockets, sin afectar su rendimiento o funcionamiento. G3D proporciona una base sólida y altamente optimizada para el desarrollo de aplicaciones gráficas [6].

Además G3D puede ser usada como una biblioteca de utilidades, pues posee características de propósito general, como por ejemplo clases para la programación multihilo, estructuras de datos rápidas, eficientes y fáciles de usar, así como también control y administración de la memoria y obtención de información relacionada con el Sistema Operativo y el Central Processing Unit (CPU) [6].

1.7.2.1 La red en G3D

G3D provee los componentes necesarios para encapsular la Capa Lógica de red, pues tiene conductos para establecer conexión entre nodos, y mecanismos de serialización de mensajes para la transmisión de datos.

Esta biblioteca cuenta con las clases necesarias para encapsular el bajo nivel de los sockets y proveer un acceso fácil al programador. También facilita la comunicación entre nodos permitiendo definir el protocolo a usar TCP o UDP. Además la serialización posibilita la transmisión de objetos por la red de manera muy sencilla, en G3D cualquier objeto que pueda ser serializado y deserializado puede ser usado con los serializadores de red, los cuales lo convierten a mensajes que pueden ser transmitidos de un nodo a otro. G3D soporta además la búsqueda automática de servidores en la red, y también posibilita obtener las propiedades de red de los nodos conectados.

1.7.3 Lenguaje de Programación C++

C++ es un lenguaje orientado a objetos de probada eficacia para generar aplicaciones de alto rendimiento como deben ser las aplicaciones gráficas. Éste aporta un nivel de productividad muy superior a C, sin comprometer la flexibilidad, el rendimiento o el control. Es uno de los lenguajes de sistemas más conocido en el mundo, altamente compatible y soporta las bliotecas gráficas Glide, OpenGL, DirectX y G3D que son muy utilizadas en la industria de los videojuegos. En general son

considerables las ventajas que este lenguaje posee, al hacer uso del paradigma de la programación orientada a objetos.

1.7.4 Rational Rose

Es una de las mejores soluciones de modelado visual en el mundo y de las mejores herramientas para traducir requisitos de alto nivel a una arquitectura flexible basada en componentes. Rational se encuentra a la cabeza en cuanto al desarrollo del Unified Modeling Language (UML), que se ha convertido en la notación estandarizada empleada en Rational Rose para especificar, visualizar y construir desarrollos de software y sistemas.

Racional Rose domina el mercado de herramientas para el análisis, modelado, diseño y construcción orientada a objetos, tiene todas las características que los desarrolladores, analistas y arquitectos exigen, soporte UML incomparable, desarrollo basado en componentes con soporte para arquitecturas líderes en la industria y modelos de componentes, facilidad de uso e integración optimizada.

Rose es una herramienta multiplataforma que ayuda a la comunicación entre los miembros del equipo, a monitorear el tiempo de desarrollo y a entender el entorno de los sistemas. Una de las grandes ventajas de Rose es que utiliza una notación estándar en la arquitectura de software (UML), la cual permite a los arquitectos de software y desarrolladores visualizar el sistema completo utilizando un lenguaje común, además los diseñadores pueden modelar sus componentes e interfaces en forma individual y luego unirlos con otros componentes del proyecto.

Capítulo 2: Características de la Arquitectura

En este apartado se define la arquitectura de comunicación a desarrollar y se fundamenta su selección. Se comienza además con el análisis del dominio y la captura de sus principales requerimientos. También se describen y modelan los Casos de Uso (CU) que representan los procesos que responden a las funcionalidades definidas en los requerimientos.

2.1 Solución Técnica

La solución propuesta es la implementación de una arquitectura genérica cliente servidor centralizada basada en la biblioteca multiplataforma G3D.

En este caso el término “genérica” se refiere a la independencia en cuanto a la disposición física de los nodos, ya sea nodos ubicados en la misma computadora o no. Esto forzará a los desarrolladores a un diseño modular, por ejemplo el diseño del cliente y el servidor no varía inclusive en juegos modo multijugador o simple, en los cuales sólo se haría uso de la red en el primer caso.

Es por ello que se necesitan dos tipos de comunicaciones: remota, la cual hace uso de la red y comunicación local para el caso de los nodos que se encuentren sobre la misma computadora (como en el modo de juego simple) usando para ello buffers de memoria.

La elección de cliente servidor es justificada por varios factores: primero es la arquitectura más popular en el área de desarrollo de videojuegos en la actualidad, esto a su vez se debe a que ésta es una arquitectura simple de implementar en relación con otras. Por otra parte el propio modelo es muy adecuado para ser del tipo centralizada, debido a las características especiales del servidor, el cual manipula y procesa toda la información que se intercambia entre los jugadores. La centralización proporciona la ventaja de garantizar una alta consistencia.

La posibilidad de dos tipos de servidores brinda flexibilidad al modelo, para el caso especial del servidor de escucha es necesario el uso de un hilo concurrente, de manera tal que este programa (el servidor) pueda correr de forma paralela al cliente que existe en el mismo proceso. Para que un

programa multihilo funcione correctamente se debe garantizar el acceso adecuado a las secciones críticas que son las zonas de memoria a las cuales pueden acceder varios hilos al mismo tiempo, para ello será usado un mecanismo de exclusión mutua.

El fichero que contiene los datos de configuración para las conexiones y el protocolo de las aplicaciones es de tipo texto, con el siguiente formato: Es un fichero basado en tokens, que soporta comentarios con la sintaxis del lenguaje de programación C++, un token es una pareja compuesta por un símbolo y un valor, las reglas para escribir los tokens son las mismas de las variables de C++. La configuración está compuesta por los siguientes tokens: GAME_PORT, APP_PROTOCOL_NAME, APP_PROTOCOL_VERSION, SERVER_BROADCAST_PORT, CLIENT_BROADCAST_PORT y SERVER_ADVERTISEMENT_PORT, cuya semántica es explicada por comentarios en el fichero.

La biblioteca G3D ofrece una infraestructura adecuada para implementar este tipo de modelo arquitectónico, en el capítulo primero se expresan las características generales de la misma.

La siguiente figura muestra un esbozo del diseño arquitectónico a implementar:

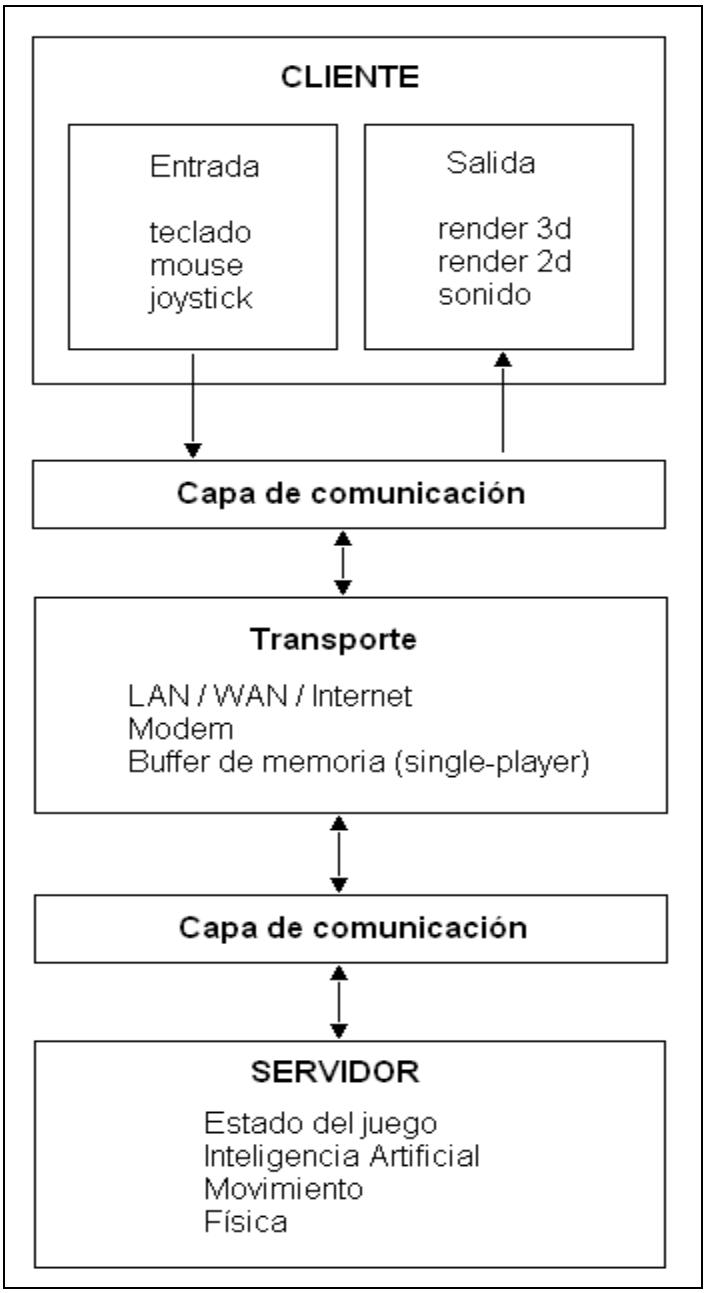


Figura 2.1 Esbozo del Diseño Arquitectónico

2.2 Análisis del Dominio

El análisis del dominio es el estudio del campo de acción en el que se desarrolla la transmisión de datos para juegos con un modelo cliente servidor, utilizando las mismas técnicas de abstracción, composición, generalización y especialización que se emplea en el análisis Orientado a Objetos (OO), pero no trata aplicaciones en particular. Un modelo de dominio sirve como base para diseñar y construir objetos del ámbito en el que se enmarca. Al desarrollar un módulo que no es más que un componente auto controlado, el cual posee interfaces bien definidas para otros componentes y es construido de manera tal que se facilite su ensamblaje y cambios en sus componentes con el menor impacto posible en la aplicación logrando así una independencia modular, se determina que la idea central es modelar el dominio utilizando programación orientada a objetos, obteniendo así, un modelo del dominio, formado por objetos muy similares a la realidad, que se rigen según las posibles características generales de las aplicaciones. El modelo de dominio describe con claridad el problema a los desarrolladores y sirve como base para diseñar y construir objetos del dominio [3].

2.2.1 Modelo del Dominio

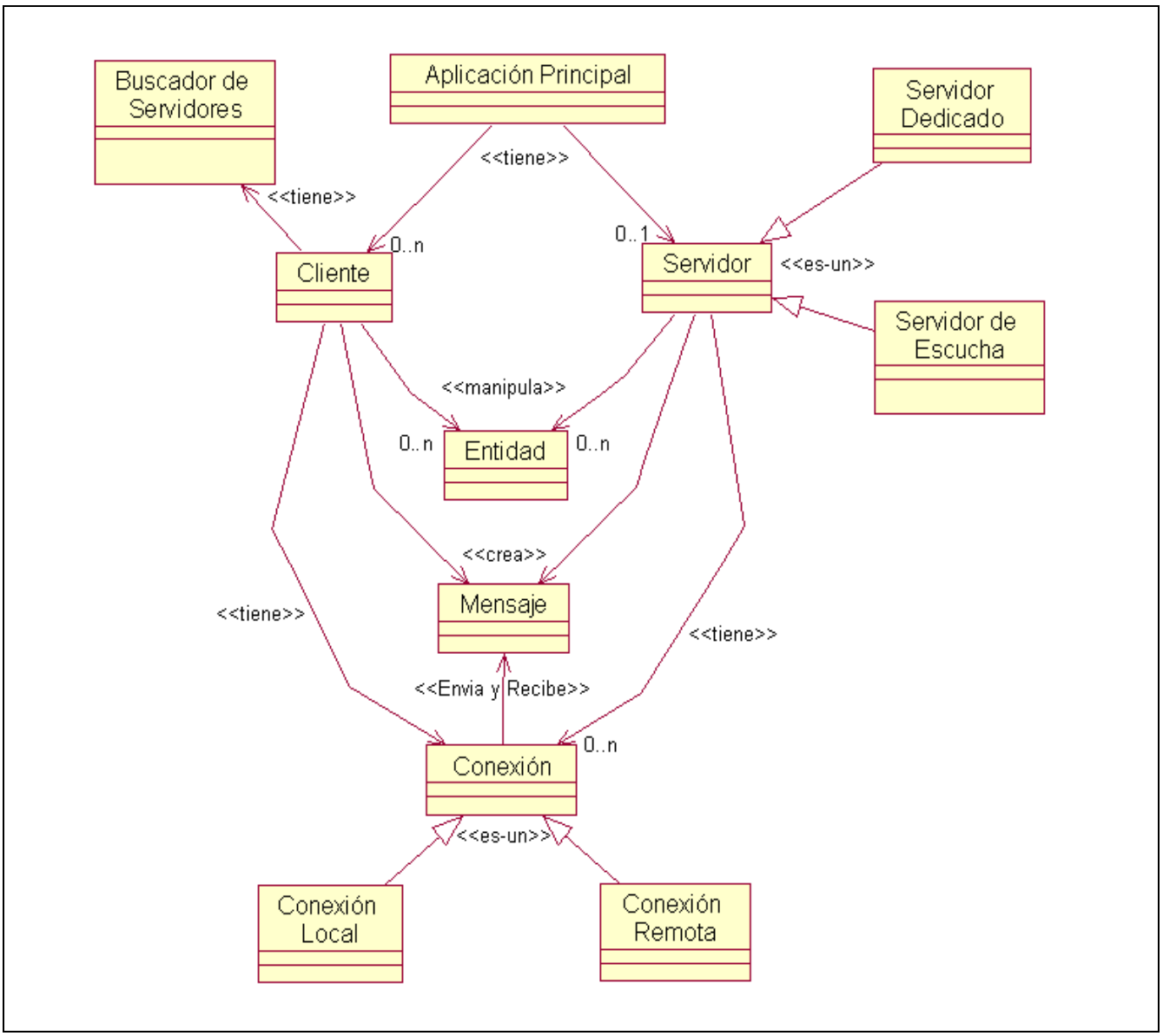


Figura 2.2 Modelo del Dominio

En el modelo anterior se muestran las relaciones de los distintos conceptos que reflejan la interacción del sistema. La Aplicación Principal, que puede ser un juego, se ejecuta siendo un cliente (ClientThread) o un servidor (ServerThread). Al ser un cliente, puede o no utilizar el Buscador de Servidores para establecer una conexión con el servidor, en caso de ser un servidor ésta espera por los clientes que se conectarán. En este proceso ambos, Cliente y Servidor, van a establecer comunicación a través de una Conexión, que no es más que un canal de comunicación.

A un nivel más abstracto están los objetos Entidad y Mensaje, los primeros expresan y modifican el estado del juego y para el intercambio de información usan a los últimos. Por lo tanto los mensajes “viajan” a través de un canal representado por la Conexión.

El tipo de Conexión depende de la ubicación física de los nodos que se comunican, o sea, si es local (en la misma computadora) o remota (en diferentes computadoras), corresponde a Conexión Local o Conexión Remota respectivamente.

El programa Servidor puede estar ejecutándose sobre el mismo proceso en el cual se ejecuta un Cliente, en este caso será un Servidor de Escucha que correrá en un hilo. El Servidor Dedicado se ejecuta sobre un proceso en el cual no existe ningún cliente.

2.2.2 Conceptos del Modelo de Dominio

Aplicación Principal: Programa principal que especializa a la infraestructura (framework) de la G3D.

Buscador de Servidores: Busca de forma automática servidores en la red.

Cliente: Representa al cliente, que establece comunicación con el servidor e interactúa con éste.

Servidor: Representa al servidor, el programa que mantiene la comunicación con los clientes y ejecuta la simulación.

Servidor Dedicado: Es un tipo especial de servidor, el cual corre en un proceso donde no existe algún cliente.

Servidor de Escucha: Es un tipo especial de servidor que corre en un proceso donde también corre un cliente.

Conexión: Representa al canal de comunicación entre el cliente y el servidor, a través del cual viajarán los mensajes.

Conexión Local: Es un tipo especial de conexión, donde los nodos que se comunican se encuentran ubicados en la misma computadora.

Conexión Remota: Es un tipo especial de conexión, en la cual los nodos involucrados se encuentran en diferentes computadoras y se comunican a través de una red.

Mensaje: Contiene información y será enviado de un nodo a otro a través de un canal de comunicación.

Entidad: Es la representación de cada elemento en el entorno que interactúa con otros, por lo que sus acciones determinan el estado del juego.

2.3 Requisitos de Software

La IEEE Standard Glossary of Software Engineering Terminology define un requisito como condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo, por esto un requisito no es más que la condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente. Los requisitos se pueden clasificar en: funcionales y no funcionales. Los Requisitos Funcionales son capacidades o condiciones que el sistema debe cumplir y los Requisitos No Funcionales son propiedades o cualidades que el producto debe tener. En este segmento se enumeran un conjunto de requisitos del campo de acción de transmisión de datos en arquitecturas cliente servidor para videojuegos, a partir de la experiencia y necesidades de los desarrolladores del proyecto Juegos Consola.

2.3.1 Requisitos Funcionales

RF1. – Buscar automáticamente servidores online.

RF2. – Establecer conexiones con el servidor.

RF3. – Enviar y recibir mensajes del servidor.

RF4. – Finalizar conexión con el servidor.

RF5. – Inicializar servidor (dedicado o de escucha).

RF6. – Establecer conexiones con los clientes.

RF7. – Enviar y recibir mensajes de los clientes.

RF8. – Notificar a todos los clientes conectados cuando se agrega un nuevo cliente.

RF9. – Notificar a todos los clientes conectados cuando se retira un nuevo cliente.

RF10. – Permitir definir un puerto para establecer la conexión.

RF11. – Notificar a todos los clientes cuando se desconecta el servidor.

RF12. – Medir latencia de los clientes en el servidor.

RF13 – Medir consumo de ancho de banda de los clientes en el servidor.

RF14 – Cargar la configuración de las conexiones desde un fichero texto.

2.3.2 Requisitos No Funcionales

1. **Usabilidad**:

   RNF1.1 – Este módulo está orientado a programadores con conocimientos básicos de redes, C++, y la biblioteca gráfica G3D.

2. **Soporte**:

   RNF2.1 – Para su correcto uso debe poseer una documentación del código y un demo ejemplificando el uso de los distintos componentes del módulo

3. **Portabilidad:**

   RNF3.1 – Debe ser portable a cualquier Sistema Operativo *Windows*, familia de *Unix* o *Mac OS.*

4. **Requisito de Software**:

RNF4.1 – Se debe desarrollar utilizando el Entorno de Desarrollo Integrado (*del inglés* Integrated Development Environment (IDE)) *VisualStudio.Net 2005*.

RNF4.2 – Se debe desarrollar utilizando las APIs que posee la biblioteca gráfica G3D.

2.4 Modelo de Casos de Uso del Sistema

2.4.1 Definición de los Actores del Sistema

Actores	Justificación
GApp	GApp es el framework de la G3D, encargado de ejecutar todas las funcionalidades del módulo.
ClientThread	GApp especializado que ejecuta funcionalidades de la sesión cliente de la arquitectura.
ServerThread	GApp especializado que ejecuta funcionalidades de la sesión servidor de la arquitectura.

2.4.2 Listado de Casos de Uso

CU-1	Descubrir servidor
Actor	ClientThread
Descripción	El cliente busca servidores. El cliente establece conexión con un servidor.
Referencia	RF-1 RF-2

CU-2	Gestionar Mensajes en el cliente.
Actor	ClientThread
Descripción	El cliente recibe los mensajes enviados por el servidor, y los procesa. El cliente envía mensajes al servidor.
Referencia	RF-3 RF-4 RF-11

CU-3	Descubrir clientes
Actor	ServerThread
Descripción	El servidor publica información por la red, usada por los clientes para establecer conexión.
Referencia	RF-6 RF-8

CU-4	Gestionar Mensajes en el Servidor.
Actor	ServerThread
Descripción	El servidor recibe mensajes enviados por los clientes y los procesa. El servidor envía mensajes a los clientes.
Referencia	RF-7 RF-9 RF-11.

CU-5	Inicializar Servidor
Actor	ServerThread
Descripción	Se inicializa el servidor: Dedicado o de escucha.
Referencia	RF-5 RF-10

CU-6	Medir latencia y consumo de ancho de banda.
Actor	ServerThread
Descripción	El servidor calcula por cada cliente el consumo de ancho de banda y la latencia.
Referencia	RF-12 RF-13

CU-7	Cargar configuración.
Actor	GApp
Descripción	Se cargan desde un fichero los datos de la configuración para establecer las conexiones.
Referencia	RF-14

2.4.3 Diagrama de Casos de Uso

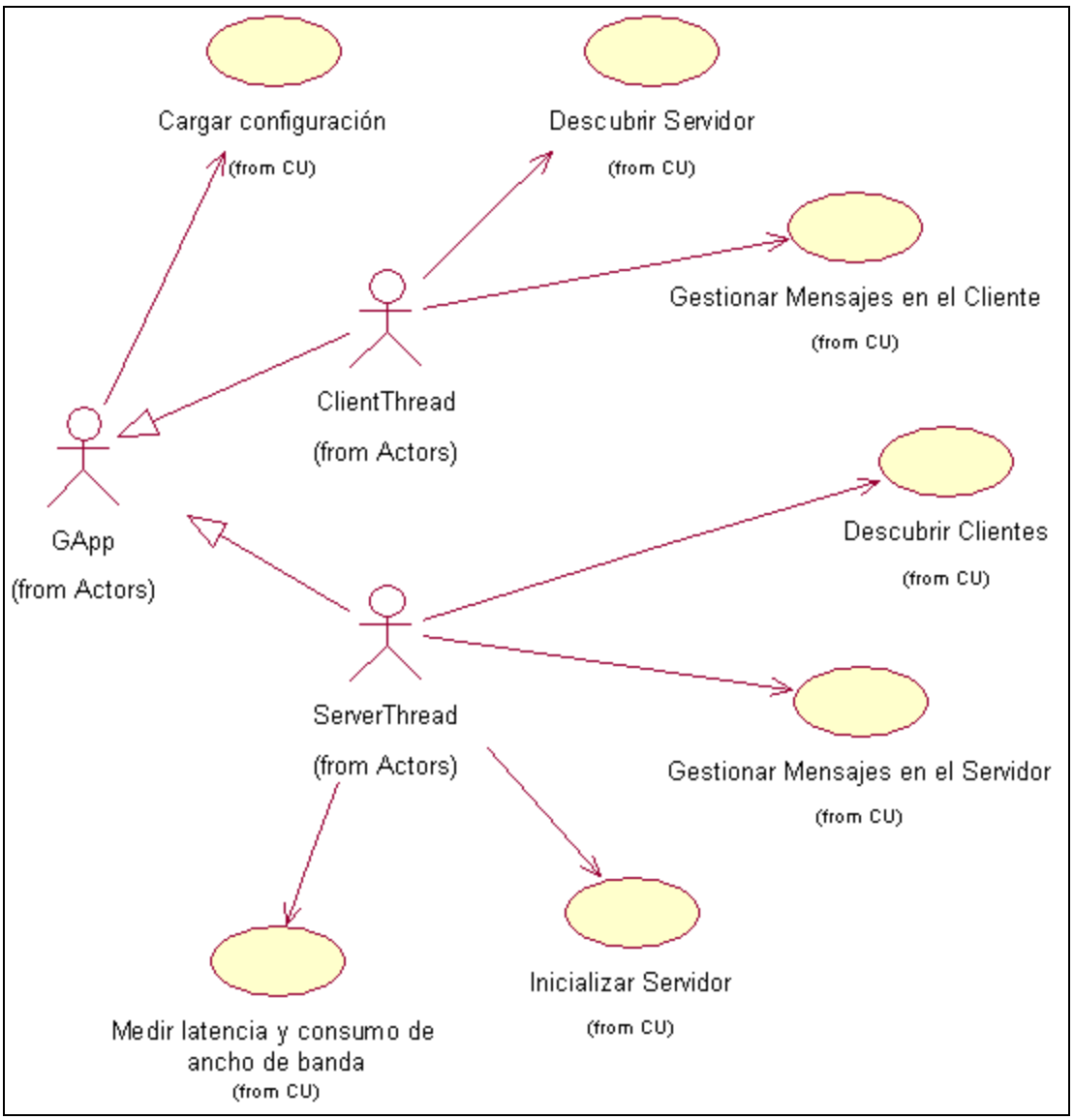


Figura 2.3 Modelo de Casos de Uso del Sistema

2.4.4 Casos de Uso expandidos

Caso de uso		
CU-1	Descubrir servidor	
Propósito	Buscar servidores y establecer conexión con un servidor por parte del cliente.	
Actores	ClientThread	
Resumen:	El cliente busca servidores y establece conexión con un servidor.	
Referencias	RF-1 RF-2	
Acción del actor		**Respuesta del sistema**
1.ClientThread invoca el evento “ejecutar red” del Buscador de Servidores. 2.ClientThread indica establecer conexión con un servidor.		1.1.El sistema busca servidores disponibles y devuelve una lista con la descripción de los mismos. 2.1. El sistema establece la conexión con el servidor especificado.
Flujo alternativo		
Acción del actor		**Respuesta del sistema**
		2.1. Si no es posible establecer la conexión con el servidor el sistema lo informa.
Puntos de extensión		

Caso de uso		
CU-2	Gestionar Mensajes en el Cliente.	
Propósito	Recibir y procesar mensajes enviados por el servidor; enviar mensajes al servidor.	
Actores	ClientThread	
Resumen:	El programa cliente chequea mensajes enviados por el servidor, los recibe y los procesa, además responde al servidor enviándole mensajes.	
Referencias	RF-3 RF-4 RF-11	
Acción del actor		**Respuesta del sistema**
1.ClientThread invoca el evento “ejecutar red” del Cliente. 2.ClientThread invoca el evento “enviar estado local” del Cliente.		1.1. El sistema verifica si hay mensajes sin procesar en la conexión del cliente. 1.2. El sistema recibe cada mensaje y lo procesa ejecutando el comando del mensaje para el cliente. 2.1. El sistema envía mensajes al servidor a través de la conexión del cliente.
Flujo alternativo		
Acción del actor		**Respuesta del sistema**
		1.1.Si la conexión no permite verificar mensajes el sistema informa que el servidor ya no está disponible.
Puntos de extensión		

Caso de uso		
CU-3	Descubrir Clientes	
Propósito	Chequear por clientes que intentan conectarse y establecer conexión con los mismos por parte del servidor.	
Actores	ServerThread	
Resumen:	El servidor establece conexión con cada cliente que intenta conectarse.	
Referencias	RF-6 RF-8	
Acción del actor		**Respuesta del sistema**
1.ServerThread invoca el evento “ejecutar red” del Servidor.		1.1. El sistema verifica si hay clientes intentando conexión con el servidor. 1.2. El sistema crea nuevas conexiones en el servidor para cada cliente entrante. 1.3. El sistema informa a los clientes previamente conectados acerca de los nuevos clientes enviándole un mensaje a los primeros.
Flujo alternativo		
Acción del actor		**Respuesta del sistema**
		1.1.Si no es posible verificar por clientes entrantes el sistema informa que no es posible realizar conexiones.
Puntos de extensión		

Caso de uso		
CU-4	Gestionar mensajes en el servidor.	
Propósito	Recibir y procesar mensajes enviados por los clientes; enviar mensajes a los clientes.	
Actores	ServerThread	
Resumen:	El programa servidor chequea mensajes enviados por los clientes, los recibe y los procesa, además responde a los clientes enviándoles mensajes.	
Referencias	RF-7 RF-9 RF-11	
Acción del actor		**Respuesta del sistema**
1.ServerThread invoca el evento “ejecutar red” del servidor. 2.ServerThread invoca el evento “enviar estado global” del servidor.		1.1. El sistema verifica si hay mensajes sin procesar en cada conexión en el servidor. 1.2. El sistema recibe por cada conexión cada mensaje y lo procesa ejecutando el comando del mensaje para el servidor. 1.3. El sistema chequea por clientes desconectados, elimina las conexiones correspondientes e informa al resto de los clientes con un mensaje. 2.1. El sistema envía mensajes de actualización a cada cliente a través de las conexiones del servidor.
Flujo alternativo		
Acción del actor		**Respuesta del sistema**
Puntos de extensión		

Caso de uso		
CU-5	Inicializar Servidor	
Propósito	Inicializar un tipo específico de servidor: Dedicado o de escucha.	
Actores	ServerThread	
Resumen:	El sistema inicializa un tipo se servidor específico.	
Referencias	RF-5 RF-10	
Acción del actor		**Respuesta del sistema**
1.ServerThread indica inicializar un Servidor de Escucha.		1.1.El sistema crea un nuevo hilo de proceso, sobre el cual ejecuta las funciones del servidor inicializado; permitiendo la ejecución de clientes sobre el mismo proceso.
Flujo alternativo		
Acción del actor		**Respuesta del sistema**
1.ServerThread indica inicializar un Servidor Dedicado.		1.1.El sistema inicializa el servidor sin crear un nuevo hilo de proceso, impidiendo que se puedan ejecutar clientes sobre el mismo proceso.
Puntos de extensión		

Caso de uso		
CU-6	Medir latencia y consumo de ancho de banda.	
Propósito	Hacer en el servidor una medición real de los parámetros de latencia y consumo de ancho de banda para cada cliente.	
Actores	ServerThread	
Resumen:	El servidor mide la latencia y el consumo de ancho de banda de cada cliente.	
Referencias	RF-12 RF-13	
Acción del actor		**Respuesta del sistema**
1.ServerThread invoca el evento “simulación” del servidor. 2.ServerThread invoca el evento “ejecutar red” del servidor.		1.1.Si ha transcurrido la unidad de tiempo el sistema calcula el consumo de ancho de banda e inicia el conteo de la latencia mediante el envío de un mensaje “ping” para todos los clientes. 2.1. Por cada cliente del cual se reciba un mensaje de respuesta “ping” se calcula la latencia.
Flujo alternativo		
Acción del actor		**Respuesta del sistema**
		1.1. Si no ha transcurrido la unidad de tiempo sólo se incrementa el tiempo transcurrido.
Puntos de extensión		

Caso de uso		
CU-7	Cargar configuración.	
Propósito	Cargar desde un fichero los datos de configuración necesarios para establecer las conexiones.	
Actores	GApp	
Resumen:	Son cargados desde fichero los datos de configuración.	
Referencias	RF-14	
Acción del actor		**Respuesta del sistema**
1.GApp crea un objeto de “configuración” e invoca “cargar desde fichero” especificando la dirección local del fichero.		1.1.El sistema abre el fichero especificado y lee los datos de configuración contenidos en el fichero, quedando inicializado el objeto “configuración”.
Flujo alternativo		
Acción del actor		**Respuesta del sistema**
		1.1. Si el fichero no existe o no se puede abrir el sistema devuelve una respuesta negativa e inicializa el objeto “configuración” con una configuración por defecto.
Puntos de extensión		

Capítulo 3: Diseño e Implementación de la Arquitectura

En este capítulo se esboza el diseño del módulo desarrollado, con el objetivo de especificar cómo solucionar los requisitos funcionales. Se especifican además los patrones aplicados en la solución y se mencionan las principales clases con sus funcionalidades para una mejor comprensión del trabajo. También se presentan los diagramas de colaboración elaborados a partir de los Casos de Uso y los diagramas de despliegue y componentes.

3.1 Patrones

Un patrón es la descripción etiquetada de un problema, de la solución, de cuándo aplicar la solución y la manera de hacerlo dentro de otros contextos [4]. Los patrones de diseño describen un problema que ocurre repetidas veces en algún contexto determinado de desarrollo de software y entregan una buena solución ya probada. Esto ayuda a diseñar correctamente en menos tiempo, ayuda a construir problemas reutilizables y extensibles y facilita la documentación.

En el presente trabajo se estudiaron y aplicaron los siguientes patrones de diseño que pertenecen al grupo de los famosos 23 patrones de Gang of Four (GOF):

Factory Method: Patrón Creacional de Clase, que define una interfaz para la creación de un objeto, el cual es creado en las subclases que implementen dicha interfaz. En este trabajo es aplicable cuando una clase no puede anticipar los tipos de objetos que usará. Este patrón ofrece una alternativa que es mucho más flexible que la creación directa de objetos, pues brinda la posibilidad de modificar la creación de los objetos en las subclases.

Prototype: Patrón Creacional de Objeto, que especifica los tipos de objetos a crear usando una instancia prototípica y crea nuevos objetos copiando (o clonando) dicho prototipo. En este trabajo es aplicable al ser el sistema independiente de la creación, composición o representación de los objetos de tipo Mensaje. Los mensajes son creados y copiados (incluyendo el estado del objeto) por el sistema sin que éste conozca los detalles específicos de estas operaciones. Este patrón ofrece muchas ventajas tales como agregar y quitar tipos de objetos al sistema en tiempo de ejecución, lo

que provoca comportamientos dinámicos y la reducción de subclases en el sistema, al no necesitar una jerarquía para la creación de objetos como en Factory Method. Prototype permite configurar una aplicación con clases dinámicamente, las clases pueden ser cargadas dinámicamente en tiempo de ejecución y el sistema no necesita hacer referencia a constructores estáticamente.

Proxy: Patrón Estructural de Objeto, que provee y controla el acceso a un objeto. En el presente trabajo, un "proxy remoto" provee una representación local para un objeto remoto, por ejemplo ambos cliente y servidor se comunican e interactúan a través de "proxies", esto oculta el hecho de que ambos objetos pueden encontrarse en diferentes nodos (o computadoras) o en diferentes espacios de memoria.

Command: Patrón de Comportamiento de Objeto, el cual encapsula una acción en un objeto. Un objeto Command puede ejecutar una acción al ocurrir un evento determinado. Este patrón es la alternativa orientada a objetos a las funciones "callback" (función registrada, que luego es ejecutada) y permite al sistema abstraerse de las acciones específicas a ejecutar al ocurrir ciertos eventos, pues el comando brinda una interfaz común para ejecutar la acción encapsulada por el mismo, esto permite un desacople entre el objeto que invoca la acción y el que "conoce" cómo ejecutarla. El patrón permite además facilidad para incrementar la funcionalidad de la aplicación al agregar nuevos comandos sin la necesidad de cambiar las clases existentes.

Template Method: Patrón de Comportamiento de Clase, el cual define el esqueleto de un algoritmo en una operación, pero el algoritmo se completa en las subclases, esto permite variaciones en el algoritmo (realizadas por las subclases) sin que sea modificada la estructura del mismo. En el presente trabajo existen algoritmos que mantienen invariables algunos pasos, los cuales son implementados una vez y las subclases sólo tienen que implementar el resto del algoritmo, esto es una buena práctica de rehúso.

3.2 Diseño por Paquetes

Dentro del ciclo de desarrollo del software el diseño juega un papel muy importante pues éste es el encargado de modelar el sistema. En el diseño se desarrolla la arquitectura y se crea la base de la implementación para que el sistema cumpla con los requisitos funcionales y no funcionales.

Como resultado del refinamiento de las etapas anteriores se identificaron las principales clases, las cuales fueron agrupadas por sus funcionalidades en tres paquetes generales (Figura 3.1). Este diagrama ofrece una vista general en cuanto a la organización o distribución de las clases por paquetes dentro del sistema.

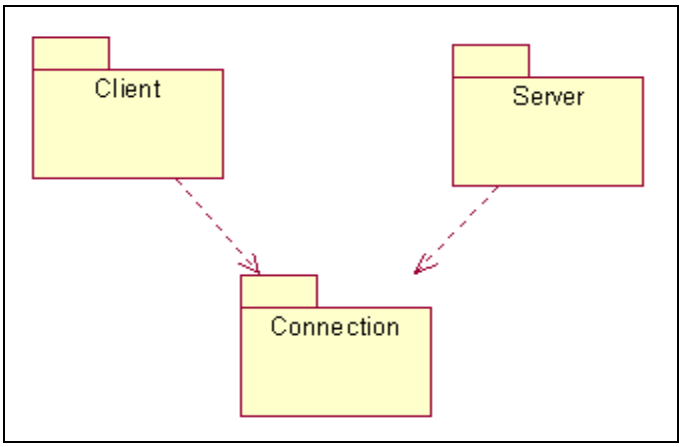


Figura 3.1 Diagrama de Paquetes de Clases del Diseño

El lenguaje UML ofrece el mecanismo paquete que permite describir los grupos de elementos o subsistemas. Un paquete es un conjunto de cualquier tipo de elementos de un modelo: clases, casos de uso, diagramas de colaboración u otros paquetes (los anidados). El paquete define un espacio de un nombre anidado, de modo que los elementos del mismo nombre pueden duplicarse dentro de varios paquetes [4].

3.2.1 Diagrama de Clases

El diagrama de clases de diseño de un sistema refleja los detalles que tienen que ver más concretamente con la implementación, mostrando la estructura interna del sistema, en cuanto a la información concerniente a cada una de las clases que forman el mismo, atributos y sus tipos de datos, métodos y sus tipos de datos de retorno y además brinda una representación gráfica de las relaciones entre todas las clases del sistema.

3.2.1.1 Paquete: Connection

Este paquete agrupa los componentes necesarios para establecer la comunicación entre los nodos de la aplicación (Figura 3.2). La clase IConnection define una interfaz que representa al canal de comunicación, sobre el cual serán transmitidos mensajes representados por la clase Message. IListener brinda una interfaz para establecer conexiones por parte del servidor.

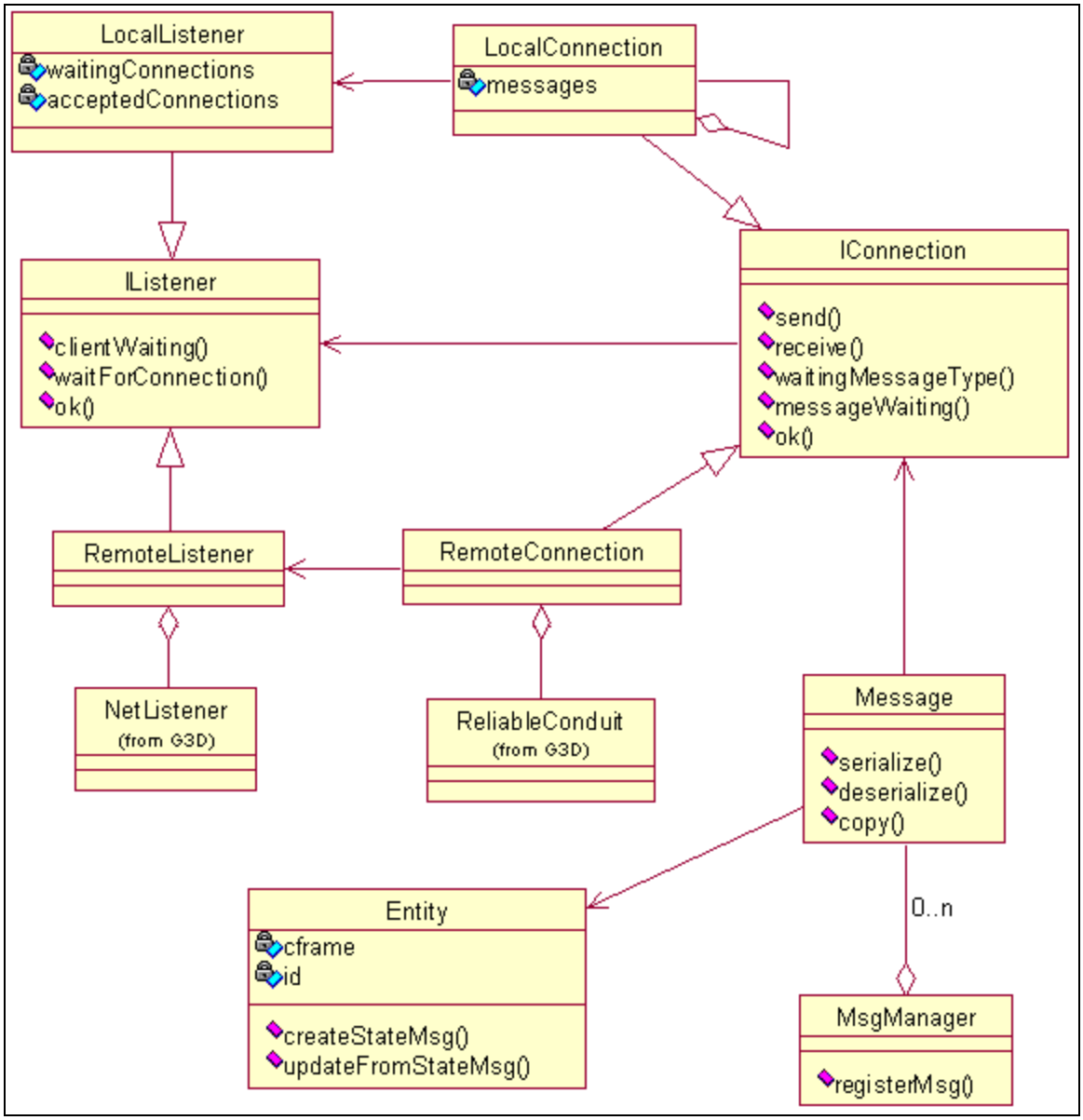


Figura 3.2 Diagrama de Clases del Paquete Connection

3.2.1.2 Paquete: Server

El paquete Server agrupa las funcionalidades de la sesión servidor de la arquitectura cliente servidor. La clase Server es el elemento más importante del paquete, la misma implementa los mecanismos y protocolos necesarios para llevar a cabo el funcionamiento de la aplicación en el servidor, gestiona la entrada y salida de clientes (ClientProxy) y la creación de nuevas entidades (Entity). Además ejecuta la simulación y la inteligencia artificial y en el caso del Servidor de Escucha (ListenServer) levanta un nuevo hilo de proceso (LServerThread). Procesa los mensajes de red recibidos ejecutando comandos (ServerCommand) y actualiza a los clientes enviándoles el estado global del juego.

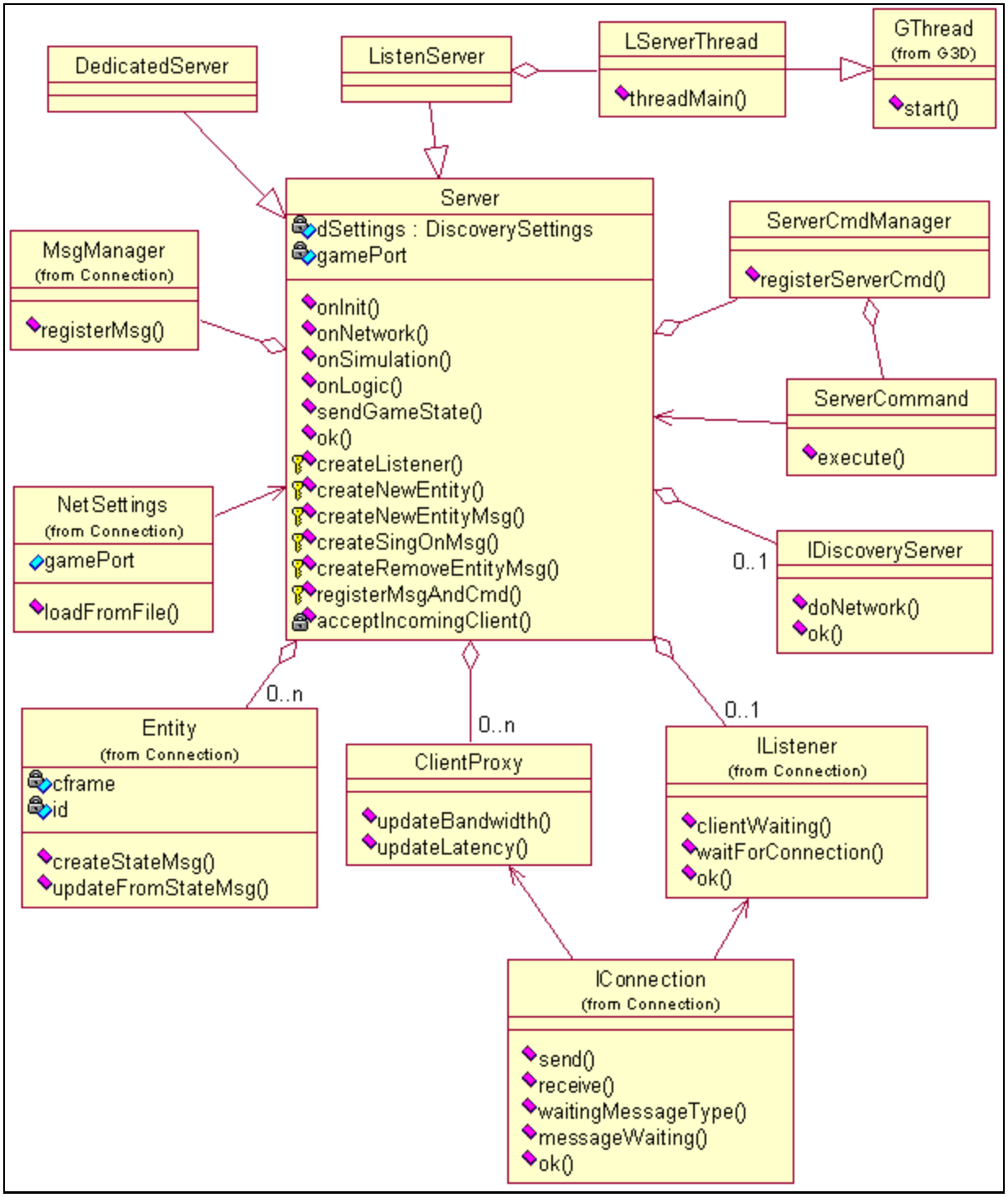


Figura 3.3 Diagrama de Clases del Paquete Server

3.2.1.3 Paquete: Client

Las funcionalidades de la sesión cliente de la arquitectura cliente servidor son agrupadas en este paquete. Client es la clase fundamental, que representa al cliente, el cual recibe la entrada del jugador y la envía al servidor por medio de ServerProxy, además actualiza la vista local del juego modificando a las entidades (Entity) al recibir mensajes del servidor, para lo cual ejecuta los comandos correspondientes (ClientCommand).

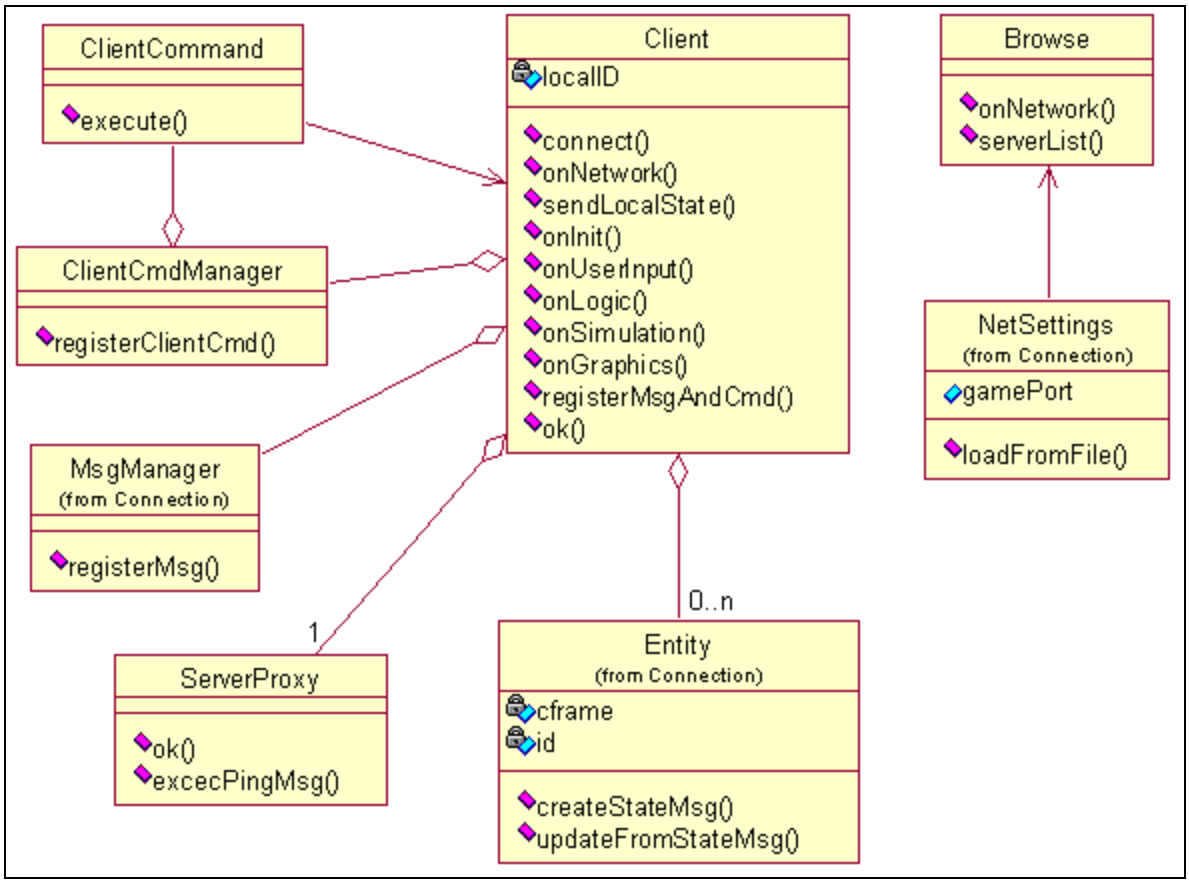


Figura 3.4 Diagrama de Clases del Paquete Client

3.3 Descripción de las Clases

La descripción de las clases describe los atributos y métodos que tiene una clase, brindando la información necesaria para lograr un completo entendimiento de la misma. En esta sección

proporcionamos la descripción de las clases Client y Server, que constituyen las principales clases del modelo pues son el núcleo de la arquitectura. La descripción del resto de las clases, cuyo conocimiento es indispensable para hacer uso del modelo creado, constituye parte de la documentación y se encuentra en formato web, generada con la herramienta Doxygen.

Nombre:	Client
Tipo de clase:	Controladora

Atributo	**Tipo**
server	ServerProxy
entityTable	EntityTable
localID	Entity::ID
msgManager	MsgManager
cmdManager	ClientCmdManager
usingUDP	bool
playerReady	bool

Responsabilidades:

Nombre:	**Descripción**:
Client (bool alsoUseUDP = false)	Constructor de la clase
onInit ()	Registra los mensajes y comandos en el cliente invocando Client::registerMsgAndCmd(). Si se sobreescribe este método entonces debe ser invocado mediante Client::onInit().
onNetwork ()	Recibe y procesa los mensajes enviados por el servidor.
onSimulation (RealTime rdt, SimTime sdt, SimTime idt)	Simulación por defecto de las entidades en el cliente ("fine-grained simulation", recordar que es el servidor el que debe ejecutar la simulación del juego). Si el cliente no realiza simulación este método se puede ignorar, y también se puede sobreescribir para una simulación personalizada.
onCleanup ()	
sendLocalState ()	Envía el estado local hacia el servidor, básicamente son las acciones del jugador.
connect (const NetAddress &address, NetworkDevice *nd)	Crea una conexión remota.
connect ()	Crea una conexión local y se conecta automáticamente al

	servidor local.
connect (LocalConnection *localConnection)	Se conecta a una conexión local.
ok () const	Conexión en correcto estado.
isPlayerReady () const	El jugador fue creado, o sea que se recibió el mensaje de CreateEntity con Entity::ID = localId. Si el jugador no ha sido creado entonces no se puede ejecutar sendLocalState() para enviar los controles del jugador.
sendClientJoinMsg ()=0	Envía un mensaje tipo ClientJoinMessage_MSG al servidor. El cliente debe invocar esta función luego de recibir un mensaje SignOnMessage_MSG.
registerMsgAndCmd ()=0	Registra los mensajes y los comandos del cliente específicos de la aplicación. Esta función es ejecutada en Client::onInit.

Nombre:	Server
Tipo de clase:	Controladora
Atributo	**Tipo**
dSettings	DiscoverySettings
gamePort	G3D::uint16
clientArray	Array <ClientProxy* >
advertisement	DiscoveryAdvertisement *
ds	IDiscoveryServer *
listener	IListener *
msgManager	MsgManager
cmdManager	ServerCmdManager
usingUDP	bool
nd	NetworkDevice *
entityTable	EntityTable

signedClientTable	ClientProxyTable
Responsabilidades:	
Nombre:	**Descripción:**
Server (NetworkDevice *_nd, const NetSettings &_netSettings, bool alsoUseUDP, DiscoveryAdvertisement *_advertisement	Constructor de la clase
onInit ()	Inicializa el Listener y los comandos del servidor
onNetwork ()	Si está permitido el broadcasting el servidor se anuncia en la red. Verifica por los clientes intentando conectarse. Recibe y procesa los mensajes enviados por los clientes. Chequea por clientes desconectados.
onLogic ()	Sobrescribir esta función para ejecutar la inteligencia artificial del juego.
onSimulation (RealTime rdt, SimTime sdt, SimTime idt)	Sobreescribir esta función para ejecutar la simulación física, pero si se desea calcular la latencia y el consumo de ancho de banda (debugMode = true) entonces se debe invocar mediante Server::onSimulation()
onCleanup ()	
sendGameState ()	Envía el estado del juego a todos los clientes.
ok () const	El mecanismo de escucha y el de discovery funcionan correctamente.
setDebugMode (bool mode)	Modifica la variable debugMode.
createListener ()=0	El servidor concreto crea un listener específico.Esta función es implementada en las subclases DServer el cual crea un RemoteListener y LServer que crea un listener dependiendo de si el juego es en red o no. Esta función es invocada en Server::onInit. Ver patrones Template Method y Factory Method de GOF.
createNewEntity (Message *joinMsg)=0	La creación de una entidad concreta depende de la aplicación final. Sobreescribir esta función devolviendo una nueva entidad que corresponderá con la entrada de un nuevo cliente al juego.Esta función es invocada en Server::acceptIncomingClient.
createNewEntityMsg (Message *msg, EntityRef entity)=0	La creación de un mensaje cuya semántica es la creación de una nueva entidad, lo que significa que un nuevo cliente se ha unido al juego. *Msg:* La instancia del mensaje a crear, *entity:* Entidad del servidor a crear del lado del cliente (que será el receptor del mensaje)
createSignOnMsg (Message *msg, Entity::ID id)=0	La creación de un mensaje cuya semántica es la entrada del cliente al juego (el cliente que recibe el mensaje es notificado). *Msg:* La instancia del mensaje a crear, id: Identificador de la

	entidad creada en el servidor que contiene información relacionada con el cliente que será inicializado.
createRemoveEntityMsg (Message *msg, Entity::ID entityId)=0	La creación de un mensaje cuya semántica es eliminar una entidad, pues un cliente se ha retirado. M*sg:* La instancia del mensaje a crear, *entityId:* El identificador de la entidad del servidor que debe ser eliminada.
registerMsgAndCmd ()=0	Registrar los mensajes y los comandos del servidor específicos de la aplicación. Esta función es ejecutada en Server::onInit.
onRemoveClient (G3D::uint clientProxyIndex)=0	Acciones específicas del servidor al remover un ClientProxy. Al ocurrir este evento, Server elimina la Entity correspondiente a este cliente e informa al resto de los clientes con un mensaje de tipo RemoveEntityMsg. *clienteProxyIndex*: Índice del cliente en el arreglo clientArray.
entities ()	Acceso a la tabla de entidades
newId ()	Genera un nuevo ID

3.4 Diagrama de Clases General

El Diagrama de clases general muestra cada una de las clases que lo componen así como las relaciones que existe entre cada una de ellas.

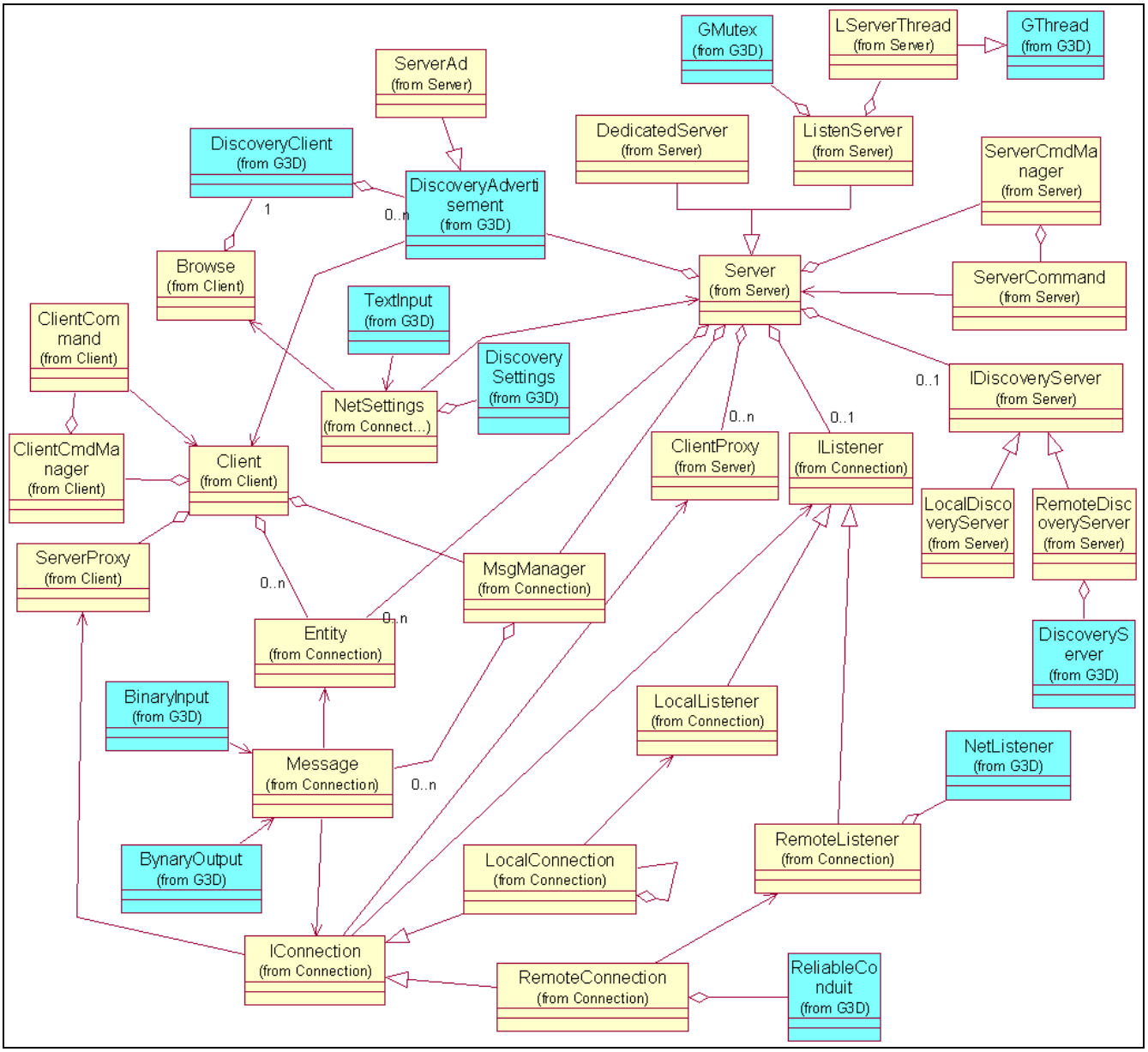


Figura 3.5 Diagrama de Clases General

3.5 Diseño por Casos de Uso

El diseño orientado a objetos tiene por objetivo definir las especificaciones lógicas del software que cumplan con los requisitos funcionales, basándose en la descomposición por clases de objetos. Los diagramas de colaboración expresan cómo interactúan estos objetos a través de mensajes, presentando el flujo de mensajes entre las instancias y la invocación de métodos. El enfoque del diseño orientado a objetos establece que los objetos colaboran a través de mensajes para llevar a cabo las tareas, este enfoque lo expresan directamente los diagramas de colaboración de UML [4].

3.5.1 CU Descubrir Servidor

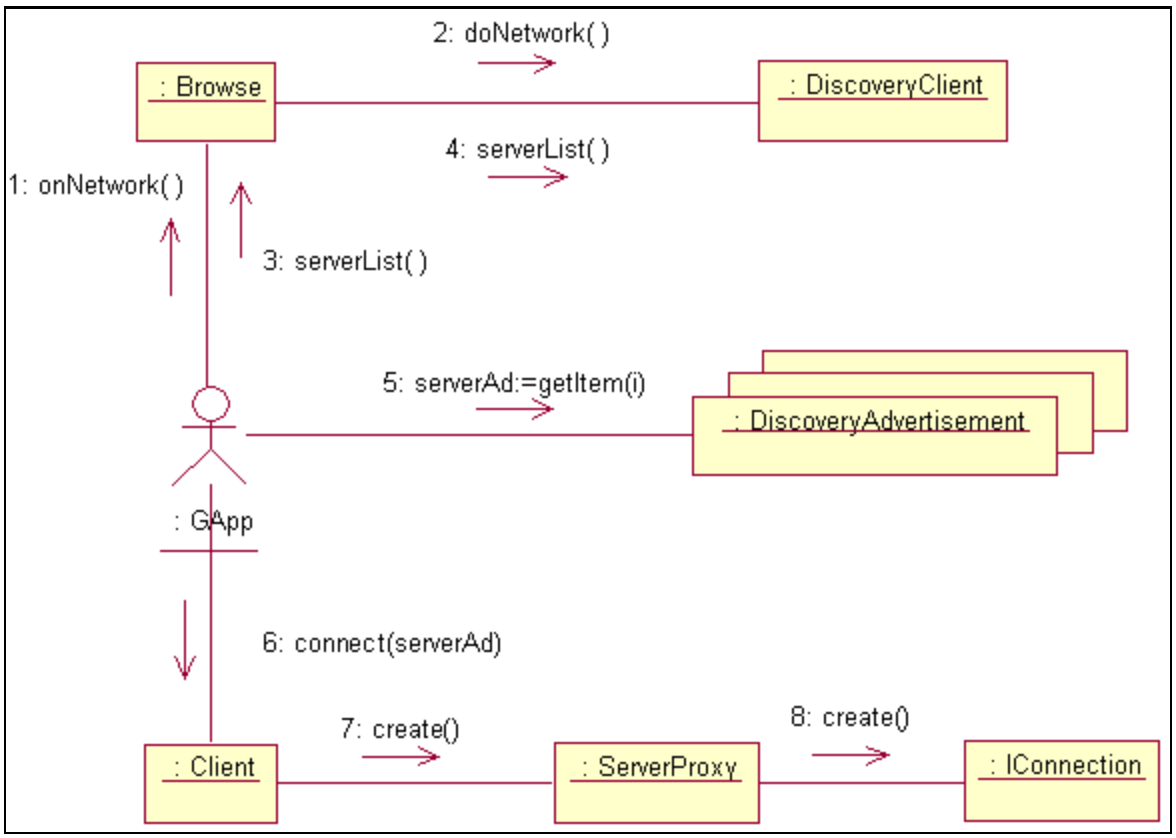


Figura 3.6 Diagrama de Colaboración del CU Descubrir Servidores

3.5.2 CU Gestionar Mensajes en el Cliente

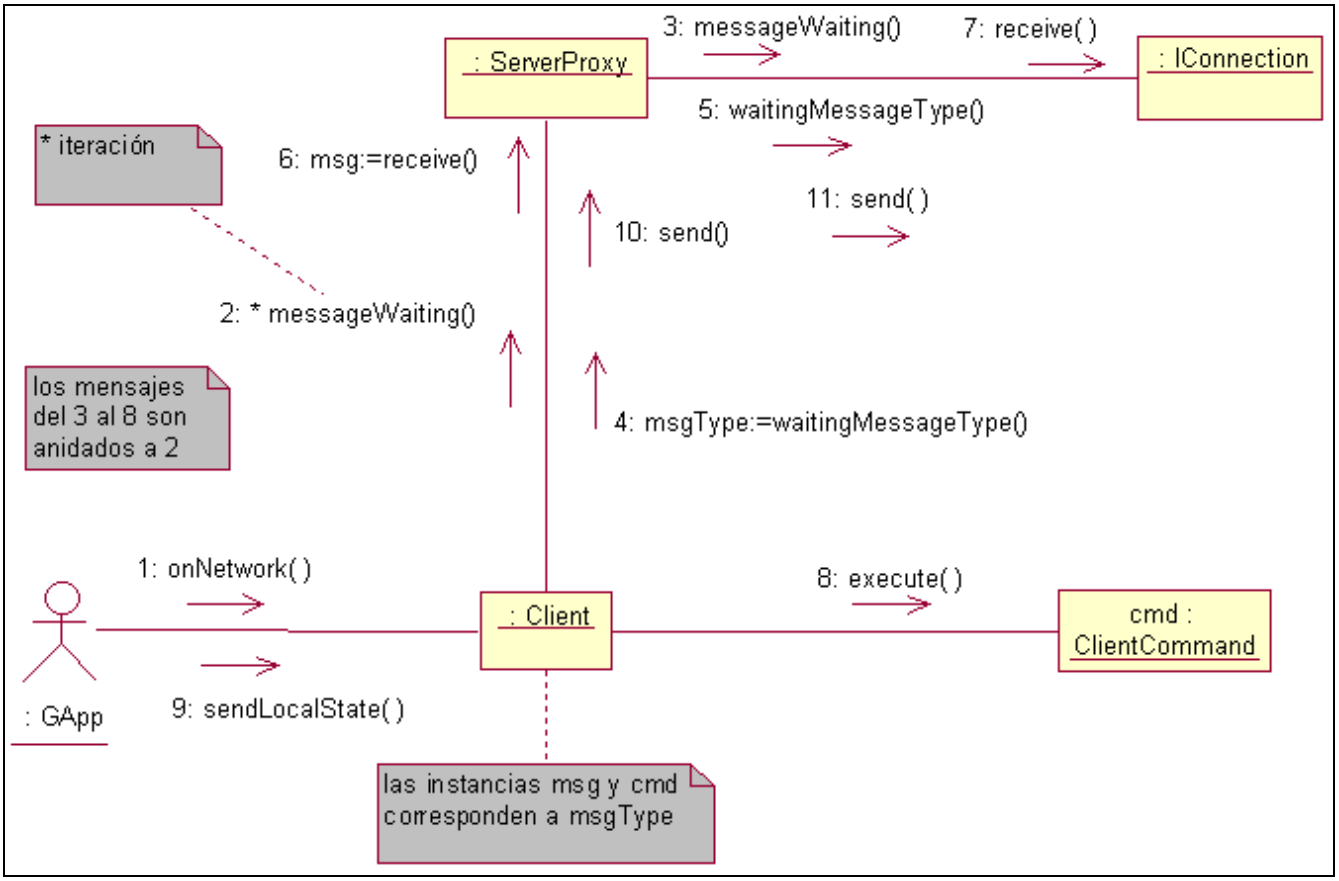


Figura 3.7 Diagrama de Colaboración del CU Gestionar Mensajes en el Cliente

3.5.3 CU Descubrir Clientes

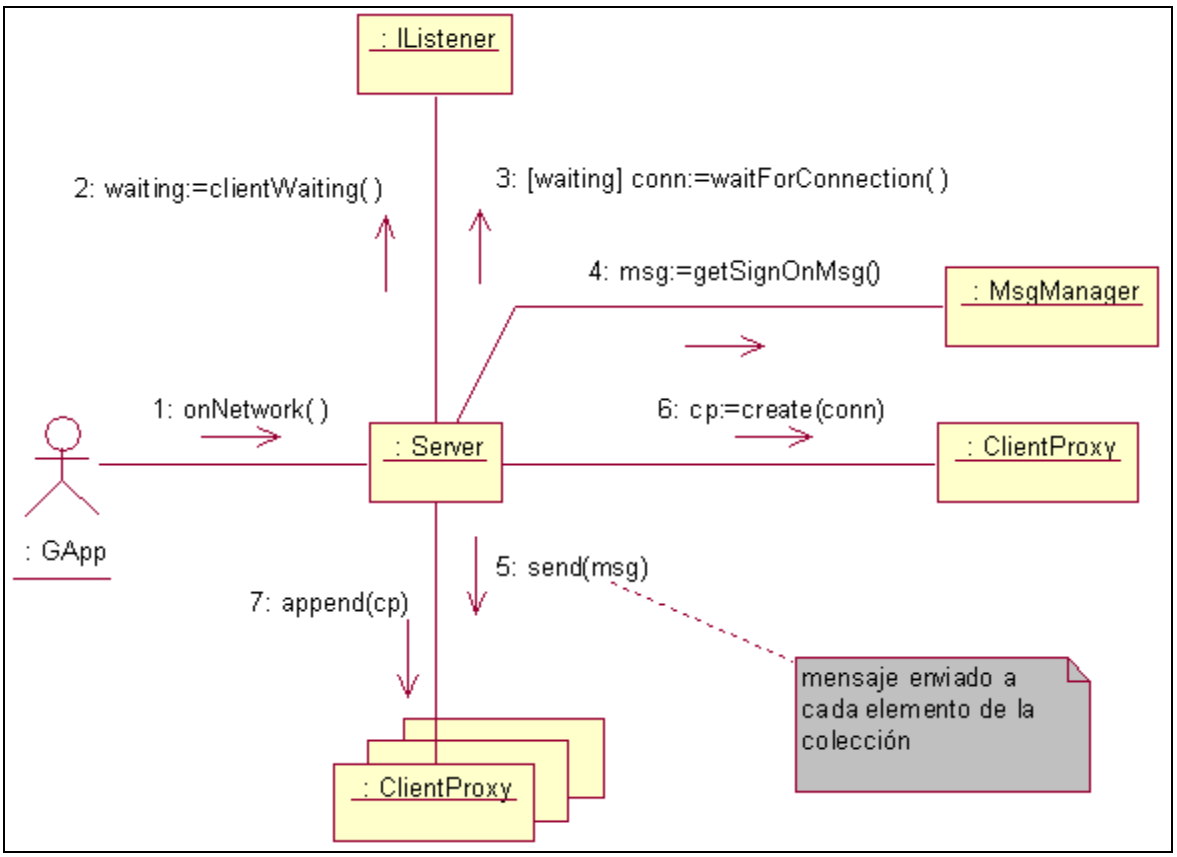


Figura 3.8 Diagrama de Colaboración del CU Descubrir Clientes

3.5.4 CU Gestionar Mensajes en el Servidor

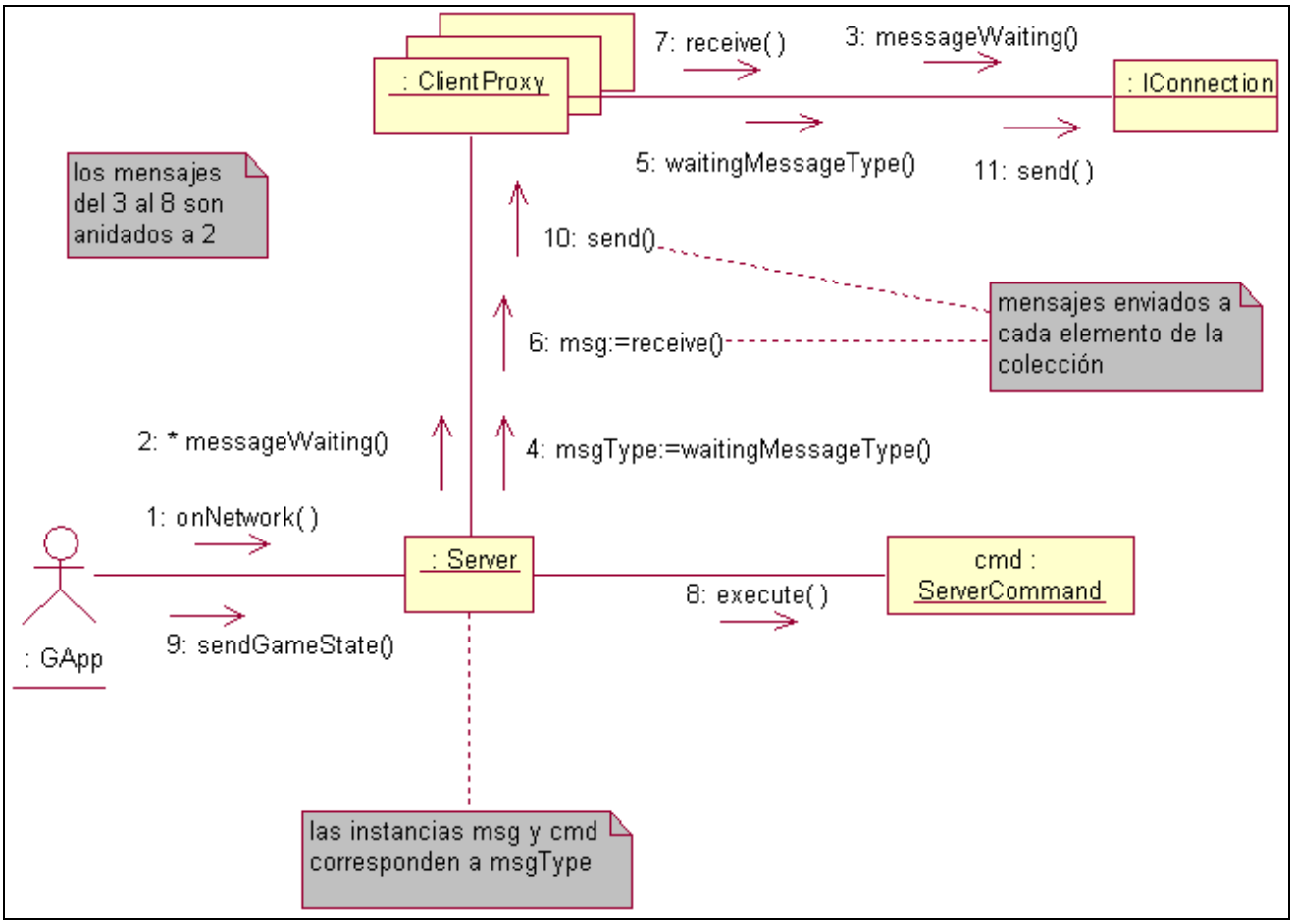


Figura 3.9 Diagrama de Colaboración del CU Gestionar Mensajes en el Servidor

3.5.5 CU Inicializar Servidor

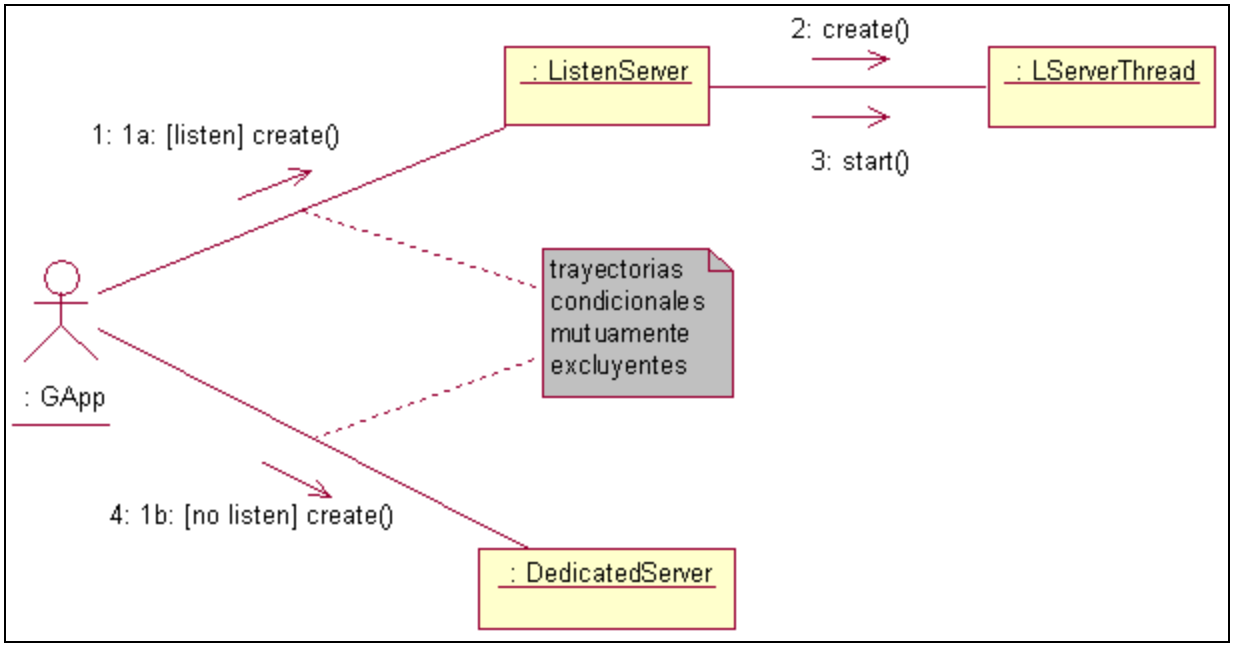


Figura 3.10 Diagrama de Colaboración del CU Inicializar Servidor

3.5.6 CU Medir Latencia y Consumo de Ancho de Banda

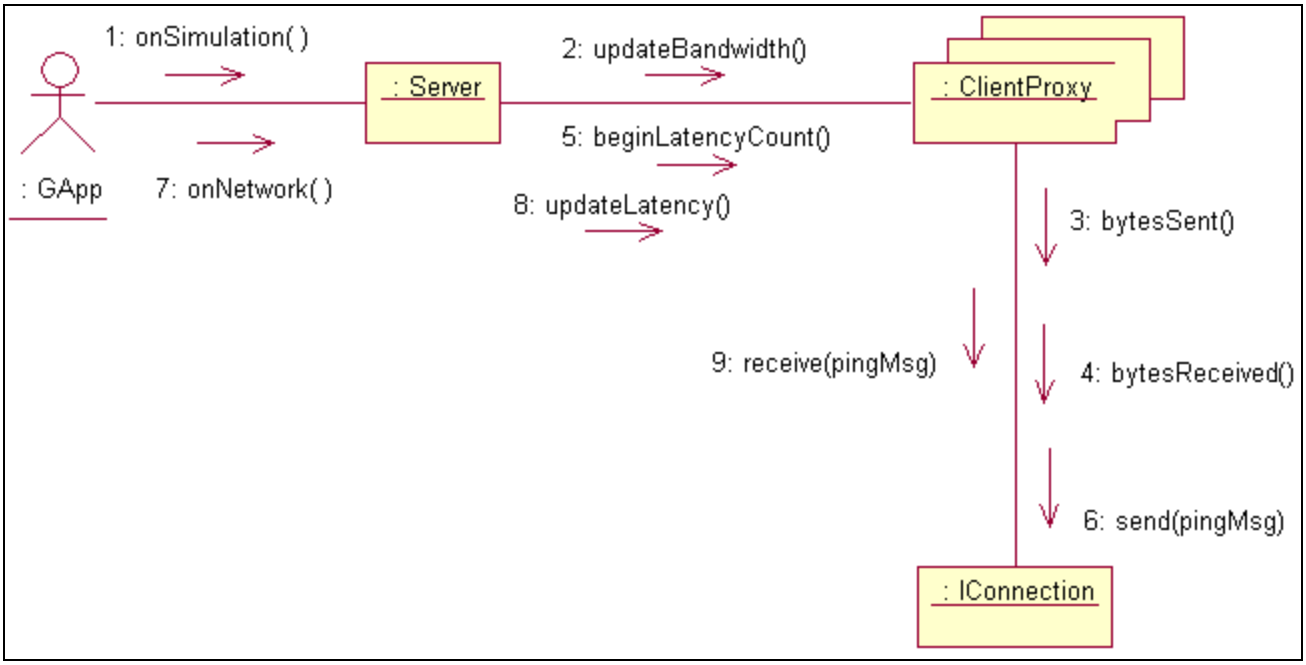


Figura 3.11 Diagrama de Colaboración del CU Medir latencia y consumo de ancho de banda

3.5.7 CU Cargar Configuración

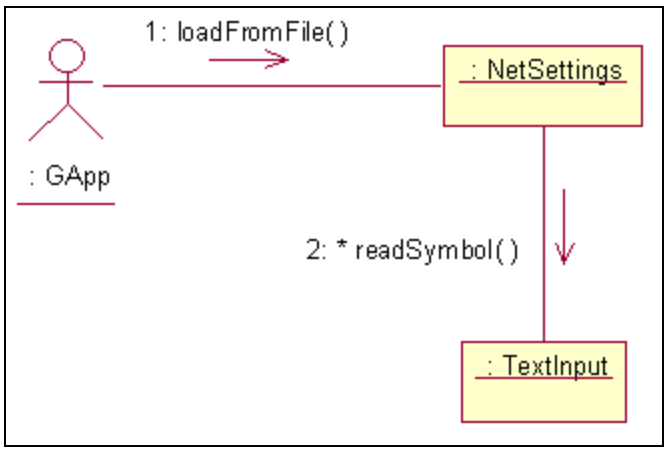


Figura 3.12 Diagrama de Colaboración del CU Cargar Configuración

3.6 El modelo de implementación

El modelo de implementación describe cómo los elementos del modelo de diseño se implementan en términos de componentes. Describiendo además cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados y cómo dependen los componentes unos de otros.

3.6.1 Diagrama de Despliegue

El modelo de despliegue es un modelo de objetos que describe la distribución física de un sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. En el Anexo I se muestra cómo sería desplegado un sistema desarrollado a partir de la arquitectura propuesta.

3.6.2 Diagramas de Componentes

El diagrama de componentes muestra las relaciones de dependencia entre las partes modulares de un sistema y encapsula la implementación. También en éste se pueden expresar las relaciones que existen entre los diferentes ficheros del sistema así como las bibliotecas empleadas para el desarrollo del mismo. En el Anexo II se muestran los diagramas de componentes del módulo desarrollado en el presente trabajo.

3.7 Estándares de codificación

Los estándares de codificación son reglas específicas a una lengua que reducen perceptiblemente el riesgo de que los desarrolladores introduzcan errores. Los estándares de codificación no destapan problemas existentes, evitan más bien que los errores ocurran. Los bugs frecuentes en programas

pueden ser detectados mucho antes o pueden ser incluso evitados totalmente. Durante el desarrollo, los estándares de codificación ayudan a los ingenieros a producir un código de alta calidad y a entender y utilizar el código de sus colegas. Pero también realzan considerablemente la capacidad de mantenimiento y rehúso a largo plazo del producto final. Tal práctica del control de bugs en el proceso del desarrollo mejora la calidad mientras que reduce el tiempo de desarrollo, el costo y el esfuerzo.

3.7.1 Reglas de codificación

Constantes

Regla: Los nombres de constantes se escriben con letras en mayúscula. En caso de estar compuesta por más de una palabra, éstas se separan por un carácter '_'.

Variables

Regla: Las variables se escriben con minúsculas, las variables con nombres compuestos, comienzan con la primera palabra enteramente en minúscula y el resto comenzando con mayúscula.

Parámetros (argumentos) de métodos

Regla: Cumplen con la misma regla de las variables.

Clases

Regla: Los nombres de clases comienzan con mayúsculas, con las palabras que la forman en minúsculas y separadas por mayúsculas.

Métodos miembros de clases

Regla: Los métodos miembros de clases siguen la notación idéntica que para las variables, es decir, comenzando con minúscula y separando las palabras con mayúsculas.

Nombres de enumerados

Regla: El nombre del tipo de enumerador cumple con la regla de las clases, y los valores con la misma notación que las constantes.

Comentarios

Regla*:* Se usan los comentarios en línea para facilitar la comprensión del código, sobre todo en procedimientos complejos.

Capítulo 4: Resultados

En este capítulo se presentan los resultados que se obtuvieron con el desarrollo de la arquitectura al elaborar una aplicación demostrativa que prueba todas las funcionalidades de la misma. Además se recogen las estadísticas de prueba para evaluar el funcionamiento del módulo.

4.1 Pruebas de la Arquitectura

Para realizar las pruebas al modelo de objetos creado se desarrolló una aplicación demostrativa que comprueba las funcionalidades que éste brinda. Esta aplicación permite conducir un auto, que es un modelo 3D, por un mundo virtual y a la vez interactuar con otros autos que son operados por una persona en otra computadora.

El demo consta de 3 modos fundamentales en los que se puede iniciar: *Servidor de Escucha* es el modo en que se crea un servidor al cual se conecta el jugador por medio de un cliente local, además se pueden conectar otros clientes por la red. *Buscar Servidores* no es más que un cliente el cual se conecta a un servidor remoto escogido de una lista de servidores online disponibles y por último *Servidor Dedicado* es el modo en el cual se crea un servidor al cual todos los clientes se conectan remoto, lleva todo el estado del juego pero no tiene ninguna interacción con el usuario que influya en dicho estado. En el Anexo III se pude consultar el diagrama de clases de la aplicación demo desarrollada.

4.2 Estadísticas de la prueba del demo

La aplicación fue realizada para explotar al máximo todos los recursos, haciendo uso de los algoritmos básicos: En el ciclo de juego del cliente por cada iteración es leída la entrada del usuario (por teclado), son recibidos y procesados todos los mensajes enviados por el servidor (quedando actualizado el estado local del juego), se envía al servidor la entrada del jugador y por último se renderean los

gráficos. Por otra parte en el ciclo de juego del servidor por cada iteración se reciben y procesan todos los mensajes enviados por todos los clientes, se actualiza el estado global del juego, se ejecuta la simulación física de las entidades (también se invoca la ejecución del modelo de inteligencia artificial) y por último es enviado a cada cliente el estado global del juego.

Es de notar que no se usaron técnicas de optimización tales como técnicas de filtrado, manipulación del tiempo o predicción que podrían incrementar el rendimiento de la aplicación pues contribuyen a consumir menos ancho de banda y a compensar la latencia.

Estadísticas de las pruebas:

Las características de la plataforma y de las computadoras en las cuales se ejecutó la aplicación son las siguientes:

Sistema Operativo: Windows 5.1 Service Pack 2

Procesador: Intel, Pentium 4, 3.00 GHz

Memoria RAM: 512 MB

GPU: Intel 945G

Versión de OpenGL: 1.4.0

Red: LAN 100 Mbps

El demo (Figura 4.1) se forzó a correr a 32 frames por segundo (fps), que es un valor aceptado para los videojuegos, fijar los fps es conveniente pues se disminuye el consumo innecesario de los recursos y se logra una sincronización en la comunicación de los clientes, ya que todos actualizan el ciclo del juego con la misma frecuencia sin que importe su capacidad de procesamiento. El requerimiento de ancho de banda para un nodo es el resultado de la suma de la cantidad de datos enviados y recibidos en un segundo, en el demo el tamaño de los mensajes para la actualización de un cliente es de 334 Bytes, estos mensajes son enviados en cada ciclo del juego, lo que da como resultado 10 KB/s aproximadamente. Éste es el factor para estimar el requerimiento de ancho de banda de la aplicación, que en los clientes es una función lineal y en el servidor cuadrática que depende del número de jugadores(N), esto es: 10*N KB/s para cada cliente y 10*N*N KB/s para el

servidor. En el demo se hace una medición real del tamaño de los mensajes enviados y recibidos en una unidad de tiempo, que coincidió con el estimado. La latencia es el tiempo que demoran los mensajes en ser transmitidos de un nodo a otro, para el cálculo de este factor se realizó una prueba en la aplicación midiendo el tiempo real que demoraba la respuesta de un mensaje, esto es el tiempo de ida y vuelta, que es un medidor de la latencia la cual resulta ser aproximadamente la mitad de este tiempo. La carga del servidor es el porciento de uso del CPU, la cual fue registrada en cada momento que se conectaba un cliente, lo cual permite comparar como se comporta el servidor y como se ve afectado el frame rate de los clientes.

Tabla 4.1 Estadísticas de la aplicación demostrativa

# PC cliente	Carga del servidor (%)	FPS en cada cliente	Requerimiento de ancho de banda por cliente (KB/s)	Requerimiento de ancho de banda para el servidor(KB/s)
1	8	32	10	10
2	13	32	20	40
3	16	32	30	90
4	19	32	40	160
5	23	32	50	250
6	29	32	60	260
7	34	32	70	490
8	43	32	80	640
9	50	32	90	910
10	60	32	100	1000

Los resultados alcanzados son satisfactorios, pues dados los recursos disponibles sobre los que se realizaron las pruebas del demo se obtuvieron valores aceptables de los parámetros que caracterizan a las aplicaciones de red en tiempo real, en especial los videojuegos. La latencia registrada en el

demo fue estable, con un valor de aproximadamente 15 milisegundos lo que es inferior a la máxima recomendada para este tipo de aplicaciones que oscila entre los 100 y 200 milisegundos. Además se puede destacar que el consumo de ancho de banda es considerablemente reducible, pues el tamaño del mensaje de actualización para un cliente se podría reducir aproximadamente 5 veces, ya que en el demo son enviados muchos datos innecesarios.

Figura 4.1 Aplicación Demostrativa

Conclusiones

La realización de este Trabajo de Diploma cumplió con los objetivos propuestos al logar transferir información entre distintas computadoras con el modelo de objetos diseñado. Se hizo un estudio de las técnicas de transmisión y de las principales arquitecturas de comunicación empleadas en el desarrollo de videojuegos, profundizando en la arquitectura cliente servidor, aspecto esencial para satisfacer las necesidades planteadas. Además se hizo un análisis del comportamiento de la latencia y el consumo de ancho de banda en esta arquitectura.

Luego de terminado el estudio de forma general y definir el dominio se comenzó con la captura de requisitos, a través de los cuales se realizaron los Casos de Uso para describir los procesos que daban solución a las funcionalidades expresadas en los requerimientos. Con estos artefactos se comenzaron a elaborar los primeros diseños del modelo, que gradualmente fue mejorado con el estudio y aplicación de patrones de diseño.

Con la realización de una aplicación demo, es apreciable destacar algunos aspectos importantes que caracterizan al módulo desarrollado, como son:

1- El uso del módulo resultó eficiente, pues a pesar de no usar ninguna técnica de optimización en el demo, las estadísticas de ancho de banda y latencia fueron admisibles.

2- Es transparente para el desarrollador de un videojuego la posibilidad de si éste va a ser en red o no, si el servidor va a ser dedicado o va a ejecutarse en un mismo proceso que el de un cliente, pues la arquitectura es implementada mediante un modelo de objetos con un diseño flexible y modular, orientado a tales características.

3- Es posible su uso en los sistemas operativos Unix, Mac OS y Windows.

Recomendaciones

Una posible ampliación de este trabajo es aumentar el conjunto de funcionalidades que el modelo brinda, como en caso de que el servidor se desconecte, el sistema se restablezca creando un nuevo servidor en un cliente. Además de incorporarle técnicas para compensar la latencia y disminuir el consumo de ancho de banda sin hacer un modelo rígido que afecte la escalabilidad y flexibilidad del sistema.

Bibliografía

Referenciada

[1]. **Money, CNN.** Video Game Sales Jump 8 Percent in 2004. [Citado el: 15 de Marzo de 2008] http://money.cnn.com/2005/01/18/technology/gamesales/.

[2]. **Neyland, DL.** *Virtual Combat: A Guide to Distributed Interactive Simulation*. Stackpole Books, Mechanicsburg, PA, USA. 1997.

[3]. **Rosanigo, I.Z.B.** *Maximizando reuso en software para Ingeniería Estructural. Modelos y Patrones, en Facultad de Informática.* La Plata : Universidad Nacional de La Plata, 2000.

[4]. **Craig, Larman.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* 1999.

[5]. **Michael, Abrash.** *Ramblings in Realtime.* 2000.

[6]. **McGuire, Morgan.** *G3D Manual and Library source code.*2007

[7]. **Armitage, Grenville; Claypool, Mark y Branch, Philip.** *Networking and Online Games: Understanding and Engineering Multiplayer Internet Games.* England : John Wiley & Sons,Ltd, 2006.

[8]. **Smed, Jouni y Hakonen, Harri.** *Algorithms and Networking for Computer Games.* Finlandia : University of Turku, 2006.

[9]. **Pellegrino, Joseph D.** *Bandwidth requirement and state consistency in three multiplayer game architectures.* Newark : Departamento de Ciencia de la Computación, Universidad de Delaware.

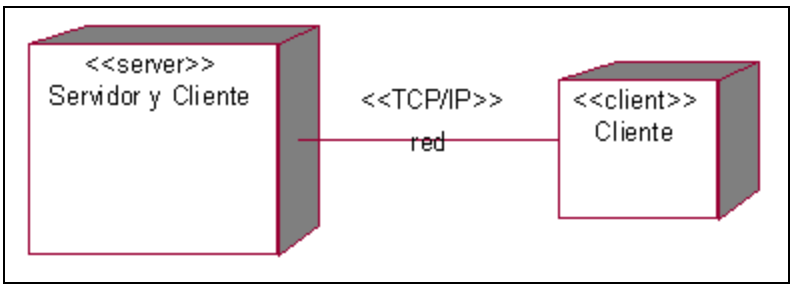
[10]. **Cronin, Eric; Filstrup, Burton y Kurc, Anthony.** *A Distributed Multiplayer Game Server System.* Universidad de Michigan, Mayo 4, 2001.

Consultada

- **Abdelkhalek, Ahmed y Bilas, Angelos.** *Parallelization and Performance of Interactive Multiplayer Game Servers.* Ontario,Canada : Universidad de Toronto, 2003.
- **Booch, Grady.** *Object- Oriented Analysis and Design.* Santa Clara, California, 1998.
- **Hawkins, Kevin y Astle, Dave.** *OpenGL Game Programming.* EEUU, 2001.
- **Hernández León, Rolando Alfredo y Coello González, Sayda.** *El Paradigma Cuantitativo de la Investigación Científica.* Ciudad de la Habana : Universidad de las Ciencias Informáticas, Noviembre,2002.
- **Gargolinski, Steven y Pierre, Christopher St.** *Better Game Server Selection.* Massachusetts : Worcester Polytechnic Institute, 2004.
- **Gamma, Erich; Helm, Richard; Johnson, Ralph** y **Vlissides, John.** *Design Patterns. Elements of Reusable Object-Oriented Software.* 1998.
- **Duch i Gavaldà, Jordi y Tejedor Navarro, Heliodoro.** *Sonido, interacción y redes.* Barcelona, 2008.
- **Género de videojuegos**. *Wikipedia* [Citado: Abril 10, 2008.] http://es.wikipedia.org/wiki/G%C3%A9nero_de_videojuegos
- **Industria de los videojuegos**. *Wikipedia* [Citado el: Enero 29, 2008.] http://es.wikipedia.org/wiki/Industria_de_los_videojuegos.
- **Sincronización por Frame Rate.** *VJuegos*.[Citado el : Mayo 02,2008.] http://prog.vjuegos.org/index.php?option=com_content&task=view&id=124&Itemid=148

Anexos

Anexo I: Diagrama de Despliegue



Videojuego multijugador

Anexo II: Diagramas de Componentes

Diagrama de Componentes del Paquete *Connection*

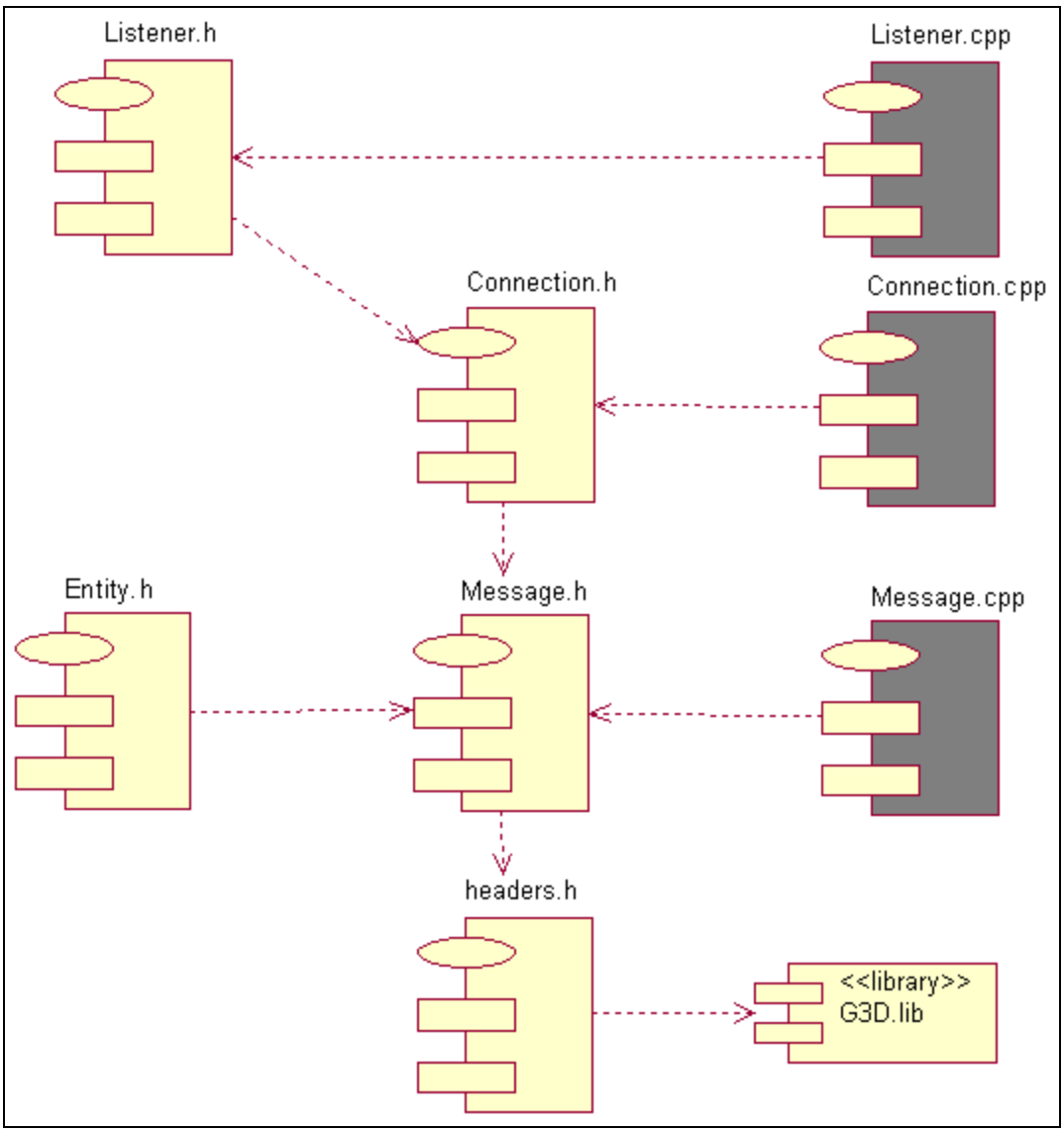


Diagrama de Componentes del Paquete *Server*

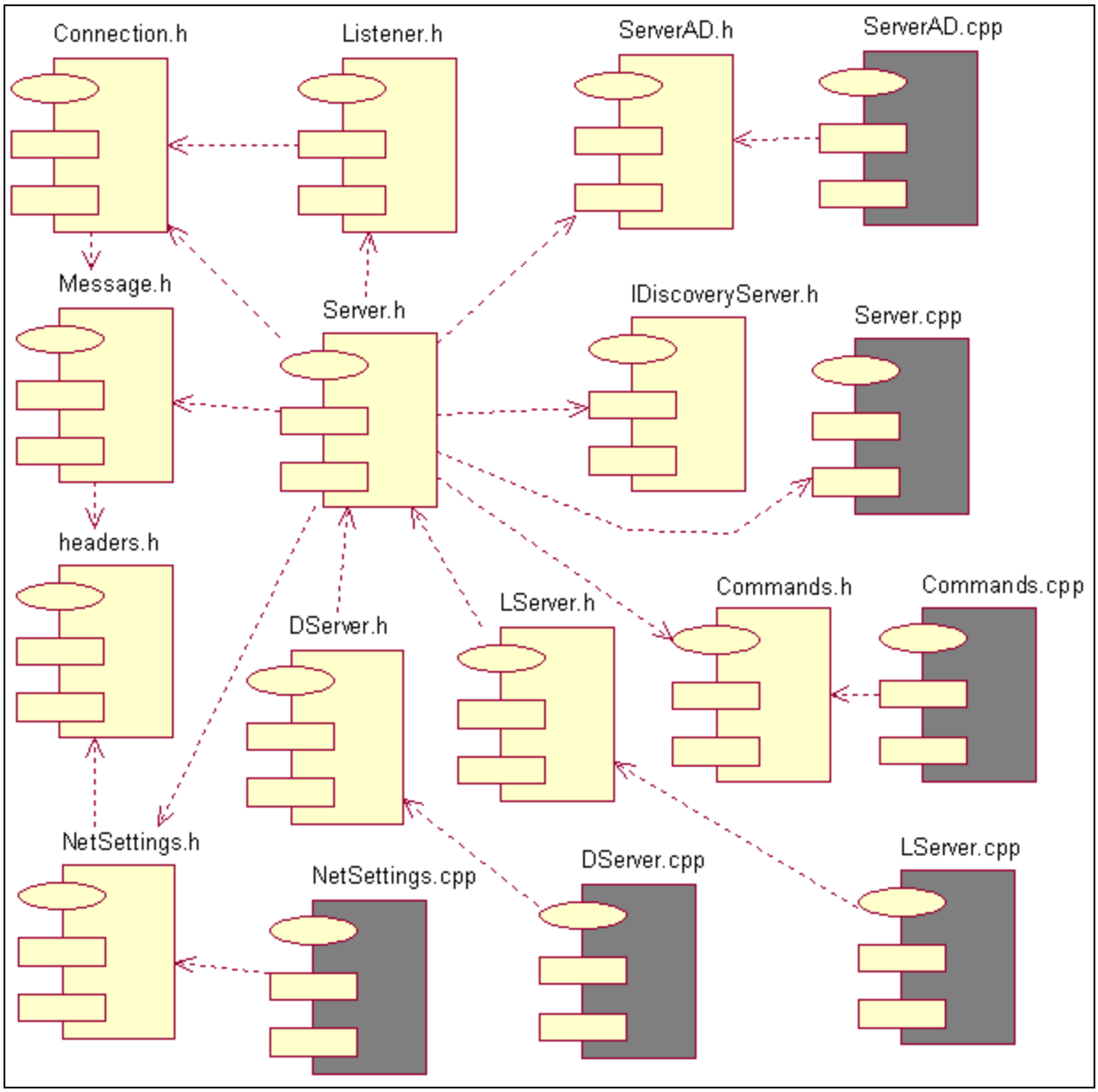
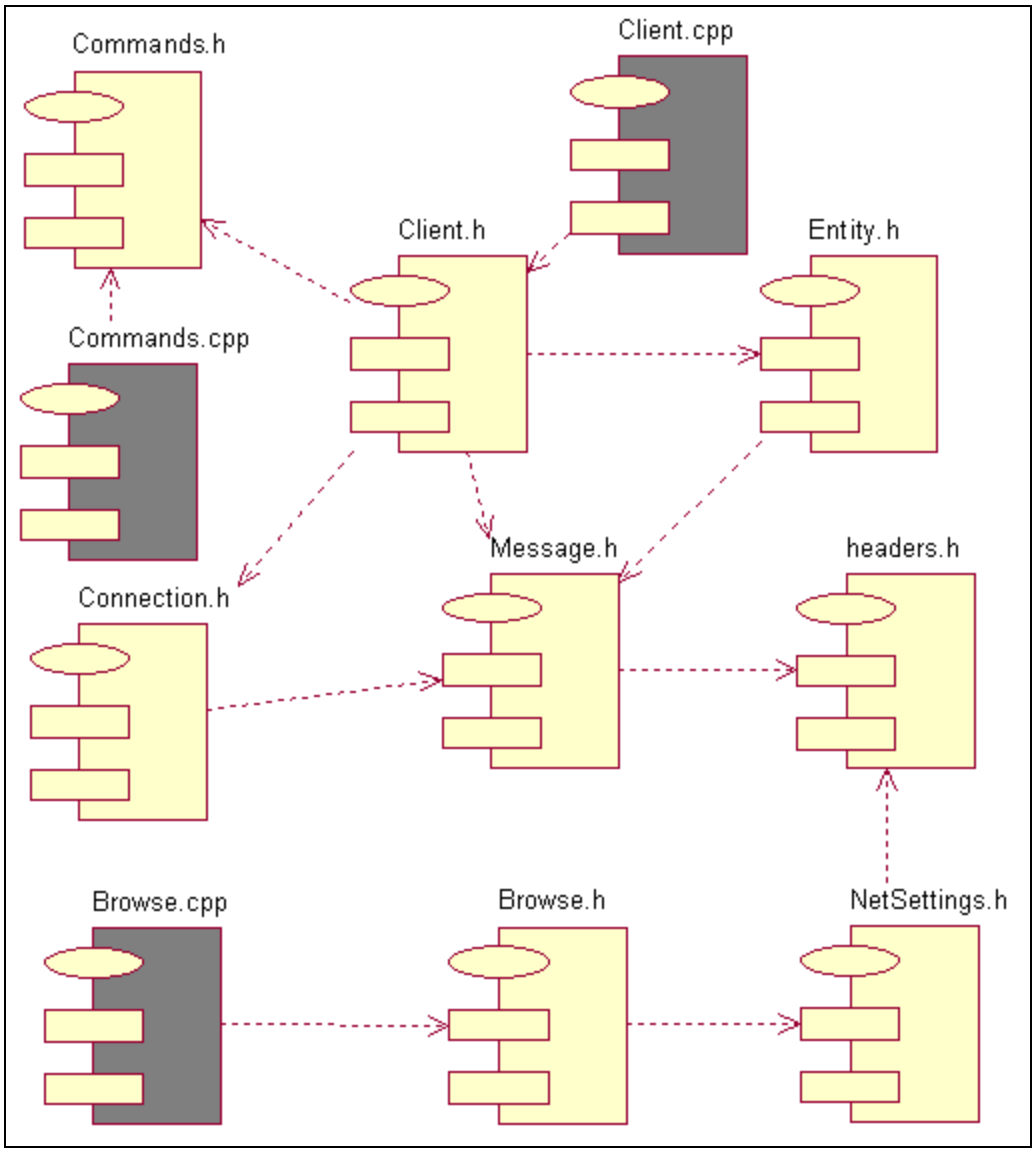
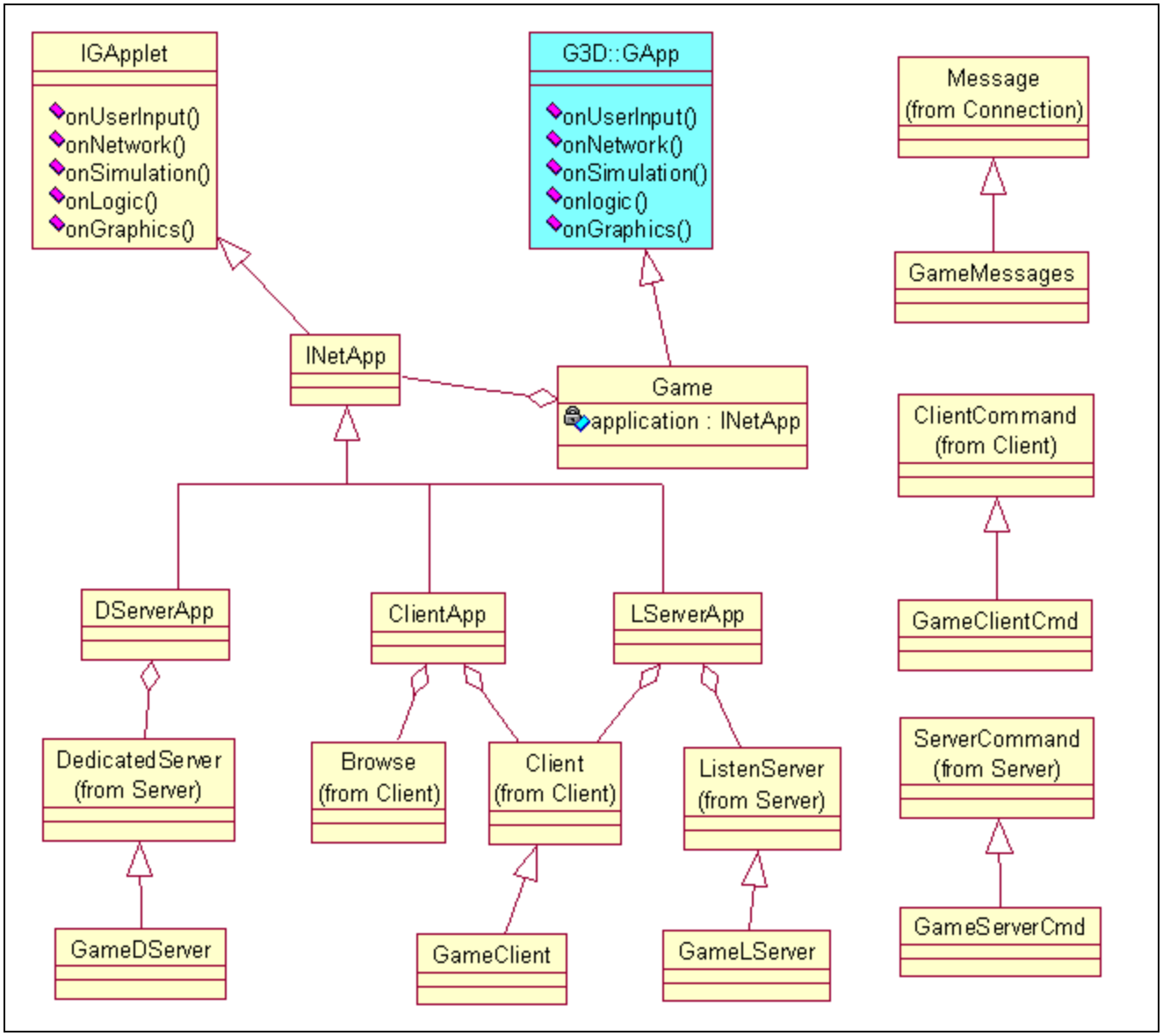


Diagrama de Componentes del Paquete *Client*



Anexo III: Diagramas de Clases de la Aplicación Demo



Glosario de Términos y Siglas

A

***Ancho de Banda*:** En las redes de ordenadores, el ancho de banda es sinónimo para la taza de transferencia de datos, o sea, la cantidad de datos que se pueden llevar de un punto a otro en un período de tiempo dado (generalmente un segundo). En ocasiones se expresa en bits por segundo (bps) o bytes por segundo (Bps).

***API*:** Application Programming Interface. Conjunto de rutinas que proveen al programador de una interfaz (de software) para acceder al hardware.

***Aplicación*:** Es un tipo de programa informático diseñado para facilitarle al usuario la realización de un determinado tipo de trabajo.

******

B

Bug: Error en la escritura de un programa que produce un defecto en el sistema.

******

C

***Consola*:** Referido a videoconsola.

***CPU*:** Acrónimo utilizado para abreviar Central Processing Unit que significa Unidad de Procesamiento Central. Parte de un computador que controla todas las demás partes. Recoge instrucciones de la memoria y las decodifica. Esto le permite transferir datos hacia o desde la memoria o activar los periféricos para que realicen acciones. También conocido como procesador o microprocesador siendo este último manufacturado con circuitos integrados.

******

D

Demo: Es una abreviatura de la palabra *demonstration* (demostración en español). Se refiere a una aplicación demostrativa.

Debugging: Proceso para la corrección de errores en un programa, generalmente haciendo uso de una herramienta (otro programa) conocida como debugger.

DirectX**:** Colección de APIs creadas y recreadas para facilitar las complejas tareas relacionadas con la programación de juegos en la plataforma Microsoft Windows.

Display**:** También conocido como monitor, dispositivo electrónico que muestra imágenes producidas por un ordenador.

******

E

Engine 3D**:** Conocido también como Game Engine (Motor de Juego), es el software principal (core o núcleo) de un videojuego, o más general de una aplicación gráfica en tiempo real (del inglés real time).

Entorno Virtual**:** Referido a Mundo virtual. Simulación de mundos o entornos denominados virtuales en los que el hombre interacciona con la máquina en entornos artificiales semejantes a la vida real. Ejemplo de aplicaciones desarrolladas sobre mundos virtuales son los simuladores y los videojuegos.

Escalabilidad**:** En general, es la capacidad que tiene un sistema informático de cambiar su tamaño o configuración para adaptarse a las circunstancias cambiantes.

******

F

***Frame*:** Fotograma o cuadro, una imagen particular dentro de una sucesión de imágenes que componen una animación. La continua sucesión de estos fotogramas producen a la vista la sensación de movimiento, fenómeno dado por las pequeñas diferencias que hay entre cada uno de ellos.

******

G

***G3D*:** Graphics Three Dimensional. Engine 3D disponible como código abierto (*del inglés* Open Source) y libre (*del inglés* free) con licencia BSD desarrollado en el lenguaje de programación C++. G3D Proporciona una base sólida y altamente optimizada para el desarrollo en general de aplicaciones gráficas y extiende las funcionalidades de OpenGL y sockets incluyendo formatos de modelos 3D y bibliotecas utilitarias.

GPU: Acrónimo utilizado para abreviar Graphics Processing Unit, que significa unidad de procesado de gráficos. Procesador dedicado exclusivamente al procesamiento de gráficos, para aligerar la carga de trabajo del procesador central en aplicaciones 3D interactivas como los videojuegos.

******

I

***IP*:** Internet Protocol (Protocolo de Internet), es un protocolo orientado a datos usado tanto por la fuente como por el destino para la comunicación de datos a través de una red de paquetes conmutados.

******

J

***Juego*:** Referido a videojuego

******

L

***Latencia*:** En redes informáticas de datos se denomina latencia a la suma de retardos temporales dentro de una red. Un retardo es producido por la demora en la propagación y transmisión de paquetes dentro de la red.

***LAN*:** Local Area Network (Red de Área Local). Es una red de ordenadores de extensión limitada (hasta algunos pocos kilómetros).

******

M

Mensaje: En el sentido más general, es el objeto de la comunicación. Está definido como la información que el emisor envía al receptor a través de un canal determinado o medio de comunicación.

Modularidad: Característica por la cual un programa de ordenador está compuesto de partes separadas llamadas módulos. Generalmente los módulos pueden ser reutilizados en varias partes de un programa o en otros programas.

***Módulo*:** Es una parte de un programa, o más general un componente de un sistema que tiene una interfaz bien definida para interactuar con otros módulos.

***Multilenguaje*:** Que soporta varios lenguajes.

***Multiplataforma*:** Se refiere los programas, sistemas operativos, lenguajes de programación, u otra clase de software, que puedan funcionar en diversas plataformas.

***Multihilo*:** Capacidad de un programa de lidiar con varios hilos de ejecución concurrentemente.

******

N

Nodo: Dispositivo conectado a una red, por ejemplo una computadora.

******

O

***OpenGl*:** Open Graphics Library, Biblioteca gráfica 3D desarrollada por Silicon Graphics Incorporated.

******

P

***Paquete*:** Cantidad mínima de datos que se transmite en una red o entre dispositivos. Tiene una estructura y longitud distinta según el protocolo al que pertenezca. También llamado TRAMA.

***Pipeline*:** Canal de proceso y comunicación en paralelo.

Paralelismo en hardware: Capacidad de ejecutar un programa de manera paralela tomando en consideración las limitaciones del hardware con que va a ser ejecutado.

Plataforma: Combinación de hardware y software usada para ejecutar aplicaciones.

******

R

***Render o Renderización*:** Es el proceso de generar una imagen desde un modelo. La palabra renderización proviene del inglés render, y no existe un verbo con el mismo significado en español, por lo que es frecuente usar las expresiones renderizar o renderear. En términos de visualizaciones en ordenador, más específicamente en 3D, la "renderización" es un proceso de cálculo complejo desarrollado por un ordenador destinado a generar una imagen 2D a partir de una escena 3D.

RUP: Rational Unified Process (Proceso Unificado de Desarrollo). Metodología para el desarrollo de Software.

******

S

Serialización: Proceso de codificación de un Objeto (programación orientada a objetos) en un medio de almacenamiento (como puede ser un archivo, o un buffer de memoria) con el fin de transmitirlo a través de una conexión en red como una serie de bytes.

***Sistema*:** Sistema informático, en general es un conjunto de hardware, software y de un soporte humano. Una aplicación ejecutándose sobre una computadora se considera un sistema.

***Sockets*:** Designa un concepto abstracto por el cual dos programas (posiblemente situados en computadoras distintas) pueden intercambiarse cualquier flujo de datos, generalmente de manera fiable y ordenada. Un socket queda definido por una dirección IP, un protocolo y un número de puerto.

Software Concurrente: Programa que tiene más de una línea lógica de ejecución, es decir, es un programa que parece que varias partes del mismo se ejecutan simultáneamente. Un programa concurrente puede correr en varios procesadores simultáneamente o no.

******

T

***Tiempo Real*:** Se refiere a aquellos sistemas que interactúan activamente con un entorno con dinámica conocida en relación con sus entradas, salidas y restricciones temporales, para darle un correcto funcionamiento de acuerdo con los conceptos de estabilidad, controlabilidad y alcanzabilidad.

***TCP*:** Transmission Control Protocol, es uno de los protocolos de comunicaciones sobre los que se basa
Internet. Posibilita una comunicación libre de errores entre ordenadores en Internet.

******

U

***UML*:** Unified Modeling Language. Es una notación estándar para modelar objetos del mundo real como primer paso en el desarrollo de programas orientados a objetos. Es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema de software.

***UDP*:** User Datagram Protocol: Es un protocolo del nivel de transporte de red basado en el intercambio de datagramas. Permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión. No tiene confirmación, ni control de flujo, por lo que los paquetes pueden adelantarse unos a otros; y tampoco se sabe si ha llegado correctamente la información.

******

V

***Videojuego*:** Programa informático creado para el entretenimiento, basado en la interacción entre una o varias personas y un aparato electrónico llamado consola (o videoconsola) que ejecuta el videojuego. Estos recrean entornos y situaciones virtuales en los cuales el jugador puede controlar a uno o varios personajes (o cualquier otro elemento de dicho entorno), para conseguir uno o varios objetivos por medio de unas reglas determinadas.

***Videojuego Multijugador*:** Videojuego diseñado para que puedan participar varios jugadores. Más concretamente este término se suele utilizar para definir videojuegos que hacen uso de Internet u otro tipo de red.

***Videoconsola*:** Sistema electrónico de entretenimiento para el hogar que ejecuta juegos electrónicos (videojuegos) que están contenidos en cartuchos o discos ópticos.

******

W

***WAN*:** Wide Area Network (Red de Área Amplia), es un tipo de red de computadoras capaz de cubrir distancias desde unos 100 hasta unos 1000 km, dando el servicio a un país o un continente.