UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 5 ENTORNOS VIRTUALES

Módulo para Transmisión de Datos en Sistemas de Realidad Virtual

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas

Autores: Arián Antonio Núñez Alonso

Sachel Hernández Pérez

Tutores: Ing. Yanoski R. Camacho Román

Lic. Andrés Martínez Morales

Ciudad de la Habana

Mayo 2007

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos al Proyecto Herramientas de Desarrollo para Sistemas de Realidad Virtual de la Facultad 5 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Autores:

Arian A. Núñez Alonso

_____

Sachel Hernández Pérez

_____

Tutor:

Yanoski Camacho Román

_____

DATOS DE CONTACTO

Nombre y Apellidos: Yanoski Rogelio Camacho Román
Edad: 26 años
Ciudadanía: cubano
Institución: Universidad de las Ciencias Informáticas (UCI)
Título: Ingeniero en Informática
Categoría Docente: Profesor Instructor
E-mail: rcamacho@uci.cu

Graduado de la CUJAE, con tres años de experiencia en el tema de la Gráfica Computacional y líder de un proyecto de Realidad Virtual en la Universidad de las Ciencias Informáticas.

Dedicatoria

A mis padres. A mi hermano.
A toda mi familia.

Arián

A Eglyta. A mis abuelos. A mis padres.
A mi hermano. A toda mi familia.

Sachel

Agradecimientos

Muchas personas han contribuido, de una u otra forma, a la elaboración de este trabajo. A todas ellas queremos expresar nuestros más sinceros agradecimientos. Quisiéramos agradecer a nuestro Comandante en Jefe Fidel por habernos dado la oportunidad de realizar estos estudios y por su eterna fe en la juventud. De forma especial, agradecer la ayuda del profesor y tutor, Ing. Yanoski R. Camacho Román, dispuesto en todo momento a atender las dudas y propuestas que iban surgiendo. Al Ing. Fernando Jiménez por sus conocimientos en ingeniería de software. A los miembros del proyecto productivo UCI-SIMPRO con los que compartimos, cuya paciencia y comentarios, sirvieron de apoyo en los momentos más extenuantes.

De Arian

A mis padres por el cariño, la confianza y la atención infinita que me han brindado a lo largo de todos estos años. A mi hermano que ha estado pendiente a entregar su ayuda en todo momento. A Yuremis por el apoyo y la comprensión que mantuvo a mi lado en las buenas y malas.

De Sachel

A mis padres quienes me infundieron la ética y los valores que guían mi transitar por la vida; por confiar, creer y apoyar siempre mis decisiones. A mi abuelo quien siempre estuvo muy orgulloso y me sirvió de inspiración para obtener mejores resultados. A mi hermano que es tan importante para mí, por apoyarme y ayudarme en todo. A Yaima, la cuál me ha brindado su cariño, compresión y apoyo en los momentos más difíciles, cuya paciencia, ha supuesto en todo momento un gran estímulo. A toda mi familia por seguir siempre mis pasos y apoyar cada una de mis decisiones. A mis amigos y hermanos Raissel y Osley quienes siguieron en todo momento los pasos de este trabajo y tanto me han apoyado en estos años. A todos mis amigos que de manera general mucho me han enseñado.

Resumen

Entre los avances más novedosos de la actualidad informática se destacan los Sistemas de Realidad Virtual (SRV). Estos ofrecen un sin número de prestaciones de manera intuitiva, segura y económica en aplicaciones que se en encuentran en ramas como la defensa, la medicina, la educación y el entretenimiento.

Casi todas las herramientas de desarrollo de entornos virtuales constan de un módulo que permite la transmisión de datos entre ordenadores de manera eficiente y en tiempo real.

El objetivo del presente trabajo es desarrollar de un módulo de clases actualizable y multiplataforma que permita el intercambio de información por la red para Sistemas de Realidad Virtual. Esta herramienta informática, permitirá al programador agilizar el desarrollo de aplicaciones que requieran la comunicación entre varias computadoras, permitiendo el intercambio de datos con una interfaz de alto nivel entre los usuarios de la herramienta y las librerías de *sockets* que brindan los sistemas operativos.

Se realizó el estudio de temas tales como, los modelos de referencia OSI (*Open System Interconnection*) y TCP/IP (*Transfer Control Protocol / Internet Protocol*), protocolos de la capa de transporte TCP (*Transfer Control Protocol*) y UDP (*User Datagram Protocol*), las arquitecturas usadas para juegos *multiplayer*, las librerías de *socket* de los sistemas operativos y las librerías de redes existentes, además de otros temas de interés para obtener un funcionamiento óptimo y el máximo aprovechamiento del ancho de banda.

Contenido

Introducción

La realidad virtual (RV) le ha dado un vuelco total a los gráficos por computadora desde finales de los años 80, logrando no solo la sustitución de los gráficos bidimensionales (2D) por los fascinantes gráficos tridimensionales (3D), sino el surgimiento de entornos generados a través de las tecnologías en los que las personas pudieran interactuar con un mundo virtual.

La RV le ha brindado al hombre la posibilidad de realizar tareas que en la vida real serían muy arriesgadas, le ha permitido entre otras cosas el entrenamiento de personal especializado sin poner en peligro su integridad física. Los simuladores de cirugía de mínimo acceso, vuelos espaciales, vuelos aéreos, tiro, y conducción son muestra de lo que se puede lograr utilizando Sistemas de Realidad Virtual (SRV).

En Cuba existe el Centro de Investigación y Desarrollo #2 (CID2), conocido como SIMPRO (Simuladores Profesionales), que ha venido desarrollando simuladores desde hace varios años y ha obtenido varios premios y reconocimientos por los productos e investigaciones realizadas. En el 2003 dicha empresa decide unir sus fuerzas de trabajo con las Universidad de las Ciencias Informáticas (UCI) para aprovechar el potencial de los estudiantes de la universidad y lograr salir al mercado internacional con productos de calidad.

En sus inicios la empresa contaba para el desarrollo de los productos con herramientas que tenían que comprar en el extranjero y las cuales no cumplían con el total de necesidades para el desarrollo de un simulador. Esto llevó al desarrollo de una herramienta básica por parte del entonces proyecto UCI-SIMPRO, que cumple con determinados requerimientos iniciales para agilizar el trabajo y lograr productos con calidad en el menor tiempo posible.

Esta herramienta o módulo de clases se ha utilizado en la propia universidad en el desarrollo de simuladores, pero se han detectado nuevos problemas no concebidos en su primera concepción.

Por ejemplo, algunos clientes han manifestado la necesidad de que la proyección de los simuladores no sea únicamente hacia una vista frontal, sino también que se proyecte hacia pantallas laterales, buscando así una mayor inmersión y realismo.

Con el hardware existente, las salidas de video desde una PC está limitada a dos, por lo que siempre que se deseen más de dos proyecciones separadas se necesitará más de una computadora; además, aunque una única PC pudiera brindar las salidas necesarias, sería demasiada la carga de cálculos y procesamiento en tiempo real para lograr la simulación deseada.

Por otra parte, facilitar la creación de juegos es otro de los fines de dicha herramienta, y unos de los requisitos fundamentales en éstos es la interacción de varios jugadores (*multiplayer*) en distintas computadoras, para lo cual es necesaria la transferencia de datos por la red en tiempo real.

Una tercera necesidad sería la producción de determinados simuladores que, aunque den una única salida de video, por la carga de procesamiento es conveniente que el trabajo se lleve a cabo en paralelo entre varias computadoras.

¿Cómo lograr la comunicación sincronizada y eficiente entre las computadoras que intervienen en la visualización de los SRV?

Para esto se plantea como objeto de estudio la transmisión de datos por red entre computadoras y como campo de acción la transmisión de datos por red sobre los protocolos de transporte *Transfer Control Protocol* (TCP) y *User Datagram Protocol* (UDP) para juegos y simuladores.

Se propone como objetivo obtener un módulo de clases que permita el intercambio de información por la red para Sistemas de Realidad Virtual.

Para satisfacer los objetivos se proponen un grupo de tareas, resumidas a continuación.

- Estudiar las características básicas de los principales módulos de redes usadas en juegos.

- Estudiar y analizar los protocolos de transmisión TCP y UDP.
- Estudiar y analizar las principales arquitecturas usadas en juegos, *Peer-to-Peer* y Cliente-Servidor.
- Analizar las librerías de sockets presente en los distintos sistemas operativos.
- Desarrollo de soluciones técnicas para alcanzar los objetivos propuestos.
- Diseñar y desarrollar un módulo de clases que dé solución a los problemas planteados.

Como resultado de este trabajo, se pretende obtener un módulo de clases actualizable y multiplataforma. Esta herramienta informática, permitirá al programador agilizar el desarrollo de aplicaciones de RV que requieran la comunicación entre varias computadoras permitiendo el intercambio de datos con una interfaz de alto nivel.

Capítulo 1 Fundamentación Teórica

Introducción

El presente capítulo tratará varios temas con respecto a la comunicación por redes entre ordenadores comenzando con el modelo de siete capas o niveles OSI (*Open System Interconnection)*, explicando de manera general en qué consiste y haciendo énfasis en algunos protocolos que son los más utilizados en la comunicación por Internet. Se explica en qué consisten las direcciones IP, su versión más utilizada IPv4 y la evolución a IPv6 por el problema actual de la escasez de direcciones y que al parecer será quien dominará dentro 20 ó 25 años aunque hay que recalcar que habrá aplicaciones que nunca dejaran de trabajar sobre IPv4. [1] Otro de los temas a tratar será el modelo TCP/IP (*Transfer Control Protocol / Internet Protocol*) que es el modelo que se usa realmente en Internet y aunque el modelo OSI es más fácil de entender, el modelo TCP/IP es quien forma la base de Internet. [2.1] Los protocolos TCP (*Transfer Control Protocol*) y UDP (*User Datagram Protocol*) serán explicados en profundidad para tener una idea más clara de cómo ocurre el proceso de enviar y recibir información por la red sin tener perdidas de la misma y ganando en velocidad.

Los juegos *multiplayer* tienen una arquitectura dividida en dos estructuras fundamentales, Cliente-Servidor y *Peer-to-Peer* (P2P), ambas estructuras serán explicadas para tener una idea de sus ventajas y sus desventajas. [3.1]

Se hablarán de los Sockets que no son más que interfaces de comunicación o un sistema de comunicación entre ordenadores, lo que un buzón o un teléfono es al sistema de comunicación entre personas: un punto de comunicación entre dos agentes (procesos o personas respectivamente) por el cual se puede emitir o recibir información. [4]

Al final del capítulo se hace un estudio de algunas librerías de red utilizadas a nivel internacional para la transmisión de datos en SRV, se ven las características fundamentales y las ventajas y desventajas de las mismas para lograr un trabajo lo más general posible y con la calidad requerida.

Modelo de Referencia OSI

En 1977 la Organización Internacional de Estandarización (*ISO* siglas en inglés) estableció un subcomité encargado de diseñar una arquitectura de comunicación. El resultado fue el modelo de referencia para la Interconexión de Sistemas Abiertos OSI, adoptado en los años 80, que establece unas bases que permiten conectar sistemas abiertos para procesamiento de aplicaciones distribuidas. Se trata de un marco de referencia para definir estándares que permitan comunicar ordenadores heterogéneos. [2.1]

Dicho modelo define una arquitectura de comunicación estructurada en siete niveles verticales. Cada nivel soluciona una serie de problemas relacionados con la transmisión de datos y proporciona un servicio bien definido a los niveles más altos. Los niveles superiores son los más cercanos al usuario y tratan con datos más abstractos, dejando a los niveles más bajos la labor de traducir los datos de forma que sean físicamente manipulables. [2.1]

Los niveles o capas del modelo OSI tienen diferentes funciones dentro del proceso de transmisión y manipulación de los datos:

- El nivel de Aplicación: proporciona un medio a los procesos de aplicación para acceder al entorno OSI. En él encontramos funciones de gestión y mecanismos útiles para soportar aplicaciones distribuidas. [5.1]
- El nivel de Presentación: se ocupa de aspectos sintácticos y semánticos de la información transmitida. [5.1]
- El nivel de Sesión: ofrece mecanismos para controlar el diálogo entre aplicaciones, utilizando testigos y mecanismos de recuperación (*checkpointing*). [5.1]
- El nivel de Transporte: realiza la comunicación extremo a extremo de forma fiable, los paquetes llegan libres de error, ordenados, sin pérdidas ni duplicados. [5.1]
- El nivel de Red: proporciona transferencia de datos transparente entre entidades de transporte. [5.1]

- El nivel de Enlace de datos: se encarga de hacer la comunicación fiable entre dos puntos y proporcionar los medios para activar, mantener y desconectar el enlace. [5.1]
- El nivel de Físico: se ocupa de la transmisión de bits a través de un canal de comunicación. [5.1]

En la Imagen 1 se muestra el modelo de referencia OSI y algunos de los protocolos más importantes existentes en cada una de las capas.

Imagen 1 Modelo OSI

En el esquema OSI se pretende implementar la comunicación de aplicaciones de usuario mediante la utilización de servicios proporcionados por los niveles inferiores. Ambas aplicaciones tendrán una unidad

de información básica a intercambiar, cumpliendo su protocolo establecido de nivel de aplicación. [6] Debemos conseguir que esta información transmitida llegue tal y como fue enviada al nivel de aplicación receptor. [6]

Sin embargo y para asegurar el cumplimiento de sus funciones, en cada nivel es necesario utilizar cierta información de control que sólo será interpretada por el nivel equivalente de la máquina receptora. Por ejemplo, para que el nivel de red pueda enviar correctamente la misma información, es necesario conocer las direcciones en la red de las máquinas origen y destino de los datos, pero esta información no tiene por qué ser conocida por el nivel de transporte ni por el de enlace de datos. De hecho, y para proteger la independencia de niveles, resulta aconsejable que cada información de control sea exclusiva del nivel que la requiera. [6]

Cada nivel, pues, tratará la información procedente del nivel superior como si fueran datos en su integridad, y añadirá su propia información de control (cabecera) antes de pasarlo al nivel inferior. [6]

El modelo OSI fue propuesto como una aproximación teórica y también como una primera fase en la evolución de las redes de ordenadores. Por lo tanto, el modelo OSI es más fácil de entender, pero el modelo TCP/IP es el más usado. Sirve de ayuda entender el modelo OSI antes de conocer TCP/IP, ya que se aplican los mismos principios, pero son más fáciles de entender en el modelo OSI. [2.1]

Modelo de Referencia TCP/IP

Los protocolos TCP/IP se crearon y normalizaron en los años 80, es decir mucho antes de que se definiera el modelo de referencia OSI de la ISO cuya arquitectura se impuso en los años 90. Los gobiernos apoyaron desde un principio los estándares OSI que representa un modelo bien definido y estructurado. Por todo ello se pensó que este modelo triunfaría sobre el modelo TCP/IP que en realidad ni siquiera representa un modelo único sino una familia de protocolos que se han ido definiendo anárquicamente y que a posteriori se han estructurado en capas. Al final lo que realmente ha triunfado ha sido Internet que usa TCP/IP. [2.1]

En el modelo OSI se describen perfectamente siete distintos niveles. En el modelo TCP/IP realmente no se describe una estructura tan precisa de niveles como en el modelo OSI aunque se pueden asimilar y comparar ambos modelos, según se puede ver en la siguiente imagen (Imagen 2).

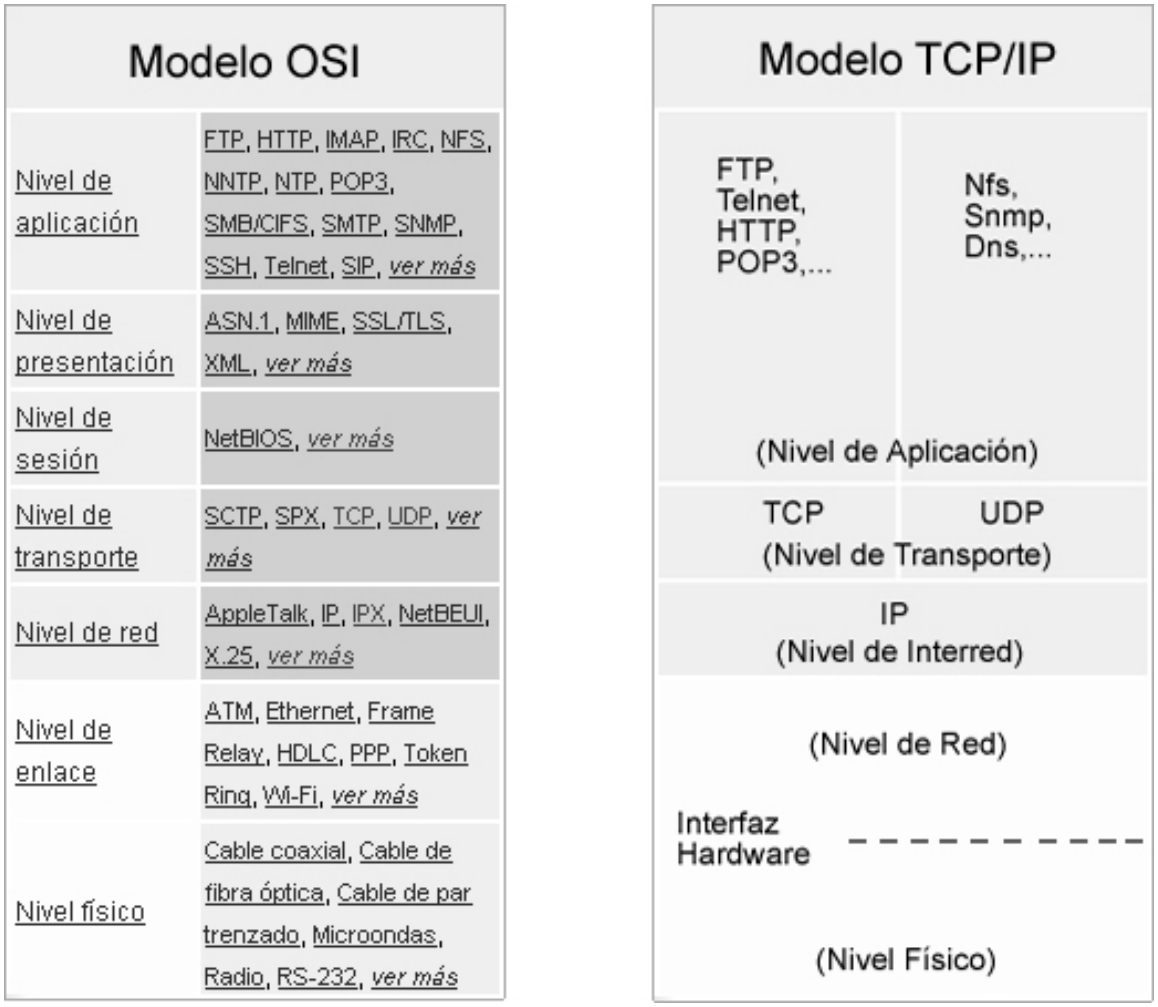

Imagen 2 Modelos TCP y OSI

En todos estos niveles se puede hablar de sus respectivos protocolos de comunicación. El término protocolo indica un conjunto de reglas perfectamente establecidas que permiten y garantizan el intercambio de información. En cada nivel del modelo la comunicación tiene su propio protocolo y su propio cometido. [9]

Los protocolos TCP están orientados a conexión con lo que se podría hablar de sesión asimilándose al nivel 5 OSI, pero UDP no está orientado a conexión y por ello no se puede hablar propiamente de sesión. Las correspondencias de niveles entre OSI y TCP/IP no son perfectas. [9]

TCP está orientado a conexión. En una primera fase se establece conexión y con ello se mantiene una sesión de trabajo que finalizará con un cierre. UDP utiliza datagramas. Es decir que en lugar de abrir una conexión con el destino para establecer un diálogo, funciona simplemente enviando a la red fragmentos de información denominados datagramas. FTP, HTTP, SMTP, y TELNET son servicios muy populares y son un buen ejemplo de protocolos orientados a conexión. SMNP, y NFS son ejemplos de servicios orientados a datagramas. [9]

TCP y UDP delegan en IP la transmisión de la información pero no asumen que IP tenga éxito. Es decir, supervisan la transmisión y se ocupan de reenviar los paquetes en caso necesario. [7.3]

En el nivel 1 OSI llamado nivel físico se definen los cables, los conectores los niveles eléctricos de las señales, la topología, etc. Pero en el modelo TCP/IP los niveles 1 y 2 quedan asimilados a un único nivel de interfaz hardware. Tampoco aquí la correspondencia de niveles es perfecta ya que en el nivel de Interfaz no se definen cables ni conectores, etc. [9]

Direcciones IP

El concepto de números o direcciones IP se puede entender mejor si se establece una analogía entre computadoras y teléfonos. Del mismo modo que cada teléfono posee un único número a nivel mundial, cada computadora conectada directamente a la red Internet tendrá asignado un único número IP a nivel mundial. Por lo tanto, cualquier computadora del planeta puede conectar con cualquier otra computadora, siempre y cuando conozca su número IP y además, exista un camino físico (formado a base de líneas conmutadas, enlaces vía satélite, líneas de fibra óptica, etc.) que conecte ambas computadoras para que puedan intercambiar información. [7.1]

IPv4

IPv4 surge como la versión número cuatro del Protocolo IP, fue la primera versión en implementarse profundamente para formar la base de Internet. Usa direcciones de 32 bits divida en cuatro octetos con un valor decimal máximo de 255 en cada octeto, esto limita a un total de 4.294.967.296 direcciones únicas. [7.1]

Inicialmente no se consideró que Internet tuviera un crecimiento tan vertiginoso, por lo que se le asignaron bloques de direcciones muy grandes (16,7 millones de direcciones) a países y empresas. Debido al desarrollo actual de Internet, ya hace algunos años se vio que las direcciones escaseaban. Esto ha traído un freno en el crecimiento de Internet y en su uso en algunos países asiáticos, especialmente en China. Se estima que las dos terceras partes de las direcciones disponibles ya están asignadas. [7.1]

Esto impulso a IPv6, que esta en las primeras fases de su implementación y el objetivo es sustituir a IPv4 en el futuro, aunque se espera que IPv4 se siga soportando hasta por lo menos el 2025, dado que hay muchos dispositivos heredados que no se migrarán a IPv6 nunca y que seguirán siendo utilizados por mucho tiempo. [7.1]

Representación de dirección IPv4: 200.55.140.181

IPv6

IPv6 surge como la versión número seis del Protocolo IP, su objetivo es sustituir a IPv4 y de esta forma resolver el problema la escasez de direcciones, además de añadir nuevas características y la mejora de las existentes. Trabaja con direcciones de 128 bits divida en ocho grupos de cuatro dígitos hexadecimales, para soportar más niveles de jerarquías de direccionamiento y más nodos direccionables. Con estas características se puede tener un total de 340.282.366.920.938.463.463.374.607.431.768.211.456 (3,4 x 1038) direcciones, si la población mundial fuera de 10 billones de habitantes habría 3,4 x 1027 direcciones por persona. O visto de otra forma habría un promedio de 2,2 x 1020 direcciones por centímetro cuadrado. Siendo así muy pequeña la posibilidad de que se agoten las nuevas direcciones. [8]

A menudo se ha argumentado que las direcciones de 128 bits son exageradas, y que Internet nunca necesitará tantas. Debería tenerse en cuenta que el fundamento principal para las direcciones de 128 bits no solo es tener suficientes direcciones disponibles de por vida, sino también asegurar que el encaminamiento podrá ser llevado a cabo eficientemente con un esquema jerárquico que mantenga al espacio de direcciones sin fragmentar. [7.1]

En muchas ocasiones las direcciones IPv6 están compuestas por dos partes lógicas: un prefijo de 64 bits y otra parte de 64 bits que corresponde al identificador de interfaz, que casi siempre se genera automáticamente a partir de la dirección MAC (*Media Access Control*) de la interfaz a la que está asignada la dirección. [7.1]

Representación de dirección IPv6: 2006:0db8:85a3:08d3:1319:8a2e:0370:7334

Protocolos IP, TCP y UDP

El protocolo IP trabaja de la siguiente manera; la capa de transporte toma los mensajes y los divide en datagramas, de hasta 64K octetos cada uno. Cada datagrama se transmite a través de la capa interred, posiblemente fragmentándose en unidades más pequeñas, durante su recorrido normal. Al final, cuando todas las piezas llegan a la máquina destinataria, la capa de transporte los reensambla para así reconstruir el mensaje original. [10]

Cuando los programadores diseñan *software* Cliente-Servidor, deben escoger entre dos tipos de interacción: orientada a conexión o no orientada a conexión. Si el cliente y el servidor utilizan UDP, la iteración es sin conexión; por el contrario, si emplean TCP, la iteración es orientada a conexión. [11.1]

Desde el punto de vista del programador de aplicaciones, la distinción entre el estilo sin conexión y orientado a conexión es crítica ya que determina el nivel de funcionalidad que proporciona el sistema. TCP proporciona toda la funcionalidad necesaria para establecer una comunicación entre ordenadores a través de Internet. Verifica que los datos llegan al destinatario y automáticamente retransmite paquetes que por cualquier motivo no llegan al destinatario o le llegan con errores. Comprueba la integridad de los datos

para garantizar que no se corrompan durante su transmisión. Emplea secuencias de números para asegurar que los paquetes de datos llegan al destinatario en el orden correcto, los paquetes duplicados son eliminados automáticamente por el protocolo TCP. Proporciona un control de flujo para asegurar que el emisor no transmita datos más rápido de lo que pueden ser consumidos por el receptor. Finalmente, TCP informa tanto al cliente como al servidor si la red deja de estar operativa por algún motivo. [11.1]

Por contraste, los clientes y servidores que emplean UDP no tienen garantías de que la información enviada a la red vaya a llegar realmente a su destinatario. Cuando un cliente envía una petición, esta puede perderse, ser duplicada, retardada o los paquetes de datos pueden llegar al destinatario fuera de orden. Del mismo modo, la respuesta del servidor puede perderse, duplicarse, retardarse o llegar desordenada. Los programas de aplicación Cliente-Servidor deben llevar a cabo las acciones oportunas para detectar y corregir tales situaciones de error. [11.1]

Sin embargo, el empleo del protocolo UDP puede ser una alternativa interesante ya que permite un transporte de información más eficaz. UDP no introduce errores, únicamente se fundamenta en la red IP para transportar paquetes. Por el contrario, IP depende del hardware de red sobre el que se asienta y los *gateways* intermedios. Desde el punto de vista del programador, la consecuencia de emplear UDP es que este trabaja bien si la red sobre la que se asienta funciona bien. Por ejemplo, UDP funciona bien en un entorno local porque los posibles errores raramente se producen. Los errores se suelen producir cuando la comunicación se expande a una red de área extendida. [11.1]

Los programadores a veces cometen el error de elegir un protocolo sin conexión (UDP), construyendo una aplicación que hace uso del mismo, pero verificando el funcionamiento de la aplicación en una red de área local. Debido a que una red de área local raramente o nunca retrasa los paquetes, los pierde o los entrega fuera de orden, la aplicación da la sensación de que funciona correctamente. Sin embargo, si se hace una prueba en una red de área extensa, puede darse el caso de que el programa falle o genere resultados incorrectos. [11.1]

Los principiantes, del mismo modo que los profesionales experimentados, prefieren emplear una comunicación orientada a conexión a la hora de diseñar sus aplicaciones de red. Un protocolo orientado a

conexión hace que la programación resulte más simple y releva al programador de la responsabilidad de detectar y corregir errores de comunicación. [11.1]

Por norma general, los programas de aplicación sólo emplean el UDP si:

- El protocolo de aplicación a implementar específica que se debe de emplear el UDP (puede ser que el protocolo de aplicación haya sido diseñado para manejar errores que se puedan producir durante la comunicación). [11.1]
- El protocolo de aplicación relega la seguridad de comunicación al hardware subyacente y no importa la pérdida de algunos paquetes de información. [11.1]
- La aplicación no puede tolerar la sobrecarga computacional o retraso requerido por los circuitos virtuales TCP. [11.1]

Resumiendo: en el diseño de aplicaciones Cliente-Servidor, a los principiantes se les recomienda utilizar el TCP ya que proporciona una conexión orientada a conexión más sencilla de programar y fiable. Los programadores únicamente hacen uso del UDP si el protocolo de la aplicación que se esté implementando lo requiere.

Números de puertos de protocolos a nivel de aplicación

La comunicación entre computadoras se lleva a cabo mediante el intercambio de paquetes de bytes. La semántica de los conjuntos de bytes que recibe una computadora viene dictada por la aplicación a la cual van destinados. Los paquetes de información que se difunden a través de una red de computadoras son encaminados hacia un *host* concreto y dentro de dicho *host* a un puerto concreto. [11.1]

Se puede pensar en un puerto como en un canal de comunicación. Cada computadora dispone de un total de 65000 canales o puertos, los cuales pueden o no estar activos. Para que un puerto esté activo es necesaria una aplicación que tome el control del mismo y sea capaz de gestionar los paquetes de bytes

que llegan por dicho puerto. Cuando un *host* recibe un paquete examina su cabecera para averiguar a qué puerto va destinado, si existe una aplicación escuchando dicho puerto, entonces se le pasan los bytes del paquete para que esta los interprete y actúe consecuentemente. El *host* no responderá a peticiones de conexión encaminadas hacia un puerto para el cual no existe ninguna aplicación escuchando. [11.1]

Es decir, de los paquetes de bytes remitidos hacia una computadora en concreto, sólo se va a atender aquellos paquetes para los cuales existe una aplicación escuchando en el puerto al cual van encaminados.

Existe una serie de puertos estándar utilizados universalmente para varios servicios. Algunos de ellos son:

Tabla 1 Puertos a Nivel de Aplicación

SERVICIO	PUERTO	DESCRIPCIÓN
FTP	21	*File Transfer Protocol*
TELNET	23	Acceso a una cuenta en un equipo remoto
SMTP	25	Para enviar correo electrónico
POP3	110	Para obtener correo electrónico del servidor
HTTP	80	Protocolo para WWW
NNTP	119	Grupos de noticias de Internet
GOPHER	70	Navegador modo texto

Arquitectura de los juegos multiplayer

Los programadores de redes escriben códigos a bajo nivel y a nivel de aplicación, para lograr correr juegos entre pocos jugadores mediante módems, red de área local (LAN) o a través de Internet.

Antiguamente los programadores tenían que dominar un gran número de protocolos como el IPX (*Internetwork Packet Exchange*) y los protocolos de módems. La gran mayoría de los juegos modernos corren exclusivamente sobre los protocolos de redes de Internet TCP y UDP. [12]

La arquitectura de los juegos multiplayer se puede dividir en dos estructuras fundamentales: *Peer-to-Peer* y Cliente-Servidor.

La estructura *Peer-to-Peer* tiene todo en la máquina del jugador, simulando su propia copia del juego y usando una gran variedad de algoritmos para mantener el estado de las diferentes computadoras tan cerca como sea posible. En esta estructura todas las máquinas pueden transmitir directamente a las otras máquinas de la red. Para utilizar este modelo de juego el ancho de banda necesita crecer exponencialmente cada vez que se agrega un jugador, esto es un desafortunado efecto secundario cuando tratas de manipular más jugadores. [12]

La estructura Cliente-Servidor divide los cálculos de las simulación del juego en un servidor, el cual manipula la simulación real y en el cliente, el que hace el papel de espectador o visualizador de los eventos que ocurren en que el mundo. Hay un gran beneficio en esta estructura, incluyendo el hecho de que el ancho de banda requiere crecer solo linealmente con el número de jugadores y el juego puede ser también protegido de varias trampas o engaños debido a estar corriendo en un servidor seguro y fiable. Recordemos que en un juego *Peer-to-Peer* cada máquina corre su propia copia del juego y tiene control sobre alguna parte del mundo. Este control puede ser fácilmente mal usado por algún programa maligno que corra en la red *Peer-to-Peer*. [12]

¿Por qué todos los juegos no son Cliente-Servidor? Dudosamente todos deberían serlo; sin embargo, depende del juego, la arquitectura Cliente-Servidor es mucho más compleja y requiere dividir la simulación y la presentación hacia un campo de trabajo más riguroso orientado a objeto. Hoy sólidamente se puede decir que los juegos *multiplayer* son los primeros ejemplos de la complicación de juegos Cliente-Servidor. Los servidores cargan con diferentes operaciones como la autenticación de jugadores, detección de trampas, simulación del juego, servicio de *chat*, transacción de base de datos y más. Los juegos *Peer-to-Peer* son mucho más similares a los tradicionales juegos de un solo jugador con la excepción de que los

juegos hacen correcciones periódicamente para estar más en línea con cada una de las vista del juego. [12]

Cliente-Servidor

Desde el punto de vista de una aplicación, TCP, al igual que muchos otros protocolos de comunicación, implementa un mecanismo fiable para la transmisión de datos entre ordenadores. En concreto, TCP permite que un programador pueda establecer comunicación de datos entre dos programas de aplicación, tanto si ambos se están ejecutando en la misma máquina, como en máquinas distintas unidas por algún camino físico (una red local, conexión telefónica directa entre computadoras, computadoras conectadas a Internet). [11.1]

Hay que tener presente que el protocolo TCP especifica los detalles y mecanismos para la transmisión de datos entre dos aplicaciones comunicantes, pero no dictamina cuándo ni porqué deben interactuar ambas aplicaciones, ni siquiera especifica como debería estar organizada una aplicación que se va a ejecutar en un entorno distribuido. Es tarea del diseñador de la aplicación distribuida el establecer un protocolo de comunicación y sincronización adecuado. [11.1]

En la práctica el esquema de programación más utilizado para la implementación de aplicaciones distribuidas es el paradigma Cliente- Servidor. [11.1]

El modelo Cliente-Servidor da solución al problema del *rendezvous* (reencuentro). Según este modelo, en cualquier par de aplicaciones comunicantes, una de las partes implicadas en la comunicación debe iniciar su ejecución y esperar (indefinidamente) hasta que la otra parte contacte con ella. Esta solución es importante, ya que TCP no proporciona ningún mecanismo que automáticamente lance procesos para atender mensajes entrantes, debe de existir un proceso en espera que sea capaz de atender dichos mensajes (peticiones de servicio). [11.1]

Muchos administradores hacen que ciertos programas de comunicaciones se inicien automáticamente cuando el sistema arranca, de este modo se aseguran que la computadora estará preparada para aceptar ciertas solicitudes de servicio. Después de iniciar su ejecución, cada uno de estos programas se queda en espera de la siguiente petición para el servicio que oferta. [11.1]

El paradigma Cliente-Servidor divide las aplicaciones comunicantes en dos categorías, dependiendo del tipo de aplicación, la que espera conexiones (servidor) o la que la inicia (cliente). [11.1]

En general, una aplicación que inicia una comunicación con otra se la califica como cliente. Los usuarios finales invocan aplicaciones cliente cuando utilizan un servicio de red. Cada vez que se ejecuta una aplicación cliente, esta contacta con el servidor, le envía una solicitud de servicio y espera la respuesta o resultados del servicio. El proceso cliente es el encargado de llevar a cabo la interacción con el usuario y de mostrar los resultados de las peticiones de servicio. En la mayoría de las ocasiones los clientes son más fáciles de diseñar que los servidores y no suelen precisar privilegios especiales del sistema para poder funcionar. [11.1]

Un servidor es un programa que espera peticiones de servicio por parte de un cliente. El servidor recibe la petición del cliente, ejecuta el servicio solicitado y retorna los resultados al cliente. No existe una interacción directa entre el usuario y el servidor, de esto ya se encarga la aplicación cliente. [11.1]

Peer-to-Peer

Una red *Peer-to-Peer* depende fundamentalmente de la capacidad de procesamiento, ancho de banda y capacidad de almacenamiento de los participantes. [13] Es una red formada por una x cantidad de nodos distribuidos que poseen información de manera descentralizada, son usadas básicamente para conectar nodos durante conexiones temporales. [14]

Se puede utilizar para varios propósitos, compartir documentos como videos, audio, datos o algo que se encuentre en formato digital, también puede ser utilizado para trasmitir datos en tiempo real, la comunicación telefónica puede realizarse mediante tecnología P2P. [13]

Una verdadera red P2P no trabaja con clientes o servidores fijos, sino una serie de nodos que se comportan a la vez como clientes y como servidores de los demás nodos de la red. [13]

Componentes:

- Nodo: es la base de esta arquitectura, el mismo es considerado y cumple la función tanto de cliente como de servidor. Se diferencian mediante un identificador. Su participación dentro de la arquitectura es temporal (o sea, por fracciones de tiempo). Cada uno aportará a la arquitectura P2P una cantidad de otros nodos.
- Ítems de Información: puede contener información pura o meta información. Es parte del índice de la red virtual. Debido a que no siempre la red tiene los mismos nodos conectados, algunos ítems en determinado momento serán obsoletos, ya que figuran en el índice pero no existen en la red. Se los identifica por medio de un identificador de datos.
- Colección: es un conjunto de ítems con alguna particularidad, que a su vez podrán tener asociados una lista de otros ítems.
- Transacción: los nodos pueden ser consultados de dos formas:

    - Palabra clave.
    - Identificador de datos.

Esto indica también que cada nodo debe mantener un índice por palabra clave y otro por identificador de datos. Desde los nodos podemos capturar o recuperar información que necesitamos por cualquiera de ellos. [14]

Peer-to-peer Centralizada

Trabaja mediante un servidor concentrador. Este se encarga de mantener el índice actualizado, teniendo en cuenta para ello los usuarios que están conectados a la red. [14]

Ventajas:

- Índice centralizado, rápido y eficaz.
- Los usuarios deben autenticarse.
- Nuevos usuarios se deben registrar previamente.

Desventajas:

- Única entrada.
- Si el servidor se cae, arrastra a toda la red.
- Actualización periódica.

Peer-to-peer Descentralizada

No utiliza servidor central. Permite que cualquier PC se conecte y que esta sea notificada a las demás como un nuevo nodo dentro de la red. El índice debe ser actualizado en los nodos que estén destinados a tal fin o en todos los nodos que componen la red. [14]

Ejemplo: una PCx se conecta a la PCa, la cual avisará a la PCb que existe un nuevo nodo activo.

PCb anunciará a PCc, PCd que un nuevo nodo se ha conectado, y así sucesivamente. Aquí se nos presenta la potencialidad de escalabilidad de la red. [14]

Ventajas:

- No depende de un servidor, por lo tanto es más robusta.
- Escalabilidad, potencialmente infinita.

Muchas empresas están desarrollando aplicaciones basadas en la tecnología P2P, algunas de ellas son:

- Proyecto JTXA, es una plataforma desarrollada por *Sun Microsystems*, orientada a resolver problemas de archivos distribuidos a través de P2P. Según *Sun* P2P no solo es una palabra de moda sino todo lo contrario, día a día se potenciará más.
- Gnutella, es un protocolo descentralizado en el cual los usuarios no se conectan a un servidor central, los mismos se comunican y transfieren información de forma horizontal.
- Frente, es la una de las novedades de *software* de red distribuida, de estructura similar Gnutella. Cada ordenador conectado a la red se convierte en un servidor de información, pero tiene la particularidad de permitir el anonimato.
- Napster, es el *software* emblemático de la nueva generación de programas informáticos P2P, el conjunto de estos programas se los denomina software de red distribuida. Es un *software* P2P centralizado.

Las redes P2P tienen varias ventajas y desventajas en comparación con las redes Cliente-Servidor:

Ventajas:

- El contenido y los recursos pueden ser compartidos desde el núcleo y los límites de la red.
- Una red de iguales es fácilmente escalable y más fiable que un solo servidor.
- Una red de iguales puede compartir su procesador, refundiendo los recursos de la computadora para las tareas distribuidas.
- Se puede acceder directamente a los recursos compartidos de los ordenadores iguales.
- Permite una comunicación multipunto eficaz apoyándose en la infraestructura multidifusión IP.
- La red P2P habilita y mejora las comunicaciones en tiempo real, la colaboración, la distribución de contenido y el procesamiento distribuido. [15]

Desventajas:

- Difusión de las peticiones a todos los vecinos (*flooding*). Consume gran cantidad de ancho de banda en las peticiones haciendo el tiempo de respuesta mayor y muy variable.
- La complejidad de encontrar otros nodos dentro de la red.
- En los juegos *multiplayer* el ancho de banda debe aumentar significativamente por cada jugador que entre en la escena.
- Los grandes riesgos de seguridad, pues los clientes son responsables de mandarles información a cada miembro de la red por lo que es vulnerable a ser hackeado.
- El bajo nivel de estandarización debido a que se necesita interactuar con aplicaciones de diferente soporte.
- La mayor desventaja en los juegos *multiplayer* son los problemas de escalabilidad, debido a que en una red que se disponga de ancho banda limitado el número de jugadores será limitado, lo que provoca el poco uso de este paradigma en los juegos. [16]

Interfaz Socket [7.2]

En el año 1980 la agencia ARPA (*Advanced Research Projects Agency*, Agencia para la Investigación de Proyectos Avanzados) fundó un grupo de investigación en la Universidad de Berkeley (California) para llevar a cabo la implementación del software TCP/IP dentro del propio núcleo del sistema operativo UNIX. Como parte del proyecto, los diseñadores crearon una interfaz para permitir que las aplicaciones pudieran acceder a los servicios que brinda el software TCP/IP. Decidieron hacer uso de las llamadas al sistema UNIX existentes siempre que fuera posible y añadir nuevas llamadas para dar soporte a las funciones TCP/IP que no se podían implementar adecuadamente con el conjunto de llamadas disponibles. Al fruto de este proyecto se le conoce como la Interfaz Socket, y el sistema operativo resultante se dio a conocer

como UNIX Berckeley o BSD UNIX. La interfaz socket desarrollada en Berkeley ha sido tan ampliamente aceptada por los grandes distribuidores de computadoras que se ha convertido en un *standard* de facto.

No hay una definición clara de que es un socket, muchas personas tiene opiniones diferentes acerca de ellos, pero básicamente un socket es un conducto entre dos computadoras de una red a través de la cual fluyen los datos. No es un cable físico ni nada que se le parezca. Un socket existe solo en el mundo de los bits y los bytes. Los dos ordenadores que intervienen en la comunicación tienen su propio socket, los cuales pueden ser identificados con los dos finales del conducto, como se muestra a continuación (Imagen 3).

Imagen 3 Socket

En la memoria de la computadora, los socket son números de 32bit. De esta forma cuando un nuevo socket es creado, esta le asigna un único número que define el socket en la computadora local.

La Interfaz Socket proporciona funciones generalizadas que dan soporte a las comunicaciones en red empleando para ello muchos de los protocolos disponibles hoy en día. Las llamadas a la Interfaz Socket hacen referencia a todos los protocolos TCP/IP como una única familia. Las llamadas permiten al programador especificar el tipo de servicio requerido, en vez del nombre de un protocolo específico.

La idea general que subyace bajo el concepto de socket es que una única llamada al sistema es suficiente para crear cualquier socket. Una vez el socket ha sido creado, una aplicación debe realizar llamadas al sistema adicionales para especificar el uso que se desea hacer del mismo.

Cuando una aplicación invoca la llamada socket, el sistema operativo aloja una nueva estructura de datos que contiene la información necesaria para comunicar.

Aunque la estructura de datos para el socket contiene muchos campos, el sistema operativo deja muchos de ellos en blanco cuando se crea el socket. Como veremos a continuación, la aplicación que creó el socket debe realizar llamadas al sistema adicionales para rellenar la información contenida en la estructura de datos socket antes de que el socket pueda ser utilizado.

Una vez el socket ha sido creado, puede ser utilizado para esperar conexiones o para iniciar una conexión. Al socket empleado por un servidor para esperar conexiones entrantes se le conoce como socket pasivo, mientras que al socket empleado por un cliente para iniciar una conexión se le conoce como socket activo. La única diferencia entre los sockets activos y pasivos radica en como son utilizados por la aplicación que los ha creado, ya que estos se crean del mismo modo.

Cuando se crea un socket, no se define información detallada acerca de como será utilizado. En concreto, el socket carece información acerca del puerto de protocolo y la dirección IP tanto de la máquina local como de la máquina remota. Antes de que una aplicación pueda utilizar un socket, debe especificar una o ambas direcciones.

Sockets *Stream* u orientados a conexión: son los más utilizados, hacen uso del protocolo TCP (Imagen 4), el cual nos provee un flujo de datos bidireccional, secuenciado, sin duplicación de paquetes y libre de errores.

Sockets *Datagram* o sin conexión: hacen uso del protocolo UDP (Imagen 4), el cual nos provee un flujo de datos bidireccional, pero los paquetes pueden llegar fuera de secuencia, pueden no llegar o contener errores. Por lo tanto el proceso que recibe los datos debe realizar resecuenciamiento, eliminar duplicados y asegurar la confiabilidad.

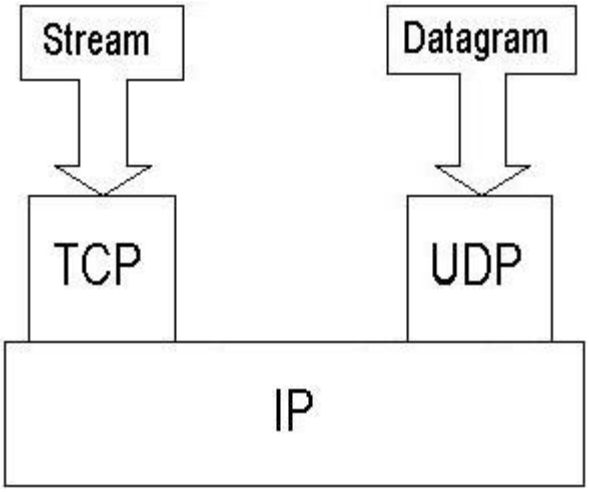

Imagen 4 Tipos de Sockets

Mucha de la complejidad a la que se enfrenta un programador que emplea la Interfaz Socket es debida al hecho de que las funciones de sistema socket poseen parámetros que permiten a los programadores utilizarlas de modos distintos. Por ejemplo, un socket puede ser empleado por un cliente o por un servidor para una comunicación orientada a conexión (TCP), o no orientado a conexión (UDP), con una dirección de punto terminal (dirección de red) específica (cliente), o con una dirección de punto terminal (dirección de red) no especificada (servidor).

Principales llamadas

En esta sección se va a explicar las llamadas que proporcionan la funcionalidad primordial que los clientes y servidores necesitan para establecer una comunicación empleando la interfaz socket. Solo se va a dar una explicación simple y clara. Para profundizar más en el uso de la interfaz socket se recomienda la lectura de un texto especializado en el tema.

Llamada al sistema: socket

Una aplicación invoca la función de sistema socket para crear un nuevo socket, el cual puede ser empleado para una comunicación en red. La llamada retorna un descriptor para el socket recién creado. Los argumentos para la llamada especifican la familia de protocolos a emplear (por ejemplo, PF_INET

para el TCP/IP) y el tipo de servicio (orientado a conexión o no orientado a conexión). Para un socket que utilice la familia de protocolos Internet, el argumento tipo de servicio determina si el socket empleará TCP o UDP. Con esta llamada solicitamos al sistema operativo que aloje todos los recursos necesarios para una comunicación en red.

Llamada al sistema: *connect*

Después de crear un socket, un cliente llama la función *connect* para establecer una conexión activa con un servidor remoto. Uno de los argumentos de *connect* permite al cliente especificar la dirección de red remota, la cual está formada por la dirección IP de la máquina remota y el número de puerto de protocolo. Una vez establecida la conexión, el cliente puede tanto transferir como recibir datos. El servidor no tiene porque estar ejecutándose en una máquina remota, también se puede encontrar en la misma máquina en que se ejecuta el cliente.

Llamada al sistema: *send*

Tanto los clientes como los servidores emplean la llamada al sistema *send* para enviar datos a través de una conexión TCP. Por norma general, los clientes emplean la llamada *send* para enviar peticiones, mientras que los servidores la emplean para enviar los resultados de un servicio solicitado previamente.

Una llamada a *send* precisa de cuatro argumentos: el descriptor de socket a través del cual la aplicación pretende enviar los datos, la dirección del buffer de memoria que contiene los datos a enviar, la cantidad de bytes a transmitir y el cuarto se utiliza para fijar la opciones de envió. Normalmente, *send* copia los datos a transmitir en *buffers* gestionados por el núcleo del sistema operativo, y permite que la aplicación continúe con su ejecución mientras el sistema operativo se encarga de transmitir los datos a través de la red. Si los *buffers* del sistema se llenan, la llamada a *send* provocará el bloqueo temporal del proceso hasta que el software TCP pueda transmitir los datos a través de la red y se libere espacio en los *buffers* para alojar nuevos datos. Esta función retorna el número de byte enviados.

Llamada al sistema: *recv*

Tanto clientes como servidores emplean la llamada *recv* para recibir datos de una conexión TCP. Normalmente, después de que se ha establecido una conexión, el servidor emplea *recv* para recibir la petición que el cliente le envía mediante la llamada *send*. Después de enviar su petición, el cliente utiliza *recv* para recibir la respuesta del servidor.

Para leer bytes de una conexión, una aplicación invoca la función de sistema *recv* con cuatro argumentos: el primero especifica el descriptor de socket a utilizar, el segundo especifica la dirección del buffer en el cual se almacenarán los bytes recibidos, el tercero indica el tamaño en bytes del *buffer* y el cuarto se utiliza para fijar la opciones de recepción. La llamada *recv* extrae los bytes que han llegado al socket y los copia al buffer indicado por el programador. Si aún no se ha recibido ningún byte, la llamada *recv* bloquea al proceso que la invoca hasta que se reciben datos por el socket. Si se reciben más bytes que los que pueden caber en el buffer, *recv* solo obtiene los justos para llenar el buffer. Si se recibe una cantidad de bytes menor que el tamaño del buffer, *recv* obtiene todos los bytes y retorna la cantidad de bytes obtenidos.

Llamada al sistema: *close*

Cuando los procesos cliente o servidor dejan de utilizar un socket, es conveniente que invoquen a la función *close* para liberar los recursos del sistema que este ocupando dicho socket. Si únicamente un proceso está utilizando el socket, *close* concluye inmediatamente la conexión y libera los recursos asignados al socket. Si varios procesos comparten un socket, *close* decrementa el contador de referencias. Sólo se liberará completamente los recursos ocupados por el socket cuando el contador de referencias alcance el valor cero. Cualquier conjunto de datos no leídos que estuviera almacenado en los *buffers* que el sistema mantiene para el socket se perderá después de la llamada a *close*.

Llamada al sistema: *bind*

Cuando se crea un socket, este no tiene asignada ninguna dirección de red, ni la dirección de red local ni la remota están especificadas. Una aplicación invoca la llamada al sistema *bind* para especificar la

dirección de red local para un socket. La llamada toma argumentos que especifican un descriptor de socket y la dirección para dicho socket. Para los protocolos TCP/IP, la dirección de red emplea la estructura *sockaddr_in*, la cual tiene campos para especificar tanto la dirección IP como el número de puerto de protocolo. Esta llamada suele ser empleada por los servidores para especificar la dirección de red (dirección IP + puerto) en la cual esperan recibir peticiones de servicio.

Llamada al sistema: *listen*

Cuando se crea un socket, este no se puede calificar ni como socket activo (listo para ser utilizado por un cliente) ni como socket pasivo (listo para ser utilizado por un servidor), hasta que la aplicación lleva a cabo una serie de operaciones. Los servidores orientados a conexión llaman a la función listen para poner un socket en modo pasivo y prepararlo para aceptar conexiones entrantes.

En el código fuente de la mayoría de los procesos servidores suele aparecer ciclos infinito encargado de aceptar las conexiones entrantes, darles servicio, y retornar los resultados al cliente que tramitó la solicitud de servicio. Incluso si el proceso de una conexión dada lleva únicamente unos pocos milisegundos, puede ocurrir que una nueva petición de servicio le llegue al servidor mientras está atendiendo la que tiene actualmente en curso. Para asegurar que no se pierda ninguna petición de conexión, el servidor debe pasar a listen un argumento que le dice al sistema operativo que encole las peticiones de conexión para un socket dado. Por lo tanto, el primer argumento de la llamada listen específica el socket a ser puesto en modo pasivo, mientras que el segundo especifica el tamaño de la cola de conexiones a ser empleada con dicho socket.

Llamada al sistema: *accept*

Después de que un servidor invoca a la función socket para crear un socket, a *bind* para especificar la dirección de red asignada a dicho socket y a listen para poner el socket en modo pasivo; el servidor utiliza la llamada al sistema *accept* para extraer la siguiente petición de conexión. Un argumento para *accept* especifica el socket del cual se aceptará la siguiente solicitud de conexión.

La llamada *accept* crea un nuevo socket por cada nueva petición de conexión y retorna el descriptor del nuevo socket a la aplicación que la invocó. El servidor utiliza el nuevo socket únicamente para la nueva

conexión, y emplea el socket original para aceptar conexiones adicionales. Una vez aceptada una conexión, el servidor puede transferir datos a través del nuevo socket. Concluido el uso del nuevo socket, el servidor lo cierra. *Accept* elimina la siguiente petición de conexión de la cola o bloquea al proceso que la invoca hasta la llegada de una nueva petición de conexión.

La llamada *accept* sólo se emplea con socket orientados a conexión (aquellos empleados con TCP).

El siguiente gráfico (Imagen 5) muestra el esquema general de interacción Cliente-Servidor con socket orientados a conexión (TCP):

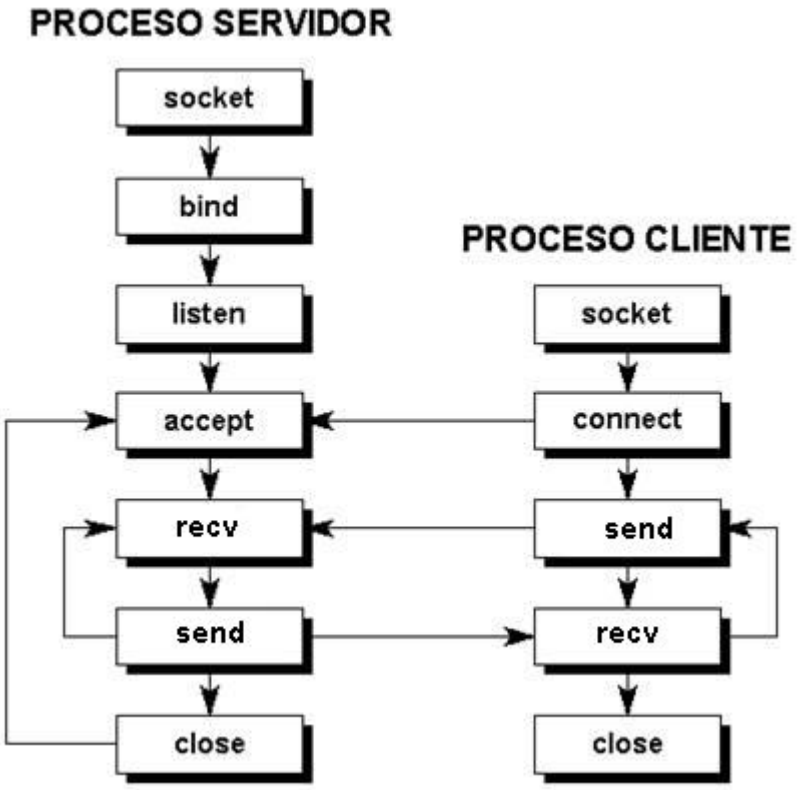

Imagen 5 Sockets TCP (Cliente-Servidor)

Llamada al sistema: *sendto*

Al igual que la función *send*, *sendto* es también usada para el envió de datos, pero para esta función no es necesario que se haya establecido una conexión previa. La función *sendto* es normalmente usada por socket no orientados a conexión (UDP).

La llamada a *sendto* precisa de seis argumentos: el descriptor de socket a través del cual la aplicación pretende enviar los datos, la dirección del buffer de memoria que contiene los datos a enviar, la cantidad de bytes a transmitir, el cuarto se utiliza para fijar la opciones de envió, el quinto es una estructura llamada *sockadddr* que contiene la dirección destino a la que se desean enviar los datos y el sexto contiene el tamaño de estructura que contiene la dirección destino. Esta función retorna el número de byte enviados.

Llamada al sistema: *recvfrom*

La llamada a la función *recvfrom* es usualmente realizada por socket no orientados a conexión (UDP) y su uso es similar al de la función *recv*.

Para leer bytes una aplicación invoca la función de sistema *recvfrom* con seis argumentos: el primero especifica el descriptor de socket a utilizar, el segundo especifica la dirección del *buffer* en el cual se almacenarán los bytes recibidos, el tercero indica el tamaño en bytes del buffer, el cuarto se utiliza para fijar la opciones de recepción, el quinto es la estructura *sockadddr* que contiene la dirección de la que provienen los datos y el sexto contiene el tamaño de estructura que contiene la dirección destino. Si no hay byte para recibir la función queda en espera, en caso de recepción devuelve el número de byte recibidos. Si el buffer no puede contener la totalidad del mensaje, *recvfrom* llena el buffer y descarta el resto del mensaje.

El siguiente gráfico (Imagen 6) muestra el esquema general de interacción Cliente-Servidor con socket no orientados a conexión (UDP):

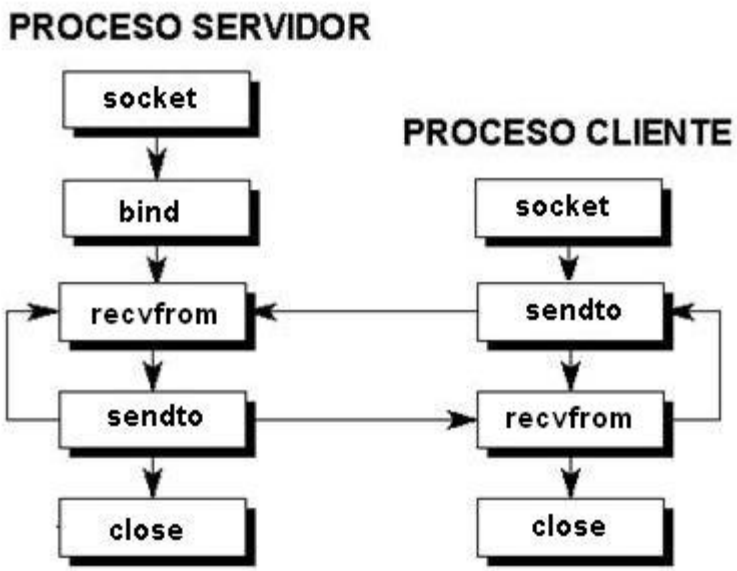

Imagen 6 Sockets UDP (cliente-servidor)

Librerías de redes orientadas a aplicaciones gráficas.

TorqueNL

TorqueNL (*Torque Network Library*) es una librería de redes multiplataforma que soporta *Windows* 98/ME/ NT/XP, *Linux* para procesadores de la familia X86 y Mac OS X, diseñada especialmente para juegos y simuladores, la arquitectura de red usada en TorqueNL ha potenciado algunos juegos de acción como TRIBES y TRIBES 2. Está basada en conexión del tipo UDP y hace un uso consecuente del ancho de banda basada en una eficiente arquitectura de flujo de paquetes. Consta de simples sentencias para lograr llamada a procedimientos remotos. La licencia comercial de esta librería cuesta alrededor de 1000 dólares. [17]

SolarSocket

Librería de comunicaciones multiplataforma programada sobre C++, de fácil operabilidad. Esta librería consta con una interfase del alto nivel para el uso de socket orientados a conexión (TCP). El uso de *threads* y el hecho de que esta orientada a eventos, la hace muy útil para proyectos que requieran intercambio de información a través de Internet, Compatible con *Linux* y *Windows*, lo que brinda la posibilidad de que el código escrito, usando *SolarSockets*, puede ser compilado sin necesidad de modificación, tanto en *Windows* como en *Linux*. La licencia comercial de esta librería cuesta alrededor de 2000 dólares. [18]

RakNet

Es una librería basada en conexiones mediante la interfase de socket no orientados a conexión (UDP) sobre C++, diseñada para aplicaciones que requieran gran velocidad de respuesta, es usada en la mayoría de los casos en juegos pero es una aplicación independiente. Raknet corre en *Windows*, *Linux* y *Macs* ha sido compilada en *Visual Studio*, GCC, DevCPP y otras. Es una librería extremadamente poderosa que puede ser integrada en cualquier tipo de aplicación en muy corto tiempo. La licencia comercial de esta librería está sobre los 4000 dólares. [19]

Conclusiones

A lo largo del capítulo se tocaron los fundamentales temas que intervienen en la transmisión de datos entre ordenadores por red. Además se hizo estudio de aspectos imprescindibles para le desarrollo del proyecto que ayudaron a entender el procedimiento básico para la conexión e intercambio de información entre computadoras y a definir los principales protocolos de comunicación a tener en cuenta para lograr un rendimiento óptimo del módulo de clases a desarrollar.

Capítulo 2 Soluciones Técnicas

Introducción

En el presente capítulo se define cual de los modelo de referencia estudiados anteriormente será el más apropiado, así como el grupo de capas y protocolos más interesante para el desarrollo del módulo de clases, se propone el uso de las librerías de sockets presente en los distintos sistemas operativos como base de la comunicación. Además se propone utilizar la arquitectura Cliente-Servidor por ser la más óptima para los Sistemas de Realidad Virtual (SRV).

Sockets

Para el desarrollo del módulo se propone como base de la comunicación el uso de las *Application Program Interface* (API) de los sockets, ya que esta interfaz permite programar aplicaciones de red. Mediante los sockets se puede lograr la comunicación entre distintos sistemas operativos. Ellos proporcionan funciones generalizadas que dan soporte a las comunicaciones en red empleando para ello muchos de los protocolos disponibles hoy en día.

Protocolos de Comunicación

En el capítulo anterior se mencionaron varios de los protocolos utilizados en la comunicación por redes y en qué capa están ubicados según los modelos de referencia estudiados. En el presente epígrafe se proponen cuáles de ellos serán utilizados en el desarrollo del módulo de clases para dar solución a las necesidades planteadas en el capítulo anterior: *Transfer Control Protocol* (TCP) y *User Datagram Protocol* (UDP) ubicados en la capa de Transporte del modelo de referencia TCP/IP.

TCP es un protocolo orientado a conexión, pues evita la duplicación y el extravío de paquetes lo que permite lograr conexiones para juegos o simuladores en los que se requiere una mayor confiabilidad.

Al contrario de TCP, UDP es un protocolo no orientado a conexión, no garantiza la duplicación de paquetes ni el extravío de los mismos, puede ser utilizado en aplicaciones que necesiten la mayor eficiencia posible y poca confiabilidad, pues logra una mayor velocidad de transmisión, es el protocolo más usado en SRV.

Arquitecturas de Comunicación

En el capítulo anterior se estudiaron las dos arquitecturas más usadas en la comunicación de Sistemas de Realidad Virtual. El presente epígrafe propone cuál de ellas es la más indicada para el desarrollo del módulo de clases, por sus características y las ventajas que posee: Cliente-Servidor.

La arquitectura Cliente-Servidor permite una mayor seguridad en los juegos *multiplayer*, evitando los trucos y trampas pues el mismo corre sobre un servidor seguro y no una copia en cada máquina como sucede en la arquitectura P2P. En esta arquitectura el consumo del ancho de banda crece linealmente cada vez que se conecta un nuevo jugador, al contrario de la *Peer-to-Peer*, en la que el crecimiento es exponencial. En los simuladores, permite que el servidor controle los cálculos principales de la aplicación y mantiene a los clientes actualizados con la situación real del entorno virtual y logrando una visualización de alta calidad, pues el procesador puede dedicarle más tiempo al *render* de la aplicación.

Consideraciones Técnicas Generales

El módulo de clases está orientado a los programadores de Sistemas de Realidad Virtual, que deben tener un conocimiento previo de programación orientada a objetos, pues el diseño e implementación se corresponderá con esta filosofía.

Para la comunicación se utilizarán los protocolos mencionados en este epígrafe, los cuales serán definidos por el programador en el momento de la inicialización de la aplicación en la que se utilizará el módulo de clases.

El módulo de clases no implementará algoritmos de detección o corrección de errores sobre el protocolo UDP en su primeras versiones, ya que en el caso de los simuladores y juegos para los cuales esta concebida, la velocidad de transferencia de los datos es más importante que su integridad. El extravió o duplicación de un paquete que se envía cada *frame* no sería percibido por el ojo humano, pues estas aplicaciones corren como mínimo de 30 a 60 *frames* por segundo. En caso de que una aplicación necesite transferir datos con un mayor grado de fiabilidad, se debe hacer uso del protocolo TCP el cual se encontrará disponible en el módulo de clases, de ahí que esta sea una de las razones fundamentales por la que el módulo será desarrollado sobre la base de los dos protocolos de comunicación, TCP y UDP.

El módulo, en respuesta a las necesidades y a las exigencias de la velocidad de transmisión, con el objetivo de lograr un mayor realismo en las aplicaciones de simulación, basará su funcionamiento en la programación multihilos.

Será adaptable a los sistemas operativos, *Windows* y familias de *Unix*, usando los sockets de los mismos, lo cual será transparente para el programador.

Todo el trabajo del módulo, se hará a través de funciones programadas puramente en el lenguaje de programación C++ *Standard*.

Conclusiones

Se quiere destacar lo novedoso de varias de las decisiones más importantes tomadas en este proyecto, como son: la posibilidad de utilizar el módulo de clases tanto en *Windows* como en *Unix*; y permitir el uso de UDP ó TCP para responder a las exigencias y necesidades según la aplicación a desarrollar.

El presente capítulo deja sentada las bases técnicas sobre las cuales será implementado el módulo de clases, para dar solución a los objetivos propuestos.

Capítulo 3 Descripción de la Solución Propuesta

Introducción

En el presente capítulo se comienza a tener una visión más práctica del módulo a desarrollar. Se definen las reglas del negocio y el modelo de dominio. Se obtienen los requisitos funcionales y los no funcionales que vienen a ser las capacidades o funciones que el sistema debe cumplir y las propiedades o cualidades que el producto debe tener. Se describen los procesos que responden a las funcionalidades definidas en los requerimientos funcionales que vienen a ser los Casos de Uso.

Reglas del Negocio

El módulo de clases trabajará sobre los protocolos de comunicación TCP y UDP.

El protocolo de transmisión a usar debe ser especificado, si no se toma por defecto UDP.

El puerto por el cual se transmiten y reciben los mensajes debe ser especificado por el usurario, si no se toma por defecto el 25000.

Los clientes deben establecer conexión después de estar activo el servidor.

Modelo del Dominio

Para el modelamiento del sistema, se seleccionó lo que se conoce como Modelo del Dominio, que viene a ser un modelo conceptual del sistema a desarrollar como se muestra en la siguiente imagen (imagen 7).

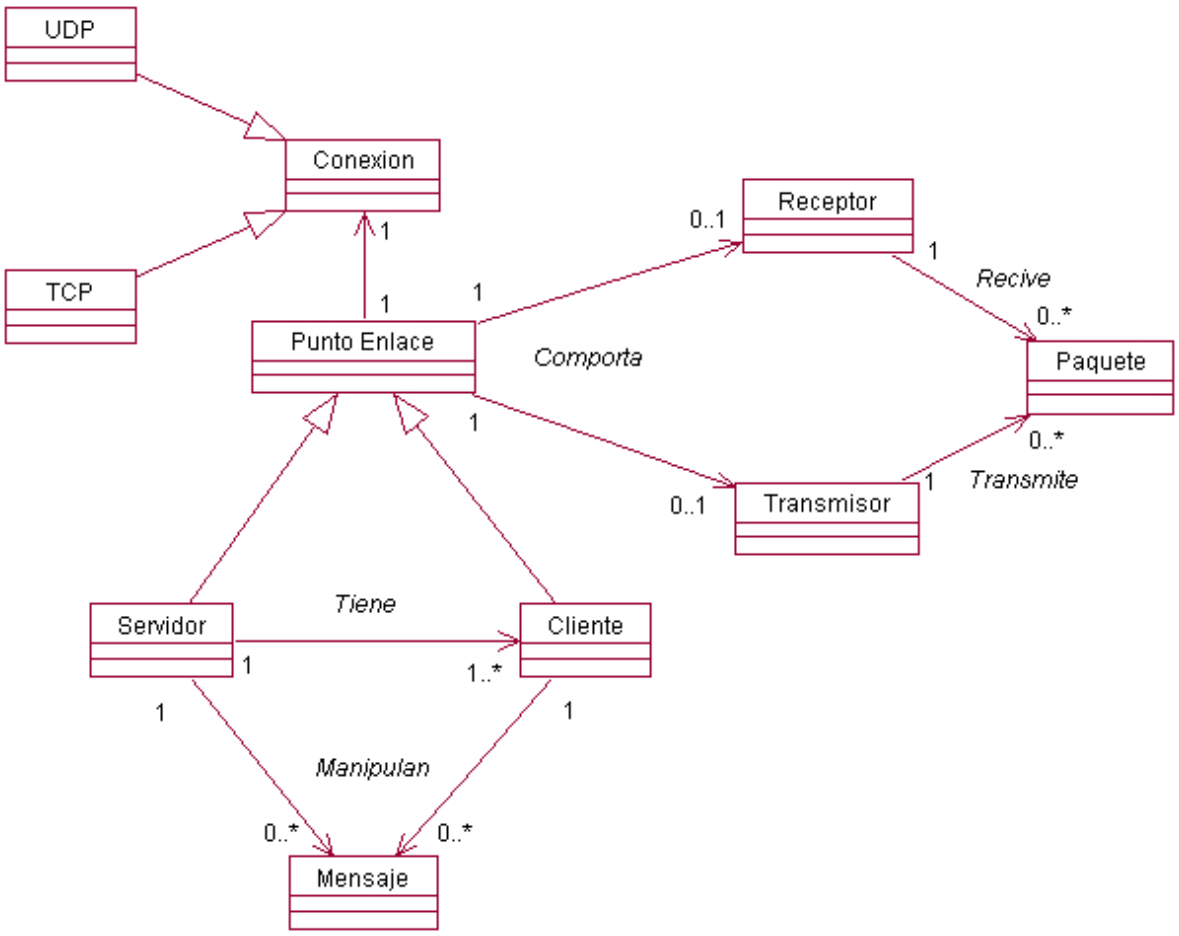

Imagen 7 Modelo del Dominio

En la imagen anterior se ve la relación de distintos conceptos que muestran la interacción del sistema. Se tiene un **Punto de Enlace** que viene a ser una computadora, este **Punto de Enlace** utiliza una **Conexión** para conectarse a otra computadora, esta **Conexión** puede ser de tipo **UDP** o **TCP**. El **Punto de Enlace** puede comportase como **Receptor** o como **Transmisor** de **Paquetes**. Los **Puntos de Enlace** pueden ser

de tipo **Servidor** o **Cliente** y manipular varios **Mensajes** para la comunicación entre ellos. Un **Servidor** tiene uno o muchos **Clientes** y un **Cliente** puede estar conectado a un solo **Servidor**.

Glosario de Términos del Dominio

Cliente*:* Un ordenador o un programa que accede a los servicios ofrecidos por otro ordenador llamado servidor.

Conexión*:* Método de comunicación entre dispositivos o software.

Mensaje*:* Está definido como la información que el emisor envía al receptor a través de un canal determinado o medio de comunicación.

Receptor*:* Es el destino de una sesión de comunicación que se encarga de recibir información.

Paquete*:* Un paquete es un pedazo de información enviada a través de la red. La unidad de datos que se envía a través de una red la cual se compone de un conjunto de bits que viajan juntos.

Punto de Enlace*:* Punto donde se realiza el enlace entre sistemas para la transferencia de información.

Servidor*:* Ordenador remoto que presta servicios a través de la red a otros ordenadores o programas llamados clientes.

TCP*:* *Transmission Control* Protocol (Protocolo de Control de Transmisión). Protocolo de redes, orientado a conexión, que se encarga chequeo de errores.

Transmisor*:* Es el origen de una sesión de comunicación que se encarga del envió de información.

UDP*:* *User Datagram Protocol* (Protocolo de Datagrama a Nivel de Usuario). Protocolo de redes no orientado a conexión, que no pide confirmación de la validez de los paquetes enviados.

Captura de Requisitos

En el presente epígrafe se expondrán los requisitos funcionales y no funcionales del sistema.

Requisitos Funcionales

1. Definir un puerto por defecto.
2. Tomar como por defecto el IP de la máquina.
3. Definir protocolo por defecto.
4. Definir cantidad de conexiones por defecto para el servidor.
5. Permitir especificar puerto.
6. Permitir especificar protocolo de conexión.
7. Permitir especificar cantidad de conexiones para el servidor.
8. Crear socket.
9. Crear cliente.
10. Establecer conexión con el servidor.
11. Crear servidor.
12. Crear lista de clientes en el servidor.
13. Aceptar conexión de los clientes.
14. Crear mensaje.
15. Enviar datos.
16. Recibir datos.
17. Calcular latencia de la red.
18. Agregar cliente a la lista de clientes.
19. Eliminar clientes de la lista de clientes.

20. Permitir obtener los clientes conectados.
21. Notificar a los clientes de la desconexión del servidor.
22. Notificar al servidor de la desconexión de los clientes.
23. Notificar a los clientes conectados cuando se agrega un nuevo cliente.
24. Notificar a los clientes conectados de la retirada de un cliente.
25. Finalizar la conexión en el Cliente.
26. Finalizar la conexión en el Servidor.

Requisitos No Funcionales

- **Usabilidad:** Los futuros usuarios del sistema serán programadores con conocimientos básicos de redes. El producto debe estar concebido para que el usuario piense en qué desea hacer y no cómo hacerlo.
- **Rendimiento:** Como aplicación de tiempo real, debe tener alto grado de velocidad en la transferencia de paquetes en por red, tiempo de respuesta y de recuperación.
- **Soporte:** En una versión inicial deberá ser compatible con la plataforma *Windows* y familias de *Unix*.
- **Legales:** Se regirá por las normas ISO 9000.
- ***Software:*** Sistema operativo *Windows* y familias de *Unix*.
- ***Hardware:*** Compatibilidad con cualquier tarjeta de red que acepte el Sistema Operativo sobre el cual se esta trabajando. Debe de existir un medio físico para la conexión entre los ordenadores que harán uso del módulo.
- **Diseño e implementación**: Debe utilizar transparentemente la interfaz de sockets que brindan los sistemas operativos y ser adaptable a trabajar con otras aplicaciones que necesiten transferencia

de datos por red. La programación del módulo se hará a través de funciones programadas puramente en el lenguaje C++ *Standard*. Se regirá por la filosofía de Programación Orientada a Objetos.

Modelo de casos de usos del sistema

En este epígrafe se reconocen los posibles actores del sistema a desarrollar y se conciben, a través de la agrupación de los requisitos funcionales anteriormente hallados, los posibles resultados de valor que le pueda brindar a sus actores, o lo que es lo mismo, los casos de uso del sistema.

Actor de sistema

Tabla 2 Actor del sistema

Actores	Justificación
Programador	Es el que hará uso y se beneficiará con las funcionalidades que brinda el módulo de clases.

Casos de uso de sistema (CUS)

1. Inicializar Conexión.
2. Inicializar Cliente.
3. Inicializar Servidor.
4. Enviar Paquetes.
5. Recibir Paquetes.
6. Gestionar Clientes.
7. Calcular Latencia.

8. Finalizar Conexión Cliente.
9. Finalizar Conexión Servidor.

Diagrama de CUS

En la siguiente imagen (imagen 8) se puede apreciar el diagrama de casos de uso del sistema.

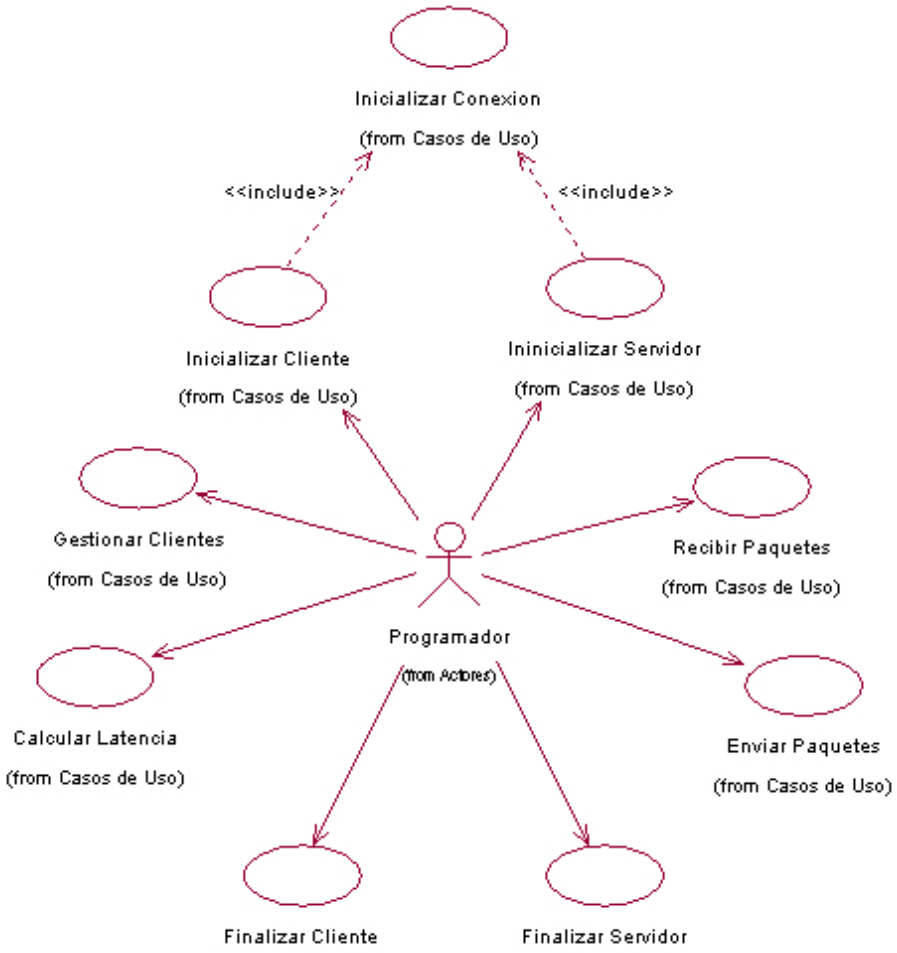

Imagen 8 Diagrama de Casos de Uso del Sistema

Especificación de los CUS en formato expandido

Tabla 3 Descripción CU Inicializar Conexión

Nombre del CU	Inicializar Conexión	
Actores		
Propósito	Realizar las llamadas básicas para la inicialización del módulo.	
Resumen	El CU se inicia cuando es llamado por el CU “Inicializar Cliente” o el CU “Inicializar Servidor”. Aquí se crea y prepara el socket para establecer conexión.	
Referencias	RF 1, 2, 3, 5, 6 y 8	
Precondiciones	El módulo de clases no puede estar inicializado.	
Curso Normal de los Eventos		
Acciones del Actor		**Respuestas del Sistema**
		1- Se inicializa la librería de socket del sistema operativo.
		2- Se crea un socket con el protocolo asignado.
		3- Se establece el socket como no bloqueante.
		4- Se crea una estructura y se le asigna el puerto asignado.
		5- Se le asigna el IP del ordenador a la estructura.
		6- Se adjunta la estructura al socket.

	7- Se devuelve le socket listo para transmisión de paquetes.
Curso Alternos de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
	2- No se define protocolo, se toma UDP por defecto.
	4- No se define puerto, se toma 25000 por defecto.
Prioridad:	Crítico

Tabla 4 Descripción CU Inicializar Cliente

Nombre del CU	Inicializar Cliente
Actores	Programador
Propósito	Preparar la aplicación que se va a comportar como cliente para la transferencia de mensajes con el servidor.
Resumen	En el CU se crea el cliente, se preparan las condiciones para establecer conexión con un servidor y se prepara la estructura que almacena la dirección destino.
Referencias	RF 9, 10 y CU “Inicializar Conexión”
Precondiciones	El módulo de clases no puede estar inicializado.
Poscondiciones	La aplicación cliente que hace uso del módulo queda en condiciones de enviar o recibir mensajes del servidor.

Curso Normal de los Eventos	
Acciones del Actor	**Respuestas del Sistema**
1- El programador solicita inicializar un cliente. Entrega el puerto de origen, el puerto y el IP del servidor al que solicitará conexión y especifica un protocolo de transmisión, el programador define un identificador para el cliente.	
	2- Se crea el cliente y se le asigna el identificador.
	3- Se inicializa la conexión. Referencia al CU "Inicializar Conexión".
	4- Se crea una estructura con la dirección IP y el puerto del servidor.
	5- Se ejecuta intento de conexión.
	6- Queda establecida la conexión con el servidor.
Curso Alterno de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
	6- Se muestra mensaje de intento de conexión fallido.
Prioridad:	Crítico

Tabla 5 Descripción CU Inicializar Servidor

Nombre del CU	Inicializar Servidor
Actores	Programador
Propósito	Preparar la aplicación que se va a comportar como servidor para la transferencia de mensajes con los clientes.
Resumen	En el CU se crea el servidor, se preparan las condiciones para recibir clientes y se determina el número máximo de clientes que se pueden conectar.
Referencias	RF 4, 7, 11, 12, 13, 18, 23 y CU “Inicializar Conexión”.
Precondiciones	El módulo de clases no puede estar inicializado.
Poscondiciones	La aplicación que hace uso del módulo queda en condiciones de intercambiar mensajes y manipular varios clientes.

Curso Normal de los Eventos	
Acciones del Actor	**Respuestas del Sistema**
1- El programador solicita inicializar un servidor. Entrega el puerto y el IP del servidor, especifica un protocolo de transmisión y el número máximo de clientes.	
	2- Se crea el servidor.
	3- Se inicializa la conexión. Referencia al CU “Inicializar Conexión”.

	4- Se especifica el número máximo de conexiones que soporta el servidor.
	5- Se crea la lista donde se almacenarán los clientes.
	6- Se espera por el intento de conexión de los clientes.
	7- Se acepta los intentos de conexión.
	8- Se verifica que el cliente no exista.
	9- Se adiciona en la lista de clientes.
	10- Se comunica al resto de los clientes la existencia de un nuevo cliente.
Curso Alternos de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
	4- Se toma 10 por defecto.
	9- Se envía un mensaje advirtiendo de la existencia del cliente.
Prioridad:	Crítico

Tabla 6 Descripción CU Enviar Paquetes

Nombre del CU	Enviar Paquetes	
Actores	Programador	
Propósito	Enviar paquetes desde el servidor o cliente.	
Resumen	El CU se inicia cuando el servidor necesita mantener actualizados a sus clientes o cuando los clientes necesitan intercambiar algún tipo de información.	
Referencias	RF 14 y 15	
Precondiciones	Debe estar inicializado el servidor y tener al menos un cliente en la lista de clientes.	
Curso Normal de los Eventos		
Acciones del Actor		**Respuestas del Sistema**
1- Se entrega los datos a enviar.		
		2- Se crea el mensaje a enviar con los datos entregados.
3- Se solicita enviar paquetes		
		4- Se envía el mensaje al ordenador de destino.
		5- Se devuelve la cantidad de bytes enviados.
Curso Alternos de los Eventos		
Acciones del Actor		**Respuesta del Sistema**
		4- Error en la transmisión.
		5- Se muestra un mensaje de error.
Prioridad:	Crítico	

Tabla 7 Descripción CU Recibir Paquetes

Nombre del CU	Recibir Paquetes	
Actores	Programador	
Resumen	El CU se inicia cuando se necesita recibir datos en los clientes para actualizar la escena con los datos enviados por el servidor ó en el servidor para procesar los datos enviados por alguno de los clientes.	
Referencias	RF 16	
Precondiciones	Debe estar inicializado el servidor y tener al menos un cliente en su lista de clientes.	
Curso Normal de los Eventos		
Acciones del Actor		**Respuestas del Sistema**
1- Solicita recibir mensaje. Se entrega el *buffer* donde se almacenaran los datos recibidos		
		2- Se recibe el mensaje.
		3- Se procesa la información recibida.
		5- Se devuelven los datos recibidos.
Curso Alternos de los Eventos		
Acciones del Actor		**Respuesta del Sistema**
		2- Error en la recepción.
		3- Se muestra un mensaje de error. Finaliza el CU
Prioridad:	Crítico	

Tabla 8 Descripción CU Gestionar Clientes

Nombre del CU	Gestionar Clientes
Actores	
Propósito	Permitir consultar y eliminar clientes de la lista de clientes del servidor.
Resumen	El caso de uso se inicia cuando se necesita eliminar clientes u obtener los clientes conectados en ese momento.
Referencias	RF 19, 20 y 24
Precondiciones	Debe estar inicializado el servidor.
Poscondiciones	Queda eliminado un cliente de la lista o se devuelven los clientes conectados hasta el momento.
Curso Normal de los Eventos	
Acciones del Actor	**Respuestas del Sistema**
1- Solicita eliminar clientes de la lista u obtener los clientes conectados hasta el momento.	
	2- Se ejecuta alguna de las siguientes acciones: a) Se decide eliminar cliente, ir a la sección “Eliminar Cliente”. b) Se decide obtener la lista de clientes conectados, ir a la sección “Obtener Clientes Conectados”.

Sección: Eliminar Cliente	
Acciones del Actor	**Respuesta del Sistema**
1- El programador solicita eliminar un cliente y entra los datos del cliente.	
	2- Se verifica que el cliente exista.
	3- Se elimina de la lista de clientes.
	4- Se envía un mensaje informando a los clientes restantes de la retirada del cliente.
Curso Alternos de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
	3- Se envía un mensaje advirtiendo que no existe el cliente.
Sección: Obtener Clientes Conectados	
Acciones del Actor	**Respuesta del Sistema**
1- Solicita obtener la lista de clientes conectados.	
	2- Se verifica que exista al menos un cliente en la lista.
	3- Se devuelve la lista de clientes
Curso Alternos de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
	3- Se envía un mensaje advirtiendo que no hay clientes conectados.
Prioridad:	Crítico

Tabla 9 Descripción CU Finalizar Conexión Cliente

Nombre del CU	Finalizar Conexión Cliente.
Actores	Programador
Propósito	Finalizar la conexión del cliente con el servidor correctamente.
Resumen	En el caso de uso se realiza el cierre de la conexión del cliente con el servidor, cerrando el socket y devolviendo los recursos usados al sistema.
Referencias	RF 22 y 25
Precondiciones	Debe estar inicializado el servidor.

Curso Normal de los Eventos	
Acciones del Actor	**Respuestas del Sistema**
1- Solicita cerrar la conexión desde el cliente.	
	2- Se envía un mensaje informando al servidor que se terminará la conexión.
3-Solicita entregar los recursos al sistema.	
	4- Se cierra el socket.
	5- Se destruye el cliente.

Prioridad:	Crítico

Tabla 10 Descripción CU Finalizar Conexión Servidor

Nombre del CU	Finalizar Conexión Servidor.
Actores	Programador
Propósito	Finalizar la conexión del servidor con los clientes correctamente.
Resumen	En el caso de uso se realiza el cierre de la conexión del servidor con sus clientes, cerrando el socket y devolviendo los recursos usados al sistema.
Referencias	RF 21 y 26
Precondiciones	Debe estar inicializado el servidor.

Curso Normal de los Eventos	
Acciones del Actor	**Respuestas del Sistema**
1- Solicita cerrar la conexión desde el servidor.	
	2- Se envía un mensaje informando a los clientes que se terminará la conexión.
3- Solicita entregar los recursos al sistema.	
	4-Se destruye la lista de clientes.
	5- Se cierra el socket.

Prioridad:	Crítico

Tabla 11 Descripción CU Calcular Latencia

Nombre del CU	Calcular Latencia
Actores	Programador
Propósito	Calcular la latencia de la red entre clientes y entre clientes y el servidor.
Resumen	El caso de uso se inicia cuando el programador necesita calcular la latencia de la red desde el servidor.
Referencias	RF 17
Precondiciones	Ya debe estar inicializado el servidor y tener al menos un cliente en la lista de clientes.

Curso Normal de los Eventos	
Acciones del Actor	**Respuestas del Sistema**
1- Solicita calcular latencia de la red desde el servidor.	
	2- Se envía un *ping* al ordenador que queremos calcular la latencia.
	3- Se escucha el *ping* de regreso.
	4- Se calcula de la diferencia de tiempo entre el *ping* que se envía y el de regreso.
	5- Se muestra la diferencia de tiempo, que es la latencia entre los ordenadores.
Prioridad:	Secundario

Conclusiones

En el presente capítulo se concretó qué es exactamente lo que espera el usuario del sistema. Para ello quedaron establecidos los requisitos funcionales y los no funcionales de éste y se describieron los casos de uso que darán solución al problema.

Capítulo 4 Diseño e Implementación del Sistema

Introducción

La primera parte del capítulo encierra el diagrama de clase del sistema, propuesto como resultado del refinamiento de las etapas anteriores. Posteriormente se muestran los diagramas de secuencia, elaborados a partir de los casos de uso que intervienen en el desarrollo del módulo.

Se presentan además los componentes físicos, que se traducen en los ficheros .h y .cpp correspondientes a la implementación en C++. Además se elabora el diagrama de despliegue del sistema.

.

Diagrama de Clases del Diseño

Antes de pasar a mostrar el diagrama de clases, se aclararán dos cuestiones importantes para la comprensión de este:

1. La nomenclatura utilizada para el diagrama de clases, se explica en el epígrafe “Estándares de codificación” del este capítulo.
2. En la realización del módulo, se generó el código con la herramienta utilizada para el proceso de desarrollo del sistema (*Rational Rose Enterprise Edition*, 2003), la cual brinda la posibilidad de generar los métodos de acceso a miembros de clases (“*gets*” y “*sets*”), constructores por defecto, constructores de copia y destructores, aun cuando no se hallan incluido en las especificaciones de las clases; por tanto, y para evitar su repetición, se omite en las clases los métodos antes mencionados.

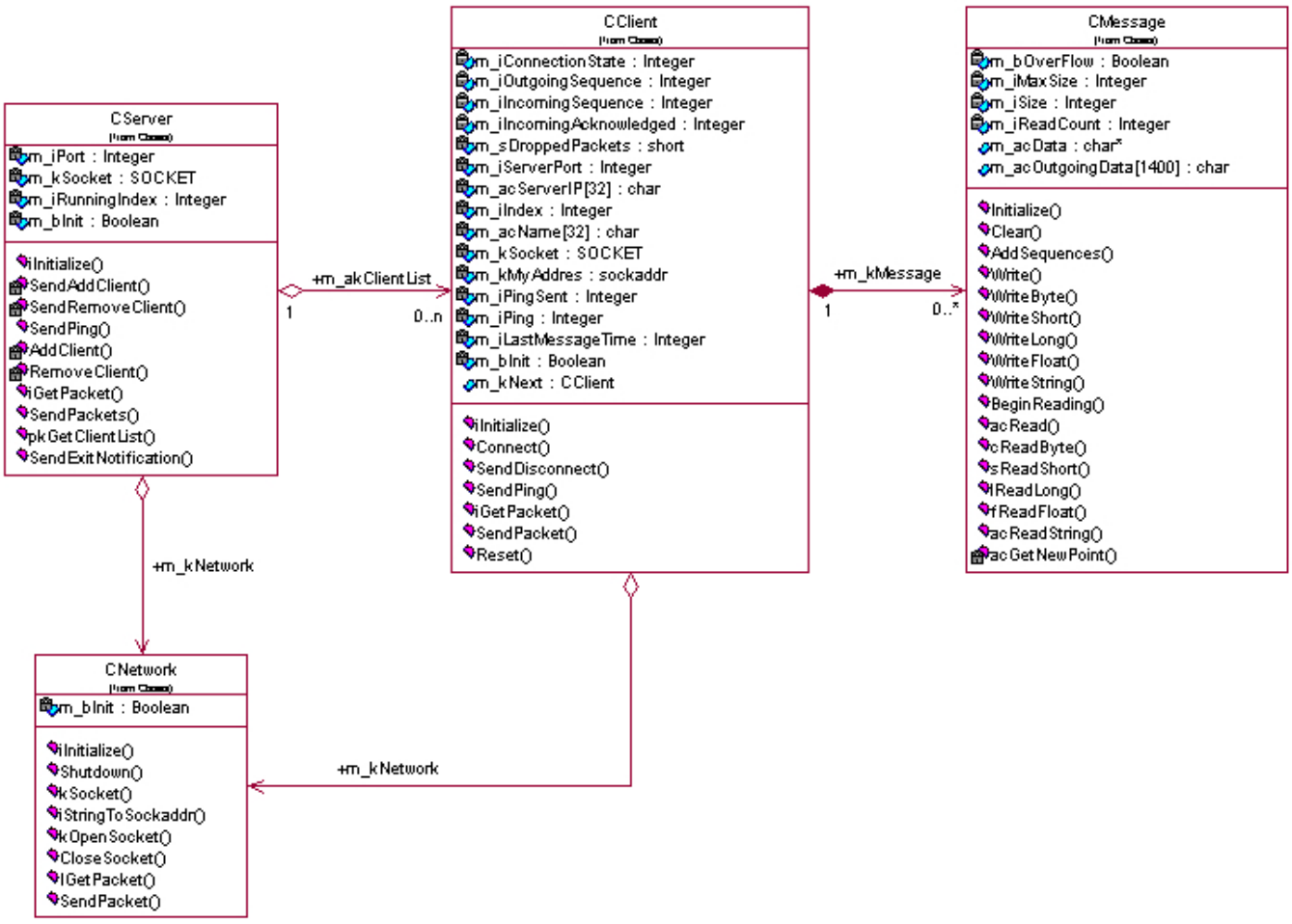

Imagen 9 Diagrama de Clases de Diseño

Descripción de las Clases de Diseño

Tabla 12 Descripción de la clase CMessage

Nombre:	CMessage	
Tipo de clase:	Entidad	
Atributo		**Tipo**
m_bOverFlow		bool
m_iMaxSize		int
m_iSize		int
m_iReadCount		int
m_acData		char*
m_acOutgoingData[1400]		char
Para cada responsabilidad:		
Nombre:	Initialize(char *arg_acBuffer, int arg_iLength)	
Descripción:	Se encarga de establecer el tamaño máximo del mensaje a partir del valor entrado por parámetro. Además recibe la dirección de memoria del buffer de datos.	
Nombre:	Clear()	
Descripción:	Se encarga de limpiar el contenido almacenado en m_acData.	
Nombre:	AddSequences(CClient *arg_pkClient)	
Descripción:	Se encarga de adjuntar al mensaje información del cliente a través del cual se enviará el mismo.	
Nombre:	Write(void *arg_pvBuffer, int arg_iLength)	
Descripción:	Se encargan de escribir información en el mensaje a partir de la dirección donde de almacenan y el tamaño que contienen los mismos.	
Nombre:	WriteByte(char arg_cBuffe)	
Descripción:	Se encarga de escribir un byte en el mensaje	
Nombre:	WriteShort(short arg_sBuffer)	
Descripción:	Se encarga de escribir un dato de tipo short en el mensaje.	
Nombre:	WriteLong(long arg_lBuffer)	
Descripción:	Se encarga de escribir un dato de tipo long en el mensaje.	

Nombre:	WriteFloat(float arg_fBuffer)
Descripción:	Se encarga de escribir un dato de tipo float en el mensaje.
Nombre:	WriteString(char *arg_acBuffer)
Descripción:	Se encarga de escribir un dato de tipo string en el mensaje..
Nombre:	BeginReading()
Descripción:	Se encarga de reiniciar la variable m_iReadCount.
Nombre:	acRead(int arg_iSize)
Descripción:	Se encarga leer determinada cantidad de bytes pasada por parámetros y devolver la información leida como arreglo de char.
Nombre:	cReadByte()
Descripción:	Se encarga de leer un byte del mensaje.
Nombre:	sReadShort()
Descripción:	Se encarga de leer y devolver un dato de tipo short del mensaje.
Nombre:	lReadLong ()
Descripción:	Se encarga de leer y devolver un dato de tipo long del mensaje
Nombre:	cReadFloat(void)
Descripción:	Se encarga de leer y devolver un dato de tipo float del mensaje
Nombre:	cReadString()
Descripción:	Se encarga de leer y devolver un dato de tipo string del mensaje

Tabla 13 Descripción de la clase CNetwork

Nombre:	CNetwork	
Tipo de clase:	Entidad	
Atributo		**Tipo**
m_bInit		bool
Para cada responsabilidad:		
Nombre:	CNetwork ()	
Descripción:	Constructor de la clase.	

Nombre:	iInitialize ()
Descripción:	Inicializa las librerías de sockets del sistema operativo.
Nombre:	Shutdown ()
Descripción:	Cierra el uso de las librerías de sockets del sistema operativo.
Nombre:	kSocket (int arg_iProtocol);
Descripción:	Método encargado de crear un socket según el tipo de protocolo pasado por parámetro.
Nombre:	iStringToSockaddr(char *arg_acAddressString, struct sockaddr *arg_pkAddress)
Descripción:	Se encarga de convertir un arreglo de char pasado como parámetro, que debe ser una dirección IP a una estructura del tipo sockaddr.
Nombre:	iGetPacket (SOCKET arg_kSock, char *arg_acData, struct sockaddr *arg_pkFrom)
Descripción:	Método encargado de recibir los paquetes enviados por el host remoto. Se le pasa como parámetro el socket del cual se debe leer, el buffer donde serán almacenados temporalmente los datos y la estructura sock addr para almacenar la dirección del emisor.
Nombre:	SendPacket (SOCKET arg_kSock, int arg_iLength, char *arg_acData, struct sockaddr kAddr)
Descripción:	Método encargado de enviar los mensajes al host remoto. Se le pasa como parámetro el socket donde debe escribir el mensaje, el tamaño del mensaje, los datos del mensaje y la estructura sockaddr con la dirección del receptor.
Nombre:	kOpenSocket (arg_iLocalPort , arg_iProtocol)
Descripción:	Prepara el socket asignándole el puerto y el protocolo a usar para la transferencia de mensajes.
Nombre:	CloseSocket (SOCKET arg_kSock)
Descripción:	Método encargado de cerrar el socket.

Tabla 14 Descripción de la clase CClient

Nombre:	CClient	
Tipo de clase:	Entidad	
Atributo		**Tipo**
m_iConnectionState		int
m_iOutgoingSequence		int
m_iIncomingSequence		int
m_iIncomingAcknowledged		int
m_sDroppedPackets		short
m_iServerPort		int
m_acServerIP[32]		char
m_iIndex		int
m_acName[32]		char
m_kSocket		SOCKET
m_kMyAddres		sockaddr
m_iPingSent		int
m_iLastMessageTime		int
m_bInit		bool
m_kMessage		CMessage
m_kNetwork		CNetwork
m_kNext		CClient
Para cada responsabilidad:		
Nombre:	CClient ()	
Descripción:	Constructor de la clase.	
Nombre:	iInitialize (int arg_iLocalPort, int arg_iRemotePort, char *arg_acRemoteIP, int arg_iProtocol)	
Descripción:	Se encarga de inicializar la conexión en el cliente, recibiendo como parámetro el puerto local, el puerto del servidor con el cual se establecerá la conexión, el IP del	

	servidor y el protocolo de comunicación a usar.
Nombre:	Connect (char *arg_acName);
Descripción:	Conecta al cliente con el servidor y se le pasa el nombre que tendrá ese cliente.
Nombre:	SendDisconnect ()
Descripción:	Envía un mensaje al servidor avisando la desconexión del cliente.
Nombre:	SendPing ()
Descripción:	Se encarga de responder la petición de ping enviada por el servidor.
Nombre:	iGetPacket (char *arg_acData)
Descripción:	Método encargado de recibir y analizar los paquetes enviado por el servidor y se le pasa el buffer donde serán almacenados temporalmente. Devuelve los bytes recibidos.
Nombre:	SendPacket (CMessage *pkMessage)
Descripción:	Método encargado de enviar los mensajes al servidor, se le pasa el mensaje a enviar.
Nombre:	Reset ()
Descripción:	Se encarga de reiniciar todos los variables.

Tabla 15 Descripción de la clase CServer

Nombre:	CServer	
Tipo de clase:	Entidad	
Atributo		**Tipo**
m_akClientList		CClient*
m_iPort		int
m_kSocket		SOCKET
m_iRunningIndex		int
m_bInint		bool
m_kNetwork		CNetwork
Para cada responsabilidad:		

Nombre:	CServer()
Descripción:	Constructor de la clase
Nombre:	Initialize(int arg_iLocalPort, int arg_iProtocol, int arg_iCantClient)
Descripción:	Se encarga de inicializar la conexión en el servidor, recibiendo como parámetro el puerto, el protocolo y la cantidad de clientes que soportara conectados.
Nombre:	SendAddClient()
Descripción:	Se encarga de enviar un mensaje a los clientes conectados notificando de la incorporación de un nuevo cliente.
Nombre:	SendRemoveClient()
Descripción:	Se encarga de enviar un mensaje a los clientes conectados notificando de la desconexión de un cliente.
Nombre:	SendPing()
Descripción:	Se encarga de calcular la latencia de red que existe entre el servidor y los clientes conectados.
Nombre:	AddClient()
Descripción:	Se encarga de adicionar un cliente nuevo a la lista de clientes conectados.
Nombre:	RemoveClient()
Descripción:	Se encarga de eliminar un cliente de la lista de clientes conectados
Nombre:	iGetPacket (char* arg_acData)
Descripción:	Se encarga de recibir datos desde cualquier cliente y almacenarlos en el arreglo de char recibido como parámetro. Retorna la cantidad de byte recibidos.
Nombre:	SendPackets ()
Descripción:	Se encarga de enviar la información entrada por parámetro al los clientes conectados.
Nombre:	pkGetClientList ()
Descripción:	Método de acceso a la lista de clientes conectados
Nombre:	SendExitNotification()
Descripción:	Se encarga de enviar un mensaje notificando a los clientes conectados de la desconexión del servidor.

Diagramas de Secuencia

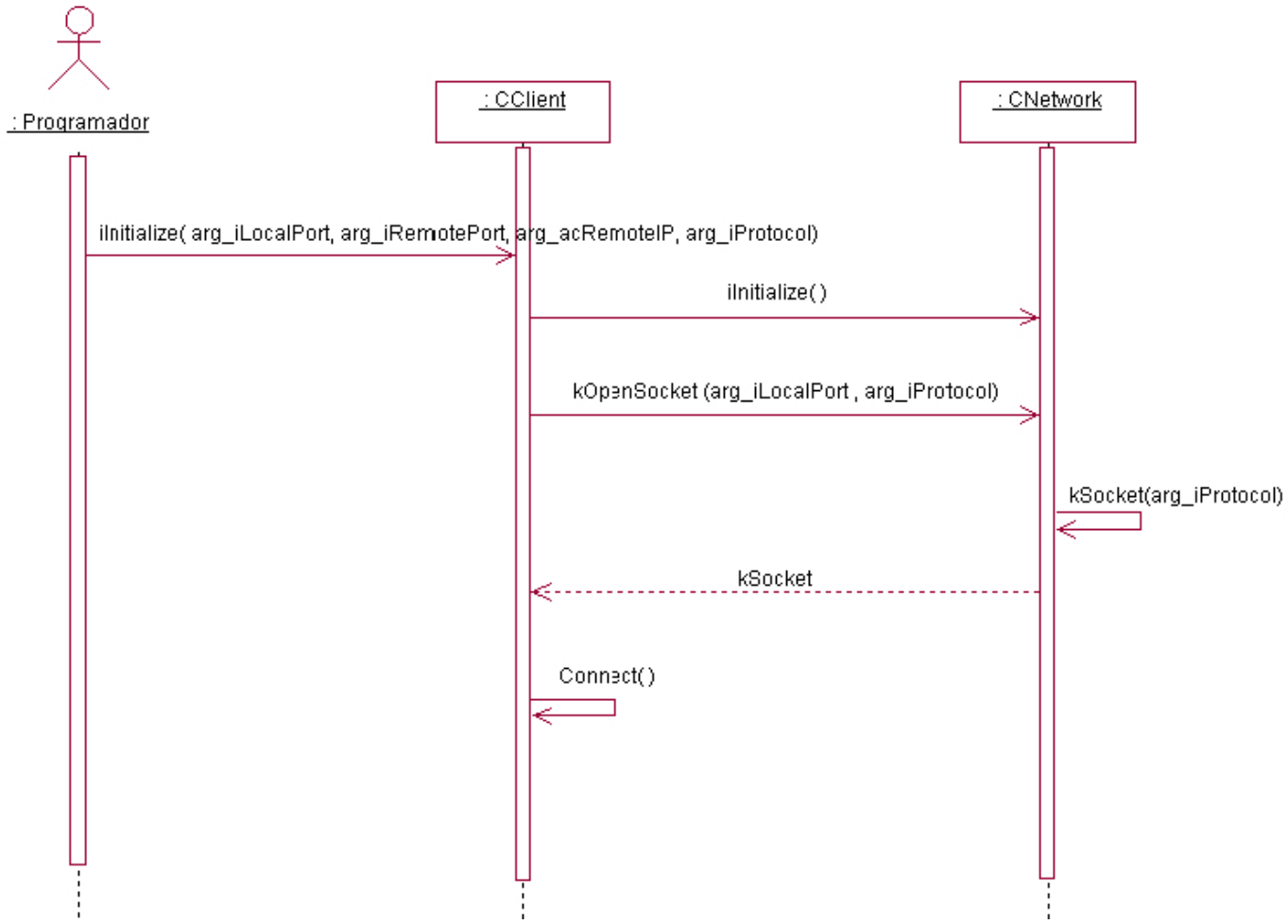

Imagen 10 Diagrama de Secuencia "Inicializar Cliente"

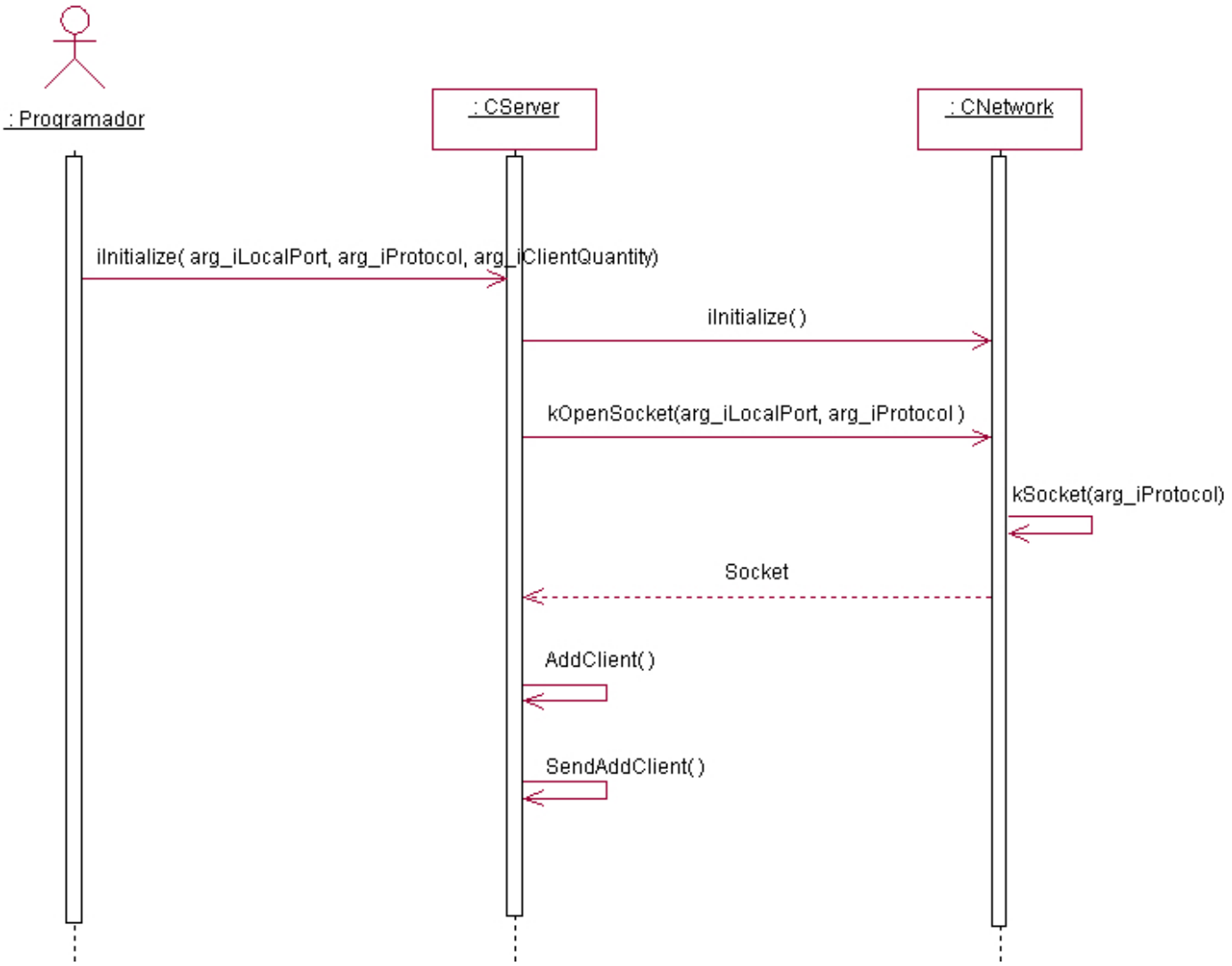

Imagen 11 Diagrama de Secuencia "Inicializar Servidor"

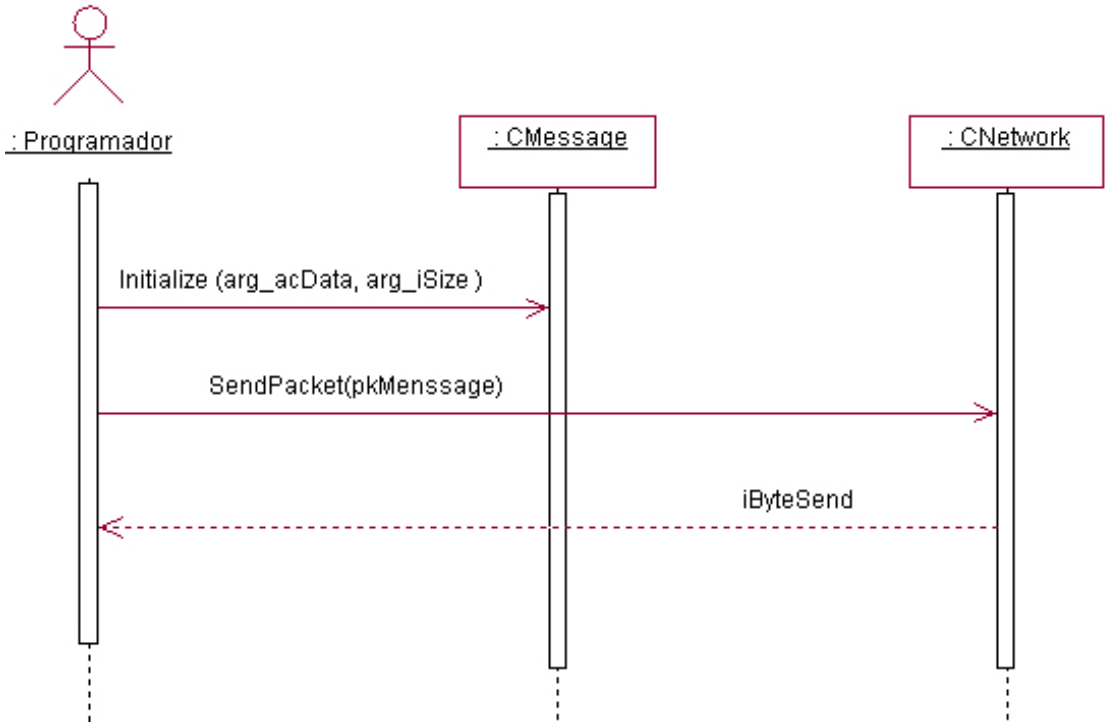

Imagen 12 Diagrama de Secuencia "Enviar Mensaje"

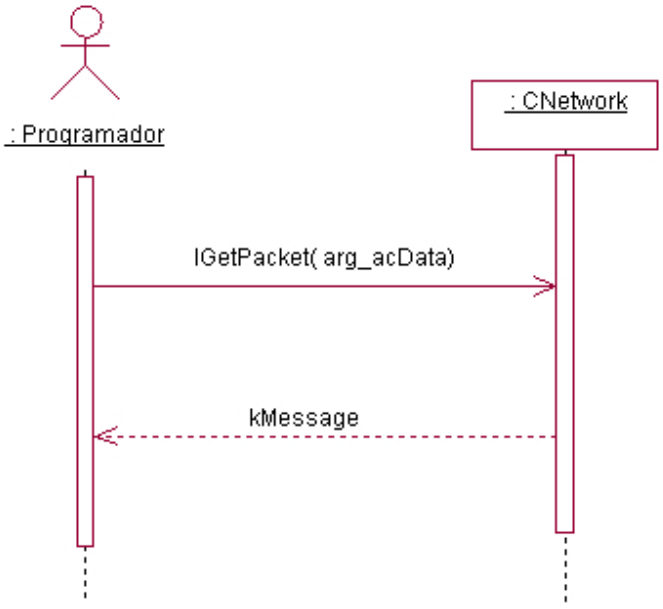

Imagen 13 Diagrama de Secuencia "Recibir Mensaje"

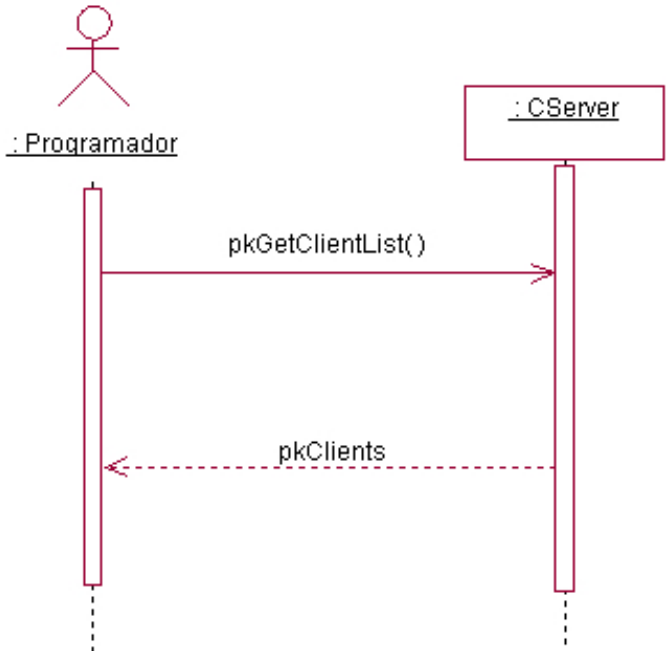

Imagen 14 Diagrama de Secuencia "Obtener Clientes Conectados”

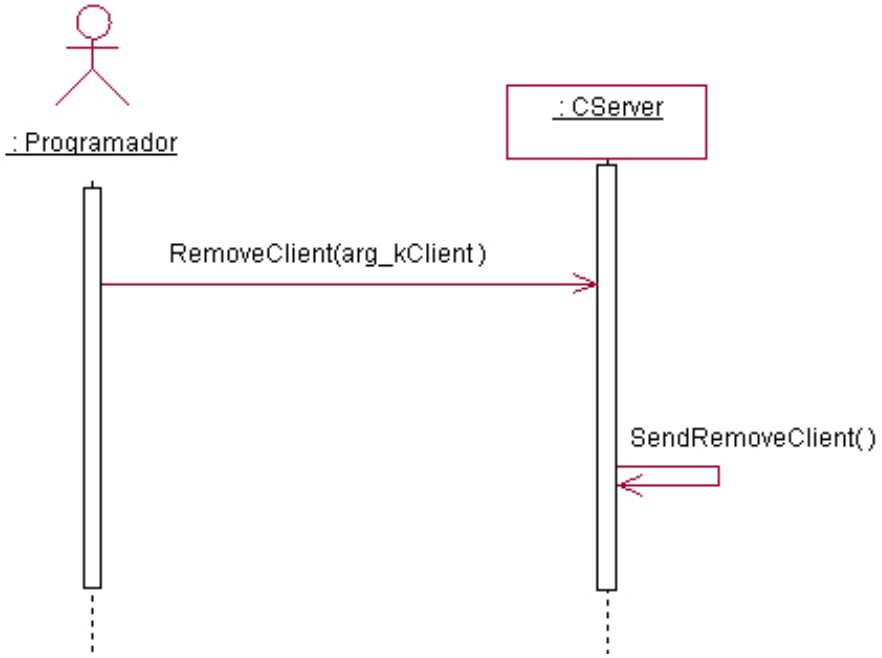

Imagen 15 Diagrama de Secuencia "Eliminar Cliente”

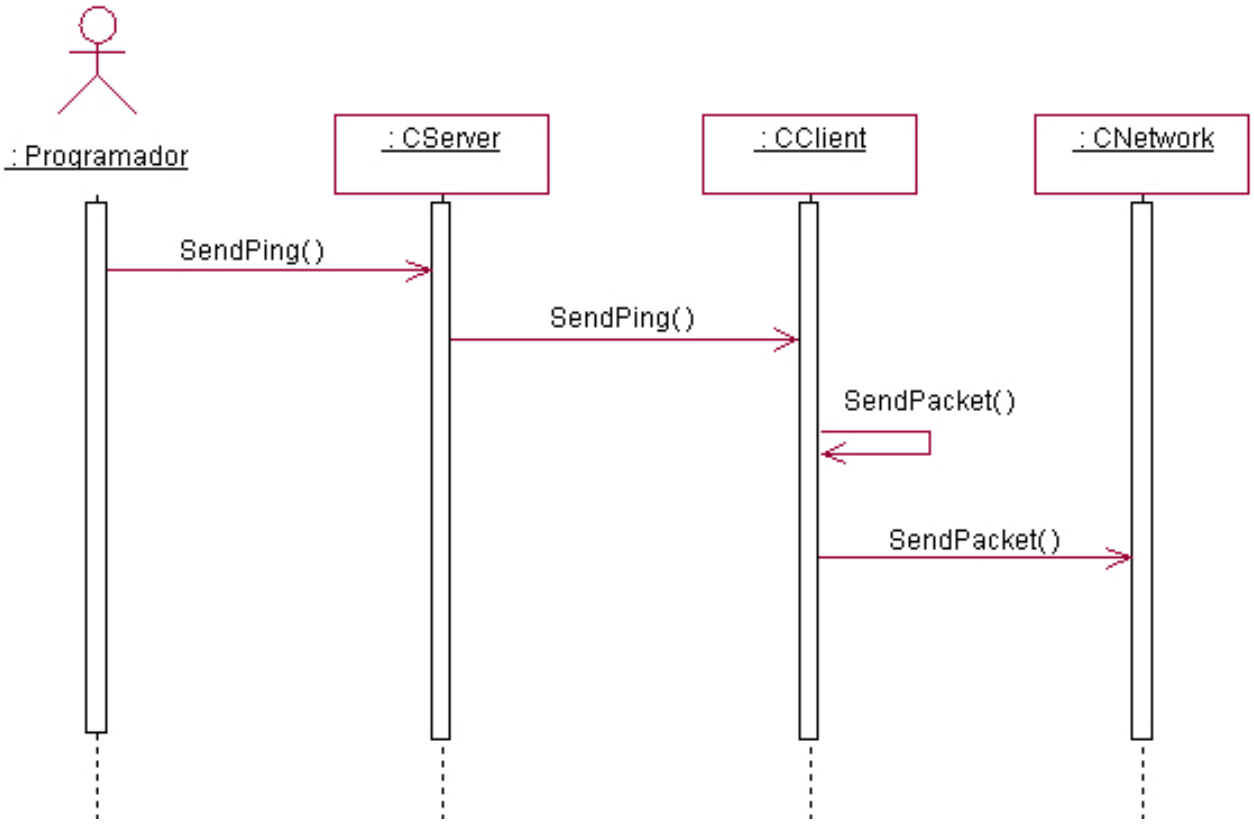

Imagen 16 Diagrama de Secuencia "Calcular Latencia"

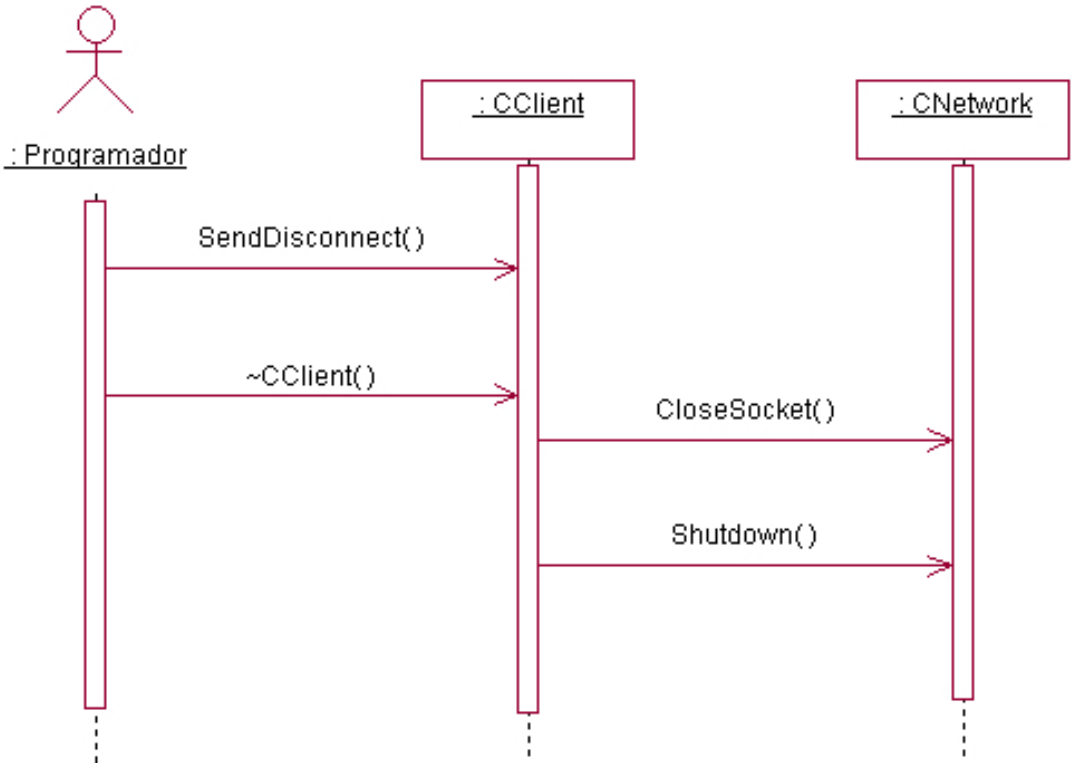

Imagen 17 Diagrama de Secuencia "Finalizar Cliente"

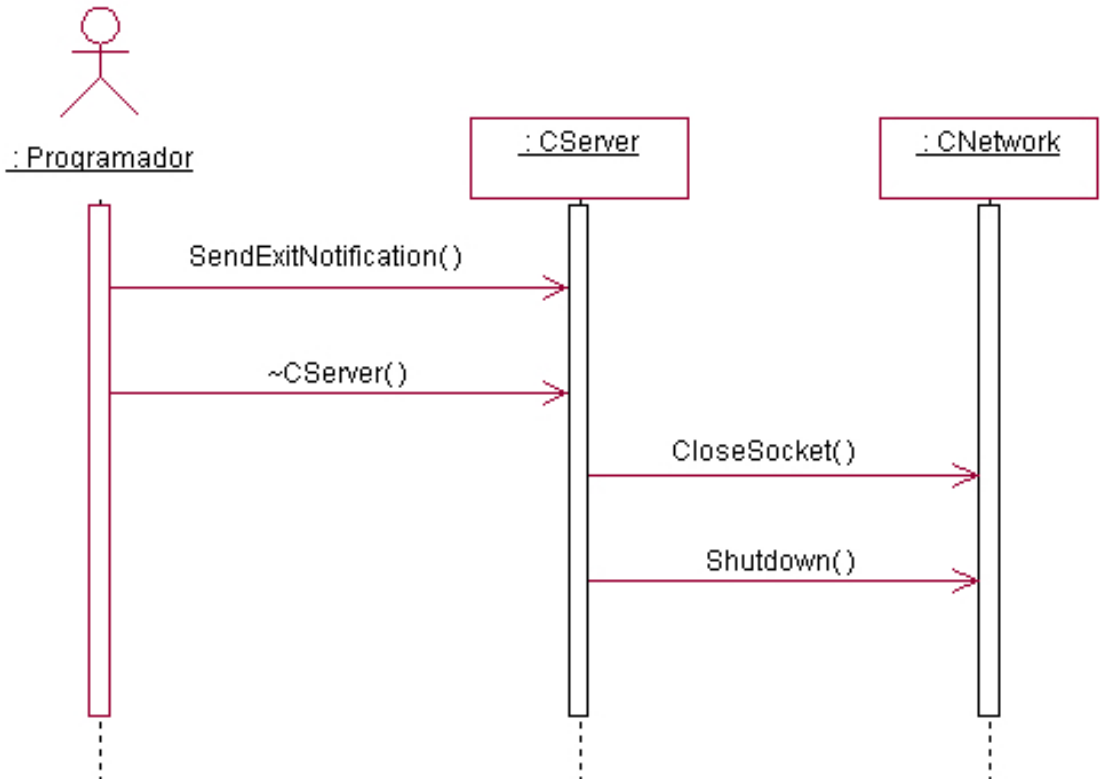

Imagen 18 Diagrama de Secuencia "Finalizar Servidor"

Estándares de codificación

El código del módulo sigue algunos estándares propuestos por el grupo de desarrollo (respetando los estándares de codificación para C++). Está programado en inglés, debido que las palabras son simples, no se acentúan y es un idioma muy difundido en el mundo informático.

El conocimiento de los estándares seguidos para el desarrollo de la misma permitirá un mayor entendimiento del código.

Nombre de los ficheros:

Se nombrarán los ficheros .h y .cpp de la siguiente manera:

SLNameOfUnits.cpp

Constantes:

Las constantes se nombrarán con mayúsculas, utilizándose el "_" para separar las palabras: *MY_CONST_ZERO* = 0;

Tipos de datos:

Los tipos se nombrarán siguiendo el siguiente patrón:

Enumerados: *enum **E**MyEnum {**ME**_VALUE, **ME**_OTHER_VALUE};*

Indicando con "**E**" que es de tipo enumerado. Nótese que las primeras letras de las constantes de enumerados son las iniciales del nombre del enumerado. Véase otro ejemplo:

*enum **E**Node**T**ype {**NT**_GEOMETRYNODE,…};*

Estructuras: *struct **S**MyStruct {…};*

Indicando con "**S**" que es una estructura. Las variables miembros de la estructura se nombrarán igual que en las clases, leer más adelante.

Clases: *class **C**ClassName;*

Indicando con "**C**" que es una clase.

Interfaces: ***I**MyInterface*

Indicando con "**I**" que es una interfaz.

Listas e iteradotes STD:

*vector<> **T**Name**List**;*

*TNameList::iterator **T**Name**ListIter**;*

Declaración de variables:

Los nombres de las variables comenzarán con un identificado del tipo de dato al que correspondan, como se muestra a continuación. En el caso de que sean variables miembros de una clase, se le antepondrá el identificador "**m**" (en minúscula), si son globales se les antepondrá la letra "**g**", y en caso de ser argumentos de algún método, se les antepondrá el prefijo "**arg**_".

Tipos simples:

*bool **b**VarName;*

*int **i**Name;*

*unsigned int **ui**Name;*

*float **f**Name;*

*char **c**Name;*

char ***ac****Name;* // arreglo de caracteres

char ***pc****Name;* // puntero a un char

*char** ***aac****Name;* // bidimensional

*char** ***apc****Name;* // arreglo de punteros

*bool **m_b**MemberVarName;* //variable miembro

*char **g**C**GlobalVarName;* //variable global, no se le antepone “”

*short **s**Name;*

Instancias de tipos creados:

*EMyEnumerated **e**Name;*

*SMyStructure **k**Name;*

*CClassName **k**ObjectName;*

*CClassName** ***pk****Name;* //puntero a objeto

*CClassName** ***ak****Name;* //arreglo de objetos

*CClassName** ***ak****Name;* // variable miembro de clase

*IMyInterface** ***pI****Name;* //puntero interfaces

Métodos:

En el caso de los métodos, se les antepondrá el identificador del tipo de dato de devolución, y en caso de no tenerlo (void), no se les antepondrá nada.

En el caso de los argumentos se les antepone el prefijo “**arg**_”

Constructor y destructor:

C*ClassName (bool* ***arg****_bVarName, float&* ***arg****_fVarName);*

*~****C****ClassName ();*

Funciones:

*b*ool **b**Function1 (…);

int **pi**Function2 (…);*

CClassName **pk**Function3 (…);*

Procedimientos:

void Procedure4 (…);

Métodos de acceso a miembros:

Los métodos de acceso a los miembros de las clases no se nombrarán "*Gets*" y "*Sets*", sino como los demás métodos, pero con el nombre de la variable a la que se accede y sin "**m**_":

int ***i****MyVar;* //variable

Obtención del valor:

```
int iMyVar();
{
   return iMyVar;
}
```

Establecimiento del valor:

```
void MyVar(char* arg_iMyVar)
{
  iMyVar = arg_iMyVar;
}
```

Obtención y establecimiento del valor:

```
int& iMyVar();
{
   return iMyVar;
}
```

Diagrama de despliegue

Imagen 19 Diagrama de Despliegue

Diagrama de componentes

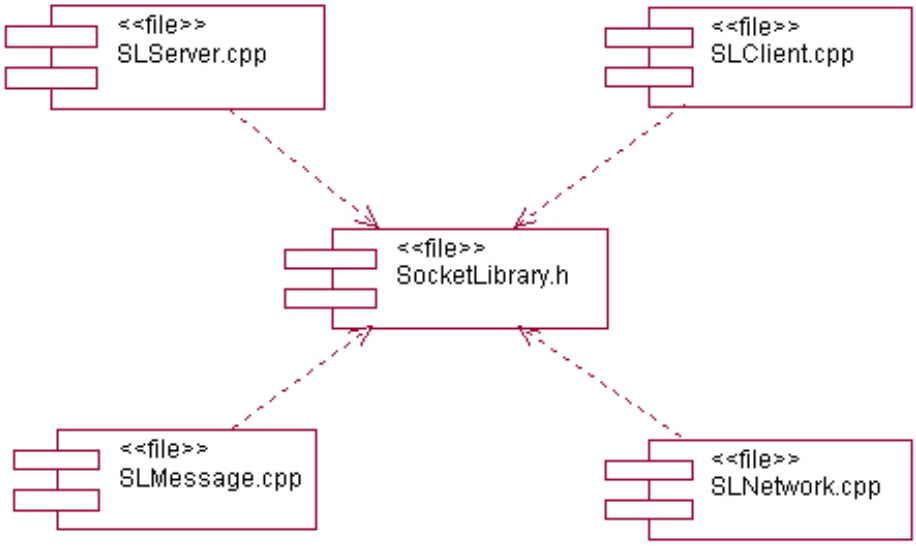

Imagen 20 Diagrama de Componentes

Conclusiones

Al concluir este capítulo se tiene concebido detalladamente el diseño completo del sistema y la secuenciación de pasos traducida a mensajes entre clases de los casos de usos a desarrollar.

Se tiene concluida la etapa de implementación, por lo que en este momento se encuentra todo preparado para pasar a la etapa de programación de los casos de uso desarrollados en el primer ciclo. Como posibilidad adicional que brinda la herramienta *Rational Rose*, ya es posible generar el código fuente de los componentes relacionados con los casos de uso a desarrollar.

Conclusiones

Se diseñó un módulo de clases mediante el cual se logra la transferencia de información de forma eficiente, cumpliendo con los objetivos propuestos en el trabajo. Para satisfacer las necesidades planteadas se hizo un estudio de las técnicas, tecnologías y tendencias actuales acerca de la transmisión de datos en tiempo real, realizando un análisis de sus principales características así como las deficiencias más comunes y la forma de erradicarlas.

Posteriormente se hizo la captura de los requisitos funcionales, no funcionales y la obtención de los casos de uso del sistema. Además quedó definida la primera fase de desarrollo del proyecto, donde se realizarán solo una parte de los casos de uso para lograr un modelo simple de conexión e intercambio de datos.

A continuación se dio paso a las etapas de diseño e implementación, utilizando los artefactos de RUP, mediante el cual se llevó a cabo la propuesta y consolidación de las clases.

Se quiere destacar lo novedoso de varias de las decisiones más importantes tomadas en este proyecto, como son: la posibilidad de utilizar el módulo de clases tanto en *Windows* como en familias de *Unix*, permitir el uso de UDP ó TCP para responder a las exigencias y necesidades según la aplicación a desarrollar y que a pesar de estar orientada a sistemas de realidad virtual puede ser usada perfectamente en cualquier tipo de aplicación.

Recomendaciones

Aspectos fundamentales que se recomiendan al trabajo:

- Migrar las funcionalidades del módulo hacia otros sistemas operativos como Solaris, Macintosh entre otros.
- Aumentar el número de funcionalidades que brinda.
- Realizar transformaciones con el objetivo de hacer mucho más sencillo el uso de módulo.
- Implementar un algoritmo de corrección de errores sobre el protocolo de comunicación UDP y permitir que su uso sea opcional.
- Mejorar las técnicas de optimización en la transferencia de mensajes.

Referencias Bibliográficas

[1].PALET, Jordi. “IPv6 es una oportunidad para la innovación”, *European* IPv6 *Task Force* & *Steering Committee IPv6 Forum*, 2003.

[2].”Redes de Computadoras”, 3ra Edición, Editorial Felix Varela, La Habana, 2004.

[2.1].Capítulo 1: Introducción.

[3].BARRON, Todd. “*Strategy Game Programming with DirectX* 9.0”, *Wordware Publishing, USA,* 2003.

[3.1]. Capítulo 14: *Network Programming Primer*.

[4].Universidad Politécnica de Catalunya. ”Sockets”, 2005, http://ariso2.ac.upc.edu/www/adjuntos/apuntes/transparencias/Tema7_JPons.pdf

[5].MEIJOMENCE, Javier. “Teleinformática y Redes”, Universidad Antonio de Lebrija, España, 1998.

[5.1]. Capítulo 2: Modelo de referencia OSI.

[6].NEO(*Networking and Emerging Optimization*). “Transmisión de datos en OSI”, 2006, http://neo.lcc.uma.es/evirtual/cdd/tutorial/modelos/trasosi.html

[7]. COMER, Douglas E. “Redes globales de información con Internet y TCP/IP”, Pueblo y Educación, Cuba, 2005, 621 páginas.

[7.1].Capítulo 29: El futuro de TCP/IP(IPng,IPv6).

[7.2].Capítulo 14: La interfaz Socket.

[7.3].Capítulo 7: Protocolo Internet.

[8].Universidad de la Republica Uruguay, “Introducción al IPv6”, 2005, http://www.rau.edu.uy/ipv6/queesipv6.htm

[9]. *SHADOWSECURITY*, “Introducción a Redes”, 2004, http://shadowsecurity.150m.com/CursoLinux/redes.html

[10].ALDUNATE, Natalia. “Seguridad de Redes”, Universidad Diego Portales, Santiago de Chile, 2004, 36 páginas.

[11]. MUNYOZ, Carles. “Redes de Ordenadores (Procesamiento paralelo)”, 2006, http://www.ctv.es/USERS/carles/PROYECTO/indice.html

[11.1].Capítulo 3: El Paradigma Cliente – Servidor.
http://www.ctv.es/USERS/carles/PROYECTO/cap3/cap3.html

[12]. CHRISTOFOLI, Justin F. “*Authority Distribution in a Proxy-Based Massively Multiplayer Game Architecture*”, *The Florida State University*, 2006.

[13]. GOUPSTATE. “*Peer-to-Peer*”, 2005, http://www.goupstate.com/apps/pbcs.dll/section?category=NEWS&template=wiki&text=Peer-to-peer

[14].MOLINA, Carlos. "*Peer to Peer*", Universidad Nacional de Lujan, Argentina, 2006.

[15].*MICROSOFT*. "Nuevas características de red en *Microsoft® Windows® XP Service Pack* 2", 2004.

[16].BUENAPOSADA, José y SERRANO, Juan. "Arquitecturas Igual a Igual (*peer to peer*)", Universidad Rey Juan Carlos, España, 2006.

[17].*GARAGAMES*. "*Torque Network Library - INDIE License*", 2007, http://www.garagegames.com/pg/product/view.php?id=27

[18]. *SOLAR-OPENSOURCE*. "*SolarSockets*", 2007, http://solarsockets.solar-opensource.com/index.php/Portada

[19].*RAKKARSOFT*. "*RakNet*", 2003, http://www.rakkarsoft.com/

Bibliografía Consultada

- ZERBST, Stefan and DÜVEL, Oliver. "3*D Engines and Game Programming*". Series Editor, André LaMothe, CEO Xtreme Games LLC. 2004. 841 páginas.
- Recopilación de autores. "*Game Programming Gems* 4". USA. *Charles River Media*. 2000. 656 páginas.
- Recopilación de autores. "*Game Programming Gems* 5". USA. *Charles River Media*. 2005. 647 páginas.
- POLLOCK, William and HUGHES Phil. "*Programming Linux Games*", Loki *Software*, Inc, USA, 2001. 395 páginas.
- MULHOLLAND, Andrew and HAKALA, Tehijo. "*Programming Multiplayer Games*", *Wordware Publishing, USA,* 2004.
- ADAMS, Jim. "*Programming Role Playing Games with DirectX*®". Series Editor, André LaMothe, CEO Xtreme Games LLC. 2004. 1021 páginas.
- MCGRAW, Hill. "*Windows* 2000 *Server* Guía de TCP/IP". Microsoft.
- MCMAHON, Richard A. "Introducción a las redes". Universidad de Huston.
- STALLINGS, Willian. "Comunicaciones y Redes de Computadoras". *Prentice Hall*.

Apéndices

Glosario de abreviaturas

API: *Application Programmer's Interface* (interfaces para programadores de aplicaciones).

ARPANET: *Advanced Research Projects Agency Network*.

BPS: *Bits Per Second* (Bits Por Segundo).

DNS: *Domain Name System* (Sistema de Nombres de Dominio).

IP: *Internet Protocol* (Protocolo de Internet).

POO: Programación Orientada a Objetos.

FTP: *File Transfer Protocol* (Protocolo de transferencia de archivos).

TCP: *Transmission Control Protocol* (Protocolo de Control de Transmisión).

SMTP: *Simple Mail Transfer Protocol* (Protocolo Simple de Transferencia de Correo)

UDP: *User Datagram Protocol* (Protocolo de Datagrama a Nivel de Usuario).

HTTP: *Hypertext Transfer Protocol* (Protocolo de Transferencia de Hipertexto).

IPX: *Internetwork Packet Exchange* (Intercambio de paquetes interred).

OSI: *Open Systems Interconnection* (Interconexión de Sistemas Abiertos).

LAN: *Local Area Network* (Red de área local).

MAC: *Media Access Control*

MBPS: *Megabits Per Seconds* (Megabits Por Segundos).

SRV: Sistemas de Realidad Virtual.

P2P: *Peer-to-Peer*.

Glosario de términos

A:

Ancho de banda: *Bandwidth* en inglés. Cantidad de bits que pueden viajar por un medio físico (cable coaxial, par trenzado, fibra óptica, etc.) de forma que mientras mayor sea el ancho de banda más rápido se obtendrá la información. Se mide en millones de bits por segundo (Mbps). Una buena analogía es una autopista. Mientras más carriles tenga la calle, mayor cantidad de tráfico podrá transitar a mayores velocidades. El ancho de banda es un concepto muy parecido. Es la cantidad de información que puede transmitirse en una conexión durante una unidad de tiempo elegida.

API: Del inglés *Application Programming Interface*. Interfaz de Programación de Aplicaciones. Un juego de rutinas usados por una aplicación para gestionar generalmente servicios de bajo nivel, realizados por el sistema operativo de la computadora. Uno de los principales propósitos de un API consiste en proporcionar un conjunto de funciones de uso general, de esta forma los programadores se benefician de las ventajas del API, ahorrándose el trabajo de programar todo de nuevo.

Aplicación: Cualquier programa que corra en un sistema operativo y que haga una función específica para un usuario.

ARPANET: Precursor del Internet desarrollado a finales de los años 60 y principios de los 70 por el Departamento de Defensa de los Estados Unidos como un experimento de una red de área, no centralizada y amplia y que resista una guerra nuclear. Posteriormente, abandonados sus propósitos defensivos, adquirió un carácter académico y comercial, evolucionando en la actual Internet.

******

B:

Bit: Dígito Binario. Unidad mínima de almacenamiento de información cuyo valor puede ser 0 ó 1 (falso o verdadero respectivamente).

Bps: Unidad de medida que indica los bits por segundo transmitidos por un equipo.

Broadcast: Es una técnica utilizada para enviar información de manera simultánea a todos los dispositivos de un segmento de red.

Buffer: En informática, un *buffer* de datos es una ubicación de memoria en una computadora o en un instrumento digital reservada para el almacenamiento temporal de información digital, mientras espera para ser procesada.

Byte: Conjunto de 8 bits. Puesto que 8 bits es la mínima cantidad requerida para representar los símbolos alfanuméricos.

******

C:

C++: Lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos (POO). C++ está considerado por muchos como uno de los lenguajes más potentes, debido a que permite trabajar tanto a alto como a bajo nivel.

Cirugía de mínimo acceso: Cirugía laparoscópica, cirugía sin ingreso o cirugía mínimamente invasiva es una técnica quirúrgica que se practica a través de pequeñas incisiones, usando la asistencia de una cámara de video que permite al equipo médico ver el campo quirúrgico dentro del paciente y accionar en el mismo, evitando los grandes cortes de bisturí requeridos por la cirugía abierta o convencional y posibilitan, por lo tanto, un periodo post-operatorio mucho más rápido y confortable.

******

D:

Datagramas: Paquete que contiene información y que se transporta por red con las direcciones de origen y de destino.

Difusión: Tipo de comunicación en que todo posible receptor es alcanzado por una sola transmisión.

DNS: Servidor que traduce las direcciones de IP en nombres más fáciles de recordar.

******

F:

Fibra óptica: Tipo de cable que se basa en la transmisión de información por técnicas optoeléctricas mediante una combinación de vidrio y materiales plásticos. A diferencia del cable coaxial y del par trenzado no se apoya en los impulsos eléctricos, sino que transmite por medio de impulsos luminosos. Se caracteriza por un elevado ancho de banda con alta velocidad de transmisión y poca pérdida de señal.

FTP: Protocolo de transferencia de archivos. FTP permite la conexión entre dos computadoras, usando por lo general el puerto 21 para conectarse (aunque se puede usar otros puertos). Por medio del Protocolo de transferencia de archivos se pueden subir y descargar archivos entre el cliente y el servidor.

******

G:

Gateways: El significado técnico se refiere a un *hardware* o *software* que traduce dos protocolos distintos o no compatibles. *Gateway* o pasarela es un dispositivo, con frecuencia un ordenador, que realiza la conversión de protocolos entre diferentes tipos de redes o aplicaciones. Por ejemplo, un *gateway* de correo electrónico, o de mensajes, convierte mensajes entre dos diferentes protocolos de mensajes.

Gopher: Herramienta de búsqueda que presenta información en un sistema de menús jerárquicos parecidos a un índice. Se trata de un método de hacer menús de material disponible a través de Internet. El *Gopher* es un programa de estilo Cliente-Servidor, que requiere que el usuario tenga un programa cliente *Gopher*.

******

H:

Hardware: Componentes físicos de una computadora o de una red (a diferencia de los programas o elementos lógicos que los hacen funcionar).

Herramienta: Colección de herramientas integradas que permiten automatizar un conjunto de tareas de algunas de las fases del ciclo de vida del sistema informático: Planificación estratégica, Análisis, Diseño, Generación de programas

Heterogéneos:

Hipertexto: Cualquier documento que contiene vínculos con otros documentos de forma que al seleccionar un vínculo se despliega automáticamente el segundo documento.

Hipervínculo: Vínculo existente en un documento hipertexto que apunta o enlaza a otro documento que puede ser o no otro documento hipertexto.

HTTP: Protocolo para transferir archivos o documentos hipertexto a través de la red. Se basa en una arquitectura Cliente-Servidor.

Host: Servidor que nos provee de la información que requerimos para realizar algún procedimiento desde una aplicación cliente a la que tenemos acceso de diversas. Al igual que cualquier computadora conectada a Internet, debe tener una dirección o número IP y un nombre.

******

I:

Implementar:

Interfaz: Zona de contacto o conexión entre dos componentes de *hardware*; entre dos aplicaciones; o entre un usuario y una aplicación. Apariencia externa de una aplicación informática.

Internet: Red virtual de recursos y servicios de telecomunicaciones nacida en 1969 en los EE.UU, a la cuál están conectadas millones de usuarios (personas, organismos y empresas) en todo el mundo desarrollado. Internet puede definirse técnicamente como la mayor red del mundo.

IP: Conjunto de reglas que regulan la transmisión de paquetes de datos a través de Internet. El IP es la dirección numérica de una computadora en Internet de forma que cada dirección electrónica se asigna a una computadora conectada a Internet y por lo tanto es única. La dirección IP esta compuesta de cuatro octetos como por ejemplo, 132.248.53.10.

IPX: Protocolo de comunicaciones *NetWare* que se utiliza para encaminar mensajes de un nodo a otro.

******

K:

Kernel: Es el corazón del sistema operativo, el cual coordina los diferentes procesos de los otros subsistemas. De una manera central, en el diseño del *kernel* están los procesos que optimizan el acceso a los servicios para la actividad del usuario.

******

L:

LAN: Red de computadoras personales ubicadas dentro de un área geográfica limitada que se compone de servidores, estaciones de trabajo, sistemas operativos de redes y un enlace encargado de distribuir las comunicaciones. Por ejemplo, computadoras conectadas en una oficina, en un edificio o en varios.

Librerías: En Inglés *library*. Cuando se habla de ordenadores, se refiere al conjunto de rutinas que realizan las operaciones usualmente requeridas por los programas. Las librerías pueden ser compartidas, lo que quiere decir que las rutinas de la librería residen en un fichero distinto de los programas que las utilizan. Los programas enlazados con bibliotecas compartidas no funcionarán a menos que se instalen las bibliotecas o librerías necesarias.

Líneas Conmutadas: *Dial Up* en Inglés. Conexión de red la cual se puede crear y desechar según se requiera que se establece usando un emulador de terminal y un módem y realiza una conexión de datos a través de una línea telefónica. Los enlaces de marcado por línea telefónica son la forma más sencilla de conexiones con acceso conmutado.

******

M:

MAC: Es un identificador hexadecimal de 48 bits que se corresponde de forma única con una tarjeta o interfaz de red.

Mbps: Unidad de medida de la capacidad de transmisión por una línea de telecomunicación donde cada megabit está formado por 1.048.576 bits.

Multidifusión: Método de difusión de información en vivo que permite que ésta pueda ser recibida por múltiples nodos de la red y, por lo tanto, por múltiples usuarios.

Módulo de clases:

Multiplataforma: Término utilizado frecuentemente en informática para indicar la capacidad o características de poder funcionar o mantener una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas.

Multiplayer: Es un modo de jugar por computadoras y videos juegos en el cual las personas pueden jugar el mismo juego al mismo tiempo usando la red.

Mundo virtual: Puede considerarse como el paradigma que actualmente rige la construcción de modelos basados en Realidad Virtual. Proponiendo un cambio fundamental en la naturaleza de la denominada interfaz usuario (que rige la orientación de la interacción hombre-máquina), desplazándola hacia un diseño centrado en lo humano, según el cual el espacio alrededor del usuario se constituye en el ambiente de computación y una entera gama de percepciones sensoriales se conjuga en torno a la nueva interfaz. Todo esto se traduce en un esfuerzo por hacer que la tecnología de computación se haga más amistosa y

accesible al usuario enfocándose hacia un planteamiento básico: si algo puede ser representado sensorialmente, es posible incorporarlo al medio computarizado.

******

N:

NetWare: Sistema operativo para redes de área local de la empresa *Novell*.

Nodo: Cualquier ordenador conectado a una red. Por definición punto donde convergen más de dos líneas. A veces se refiere a una única máquina en Internet. Normalmente se refiere a un punto de confluencia en una red.

******

O:

Octetos: Término utilizado para referirse a los ocho bits que conforman un byte. No obstante, este término se usa a veces en vez de byte en la terminología de redes porque algunos sistemas tienen bytes que no están formados por 8 bits.

Optoeléctricas:

OSI: Interconexión de sistemas abiertos. Son estándares de redes de ordenador desarrollados con el fin de crear estándares comunes de comunicación entre programas y ordenadores creados por distintos fabricantes. Se estructura en siete niveles (Presentación, Aplicación, Sesión, Transporte, Red, Nivel físico y Enlace de datos) que definen normas en cada uno de ellos desde las conexiones puramente físicas hasta las relaciones entre las aplicaciones.

******

P:

Paquete: Un paquete es un pedazo de información enviada a través de la red. La unidad de datos que se envía a través de una red la cual se compone de un conjunto de bits que viajan juntos. En Internet la información transmitida es dividida en paquetes que se reagrupan para ser recibidos en su destino. Ver también conmutación de paquetes.

Peer-to-Peer: P2P. Comunicación bilateral exclusiva entre dos ordenadores a través de Internet para el intercambio de información en general y de archivos en particular.

Ping: Es una utilidad que comprueba el estado de la conexión con uno o varios *hosts* remotos.

Procesamiento en tiempo real: Técnica de procesamiento en que la actualización de los datos afectados por un evento se realiza a medida que sucede el evento causante.

Proceso: Es un conjunto de actividades o eventos que se realizan o suceden con un determinado fin. Este término tiene significados diferentes según la rama de la ciencia o la técnica en que se utilice.

Programación Orientada a Objetos: POO es una filosofía de programación que se basa en la utilización de objetos. El objetivo de la POO es "imponer" una serie de normas de desarrollo que aseguren y faciliten la mantenibilidad y reusabilidad del código.

Protocolo: Definición del sistema de comunicación de una computadora. Acuerdo entre diferentes sistemas para trabajar conjuntamente bajo un estándar común. Conjunto de normas que permiten estandarizar un procedimiento repetitivo.

Proyección: Conversión de coordenadas 3D del mundo a las coordenadas 2D de un plano.

Puerto: Es una forma genérica de denominar a una interfaz por la cual diferentes tipos de datos pueden ser enviados y recibidos. Dicha interfaz puede ser física, o puede ser a nivel de *software*.

* * * * * *

R:

Realidad Virtual: Término futurista el cuál pretende describir la interacción de los seres humanos en mundos virtuales o simulados.

Red: *Network* en inglés. Sistema de comunicación de datos que conecta entre sí sistemas informáticos situados en lugares más o menos próximos. Puede estar compuesta por diferentes combinaciones de diversos tipos de redes.

Render: Proceso mediante el cual el ordenador crea una imagen partiendo de la descripción de las características de los objetos a visualizar.

Router: Un dispositivo que conecta dos redes; opera como un puente pero también puede seleccionar rutas a través de una red.

* * * * * *

S:

Servidor: (ver *host*).

SMTP: Se usa para la transferencia de correo electrónico entre computadoras. Es un protocolo de servidor a servidor, de forma que para poder leer los mensajes se deben utilizar otros protocolos.

Simulación: Programa computacional basado en cálculos y modelos estadísticos, usados para representar un escenario determinado.

Sistemas de Realidad Virtual: Sistema informático interactivo que ofrece una percepción sensorial al usuario de un mundo tridimensional sintético que suplanta al real.

Sincronizar: Lograr que varias actividades o procesos ocurran en el mismo instante de tiempo.

Sockets: Interfaz de comunicación que ofrece un mecanismo de comunicación general entre dos procesos cualquiera que pertenezcan a un mismo sistema, a sistemas diferentes o a sistemas en ordenadores diferentes.

******

T:

TELNET: Protocolo de Internet que permite entrar en una computadora remota, operándola como una terminal.

TCP: Protocolo de redes, orientado a conexión y confiable, que forma parte del conjunto de protocolos de TCP/IP.

TCP/IP: Familia de protocolos sobre los cuales funciona Internet, que se ha convertido en el estándar actual de comunicación entre computadoras. Conocida por estas siglas debido a que los dos protocolos más importantes son: el protocolo IP, que se ocupa de transferir los paquetes de datos hasta su destino adecuado y el protocolo TCP, que se ocupa de garantizar que la transferencia se lleve a cabo de forma correcta y confiable.

Thread: Hilo de ejecución en español. En sistemas operativos, es similar a un proceso en que ambos representan una secuencia simple de instrucciones ejecutada en paralelo con otras secuencias. Los hilos permiten dividir un programa en dos o más tareas que corren simultáneamente. En realidad, este método permite incrementar el rendimiento de un procesador de manera considerable. En todos los sistemas de hoy en día los hilos son utilizados para simplificar la estructura de un programa que lleva a cabo diferentes funciones.

******

U:

UDP: Perteneciente a la familia de protocolos TCP/IP. Este protocolo no es tan fiable como TCP, pues se limita a recoger el mensaje y enviar el paquete por la red. Para garantizar el éxito de la transferencia, UDP hace que la máquina de destino envía un mensaje de vuelta. Si no es así, el mensaje se envía de nuevo. Con este protocolo no se establece una conexión entre las dos máquinas.

Unix: Sistema operativo multiusuario y multitarea, desarrollado originalmente por *Ken Thompson* y *Dennis Ritchie* en los laboratorios *Bell* en 1969, para su uso en minicomputadoras. Ofrece múltiples ventajas y se considera potente, más portable e independiente de equipos concretos que otros sistemas operativos.

******

V:

Virtual: Término utilizado para hacer referencia a algo que no tiene existencia física o real, solo aparente.

******

W:

WWW: Un sistema de intercambio de información capaz de manipular varios tipos de medios, basado en el protocolo HTTP. La característica principal del *Worl Wide Web* es que los diferentes sitios, identificados por un URL único, pueden referirse de forma cruzada por elementos de texto "activo" conocidos como hipervínculos o enlaces. El *World Wide Web* es una de las bases de Internet.

******

Índice de Figuras y Tablas

Índice de Figuras

Índice de Tablas