UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 5

Módulo de Transmisión de datos por Red para el proyecto "Prolavi".

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

Autores:

Yaniert Pascual Aguila.

Yadiel Pérez Artiles.

Tutor: Ing. Saylin Salas Hechavarría.

Co-tutor: Ing. Yessy Cedeño González.

Junio 2009

DATOS DE CONTACTO

Tutores

Nombre: Sailyn Salas Hechevarría
Institución: Universidad de las Ciencias Informáticas (UCI)
Título: Ingeniero en Informática.
Categoría Docente: Profesor Instructor
Correo electrónico: ssalas@uci.cu

Nombre: Yessy Cedeño Gonzalez
Institución: Universidad de las Ciencias Informáticas (UCI)
Título: Ingeniero en Informática.
Categoría Docente: Profesor Instructor
Correo electrónico: ycgonzalez@uci.cu

AGRADECIMIENTOS

Yaniert:

En especial a mi padre Armando por guiarme por el camino correcto, por ser mi ejemplo a seguir. A mi madre por su preocupación y amor infinito. A mis hermanas que siempre han estado a mi lado, dándome su apoyo, a mis cuñados Ronald y Adrian por ayudarme en todo momento. A mi familia en general. A mis colegas de la UCI por su ayuda y comprensión. A mi eterno amigo David de Camagüey, por su apoyo incondicional. A la Revolución y la UCI por darme la oportunidad de estudiar y prepararme para la vida.

Yadiel:

A la UCI, por acogerme estos cinco años…han sido maravillosos.

A todos mis amigos...porque han hecho de cada día una aventura, de cada obstáculo una experiencia, porque son la única manera de mezclar sueño, ojeras, trabajo y el peso inestimable del tiempo que se acaba, y obtener milagrosamente: risa.

A mi madre...a quien admiro más que a nadie en el mundo, le debo lo que soy y lo que he logrado…gracias por siempre.

A mi hermana Lumey, mi ejemplo a seguir, siempre dio el primer paso, y me apoyó a lo largo del camino.

A mi hermana Yarelis, porque su inocencia brinda lecciones conocidas cuando niños pero olvidadas al crecer, por hacerme reír.

A mi familia.

DEDICATORIA

…A mi padre Armando por su ejemplo y cariño, siempre preocupándose por mí,

…A mi madre por darme la vida, a ella le debo todo,

….A mis hermanas Dagmara y Daineris por apoyarme en todo momento.

…A toda mi familia, mis amigos.

Yaniert

…A mi madre, a mis hermanas, a mis amigos.

Yadiel

RESUMEN

La educación apoyada en medios digitales plantea ineludibles cambios en los paradigmas educativos, mayormente reflejados en la búsqueda de nuevas tecnologías que proporcionen mejor soporte al proceso de enseñanza-aprendizaje en las distintas áreas y especialidades. Entre estas nuevas tecnologías encontramos el uso del chat, chat de voz y videoconferencias, su utilización con fines educativos constituye un campo abierto a la investigación.

En el presente trabajo se aborda el diseño e implementación de un módulo de transmisión de texto, sonido y video por red, basado en una arquitectura cliente–servidor. Para alcanzar esta meta se estudiaron las características de diferentes bibliotecas de red, que en conjunto con las necesidades del proyecto influyeron en la selección de las bibliotecas a utilizar. Además se hizo un estudio de los principios básicos, conceptos, características y componentes de la programación en redes.

El módulo resultante de este trabajo permitirá ser acoplado a las prácticas de laboratorios del Proyecto Prolavi, constituyendo el soporte para la comunicación entre sus usuarios y la transmisión de datos en general.

PALABRAS CLAVE

Chat, chat de voz, videoconferencia, módulo, arquitectura cliente-servidor, bibliotecas de red, programación en redes, prácticas de laboratorio, Prolavi.

ABSTRACT

Supported education in digital media, raises the inevitable changes in educational paradigms, mostly reflected in the search of new technologies that provide better support to the teaching-learning process in different areas and specialties. Among these new technologies we can find the use of chat, voice chat and video conferences, its use for educational purposes constitutes an open field for research.

The present work aims to design and implement a module for transmitting data, video and sound, based on client-server architecture. To achieve this goal we study the characteristics of different network libraries, which together with the needs of the project will influence on the selection of the libraries to use for the work. Furthermore, we do a study of the basic principles, concepts, characteristics and components of the network programming.

The module resulting from this work will be coupled to the laboratories practices of the Prolavi Project, this constitute the support for communication between users and data transmission in general.

KEYWORDS:

Chat, voice chat, videoconference, module, client- server architecture, network libraries, network programming, laboratories practices, Prolavi.

TABLA DE CONTENIDOS

INTRODUCCIÓN

En la última década el desarrollo de la Realidad Virtual ha sido vertiginoso, nuevas técnicas, nuevas aplicaciones para modelar entornos, nuevas formas de crear ambientes de un realismo increíble, y sobre todo, nuevos usos para estos entornos virtuales. La aplicación de esta disciplina de la informática avanza en distintas esferas, ejemplo: en videojuegos, en el ámbito militar, en el área del entrenamiento profesional (simuladores quirúrgicos, simuladores de vuelo), y en el panorama educativo.

La dupla Educación-Tecnología tiene un proceso de retroalimentación en el que ambas partes se benefician y desarrollan. Sin embargo, en Cuba la Realidad Virtual ha sido poco explotada como técnica válida en el proceso educativo. La utilización de escenarios virtuales para el estudio de distintas ramas de la educación puede ser un gran apoyo para el aprendizaje del estudiante, así como una opción viable ante la falta de recursos que impidan hacer de forma tradicional los ejercicios que se están simulando.

Estas ideas fueron la base para la creación del proyecto Prolavi, el cual es parte del polo de Realidad Virtual de la facultad 5 en la UCI, su objetivo principal es crear prácticas de laboratorio virtuales para algunas asignaturas como Física, Biología y Química, y así suplir la necesidad de medios didácticos interactivos en los institutos educativos. En estas prácticas los estudiantes interactuarán con entornos virtuales del mayor realismo posible para realizar distintos ejercicios de laboratorio.

En el proyecto Prolavi no solo se desea lograr la interacción del estudiante con el entorno virtual, también se hace necesario que este se comunique con otros estudiantes, o con el supervisor de la práctica, y que el trabajo se desarrolle de forma conjunta. Para lograr la comunicación necesaria que permita desarrollar estas prácticas conjuntas se requiere de una interfaz de comunicación por red que brinde las funcionalidades necesarias para ello. Esta interfaz de comunicación entre los usuarios de un entorno virtual, específicamente para los usuarios de las prácticas creadas por el proyecto Prolavi, no ha sido diseñada ni implementada aún.

Debido a esto surge el siguiente **problema científico:** ¿Cómo enlazar los distintos usuarios de las prácticas de laboratorio virtuales desarrolladas por el proyecto Prolavi?

Para esto se plantea como **objeto de estudio** las técnicas de trasmisión y recepción de datos en arquitecturas de red y **como campo de acción,** la transmisión y recepción de datos en arquitecturas cliente-servidor.

Este trabajo se propone como **objetivo:** Diseñar e implementar un módulo de transmisión de datos para enlazar los usuarios de las prácticas de laboratorio virtuales desarrolladas por el proyecto Prolavi.

En correspondencia con el problema científico planteado se trazaron **tareas** que contribuyen a darle cumplimiento al objetivo general. A continuación se relacionan las tareas:

- Reconocer los principios básicos, conceptos, características y componentes de la programación en redes.
- Analizar las tendencias actuales de las bibliotecas de red existentes para la transmisión de texto, video y sonido.
- Seleccionar las bibliotecas de red necesarias para la transmisión de texto, video y sonido.
- Hacer el diseño de las clases, paquetes y sub-paquetes del módulo de transmisión de texto, video y sonido por red.
- Codificar cada uno de los componentes del módulo de transmisión de datos.
- Implementar un prototipo de interfaz operacional para el módulo de transmisión de datos.

La idea fundamental a defender de este trabajo es: Mediante la utilización del módulo de transmisión de datos por red se logrará la comunicación entre los usuarios de las prácticas de laboratorio virtuales del proyecto Prolavi.

Durante el desarrollo de la investigación, se utilizó un conjunto de métodos, técnicas y procedimientos para la recopilación, el análisis, el procesamiento y la valoración de la información.

Métodos de la Investigación:

Los **métodos teóricos** posibilitaron analizar los resultados obtenidos luego de aplicar los métodos empíricos durante la realización de las tareas investigativas; permitieron establecer conclusiones fiables que posibilitaron dar solución al problema planteado.

Analítico – sintético: Este método se utilizó para realizar una profunda investigación sobre el objeto de estudio y el campo de acción. Se analizaron documentos, teorías y características sobre redes para así extraer de todo lo estudiado los elementos más importantes y que más se relacionan con nuestro trabajo.

Histórico-lógico: Este método se utilizó para profundizar en los antecedentes del desarrollo de software educativos a nivel nacional e internacional y la evolución de la programación en redes.

El presente Trabajo de Diploma consta de cuatro capítulos:

Capítulo 1. Fundamentación Teórica: En este capítulo se exponen los conceptos fundamentales que sirven de referentes para la programación en redes, se enumeran las características fundamentales de diferentes bibliotecas de red, y se describe un sistema de transmisión de video en tiempo real.

Capítulo 2. Soluciones Técnicas: En este capítulo se especifican los métodos que se utilizarán y las bibliotecas de red que más se adecuan a los requerimientos de nuestro trabajo, que deberán ser tratadas para ofrecer una solución al problema científico de la investigación y cumplimentar el objetivo general.

Capítulo 3. Características del sistema: En este capítulo se profundiza en las características del sistema, se crea el modelo del dominio, se hace el levantamiento de requisitos, se crea el modelo de casos de uso y se hace la descripción de los casos de uso de los subsistemas del módulo de transmisión de datos.

Capítulo 4. Diseño e implementación: En este capítulo se muestra cómo se diseñaron cada uno de los dos subsistemas que constituyen el módulo. Se muestran los diagramas de diseño, de secuencia, y de componentes correspondientes a cada uno de los subsistemas del módulo de transmisión de datos.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo, para una mejor comprensión del problema, se introducen una serie de conceptos básicos para el trabajo con redes, tales como protocolos de red, técnicas de transmisión, y arquitecturas de comunicación. Se hace una reseña de las bibliotecas de red más utilizadas a nivel internacional y nacional, haciendo énfasis en sus características, lo que nos permitirá seleccionar la que se ajuste más a nuestras necesidades de trabajo. Además se hace un análisis de un sistema de transmisión de video en tiempo real, incluyendo aspectos como la captura de video, codificación de video, transmisión en tiempo real, decodificación de video, y reproducción.

1.1 Redes

Una red de computadoras o red informática es un conjunto de equipos (computadoras y/o dispositivos) conectados por medio de cables, señales, ondas o cualquier otro método de transporte de datos, que comparten información (archivos), recursos (CD-ROM, impresoras, y otros) y servicios (acceso a internet, e-mail, chat, juegos) [1].

1.1.1 Protocolos de red

Los protocolos de red se definen como un conjunto de reglas que dos aplicaciones pueden seguir para establecer la comunicación entre ellas. Podemos encontrar protocolos de bajo nivel, estos guían los mensajes desde el origen hasta el destino sin mostrar la dirección en que se realiza la trasmisión, estos protocolos se encuentran reunidos en el Protocolo de Internet (Internet Protocol (IP)). El protocolo IP generalmente no se usa en las aplicaciones de red de forma directa, en ellas se utilizan protocolos que estén por encima de IP, los de uso más común son el Protocolo de Datagrama de Usuario (UDP) y el Protocolo de Control de Transferencias (TCP).

- **TCP:** Este protocolo une la capa de aplicación a la capa de red, además se asegura que los datos tengan el tamaño adecuado, que se coloquen correctamente en paquetes y que se coloquen en el orden adecuado cuando se reciben, de allí que sea un protocolo fiable u orientado a la conexión. Pero requiere mayor ancho de banda, es más lento y utiliza paquetes más grandes [1].

- **UDP:** Una alternativa a TCP para comunicarse en la capa de transporte, es el protocolo de datagrama de usuario (UDP). UDP es un protocolo sin conexión (como IP) suele ser más rápido que TCP ya que no tiene que abrir una conexión con el receptor y no tiene que realizar ninguna corrección de error. La trasmisión se realiza a una mayor velocidad, y la información se procesa con mayor facilidad pues no le incorpora a los paquetes de datos información adicional [1].

1.1.2 Técnicas de trasmisión

Las técnicas de trasmisión para mensajes se pueden dividir en distintos tipos, cada uno de ellos con características distintivas que los diferencian e identifican:

- **Unicasting:** En esta técnica se establece una conexión entre dos puntos a los cuales se les identifica como nodo receptor y nodo emisor, mediante la misma se puede realizar un control y dar dirección a los paquetes enviados. No es eficiente, en determinadas ocasiones hace un uso excesivo de ancho de banda para enviar un mensaje, por ejemplo si ese mensaje es requerido por más de un receptor el emisor lo envía de forma repetitiva malgastando recursos [2].
- **Multicasting:** En esta técnica se agrupan los receptores en grupos específicos, a los cuales les interesen mensajes comunes, para hacer el uso más eficiente que se pueda de la red. La conexión se establece entre un emisor y varios receptores agrupados como se dijo anteriormente, de manera que cuando se envía un mensaje determinado este llega a los receptores del grupo interesado y no hay necesidad de repetir o duplicar el mensaje. Se considera una buena técnica pera enviar información a grandes números de usuarios o nodos [2].
- **Broadcasting:** En esta técnica la comunicación es establecida entre un emisor y todos los nodos restantes de la red, esto trae como consecuencia que cada nodo tenga que realizar el procesamiento de cada mensaje emitido, debido a esta situación en grandes redes en las cuales se encuentren conectados un gran número de usuarios no se garantiza el broadcasting [2].

1.1.3 Arquitecturas de comunicación

Las arquitecturas de red se pueden organizar de acuerdo a su grado de despliegue y pueden ser utilizadas de acuerdo a las exigencias del cliente o de la tecnología disponible, así como la eficiencia que tendrá de acuerdo con las características específicas de la red a la cual va a ser aplicada.

- **Arquitectura punto a punto (peer to peer):** En las redes con esta arquitectura todos los nodos están conectados a los restantes nodos que integran la red, no existe intermediario entre ellos y cada uno puede emitir mensajes a los restantes, esto afecta de manera positiva a la red con respecto al problema de la latencia. Este tipo de red no tiene jerarquía entre los nodos por lo cual no resulta escalable. Muy útil cuando se aplica a un número no muy grande de computadoras conectadas y en redes locales LAN [2].
- **Arquitectura cliente servidor:** En esta arquitectura se escoge un nodo específico el cual hará la función de servidor, el resto de los nodos en la red son receptores, el nodo servidor controla y administra la comunicación en la red. La comunicación directa entre los nodos receptores no existe, al servidor controlar todos los envíos realizados hay un retardo en el flujo de paquetes, sin embargo, hay un mejor uso de recursos porque el mismo mensaje no tiene que ser enviado a todos los nodos [2].
- **Arquitectura red de servidores:** Está constituida por un grupo de nodos servidores conectados punto a punto, a cada uno de estos servidores hay un grupo de nodos clientes conectados, es decir un conjunto de subredes que están conectadas a través de los nodos servidores. Esta arquitectura brinda una gran escalabilidad y disminuye los problemas de capacidad que brindaba un solo servidor, sin embargo cuando llega a gran escala surgen problemas para el control y manipulación de datos por la red [2].

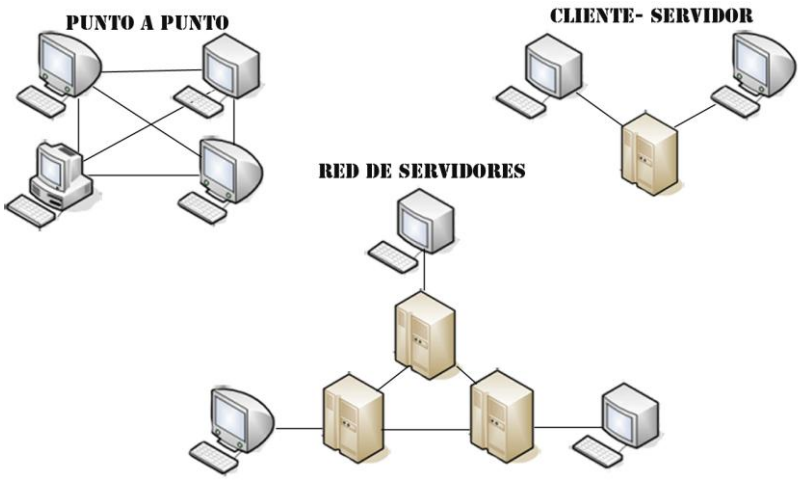

Figura 1. Arquitecturas de comunicación

1.2 Bibliotecas de red

En la actualidad el desarrollo en las redes es enorme, el auge de Internet impone un desarrollo acelerado para el control y envío de la gran cantidad de información que viaja por la red. Los juegos online y multiplayer exigen constantemente nuevas técnicas para manipular los datos y establecer comunicación con el servidor. Surgen nuevas opciones de negocio y comunicación, se generaliza el uso de la video-conferencia y la trasmisión de video y audio en tiempo real. Los simuladores avanzan hacia simulaciones colectivas y de forma conjunta necesitando no solo la conexión entre los elementos simulados sino también opciones válidas de comunicación para los participantes de la simulación. Esto ha traído como consecuencia el desarrollo de un enorme número de bibliotecas que se dedican al control y manipulación de datos para el envío de estos por la red.

La programación de la red para el módulo de transmisión de datos requiere de dos elementos imprescindibles. En primer lugar necesitamos crear las conexiones entre los clientes. Y en segundo lugar necesitamos diseñar un protocolo que defina cómo se comunican las aplicaciones clientes entre sí, es decir tenemos que crear un lenguaje que permita describir los estados de los clientes. Para lograr esto hacemos un estudio de varias bibliotecas de red.

1.2.1 Biblioteca HawkNL

HawkNL es una API de red orientada a juegos, es libre, de código abierto, liberada bajo la licencia GNU LGPL (GNU Library General Public License), se considera una API de bajo nivel ya que trabaja bastante cercano a los sockets, tiene funciones sobre Berkeley/Unix Sockets y WinSock. HawkNL también brinda otras ventajas incluyendo soporte para distintos sistemas operativos, grupos de sockets, un temporizador de alta exactitud, estadísticas de los sockets, macros para leer y enviar datos en paquetes con conversión “endian” y soporte para múltiples transportes de red. Ha sido probada en Windows 9X/ME/NT/2000/XP/CE, Linux, IRIX, AIX, BSDs, Mac OS. Esta biblioteca trabaja básicamente con los sockets brindando funciones para la conexión, la lectura y escritura, los cierres de conexión y todo lo referente con el envío de estos datos, haciendo más fácil el trabajo. Es una de las bibliotecas más usadas internacionalmente como base para las conexiones en juegos online. Posee comunicación de voz como software propietario [3].

Entre las funciones que brinda se encuentran:

- **nlAcceptConnection (nlSocket Socket):** esta función devuelve un nuevo objeto socket si una nueva conexión es aceptada, este socket recibirá solo paquetes de la dirección desde la cual se realizó la conexión. Para habilitar las conexiones entrantes se tiene que hacer una llamada a nlListen [3].

- **nlListen (nlSocket Socket):** habilita a los sockets para que atiendan nuevas conexiones.[3]
- **nlAddrtoString y nlStringtoAddr**, estas funciones de gran utilidad a la hora de establecer las conexiones ya que permiten la transformación de cadenas de caracteres a una dirección con el formato xxx.xxx.xxx.xxx : xxxx, o sea que también incluye el puerto por el cual se está realizando el envío de información, al convertirse al tipo de dato string una dirección también se hace más fácil lo referente al almacenamiento de esta información [3] .
- **nlConnect** Intenta hacer la conexión de un socket a una dirección dada, para ejecutar la mayor parte de las funciones de este biblioteca esta función tiene que haberse ejecutado con éxito [3].
- **nlWrite y nlRead** Estas funciones son las que se encargan de escribir o leer desde un buffer determinado con los datos que se quieren enviar o se están recibiendo [3].
- **nlGetLocalAddr** Obtiene la dirección local en el formato NLAddress de un socket, esta puede ser convertida a una cadena de caracteres con otra funciones de la biblioteca [3].

1.2.2 Biblioteca STKNET

STKNET es una biblioteca libre desarrollada en la Universidad de las Ciencias Informáticas por el ingeniero Yessy Cedeño. STKNET brinda funcionalidades para envío y procesamiento de datos, actualmente está en desarrollo y se propone ser la base de los módulos de red en los proyectos del polo de Realidad Virtual. Su diseño orientado a objeto permite a los programadores no tener que interactuar con las funciones de bajo nivel o trabajar directamente con los sockets de la HawkNL, que es la biblioteca que sirve de base a STKNET y soporta todas las rutinas de conexión y lectura/escritura de datos. También cuenta con un buffer en cuya estructura se determina como serán interpretados los datos, especificando claramente en las funciones de envío y lectura de datos como será la sintaxis del buffer, se pueden enviar en este una gran cantidad de datos, convirtiendo el uso de este recurso en un elemento flexible y multiuso, dando al programador la

oportunidad de escribir sus propias funciones, obteniendo módulos personalizados en cada proyecto que use la biblioteca.

En STKNET encontramos varias clases que brindan funciones de apoyo y que sirven de estructura base a la hora de conformar los paquetes o de efectuar el envío o recepción de datos. Sin embargo el núcleo funcional lo constituyen las clases STKPeer, STKServer y STKClient. STKPeer es una clase abstracta de la cual heredan STKServer y STKClient y brinda las funciones básicas de la biblioteca.

STKServer y STKClient también son clases abstractas pero ya se enfocan en la función específica que van a desarrollar como cliente o servidor, al trabajar con la biblioteca STKNET hay que heredar de estas dos clases para crear clases cliente y servidor propias, donde se implementan los eventos según las necesidades del proyecto para el que se use, personalizando de este modo el transporte de datos.

1.2.3 Biblioteca Raknet

Raknet es software propietario, está programada en C++, constituye un motor (en inglés engine) para la gestión y trabajo con redes, principalmente en juegos. Está diseñada para tener un alto desempeño, fácil de integrar y constituye una solución completa para juegos y otras aplicaciones. Posee un sistema de replicación de objetos que automáticamente crea, destruye, serializa y trasmite los objetos de juego. Tiene una robusta capa de comunicación con un control de flujo automático, ordenamiento de mensajes por múltiples canales, reagrupamiento del mensaje, fraccionamiento y el posterior reensamblaje del mensaje. Llamadas remotas a procedimientos propios de C y C++, con una lista automática de parámetros serializados. También posee comunicación de voz, por lo que incluye estructuras de unión con audio para Port Audio, FMOD y Direct Sound [4].

1.2.4 Biblioteca DataReel

DataReel es una plataforma desarrollada en C++ que constituye una herramienta de desarrollo usada para crear bases de datos multi-hilos y aplicaciones de comunicación. DataReel Produce una rápida ejecución en programas compilados y ofrece poderosas capacidades para la programación. Usando DataReel se puede potenciar el lenguaje de programación C++ utilizando interfaces de alto nivel para la programación de bases de datos, comunicaciones y programación multi-hilo. El paquete de desarrollo de DataReel fue producido por trabajo independiente y bajo contrato y fue liberado al publico bajo una licencia de no exclusividad. El trabajo de creación comenzó de manera independiente en 1997 y fue aumentado del 1999 al 2004 por código bajo

contrato para servir de soporte a varias aplicaciones. Son muchos los desarrolladores a través del mundo que han aportado código para hacer de DataReel una biblioteca robusta. En 2005 el código de DataReel se puso bajo el escrutinio para producir un código fuente estable y seguro que permitiera su uso en complejos sistemas comerciales.

Entre las ventajas de DataReel tenemos que simplifica la complejidad temporal consumida al trabajar con bases de datos, sockets y programación multi-hilo brindando una interfaz de programación semejante a la de JAVA para la programación de estos. Es flexible, modular, portable y constituye un acercamiento a la programación para redes y bases de datos [5].

1.2.5 Biblioteca Libtcp++

Libtcp++ es una biblioteca de clases de C++ que facilita la creación de clientes y servidores TCP/IP. Esta biblioteca contiene tres clases, TcpClient, TcpServer y TcpRuletSet. TcpServer ha construido un método para la detección de las capacidades de conexión de una computadora "peer" (computadora conectada al resto de los nodos, puede enviar y recibir mensajes directamente), y un control de acceso basado en IP para regular la funcionalidad del servidor. La clase de TcpClient brinda en la función connect () un parámetro timeout que resulta útil para los escaneos de puertos de host, y para otras situaciones en las que estés tratando de conectarte a host apagados o a puertos protegidos por reglas de cortafuegos (en inglés firewall). La biblioteca ha sido probada en Linux y BSD, en otros sistemas aun no ha sido probada, con el ajuste de algunas líneas debe ser posible correrla en distintos compiladores y sistemas operativos. Brinda el código libre y se pueden realizar las modificaciones necesarias que estime el usuario [6].

1.2.6 Biblioteca Zoidcom

Zoidcom es una biblioteca de red de alto nivel libre para el uso no comercial, esta biblioteca basada en el protocolo UDP provee ventajas para la replicación de objetos de juego y la sincronización de sus estados sobre la conexión de red de una manera altamente eficiente referente al ancho de banda. Esto se logra al multiplexar y demultiplexar la información de los objetos desde y hasta secuencias de bits, lo que hace posible evitar el envío de datos redundantes. Los booleanos necesitan un solo bit, los datos enteros y flotantes se separan en cuantos bits sean necesarios.

Entre las ventajas de Zoidcom tenemos:

- **Fácil de usar y flexible:** Es una API simple y directa hecha en C++, posee una extensa documentación, hay una gran cantidad de programas ejemplos disponibles, no se necesitan pasos de construcción adicionales, administración de memoria personalizada.
- **Conectividad:** Rápida, basada en el protocolo de red UDP usando secuencias de bits para el envío aunque se tiene muy poco control de los datos enviados. El sistema completo puede operar sobre un solo puerto UDP, de forma automática crea los paquetes de la talla requerida. Tiene una distribución dinámica y limitación del ancho de banda según las necesidades y envía paquetes UDP a destinos arbitrarios.
- **Eventos de Objetos:** No necesariamente tienen que pasar los mensajes de objeto a través de toda la aplicación. Control total de los objetos, se puede interceptar los eventos de objetos, monitorear y proteger contra las trampas.
- **Sincronización de estado de los objetos automática:** Sincroniza los tipos de datos entero, bool y flotante y de forma automática realiza interpolación entre las actualizaciones de los estados. Implementa en solo minutos replicadores de tus tipos de datos y solamente envía la información que cambia realmente [7].

1.3 Biblioteca de audio

Uno de los aspectos más importantes para lograr la calidad tanto en un juego como en un simulador es el sonido, de un buen efecto de sonido depende en gran parte el realismo que se le puede imprimir al entorno. También es necesario una biblioteca que sea capaz de trasmitir, guardar y controlar los tipos de datos relacionados con el sonido, tanto del entorno virtual como de medios de comunicación que se puedan establecer entre los usuarios.

Open Audio Library (OpenAL):

Cuando se habla de OpenAL no se puede dejar de mencionar su característica distintiva: esta API es capaz de proporcionar sonido tridimensional (sonido 3D). ¿Qué quiere esto decir? El sonido se oirá de formas distintas de acuerdo a disímiles parámetros y situaciones. Dependerá de la posición de la fuente con respecto al oyente, con su movimiento, con la velocidad entre ambos, y el tipo de entorno, entre muchos otros parámetros.

En OpenAL también se inicializa el buffer de sonido y la fuente de este, a la hora de cargar los sonidos hay distintas variables que son imprescindibles: tamaño, frecuencia, el formato y los datos en específico. La forma más común es utilizar la misma biblioteca como dispositivo de salida, sin embargo hay otros dispositivos de salida con los cuales la calidad y elegancia del sonido aumentan, tal es el caso de DirectSound3D, SDL, ALSA y MMSytem [9].

Esta biblioteca constituye una buena opción para el trabajo con sonidos, tanto para el manejo de datos de audio para su posterior envío o almacenamiento, así como para aplicarle sonido 3D a los entornos creados en simuladores y juegos.

1.4 Sistema de trasmisión de video en tiempo real

En la actualidad el desarrollo de Internet y de las tecnologías multimedia ha traído como consecuencia que la trasmisión de video, audio y animaciones multimedia en tiempo real se haya convertido en centro de debate y desarrollo constante de tecnologías y protocolos que trabajan con este tipo de transmisión. A esta tecnología se le conoce como tecnología de transmisión de flujo multimedia, su función principal es mandar en tiempo real datos de videos, sonido y animaciones multimedia a los usuarios desde servidores, de esta forma los usuarios no tienen que esperar a que se descarguen todos los datos, sino solo unos segundos y comienzan a visualizar en tiempo real el flujo multimedia.

Un sistema de trasmisión de video en tiempo real incluye la captura del video (en caso del uso de cámaras web (en inglés webcam), en otros casos se ubican las secuencias de video digital a trasmitir), la codificación del video en caso de ser necesario, la trasmisión por la red en tiempo real y finalmente la decodificación y reproducción del video.

1.4.1 Captura del video

Para realizar la captura de video desde un webcam es necesario instalar los drivers necesarios según el sistema operativo que se use para el correcto desempeño de la herramienta, una vez instalados se prepara una aplicación para la captura de la secuencia de video. Estos pasos proveen un video analógico o ya directamente un video digitalizado que se utilizará para la trasmisión [9].

1.4.2 Codificación del video

En el momento de trasmitir la información, esta debe ser comprimida y codificada para generar una secuencia de video orientada a la comunicación por la red y que presente las características requeridas por los estándares de trasmisión y de la red. Existen distintos APIs que brindan estas funcionalidades, por ejemplo la biblioteca **libavcodec**, contiene todos los codificadores y decodificadores de audio y video FFmpeg.

Un estándar que se debe tener en cuenta para llevar a cabo este trabajo es el MPEG-4, MPEG-4 requiere una baja tasa de trasmisión y usa eficientemente el ancho de banda, comprime los datos a través de la reconstrucción de frames y obtiene una mejor calidad de imagen usando menor cantidad de información, provee una gran tasa de compresión y al mismo tiempo hay una menor pérdida de datos, manteniendo de esta forma la calidad del video [9].

1.4.3 Trasmisión en tiempo real

En este paso se envía la secuencia de bits codificados a través de un trasmisor de tiempo real por la red. Esta trasmisión es regida por el Protocolo de Transporte en Tiempo Real (RTP), que constituye el estándar a utilizar y la tecnología clave en este tipo de trasmisión [9].

Una de las bibliotecas que se utilizan para este trabajo es la **JRTPLIB**, esta biblioteca maneja internamente el Protocolo de Control de RTP (RTCP), y brinda las funcionalidades necesarias para llevar a cabo la trasmisión de datos.

1.4.4 Decodificación del video y reproducción

Finalmente se decodifica la secuencia de video, se reconstruye la señal y se reproduce en las herramientas de salida del receptor. Para llevar a cabo este proceso es muy útil el uso de las APIs de Simple DirectMedia Layer (SDL).

SDL - Simple DirectMedia Layer:

SDL es una biblioteca multimedia con licencia GNU LGPL (GNU Library General Public License) que permite que los programas usen audio, el teclado, mouse, joystick, 3D hardware (vía OpenGL) y un buffer de fotogramas de video 2D. Esta biblioteca es utilizada en emuladores y en varios juegos. Soporta una gran cantidad de sistemas operativos entre los que se incluyen Linux, Windows, BeOS, MacOS Classic, Mac OS X,

FreeBSD, OpenBSD, Solaris, IRIX y QNX. Se puede acceder a la biblioteca a través de C, C++, Ada, Eiffel, Java, Lua, ML, PHP, Python y otros lenguajes de programación [9].

1.5 Protocolo de transporte en tiempo real (RTP)

Este es un protocolo a nivel de sesión, utilizado para la trasmisión de datos en tiempo real, de uso muy difundido en las aplicaciones de audio y video online, especialmente en las videoconferencias. El estándar RTP fue publicado por primera vez en 1996, desarrollado por el grupo de trabajo de trasmisión de Audio y Video de IETF (Grupo de Trabajo en Ingeniería de Internet, en inglés ***I**nternet **E**ngineering **T**ask **F**orce*) y apareció bajo la RFC (Petición De Comentarios, en inglés **R**equest **F**or **C**omments) 1889. Durante los años siguientes siguió un curso de desarrollo y mejora y en el año 2003 salió la RFC 3550 la cual constituye el estándar de Internet STD 64. En un principio se publicó como un protocolo multicast pero después ha sido usado en aplicaciones unicast, su uso más frecuente es en sistemas de control de flujo y transmisión, videoconferencias, sistemas para trasmitir audio en tiempo real y otras aplicaciones online. En la RFC 3711 se define el SRTP (Secure Real-Time Transport Protocol, Protocolo seguro de transporte en tiempo real) el cual brinda opciones de confidencialidad, autentificación de mensajes y protección de reenvíos para flujos de audio y video.

El protocolo RTP es de implementación simple y flexible sin imponer o indicar determinados algoritmos con un frente de transporte neutral, es escalable (unicast, multicast y broadcast), realiza por separado el manejo de datos y el control de estos y brinda opciones de seguridad como son la encriptación y la autenticación.

1.5.1 Protocolo de Control de RTP (RTCP).

Este protocolo se encuentra relacionado de manera estrecha con el RTP, se definió en la RFC 3550 y provee un control fuera de banda de la información en flujos RTP. Acompaña al protocolo de transporte en la entrada y empaquetamiento de datos multimedia aunque en sí mismo no transporta ningún dato. Se usa para emitir de manera periódica paquetes de control que participan en el flujo de sesiones multimedia. Su principal función es proveer una retroalimentación de la calidad del servicio que está brindando la RTP.

RTCP recoge información de una conexión multimedia y de los datos enviados tales como bytes enviados, paquetes enviados, paquetes perdidos, inestabilidad, retroalimentación y la demora de cada ronda de envío, esta información puede usarse para mejorar la calidad del servicio y tomar distintas decisiones sobre los límites del flujo y los códec que se utilizan.

Existen distintos tipos de paquetes RTCP, están los paquetes de reporte de envío, los paquetes de reporte de llegada, paquetes de descripción de la fuente RTCP, paquete “GoodBye” RTCP y paquetes de aplicaciones específicas RTCP.

1.6 Biblioteca JRTPLIB para la transmisión en tiempo real

Biblioteca orientada a objetos escrita en C ++, desarrollada en el Instituto para el Conocimiento Tecnológico en cooperación con la Universidad de Hasselt y la Universidad de Maastricht por Jori Liesenborgs. Esta biblioteca brinda soporte al Protocolo de Transporte en Tiempo Real (RTP), y hace sencillo el envió de paquetes RTP manejando de forma interna las funciones del RTP Control Protocol. Ha sido probada en las plataformas de GNU/Linux, MS-Windows y Solaris.

El objetivo de la biblioteca es ayudar a los usuarios a enviar y recibir datos usando protocolo RTP, sin tener que preocuparse acerca de las colisiones de los paquetes, el ordenamiento de los envíos o la trasmisión. La biblioteca provee varias clases de utilidad a la hora de crear aplicaciones de RTP, la mayor parte de los usuarios utilizan la clase RTPSession en la construcción de su aplicación, esta clase brinda las funciones necesarias para el envío de datos RTP y el manejo interno del RTCP. El código que es específico para el protocolo subyacente a través del cual los paquetes RTP son transportados, está agrupado en clases que heredan su interfaz de una clase llamada RTPTransmitter, esto facilita el soporte para diferentes protocolos subyacentes. Soporta UDP sobre IPv4 y UDP sobre IPv6.

Los permisos son concedidos a todas las personas libre de cargo para obtener una copia del software y la documentación asociada, manejar el software sin restricciones para el uso, modificación, mezcla, publicación, distribución, sublicencias.

La clase principal de esta biblioteca es la RTPSession como se había mencionado, para el uso de RTP se necesita crear un objeto de RTPSession:

RTPSession

Para crear realmente una sesión se necesita llamar a la función Create () la cual toma tres argumentos, el primero de ellos es de tipo RTPSessionParams y en él se especifican las opciones generales para la sesión, uno de los argumentos de esta clase debe ser establecido de manera explícita de otra manera la sesión no se creará de manera exitosa, este parámetro es el “timestamp unit” del dato que vas a enviar, este puede ser

calculado dividiendo un intervalo de tiempo (en segundos) entre el numero de muestras de ese intervalo, ejemplo: Para enviar 8000 Hz de datos de voz se podría utilizar el siguiente código para establecer el timestamp,

```
RTPSessionParams sessionparams;

sessionparams.SetOwnTimestampUnit (1.0/8000.0);
```

El segundo argumento de la función Create() es un puntero a una instancia de RTPTransmissionParams y describe los parámetros para el componente de trasmisión. El tercer parámetro selecciona el tipo de componente de trasmisión que vamos a utilizar, por defecto se utiliza el UDP sobre IPv4, para este componente en específico se utiliza un parámetro del tipo RTPUDPv4TransmissionParams. Por ejemplo asumiendo que queremos establecer como nuestro puerto base el 8000 se haría de la siguiente manera:

```
RTPUDPv4TransmissionParams transparams;

transparams.SetPortbase (8000);
```

Después de tener esos parámetros ya se puede llamar a la función Create (), esta retorna un valor entero que se guarda en la variable status para poder chequear si todo salió sin error.

```
int status = session.Create(sessionparams,&transparams);

if (status < 0)

{

    std:cerr << RTPGetErrorString (status) << std::endl;

    exit (-1);

}
```

Una vez creada la sesión se especifica el destino al cual se enviarán los paquetes RTP y RTCP, la función para ello es AddDestination que toma un parámetro del tipo RTPAdress, esta es una clase abstracta y para el protocolo UDP sobre IPv4 se utiliza RTPIPv4Adress.

La biblioteca posee varias funciones que facilitan el trabajo y el acceso a los datos que se están enviando, es una buena opción para el trabajo con el envío de datos en tiempo real y es muy utilizada en las aplicaciones de este tipo.

1.7 Biblioteca EMIPLIB para transmisión de audio y video

Esta biblioteca brinda muchas ventajas que simplifican el trabajo de captura y transmisión de datos como video y sonido. Puede ser utilizada en sistemas operativos Linux, MacOS, y Windows lo que hace que las aplicaciones resultantes sean fáciles de transportar para otro sistema operativo sin hacer cambios en el código. EMIPLIB se organiza en pequeños componentes que se conectan en cadena (Figura 2). Por ejemplo, un componente de captura de audio, otro para comprimir y otro para reproducir.

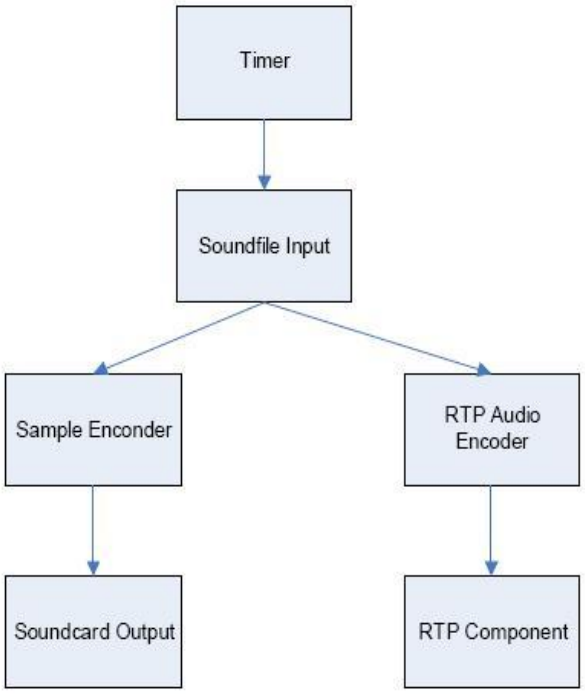

Figura 2 - Ejemplo de una cadena de componentes.

Esta biblioteca permite:

- Captura de sonido (a través de OSS o WinMM)
- Reproducir sonido (a través de OSS, ALSA, ESD o WinMM)
- Entrada de archivos WAV (a través de libsndfile, libaudiofile, o un lector de WAV interno)

- Archivos de salida WAV (a través de libsndfile)
- Captura con webcam (a través de Video4linux y DirectShow)
- Compresión speex
- La compresión H.263 + (a través de libavcodec)
- Mezcla del audio recibido
- Efectos de sonido 3D.

1.8 Conclusiones del capítulo

A lo largo de este capítulo, como base para el estudio del tema en que se desenvolverá nuestro trabajo, se introdujo al lector en una serie de conceptos básicos como protocolos de red, técnicas de transmisión de datos por la red y arquitecturas de comunicación, que pensamos son fundamentales para entender lo expuesto. Además se estudiaron características distintivas de varias bibliotecas de red, las que constituyen la base para escoger la que más se ajusta a las necesidades de nuestro trabajo. También se profundizó en el tema de transmisión de video por la red en tiempo real, teniendo en cuenta aspectos como la captura, decodificación, transmisión por la red, codificación y reproducción del video.

CAPÍTULO 2: SOLUCIONES TÉCNICAS

En el capítulo anterior estudiamos las bibliotecas de red más utilizadas para el envío de datos, sus características particulares, así como las técnicas y sistemas utilizados en la trasmisión de información desde archivos y en sesiones en tiempo real. En este capítulo se proponen las soluciones técnicas para la creación del módulo de trasmisión de datos, los métodos específicos que se utilizarán y las bibliotecas más adecuadas según los requerimientos de nuestro trabajo.

2.1 Bibliotecas de red y códec

Teniendo en cuenta las características de las bibliotecas estudiadas, se propone el uso de STKNET para la aplicación de comunicación, así como el uso de EMIPLIB que se sirve de JRTPlib para el manejo de las sesiones de sonido y video en tiempo real. Como códec de audio se propone el uso de Speex.

2.1.1 Biblioteca STKNET

Para realizar nuestro trabajo es necesario escoger una biblioteca que brinde las funcionalidades necesarias para el envío y procesamiento de datos (texto de mensajes, direcciones de ip, nombres de usuarios entre otros). En el capítulo anterior realizamos un estudio de las bibliotecas más usadas en sistemas de realidad virtual y juegos, decidiendo finalmente usar STKNET, la cual tiene dependencia de HawkNL, las funciones de la HawkNL pueden ser usadas directamente en ella. En este caso se utilizará esta biblioteca para la aplicación de comunicación de chat y para el intercambio de información referente a los clientes conectados a la aplicación. Estas funciones van a ser necesarias para el envío de sonido y video, ya que con ellas se obtendrán los destinos (direcciones de ip) para las sesiones de transmisión de audio y video.

STKNET brinda una clase **CBuffer,** la cual constituye una estructura flexible para almacenar en memoria, así como varias funciones de lectura y escritura de tipos primitivos de **HawkNL**, también implementa un constructor de copia y un operador de asignación que realiza una copia profunda e inmediata de los objetos buffer. La estructura estándar de los objetos de CBuffer es la siguiente:

- Puntero char * (“m_acData”) al segmento de memoria que se utiliza para almacenar los datos, si este puntero es NULL (0) todavía no se ha reservado memoria para almacenar los datos.

- Un short (“m_uiAlen”) donde se almacena el tamaño de “m_acData”, en bytes. El tamaño del segmento interno de memoria reservada es manejado de manera automática por la clase CBuffer mientras se ingresan más datos.
- Un short (“m_uiUlen”) donde se cuenta el número de bytes de los datos válidos almacenados en el buffer, comenzando por el inicio del arreglo char interno. Entiéndase por datos validos a todos aquellos ingresados por cualquiera de las operaciones put(), o por las copias mediante los constructores.
- Un short (“m_usPos”) que apunta a la posición de lectura/escritura en el buffer. Este puntero se utiliza en las ambos tipos de operaciones (lectura/escritura) y generalmente su valor no excede al valor de de m_uiUlen, es común encontrarse que m_usPos == m_uiUlen , esto puede significar que se está escribiendo en dicho buffer.

Las clases en la siguiente figura representan el núcleo funcional de **STKNET:**

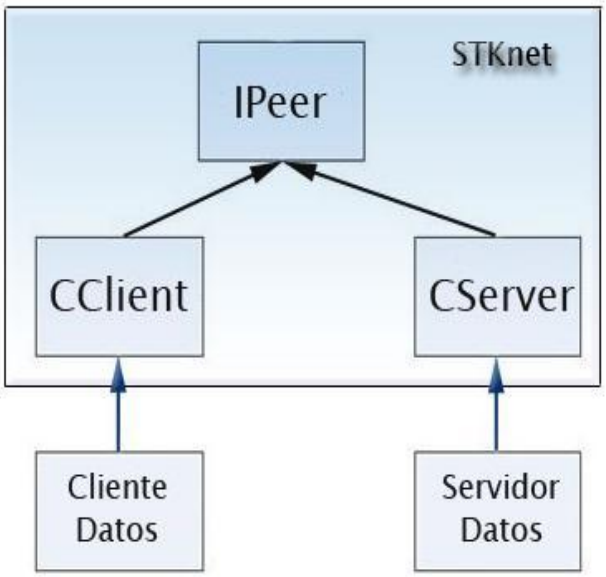

Figura 3. Estructura del núcleo funcional de STKNET.

La clase **IPeer** es una clase abstracta donde se declaran funciones bases, de utilidad tanto para la clase **CClient** como para **CServer** , las cuales se redefinen e implementan de forma particular en cada caso. **CCliente** y **CServer** son también clases abstractas de las cuales heredan las clases que el usuario vaya a utilizar para su propio programa. Haciendo uso de CBuffer se implementan unas clases Paquetes, que poseen

un encabezado particular para cada tipo, y esto permite que en la estructura interna de lectura de paquetes de CClient o CServer se extraigan los datos más fácilmente.

El tipo de paquete en que viajarán los datos del cliente es el Special Packet (especial), cuya estructura es la siguiente:

```
//                    1 Byte        1 Byte            2 Bytes              1 Byte           2 Bytes
//                  ---------------------------------------------------------------------------------------
// special  packet |  SPECIAL    |  HEAD LEN   | Emitter ID(server/client) | CODED MESAGE |        LENGTH
//                  ---------------------------------------------------------------------------------------
```

Figura 4. Estructura de la cabecera del paquete especial.

En todos los paquetes, en el primer byte estará el tipo de este, de acuerdo con esto se esperará que la estructura que sigue en el mensaje esté de acuerdo con la cabecera definida para el tipo de mensaje. El grueso del intercambio de información se desarrolla a través del tipo de paquete especial, sin embargo dominar los otros paquetes, que se denominan como internos, es muy útil a la hora de implementar eventos de desconexión, o de refutar el servicio desde el Server a un cliente dado, según condiciones establecidas con anterioridad (rangos de ip, ip baneados, repetición del nombre), a continuación mostraremos la estructura de estos paquetes:

```
*** INTERNAL PACKETS ***
                    1 Byte          1 Byte              2 Bytes               1 Byte            2 Bytes
                  ------------------------------------------------------------------------------------------
connect  packet |  CONN_REQ   |  HEAD LEN   |     Random client ID      | CODED MESAGE |   Attempt   |
                  ------------------------------------------------------------------------------------------
                  ------------------------------------------------------------------------------------------
accept   packet |  CONN_ACPT  |  HEAD LEN   |        SERVER_PEER        | CODED MESAGE |  Client ID rectified
                  ------------------------------------------------------------------------------------------
                  ------------------------------------------------------------------------------------------
reject   packet |  CONN_REJ   |  HEAD LEN   |        SERVER_PEER        | CODED MESAGE |
                  ------------------------------------------------------------------------------------------
                  ------------------------------------------------------------------------------------------
abort    packet |  ABORT      |  HEAD LEN   | Emitter ID(server/client) | CODED MESAGE |
                  ------------------------------------------------------------------------------------------
                  ------------------------------------------------------------------------------------------
info req packet |  INFO_REQ   |  HEAD LEN   | Emitter ID(server/client) | CODED MESAGE |
                  ------------------------------------------------------------------------------------------
                  ------------------------------------------------------------------------------------------
info     packet |  INFO       |  HEAD LEN   | Emitter ID(server/client) | CODED MESAGE |  INFO FIELDS......AUN PC
                  ------------------------------------------------------------------------------------------
                  ------------------------------------------------------------------------------------------
ping     packet |  PING       |  HEAD LEN   |        SERVER_PEER        | CODED MESAGE |
                  ------------------------------------------------------------------------------------------
                  ------------------------------------------------------------------------------------------
pong     packet |  PONG       |  HEAD LEN   |        client ID          | CODED MESAGE |
                  ------------------------------------------------------------------------------------------
                  ------------------------------------------------------------------------------------------
ACK      packet |  ACK        |  HEAD LEN   | Emitter ID(server/client) | CODED MESAGE |  ACK byte   |
                  ------------------------------------------------------------------------------------------
```

Figura 5. Cabeceras de los paquetes internos de la biblioteca.

Para los fines propios del módulo se ha definido una estructura en los datos que se enviarán en el paquete especial, de esta forma no solo se obtendrá información del cuerpo del paquete sino que también el orden en que los datos se envían tendrán significado implícito. La estructura del paquete se ha definido de la siguiente manera:

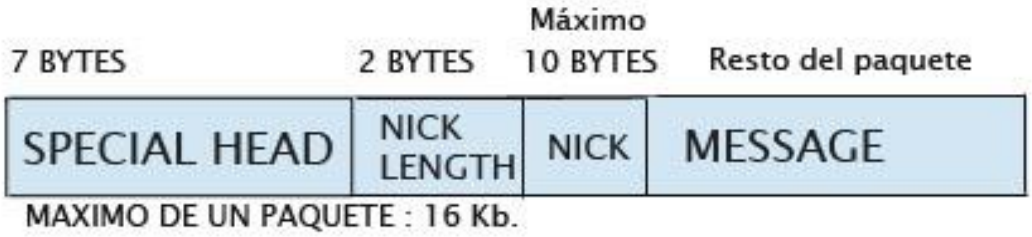

Figura 6. Estructura del paquete utilizado para enviar datos.

Primero tenemos los 7 bytes de la cabecera del tipo especial, después está en 2 bytes la longitud del “nick” (nombre del que está enviando el mensaje), a continuación el “nick” y después el texto del mensaje. El campo nick tendrá un máximo de 10 bytes, de esta manera se sabe que el número que se encuentra en el principio del paquete será como máximo 10, y se utilizó este mismo campo para enviar en él códigos que especifiquen un tipo de mensaje dado, y al ser este identificado por el cliente o por el servidor, se ejecutan las acciones necesarias para su procesamiento. Por ejemplo, cuando un cliente se quiere conectar escoge un nombre, y en la rutina de conexión se envía un mensaje con el código “17” (comprobar si nombre está en uso) en el primer campo del mensaje, el servidor identifica este mensaje y verifica los nombres de los clientes conectados, desconectando al cliente en caso de tener un nombre repetido.

Como se ha mencionado anteriormente STKNET utiliza a HawkNL, esta biblioteca es una API orientada a juegos, por lo que muchas de sus funcionalidades y ventajas pueden ser utilizadas en los simuladores por su semejanza con la arquitectura de algunos juegos. Es de código abierto, y constituye una biblioteca de bajo nivel que trabaja a nivel de sockets, haciendo la conexión sencilla y brinda ventajas para el envío de datos.

2.1.2 Biblioteca EMIPLIB

Esta biblioteca brinda una gran cantidad de ventajas a la hora de transmitir datos en tiempo real, especialmente audio y video, agrupa y usa otras bibliotecas brindando un interfaz a un nivel más alto, haciendo su uso sencillo para los usuarios. Es de código abierto, se ha probado en distintas plataformas (GNU/Linux, Mac OS, Win32 y WinCE), brinda ejemplos junto con su código, posee una documentación amplia y bien estructurada, por estas razones EMIPLIB constituye una opción ideal para usarla en nuestro trabajo como soporte para la transmisión y manejos de datos en tiempo real.

EMIPLIB es un framework flexible, que brinda una gran cantidad de pequeños componentes, donde cada uno de ellos realiza una función en específico. Estos componentes se pueden unir en una cadena donde se pueden intercambiar mensajes, de esta manera se pueden construir aplicaciones más complejas a partir de los componentes básicos.

También encontramos varias clases wrapper (cubierta), en las cuales se unen internamente los componentes necesarios para hacerlas funcionar y el usuario no tiene que crear las cadenas directamente, esto hace que crear aplicaciones de VoIP y similares sea bastante sencillo.

En núcleo de este biblioteca cuenta de tres elementos:

- Componentes.
- Cadenas de Componentes.
- Mensajes.

Los componentes se describen en clases que derivan de la clase **MIPComponent**, pueden ser colocados en una cadena que es descrita por la clase **MIPComponentChain**, cuando esta cadena se activa los mensajes son distribuidos a través de los vínculos, estos mensajes son descritos por clases que derivan de **MIPMessage**. Generalmente los mensajes en una cadena se pasan a los elementos que se encuentran a continuación en esta, pero hay varias razones por las que se necesitan en algunos casos cadenas de retroalimentación, esto se implementa con clases de mensajes derivadas de **MIPFeedback**.

Antes de comenzar a trabajar directamente con los componentes y las cadenas es necesario hacer un manejo de errores, de esta manera se sabe si hay algún error, en qué componente fue y así tratar de solucionarlo directamente.

Entre estas clases wrappers de las que se habla anteriormente encontramos:

- MIPAudioSession.
- MIPVideoSession.

Estas brindan una secuencia de componentes unidos en cadenas que ya tienen las funcionalidades necesarias para transmitir audio y video. También brinda unas clases para el trabajo en archivos wav.

Esta biblioteca depende de JRTPLIB y de JTHREAD, también si se van a usar las clases para la transmisión de voz por la red, es necesario tener instalado el códec Speex, ya que se utiliza para la codificación y decodificación de los datos de voz. Para el trabajo de la captura de video es necesario tener instalado Video4Linux para los sistemas de tipo UNIX y DirectShow para sistemas Windows.

Una de las funcionalidades que brinda EMIPLIB es la trasmisión de sonido y video en tiempo real, para esto hace uso de la biblioteca **JRTPLIB**.

- **JRTPLIB**:

Esta es una biblioteca orientada a objetos, donde se maneja de forma interna el control del flujo de paquetes, y usa el mecanismo de sockets pera la conexión en la red.

La API que brinda esta biblioteca está compuesta por las clases **RTPSession**, **RTPSourceData** y **RTPPacket**. La clase **RTPSession** constituye la clase principal, es donde se crean y se cierran las sesiones RTP, donde se seleccionan o se agregan nuevos destinos, y es la clase que permite enviar y recibir paquetes RTP.

Los paquetes enviados viajan como instancias de la clase **RTPPacket**, y la información sobre los participantes de la sesión RTP se almacenan en instancias de la clase **RTPSourceData** [10].

2.1.3 Códec Speex

Speex es un formato de compresión de audio diseñado para optimizar el trabajo con la voz, es software libre de código abierto y patente libre. Es un proyecto que intenta eliminar las barreras que existen para las aplicaciones de entrada de voz proveyendo una alternativa libre a los caros codecs propietarios que existen para este fin. Es parte del proyecto GNU y brinda ventajas que no se encuentran en casi ninguno de los codecs de este tipo. Está diseñado para comprimir voz a un bitrate (tasa de bits por segundo) de 2 hasta 44 kbps.

Entre las ventajas que brinda Speex encontramos:

- Ocultamiento de paquetes perdidos.
- Cancelación del eco acústico.
- Detección de actividad de voz.
- Transmisión discontinua.
- Supresión de ruido.

Este códec es muy usado en aplicaciones VoIP (Voice over IP, voz sobre IP), para archivar datos, como correo de voz. Hay varios proyectos que están usando este software en la actualidad entre ellos encontramos LinPhone, Ekiga, y Asterisk. EMIPLIP lo utiliza para la codificación y decodificación de sonido.

2.2 Protocolo de comunicación

Un protocolo de comunicación no es más que un conjunto de reglas que permiten la comunicación entre dos o más computadoras, o permite el intercambio de información entre dos aplicaciones. Para poder establecer el

intercambio de datos entre dos computadoras es necesario un conjunto de reglas estrictas para el procesamiento de los mensajes [11].

La clave de cualquier mensaje esta en los encabezados, cada uno de los protocolos añade encabezados que contienen información para su propio uso, de esta manera el mensaje que se envía siempre es mayor que el que se recibe de la capa superior [11].

De esta manera, la estructura del buffer a través del cual se transmiten los datos con la biblioteca STKNET representa el medio de comunicación para el servidor y el cliente. Conectarse a la aplicación cliente con el servidor no basta para iniciar el intercambio de información, ambas aplicaciones tienen que "hablar" el mismo idioma, es decir, tienen que conocer la estructura de los mensajes que se intercambian, y de acuerdo a esta estructura establecer un conjunto de reglas para su procesamiento.

Los mensajes que serán enviados tanto por el Cliente o por el Servidor del módulo se dividen en dos grupos, mensajes internos, estos son enviados y procesados por la biblioteca STKNET, y mensajes especiales, aquí es donde se definirá el protocolo propio para el intercambio de información de los componentes del módulo.

Estructura de los paquetes enviados por el cliente:

- Paquete "Mensaje":

  Este constituye el paquete principal del cliente, se estructura de la manera siguiente:

  - Cabecera del mensaje, un short (2 bytes), para este mensaje es un número del 1 al 10, y significa la longitud del nombre de quien envía el mensaje.
  - Seguido a este campo se encuentra el nombre del que envía el mensaje, su longitud depende del número almacenado en la cabecera.
  - Al final y constituyendo el resto del paquete se encuentra el texto a transmitir.

- Paquete " Nombre":

  Se usa para verificar si el nombre escogido por el cliente para la conexión no está en uso, y para llenar los campos de información que se guardan en el servidor sobre los clientes. Estructura:

  - Cabecera de dos bytes con el código "Nombre".
  - Seguido el nombre del cliente a conectarse.

- Paquete "Audio":

  Solicita las direcciones de los clientes para transmitir Audio hacia estos. Estructura:

  - Cabecera de dos bytes con el código "Audio".

- Paquete "Video":

  Solicita las direcciones de los clientes para transmitir Video hacia estos. Estructura:

- Cabecera de dos bytes con el código “Video”.

- Paquete “CON”:

  En este paquete se hace una solicitud al servidor para obtener una lista de las direcciones y puertos de todos los clientes conectados. Estructura:

  - Cabecera de dos bytes con el código “CON”.

- Paquete “Users”:

  En este paquete se hace una solicitud al servidor para obtener una lista de los nombres de todos los clientes conectados. Estructura:

  - Cabecera de dos bytes con el código “Users”.

Estructura del los paquetes enviados por el servidor:

- Paquete “Mensaje”:

  Este paquete no sufre cambios en su estructura, se mantiene como los envió al cliente y se redirecciona al resto de los clientes conectados.

- Paquete “Audio”:

  El objetivo de este paquete es brindar la información necesaria para transmitir audio al cliente que lo solicite. Estructura.

  - Cabecera de dos bytes con el código “Audio”.
  - A continuación número de clientes conectados.
  - Lista de direcciones de los clientes conectados.

- Paquete “Video”:

  El objetivo de este paquete es brindar la información necesaria para transmitir video al cliente que lo solicite. Estructura.

  - Cabecera de dos bytes con el código “Video”.
  - A continuación número de clientes conectados al servidor.
  - Lista de direcciones de los clientes conectados.

- Paquete “RTPData”:

  El objetivo de este paquete es el de informar a los clientes que uno de los clientes va a transmitir audio o video, para que estos inicien las sesiones que le permite capturar estas transmisiones. Estructura:

  - Cabecera de dos bytes con el código “RTPData”.

- Paquete “CON”:

El objetivo de este paquete es enviar una lista de las direcciones de los clientes conectados al cliente que lo solicita. Estructura:

- Cabecera de dos bytes con el código “CON”.
- A continuación el número de clientes conectados al servidor.
- En el resto del paquete un lista con las direcciones y puertos de cada cliente.

- Paquete “Users”:

  El objetivo de este paquete es enviar una lista con los nombres de los clientes conectados al cliente que lo solicita. Estructura:

  - Cabecera de dos bytes con el código “Users”.
  - A continuación el número de clientes conectados al servidor.
  - En el resto del paquete un lista con los nombres de cada cliente conectado.

2.3 Metodologías a utilizar

2.3.1 Lenguaje UML

Unified Modeling Language (UML) es un lenguaje que se ha convertido en la notación estándar para definir, organizar y visualizar los elementos que configuran la arquitectura de un sistema de software. Mediante las combinaciones de sus elementos gráficos UML nos permite conformar una serie de diagramas de manera estándar con los que podemos generar un anteproyecto fácil de entender para cualquier persona involucrada en el proceso de desarrollo.

Con la notación UML podemos compartir y comunicar el conocimiento de una arquitectura gracias a la combinación simultánea de cinco perspectivas:

- **Definir:** Fijar, determinar, decidir, explicar un concepto a través de sus atributos distintivos. Señalar sus límites y dar una idea exacta de lo que es esencial y de lo que es circunstancial.
- **Organizar:** Establecer unos recursos, disponer un orden de responsabilidades y formalizar unas reglas de relación y actuación; todo ello orientado a conseguir un propósito.
- **Visualizar**: Representar mediante imágenes y/o símbolos el contenido y la organización de los conceptos que configuran un sistema. Hacer visible su naturaleza y su complejidad.

- **Actuar:** Pensar y tomar decisiones de manera ágil y sistemática, siguiendo un método; éste a su vez, define el modo de actuar en base a la relación de un conjunto de actores, actividades, entregables y certificaciones posibles en un escenario concreto.
- **Certificar:** Comprobar de manera fehaciente que un entregable es completo, coherente y usable para el propósito que ha sido creado.

El resultado, es una mayor comprensión y claridad sobre la naturaleza de los objetos, eventos y hechos que tienen consecuencias dentro de un dominio.

2.3.2 Proceso Unificado de Desarrollo

Proceso Unificado de Desarrollo o RUP (en inglés Racional Unified Process) es una metodología cuyo fin es entregar un producto de software donde se estructuran todos los procesos y se mide la eficiencia de la organización. Utiliza el lenguaje unificado de modelado UML para conformar los esquemas y una serie de metodologías adaptables al contexto y necesidades de cualquier sistema de software.

RUP constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Con esta garantizamos una forma disciplinada de asignar tareas y responsabilidades además de verificar la calidad del software.

Características de RUP:

- **Iterativo e Incremental:** En el Proceso Unificado los proyectos se entregan en etapas iteradas. En cada iteración se analiza la opinión, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados. Esta serie de iteraciones trae como resultado un incremento del producto desarrollado que añade o mejora las funcionalidades del sistema en desarrollo.
- **Dirigido por casos de uso:** En el Proceso Unificado los casos de uso se utilizan para capturar los requisitos funcionales y para definir los contenidos de las iteraciones. Cada iteración toma un conjunto de casos de uso o escenarios y desarrolla todo el camino a través de las distintas disciplinas: diseño, implementación, prueba, etc.
- **Centrado en la arquitectura:** El Proceso Unificado asume que no existe un modelo único que cubra todos los aspectos del sistema. Por dicho motivo existen múltiples modelos y vistas que definen la arquitectura de software de un sistema.

2.4 Herramientas a utilizar

2.4.1 Qt Designer

Para la construcción de la interfaz gráfica se utilizará **Qt Designer** una herramienta para el diseño y construcción de Interfaces Gráficas de Usuario (GUIs), completamente gratuita para aplicaciones de código abierto. Este software está provisto de herramientas que facilitan la creación de formas, botones, ventanas de diálogo muy elegantes, además permite su previsualización, lo que ayuda en gran medida a crear mejores diseños. Qt Designer se puede descargar de Internet para casi todas las plataformas, ya sean sistemas Linux, MacOS, o Windows, por lo que la aplicación resultante puede ser compilada y utilizada en cualquier sistema operativo sin necesidad de cambiar el código. Qt hace uso de una extensa biblioteca de clases y herramientas, las cuales se encuentran bien documentadas en la ayuda del software.

2.4.2 Code::Blocks

Code::Blocks es una herramienta de entorno de desarrollo integrado (en inglés IDE) libre para el desarrollo de programas. Está basado en la plataforma de interfaces gráficas WxWidgets, lo que le permite correr libremente en diversos sistemas operativos.

Code::Blocks es un núcleo abstracto donde los plugins se convierten en una parte vital del sistema. Esto lo convierte en una plataforma muy dinámica y potente, no solo por la facilidad con que pueden ser incluidas nuevas funcionalidades, sino por la capacidad de poder ser usada para construir otras herramientas de desarrollo tan solo añadiendo plugins.

2.4.3 Visual Paradigm

Visual Paradigm for UML (VP-UML) es una poderosa herramienta de modelado visual, está diseñada para apoyar a arquitectos, desarrolladores, diseñadores, analistas de procesos de negocio y modeladores de datos. Este software de modelado UML constituye una gran ayuda en la rápida construcción de aplicaciones de calidad a un menor costo. Además de su Interfaz amigable y fácil de usar (VP-UML EE) nos brinda motores de generación de código para diez lenguajes de programación diferentes entre los cuales se encuentra el C++ que es el lenguaje en el que se programará. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Algunas de sus principales características:

- Ingeniería Inversa, Código a modelo, código a diagrama.
- Generación de Código: Modelo a código, diagrama a código.
- Generación de Informes para elaboración de documentación.
- Distribución automática de diagramas, Reorganización de las figuras y conectores de los diagramas UML.
- Carácter multiplataforma ya que se puede usar tanto en sistemas Linux, Unix o Windows.

2.5 Lenguaje de Programación

C++ se utilizará como lenguaje de programación, primero que todo porque es el lenguaje de programación de nuestra facultad 5, a lo largo de la carrera hemos aprendido que con C++ se pueden hacer todo tipo de programas, desde los más simples hasta los más complejos. C++ es un lenguaje de programación orientado a objeto lo cual es esencial para incrementar la productividad, calidad y reutilización del software. Ofrece gran riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además es libre, multiplataforma, y robusto.

2.6 Conclusiones del Capítulo

En este capítulo sentamos las bases técnicas sobre las cuales se va a implementar el módulo, que dará solución al objetivo propuesto. Se escogieron las bibliotecas de red necesarias, así como las metodologías y herramientas que se van a utilizar. Es importante destacar que todo el software que se utilizará es software libre, de código abierto y multiplataforma, por lo que el módulo resultante no necesitará ninguna licencia para su uso y podrá ser portado fácilmente a otros sistemas operativos.

CAPÍTULO 3: CARACTERÍSTICAS DEL SISTEMA

En este capítulo se comienza a tener una visión más clara del módulo que será implementado, las necesidades del cliente que este satisface, sus características y funcionalidades. Se determinan las reglas del negocio y se hace el levantamiento de requisitos funcionales (capacidades o funciones que el sistema debe cumplir) y los no funcionales (propiedades o cualidades que el producto final debe presentar). También se presenta el modelo del dominio para representar los conceptos principales que rigen y se manejan en el sistema.

3.1 Reglas del negocio

- Los datos de audio transmitidos serán codificados con el códec Speex.
- Las direcciones de destino para los envíos de audio y video serán instancias de RTPIPv4Address.
- Los videos trasmitidos serán codificados con H .263+.
- El tamaño máximo del buffer de datos será de 16 KB.

3.2 Modelo del dominio

A continuación se presentan los principales conceptos para el sistema que se va a construir. Este modelo nos sirve como base para diseñar el módulo. Se utilizarán técnicas para modelar con similitud a las de la programación orientada a objetos, creando de esta forma un modelo con objetos del dominio que tengan un gran parecido a los objetos reales que se van a construir e identifiquen claramente el problema a resolver.

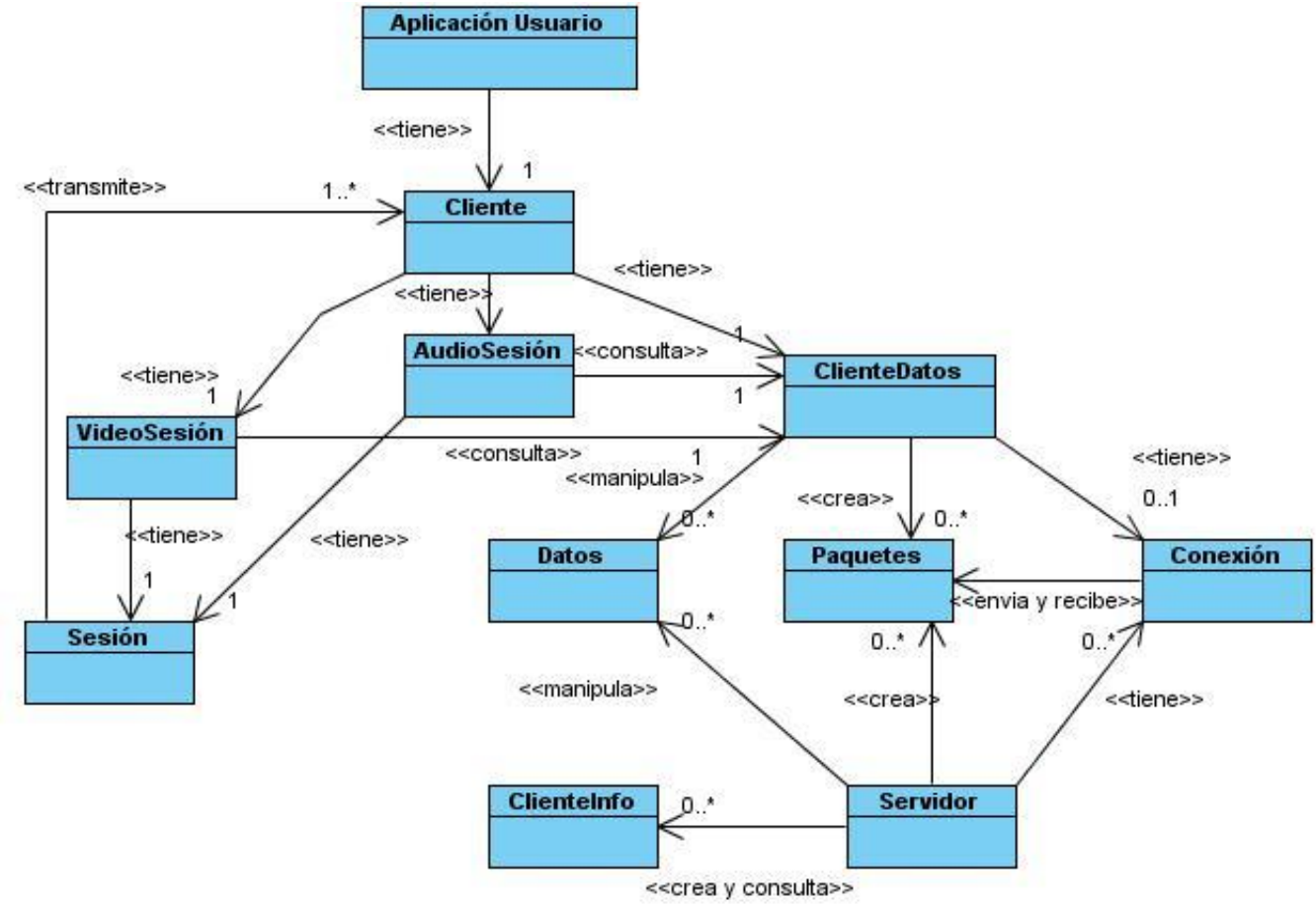

Figura 7. Modelo del Dominio.

En este modelo presentamos las relaciones entre los conceptos manejados para reflejar la funcionalidad del sistema. La **Aplicación Usuario** es aquella que se va a servir de las funciones del **Cliente**, en **Cliente** se encuentran **ClienteDatos**, **AudioSesión** y **VideoSesión** que es donde se desarrolla la lógica de trasmisión de datos según qué tipo de datos se le pida trasmitir a **Cliente**. **ClienteDatos** manipula los **Datos**, es el encargado de estructurar los **Paquetes** y a través de **Conexión** establece la comunicación con **Server**. **AudioSesión** y **VideoSesión** establecen la comunicación con los otros clientes mediante **Sesión** que es el método que utilizan las bibliotecas usadas para la trasmisión de video y sonido en tiempo real. En **Sesión** se crea una conexión directa entre los clientes para el envío de sonido y video en tiempo real.

3.3 Glosario de términos del Modelo del dominio.

- **Aplicación Usuario:** Aplicación que se va a servir de los servicios de trasmisión del módulo de red, pasándole a este los datos que quieren que sean transportados.
- **AudioSesión:** Se encarga del envío de audio, se inicializan los parámetros y componentes necesarios para establecer una **Sesión** de comunicación con el resto de los clientes que van a recibir la transmisión.
- **Cliente:** Brinda una interfaz a la **Aplicación Usuario** haciendo transparente a esta toda la lógica referente a la conexión y envío de datos.
- **ClienteInfo:** Representa la estructura en la que el servidor almacena los datos de los clientes conectados en ese momento.
- **ClienteDatos:** En él se extraen los datos recibidos de los paquetes y se manipulan los datos a enviar para conformar los paquetes, por medio de **Conexión** efectúa la lógica de conexión, trasmisión y recepción de paquetes. También a través de este se obtiene la información necesaria para la trasmisión de sonido y video.
- **Conexión:** Medio de comunicación entre **ClienteDatos** y **Servidor**, a través de este se envían y reciben los datos.
- **Datos:** Representan la información que se quiere trasmitir, los datos que conformarán los paquetes de comunicación entre el servidor y el cliente.
- **Paquetes:** Buffer dentro del cual se transportan los datos, la sintaxis de construcción y lectura del buffer determina el uso y funcionalidad de los datos que son transportados.
- **Servidor:** Asegura la conexión entre los clientes, permanece escuchando en espera de nuevas conexiones y de paquetes entrantes y los redirecciona hacia los destinatarios. Atiende los llamados especiales de los clientes cuando estos piden datos que se necesitan para la trasmisión interna de video o sonido y le brinda la información necesaria para inicializar las sesiones de transmisión.
- **Sesión:** Conexión que se estable entre los clientes a través de la cual se hace el envío de audio o video en tiempo real.
- **VideoSesión:** Se encarga de la trasmisión de video en tiempo real, se inicializan los parámetros y componentes necesarios para iniciar una **Sesión** de trasmisión de video.

3.4 Captura de requisitos.

Los requisitos son las condiciones o capacidades que deben ser alcanzadas o poseídas por un sistema para satisfacer las necesidades del cliente. Se pueden clasificar como funcionales y no funcionales. Los funcionales representan las capacidades y condiciones que deben ser brindadas por el módulo, y los no funcionales son las propiedades y cualidades del producto. En este trabajo se han definido claramente dos estructuras funcionales, que se podrían clasificar como subsistemas dentro del módulo (subsistema cliente y subsistema servidor), cada uno posee sus propios requisitos, debido a esto y para una mejor comprensión especificaremos a que cual de los dos pertenece cada requisito.

3.4.1 Requisitos funcionales

Requisitos funcionales del subsistema Cliente:

- RF1C - Poner el usuario en línea (online), abrir correctamente un puerto y estar listo para conectarse.
- RF2C - Conectarse al servidor cuya dirección IP y puerto se especifiquen.
    - RF2C.1 – Informar si se pudo o no realizar la conexión al servidor.
- RF3C - Enviar datos del tipo texto al servidor.
    - RF3C.1 – Crear el paquete con los datos a enviar.
- RF4C – Recibir los datos (texto) enviados por el servidor.
- RF5C – Enviar datos de tipo audio a los otros clientes.
- RF6C – Enviar datos de tipo video a los otros clientes.
- RF7C – Recibir audio.
- RF8C – Recibir video.
- RF9C – Obtener del servidor los datos necesarios de los otros clientes para establecer las sesiones de transmisión de audio y video al resto de los clientes.
- RF10C – Mostrar las direcciones IP de los otros clientes conectados al servidor.

- RF11C – Mostrar el nombre de los otros usuarios conectados al servidor.
- RF12C – Terminar la conexión con el servidor.
    - RF12C.1 - Avisar al servidor cuando abandone la conexión.

Requisitos funcionales del subsistema Servidor.

- RF1S – Estar en línea (online), abrir correctamente un puerto y permanecer esperando la conexión de los clientes.
- RF2S – Aceptar la conexión de un nuevo cliente.
    - RF2S.1 – Verificar la información enviada por el cliente, comprobar si puede ser aceptado o no este nuevo cliente.
    - RF2S.2 – Guardar la información de los clientes aceptados.
    - RF2S.3 – Notificar al resto de los clientes cuando se ha conectado un nuevo cliente.
- RF3S – Recibir y redireccionar los paquetes que envíen los clientes.
- RF4S – Enviar a los clientes que lo soliciten la información de los clientes para la transmisión de audio o video.
    - RF4S.1 – Avisar a los clientes cuando un cliente va a transmitir datos en tiempo real, para que estos inicialicen las sesiones para recibir audio y video.
- RF5S – Notificar a los clientes cuando el servidor deje de prestar servicios.
- RF6S – Notificar a los clientes cuando un cliente se desconecta.
- RF7S – Mostrar todas las acciones realizadas por los clientes a través del servidor (server log).

3.4.2 Requisitos no funcionales

Software:

- Para la codificación y decodificación de video deben estar disponibles: Sistema Windows: AVCodec y DirectShow, familia UNIX: Video4Linux
- Para la codificación y decodificación de sonido debe estar disponible el códec Speex para el sistema operativo que se esté utilizando.

Hardware:

- Tarjeta de Red.
- Canal físico de comunicación.

Para transmitir video:

- Webcam.
- Tarjeta de video.

Para transmitir sonido:

- Micrófono.
- Tarjeta de sonido.

Diseño e implementación:

- Lenguaje de programación **C++**.
- Se utilizará la herramienta de desarrollo integrado **Code::Blocks** para programar.
- Se utilizará la aplicación **QT Designer** para diseñar la interfaz gráfica del demo.
- Se utilizará la biblioteca **STKNET** para gestión de conexiones y transmisión de paquetes de texto por la red. STKNET hace uso de HawkNL que es una biblioteca libre de más bajo nivel.
- Se utilizará la biblioteca **Emiplib** para la transmisión de audio y video por la red en tiempo real. Emiplib a su vez hace uso de JRTPlib y JThread.
- Se utilizará la biblioteca de clases **QT** para construir la aplicación visual del servidor.

3.5 Modelo de casos de uso del sistema.

En este trabajo se han definido dos subsistemas funcionales (cliente y servidor), los cuales al interactuar brindan todas las funcionalidades del módulo. En cada uno de estos subsistemas se definen Actores del Sistema y Casos de Uso del Sistema.

3.5.1 Definición de actores del Sistema

- **SUBSISTEMA CLIENTE**

Actores	Justificación
Aplicación Usuario	Aplicación que interactúa con las funcionalidades que brinda el cliente
Servidor	Aplicación que envía mensajes especiales para iniciar funcionalidades en el cliente.

Tabla 1: Actores del sistema (cliente).

- **SUBSISTEMA SERVIDOR**

Actores	Justificación
Administrador	Usuario que se encarga de levantar y retirar los servicios del servidor.
Cliente	Aplicación que envía mensajes al servidor para obtener los servicios de trasmisión de datos de este.

Tabla 2: Actores del sistema (servidor).

3.5.2 Casos de uso del sistema

- **SUBSISTEMA CLIENTE**

CU1	Conectar Cliente
Actor	Aplicación Usuario
Descripción	Inicializa el cliente, intenta conectarse con el servidor e informa el resultado.
Referencia	RF1C, RF2C, RF2C.1

Tabla 3: CU1 Conectar Cliente.

CU2	Enviar texto
Actor	Aplicación Usuario
Descripción	Crea paquetes con el texto que se quiere transmitir y los envía al servidor.
Referencia	RF3C, RF3C.1

Tabla 4: CU2 Enviar texto.

CU3	Recibir texto
Actor	Servidor
Descripción	Identifica el paquete de texto enviado, extrae los datos y los pone a disposición de la aplicación.
Referencia	RF4C

Tabla 5: CU3 Recibir texto.

CU4	Enviar Audio
Actor	Aplicación Usuario
Descripción	Envía el audio capturado a través del micrófono a los demás clientes conectados al servidor.
Referencia	RF5C, RF9C

Tabla 6: CU4 Enviar Audio.

CU5	Enviar Video
Actor	Aplicación Usuario
Descripción	Envía el video capturado con webcam a los demás clientes conectados al servidor.
Referencia	RF6C, RF9C

Tabla 7: CU5 Enviar Video.

CU6	Recibir Audio
Actor	Servidor
Descripción	Inicializa los componentes necesarios para reproducir audio.
Referencia	RF7C

Tabla 8: CU6 Recibir Audio.

CU7	Recibir Video
Actor	Servidor
Descripción	Inicializa los componentes necesarios para reproducir video.
Referencia	RF8C

Tabla 9: CU7 Recibir Video.

CU8	Mostrar Dirección
Actor	Aplicación Usuario
Descripción	Muestra un listado con las direcciones IP de los usuarios conectados
Referencia	RF10C

Tabla 10: CU8 Mostrar Dirección.

CU9	Mostrar Nombre
Actor	Aplicación Usuario
Descripción	Muestra un listado con los nombres de los usuarios conectados
Referencia	RF11C

Tabla 11: CU9 Mostrar Nombre

CU10	Desconectar
Actor	Aplicación Usuario
Descripción	Cierra la conexión con el servidor
Referencia	RF12C, RF12C.1

Tabla 12: CU10 Desconectar.

- **SUBSISTEMA SERVIDOR**

CU1	Inicializar el Servidor
Actor	Administrador
Descripción	Abre un puerto para la conexión y comienza a mostrar todas las acciones Realizadas por los clientes que se conecten a través de él.
Referencia	RF1S, RF7S

Tabla 13: CU1 Inicializar Servidor.

CU2	Procesamiento de paquetes
Actor	Cliente
Descripción	Identifica el paquete enviado por el cliente y de acuerdo a este realiza las Acciones definidas para responder a la petición.
Referencia	RF2S, RF2S.1, RF2S.2, RF2S.3, RF3S, RF4S, RF4S.1, RF6S

Tabla 14: CU2 Procesamiento de paquetes.

CU3	Desconectar
Actor	Administrador
Descripción	Termina los servicios del servidor, informa a los clientes que se ha cerrado.
Referencia	RF5S

Tabla 15: CU3 Desconectar.

3.6 Diagrama de casos de uso del sistema

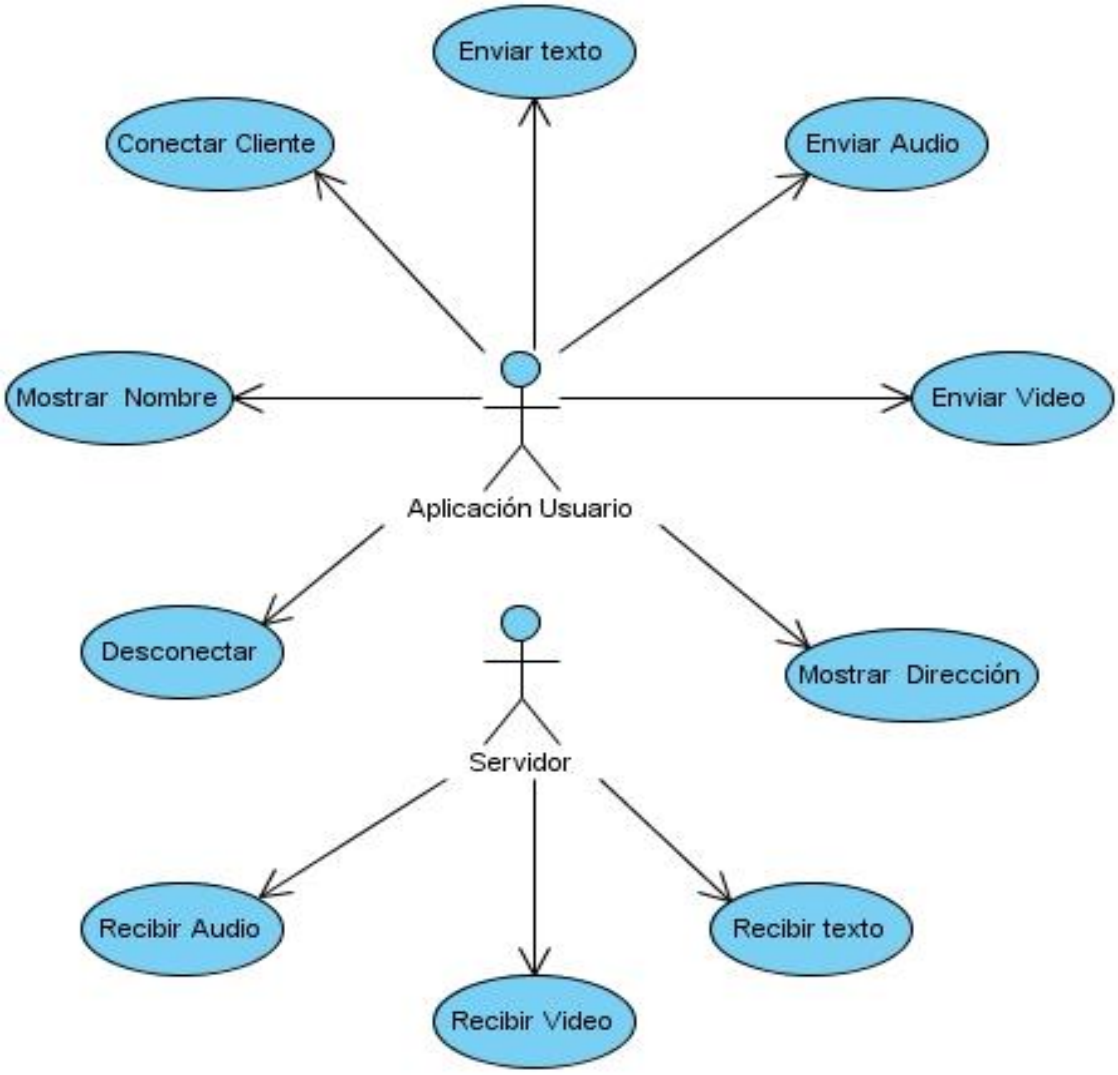

Figura 8. Diagrama de casos de uso del sistema (subsistema cliente).

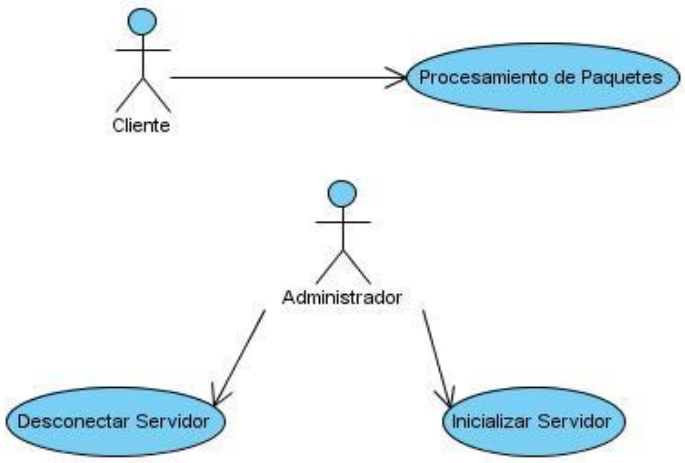

Figura 9. Diagrama de casos de uso del sistema (subsistema servidor).

3.7 Expansión de casos de uso

- **SUBSISTEMA CLIENTE**

Caso de Uso	
CU1	**Conectar Cliente**
Propósito	Conectar la Aplicación al servidor
Actores	Aplicación Usuario.
Resumen	El caso de uso se inicia cuando la Aplicación Usuario desea conectarse al servidor usando la función de conexión que brinda el Cliente del módulo.
Referencias	RF1C, RF2C, RF2C.1
Precondiciones	-

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. 1.La Aplicación Usuario indica la dirección del Servidor y el nombre de usuario, y pide conectarse.	1.1- Abre el puerto de conexión, se pone en línea y guarda el nombre de usuario.

	1.2- Se conecta al servidor
	1.3- Informa que la conexión se estableció.
Flujo Alterno 1	
Acción del Actor	**Respuesta del Sistema**
	1.1- No se puede abrir el puerto para la conexión.
	1.2- Informa un error al abrir el cliente.
Flujo Alterno 2	
Acción del Actor	**Respuesta del Sistema**
	1.2 No se puede establecer conexión con el servidor.
	1.3 Informa que no se pudo conectar al servidor indicado.
Poscondiciones	Cambia el estado a conectado y se establece la conexión con el servidor.
Prioridad	Crítico.

Tabla 16: Descripción del CU1 Conectar Cliente.

Caso de Uso	
CU2	**Enviar texto**
Propósito	Transmitir los datos de tipo texto al resto de los clientes.
Actores	Aplicación Usuario.
Resumen	El caso de uso se inicia cuando la Aplicación Usuario llama la función de enviar texto del Cliente.
Referencias	RF3C, RF3C.1
Precondiciones	Estar conectado al Servidor.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**

1. 1. La Aplicación Usuario le pasa la función de enviar texto del Cliente el texto que desea transmitir.	1.1- Crea el paquete a enviar, incorporando el nombre del usuario y el texto a transmitir.
	1.2- Envía el paquete al servidor.
Poscondiciones	Se envían los datos al servidor.
Prioridad	Crítico.

Tabla 17: Descripción del CU2 Enviar texto.

Caso de Uso	
CU3	**Recibir texto.**
Propósito	Recibir los datos tipo texto que se envían desde el servidor.
Actores	Servidor.
Resumen	El caso de uso se inicia cuando el Servidor envía un paquete especial del tipo “Mensaje”.
Referencias	RF4C
Precondiciones	Estar conectado al servidor.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. 1. El Servidor envía un paquete especial del tipo “Mensaje” al Cliente.	1.1- Identifica que el tipo de paquete recibido es “Mensaje”.
	1.2- Extrae del paquete el nombre de quien lo envió y el texto del mensaje.
	1.3- Retorna la información obtenida.
Poscondiciones	Se obtienen los datos enviados por el servidor.
Prioridad	Crítico.

Tabla 18: Descripción del CU3 Recibir texto.

Caso de Uso		
CU4	**Enviar Audio**	
Propósito	Transmitir el sonido capturado a través del micrófono al resto de los clientes.	
Actores	Aplicación Usuario.	
Resumen	El caso de uso se inicia cuando la Aplicación Usuario llama la función de transmitir Audio del Cliente.	
Referencias	RF5C, RF9C	
Precondiciones	Estar conectado al Servidor.	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Sistema**
1. 1. La Aplicación Usuario llama a la función de enviar Audio.		1.1- Crea un paquete especial tipo “Audio” y lo envía al servidor, informando que se dispone a transmitir audio.
		1.2- Envía el paquete al Servidor.
		1.3 Recibe el paquete especial tipo “Audio” enviado por el Servidor y extrae de este las direcciones IP de los otros clientes conectados.
		1.4 Añade la dirección de los clientes como nuevos destinos para la transmisión de Audio.
		1.5 Inicializa los dispositivos de entrada para capturar sonido a través del micrófono e inicia la sesión de transmisión.
Poscondiciones	Se comienza a transmitir audio en tiempo real a los clientes.	
Prioridad	Crítico.	

Tabla 19: Descripción del CU4 Enviar Audio.

Caso de Uso		
CU5	**Enviar Video**	
Propósito	Transmitir el video capturado a través de una webcam al resto de los clientes.	
Actores	Aplicación Usuario.	
Resumen	El caso de uso se inicia cuando la Aplicación Usuario llama la función de transmitir Video del Cliente.	
Referencias	RF6C, RF9C	
Precondiciones	Estar conectado al Servidor.	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Sistema**
1. 1. La Aplicación Usuario llama a la función de enviar Video.		1.1- Crea un paquete especial tipo “Video” y lo envía al servidor, informando que se dispone a transmitir video.
		1.2- Envía el paquete al Servidor.
		1.3 Recibe el paquete especial tipo “Video” enviado por el Servidor y extrae de este las direcciones IP de los otros clientes conectados.
		1.4 Añade la dirección de los clientes como nuevos destinos para la transmisión de Video.
		1.5 Inicializa los dispositivos de entrada para recibir video a través de una webcam e inicializa la sesión de transmisión.
Poscondiciones	Se comienza a transmitir video en tiempo real a los clientes.	
Prioridad	Crítico.	

Tabla 20: Descripción del CU5 Enviar Video.

Caso de Uso		
CU6	**Recibir Audio.**	
Propósito	Recibir el audio enviado por otros clientes.	
Actores	Servidor.	
Resumen	El caso de uso se inicia cuando el Servidor envía un paquete especial del tipo “RTPData”.	
Referencias	RF7C	
Precondiciones	Estar conectado al Servidor.	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Sistema**
1. 1. El Servidor envía un paquete especial del tipo “RTPData”.		1.1- Identifica el paquete “RTPData”.
		1.2- Inicializa los componentes para recibir la transmisión de audio.
Poscondiciones	Se reciben el audio que es transmitido en tiempo real por otros clientes.	
Prioridad	Crítico.	

Tabla 21: Descripción del CU6 Recibir Audio.

Caso de Uso	
CU7	**Recibir Video.**
Propósito	Recibir el audio y el video enviado por otros clientes.
Actores	Servidor.
Resumen	El caso de uso se inicia cuando el Servidor envía un paquete especial del tipo “RTPData”.
Referencias	RF8C
Precondiciones	Estar conectado al Servidor.
Flujo Normal de Eventos	

Acción del Actor	Respuesta del Sistema
1. 1. El Servidor envía un paquete especial del tipo “RTPData”.	1.1- Identifica el paquete “RTPData”.
	1.2- Inicializa los componentes para recibir la transmisión de video.
Poscondiciones	Se reciben el video que es transmitido en tiempo real por otros clientes.
Prioridad	Crítico.

Tabla 22: Descripción del CU7 Recibir Video.

Caso de Uso	
CU8	**Mostrar Dirección.**
Propósito	Brindar información sobre el resto de los usuarios conectados.
Actores	Aplicación Usuario.
Resumen	El caso de uso se inicia cuando la Aplicación Usuario introduce como texto a enviar el comando “/con” para obtener un listado con las direcciones de los clientes.
Referencias	RF10C
Precondiciones	Estar conectado al Servidor.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. La Aplicación Usuario entra como texto a enviar el comando “/con”.	1.1- Identifica el comando “/con” y crean un paquete especial del tipo “CON”.
	1.2- Envía el paquete al servidor.
	1.3- Identifica el mensaje especial “CON” enviado por el servidor y extrae la lista con las direcciones IP y el puerto de los clientes

	conectados.
	1.4- Retorna la información recibida.
Poscondiciones	Se obtiene el listado de direcciones de los clientes conectados.
Prioridad	Secundario.

Tabla 23: Descripción del CU8 Mostrar Dirección.

Caso de Uso	
CU9	**Mostrar Nombre.**
Propósito	Brindar información sobre el resto de los usuarios conectados.
Actores	Aplicación Usuario.
Resumen	El caso de uso se inicia cuando la Aplicación Usuario introduce como texto a enviar el comando "/users" para obtener un listado con los nombres de los clientes.
Referencias	RF11C
Precondiciones	Estar conectado al Servidor.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. La Aplicación Usuario entra como texto a enviar el comando "/users".	1.1- Identifica el comando "/users" y crean un paquete especial del tipo "Users".
	1.2- Envía el paquete al servidor.
	1.5- Identifica el mensaje especial "Users" enviado por el servidor y extrae la lista con los nombres de los clientes conectados.
	1.6- Retorna la información recibida.
Poscondiciones	Se obtiene el listado de nombres de los clientes conectados.
Prioridad	Secundario.

Tabla 24: Descripción del CU9 Mostrar Nombre.

Caso de Uso		
CU10	**Desconectar.**	
Propósito	Cerrar la conexión con el servidor.	
Actores	Aplicación Usuario.	
Resumen	El caso de uso se inicia cuando la Aplicación Usuario quiere terminar la conexión y llama a la función desconectar.	
Referencias	RF12C, RF12C.1	
Precondiciones	Estar conectado al Servidor.	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Sistema**
1. La Aplicación Usuario llama la función desconectar Cliente.		1.1 Manda un paquete al servidor para avisar que Va a cerrar la conexión.
		1.2 Envía el paquete al servidor y cierra el puerto de la conexión, el estado del cliente cambia a offline (fuera de línea).
Poscondiciones	Se cierra la conexión con el Servidor.	
Prioridad	Crítico.	

Tabla 25: Descripción del CU10 Desconectar.

- **SUBSISTEMA SERVIDOR**

Caso de Uso	
CU1	**Inicializar Servidor**
Propósito	Abrir correctamente el puerto para brindar servicio a los clientes, ponerse en línea (online).
Actores	Administrador.
Resumen	El caso uso se inicia cuando el Administrador selecciona la opción

	de poner en línea el Servidor.	
Referencias	RF1S, RF7S	
Precondiciones	-	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Sistema**
1. 1. El Administrador selecciona la opción de poner en línea el Servidor.		1.1- Abre el puerto usado para la conexión Correctamente.
		1.2- Cambia su estado a online (en línea).
		1.1- Comienza a mostrar las acciones de los clientes conectados.
Flujo Alterno 1		
		1.1 No se puede abrir el puerto para la conexión.
		1.2 Informa error al iniciar el servidor.
Poscondiciones	Se inician los servicios del Servidor.	
Prioridad	Crítico.	

Tabla 26: Descripción del CU1 Inicializar Servidor.

Caso de Uso		
CU2	**Procesamiento de paquetes.**	
Propósito	Procesar todos los paquetes que se reciben de los clientes.	
Actores	Cliente.	
Resumen	El caso uso se inicia cuando el Cliente envía un paquete al servidor.	
Referencias	RF2S,RF2S.1,RF2S.2,RF2S.3,RF3S, RF4S, RF4S.1,RF6S	
Precondiciones	El Servidor se encuentra en línea (online).	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Sistema**

1- El Cliente envía un paquete al servidor.	1.1 El Servidor identifica el tipo de paquete.
	1.2- Si el tipo de paquete identificado es : a) - "Request Packet" ir a la Sección Aceptar Cliente. b) – Si el tipo de paquete es "Mensaje" ir a la Sección Enviar Texto. c) – Si el tipo de paquete es "Audio" ir a la Sección Audio. d) – Si el tipo de paquete es "Video" ir a la Sección Video. e) – Si el tipo de paquete es "CON" ir a la Sección Conectados. f) – Si el tipo de paquete es "Users" ir a la Sección Usuarios. g) – Si el tipo de paquetes es "Disconnected" ir a la Sección Desconectado.
Sección "Aceptar Cliente"	
	1.1 - Extrae la información del paquete.
	1.2 – Verifica la información del cliente.
	1.3 – Acepta al cliente.
	1.4 – Almacena la información de este nuevo cliente.
	1.5 – Informa al resto de los clientes que se ha conectado uno nuevo.
Flujo Alternativo 1	
	1.3 – No se acepta al cliente.
	1.4 – Se le informa al cliente que no fue aceptado y el motivo.
Sección "Mensaje"	

	1.1 – Se busca la dirección de cada cliente conectado.
	1.2 – Se reenvía el mensaje a los demás clientes.
Sección “Audio”	
	1.1 – Se busca la dirección del resto de los clientes.
	1.2 – Se crea un paquete especial del tipo “Audio” que contiene las direcciones de los clientes y se le envía al cliente que envió el paquete “Audio” al servidor.
	1.3 – Se crea un paquete especial “RTPData” y se envía al resto de los clientes para avisarles que se disponen a transmitir Audio o Video.
Sección “Video”	
	1.1 - Se busca la dirección del resto de los clientes.
	1.2 – Se crea un paquete especial del tipo “Video” que contiene las direcciones de los clientes y se le envía al cliente que envió el paquete “Video” al servidor.
	1.3 – Se crea un paquete especial “RTPData” y se envía al resto de los clientes para avisarles que se disponen a transmitir Audio o Video.
Sección “Conectados”	
	1.1 – Busca la dirección y el puerto de los clientes conectados.
	1.2 – Crea un paquete especial tipo “CON” con las direcciones y puertos de los clientes

	conectados.
	1.3 – Envía el paquete al usuario que envió el paquete “CON”
Sección “Usuarios”	
	1.1 – Busca el nombre de los clientes conectados.
	1.2 – Crea un paquete especial tipo “Users” con los nombres de los clientes conectados.
	1.3 – Envía el paquete al usuario que envió el paquete “Users”
Sección “Desconectado”	
	1.1 – Extrae el nombre del cliente que se desconecta.
	1.2 – Informa al resto de los clientes el nombre del cliente que se desconectó.
	1.3 – Elimina la información del cliente desconectado.
Poscondiciones	Se procesan los paquetes del Cliente según su tipo.
Prioridad	Crítico.

Tabla 27: Descripción del CU2 Procesamiento de paquetes.

Caso de Uso	
CU3	**Desconectar**
Propósito	Terminar los servicios del servidor.
Actores	Administrador.
Resumen	El caso uso se inicia cuando el Administrador selecciona la opción Cerrar del Servidor.

Referencias	RF5S	
Precondiciones	Estar online (en línea).	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Sistema**
1. El Administrador selecciona la opción cerrar del Servidor.		1.1 Se informa a los clientes que el servidor va a finalizar los servicios.
		1.2- Cambia su estado a offline (fuera de línea) y se cierra el servidor.
Poscondiciones	Se finalizan los servidos del Servidor.	
Prioridad	Crítico.	

Tabla 28: Descripción del CU3 Desconectar.

3.8 Conclusiones del Capítulo

En este capítulo se definieron las funcionalidades del sistema a partir de la recogida de los requisitos funcionales, a los que el módulo debe responder para dar total cumplimiento a los objetivos propuestos. Se dividió la estructura del módulo en dos subsistemas funcionales, con el objetivo de encapsular las responsabilidades del cliente y el servidor, y de esta forma independizar su desarrollo.

CAPÍTULO 4: DISEÑO E IMPLEMENTACIÓN DEL SISTEMA

En el capítulo anterior se hizo una captura de los requisitos funcionales y no funcionales para el módulo a desarrollar, también se ha tenido un acercamiento a las características principales del sistema y el agrupamiento de sus funcionalidades en dos subsistemas. En este capítulo se traducirán esos requisitos del módulo en especificaciones que hagan una descripción de cómo se pueden implementar cada una de estas funcionalidades, estas especificaciones constituyen el diseño del sistema. Luego de tener este diseño se hará el paso de las clases del diseño a componentes físicos, al utilizar el lenguaje C++ estos componentes físicos serán ficheros .h y .cpp. También se elabora el diagrama de despliegue para el módulo, presentando la ubicación física de los componentes del sistema.

4.1 Vista general de la Arquitectura del Módulo

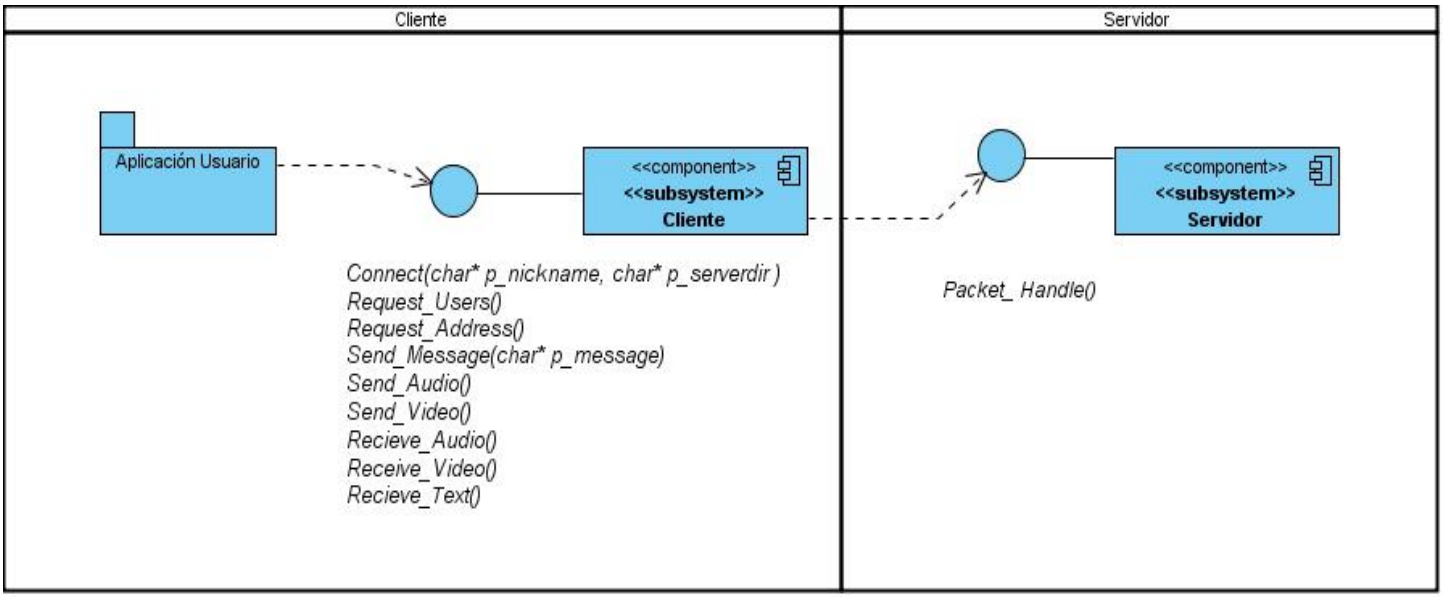

Figura 10. Vista general de la arquitectura.

En esta figura se puede apreciar claramente el uso de la arquitectura cliente-servidor, mostrando los subsistemas Cliente y Servidor. El subsistema Cliente brinda una interfaz operacional a la Aplicación Usuario haciendo totalmente transparente para ella la lógica de la conexión y la transmisión de datos. A la hora del intercambio de información se crean los paquetes en el subsistema Cliente, de acuerdo al protocolo definido

para la comunicación con el subsistema Servidor. El Cliente manipula los datos a transmitir y procesa los datos que arriban desde otros clientes a través del Servidor. Por su parte el subsistema Servidor está siempre a la escucha de los paquetes que envían los clientes, y los procesa respondiendo así a cada una de las peticiones. Las acciones del Servidor dependen del tipo de paquete que arriben a éste, el tipo se encuentra en la cabecera del paquete y las reglas para la composición posterior del mensaje se definen en el "Protocolo de Comunicación" especificado en el Capítulo 2.

4.2 Diagramas de paquetes del diseño

- SUBSISTEMA CLIENTE.

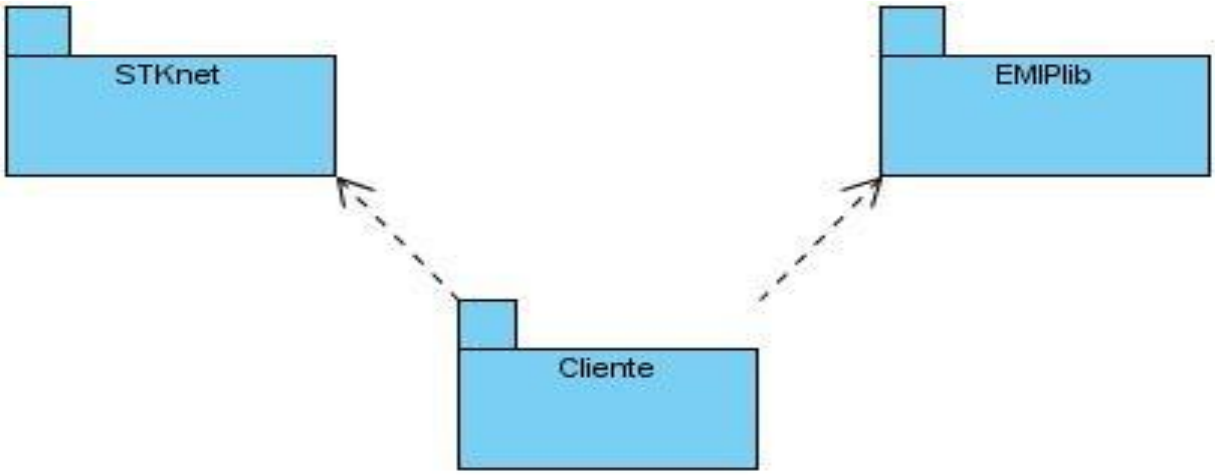

Figura 11. Diagrama de paquetes del subsistema Cliente.

- SUBSISTEMA SERVIDOR.

Figura 12. Diagrama de paquetes del subsistema Servidor.

Tanto en el Cliente como en el Servidor la interacción de las clases definidas con las de las clases de las bibliotecas es muy estrecha, por lo que se consideró necesario mostrar en los diagramas las clases con las que los subsistemas interaccionan directamente.

4.3 Diagrama de Clases del Diseño del subsistema Servidor

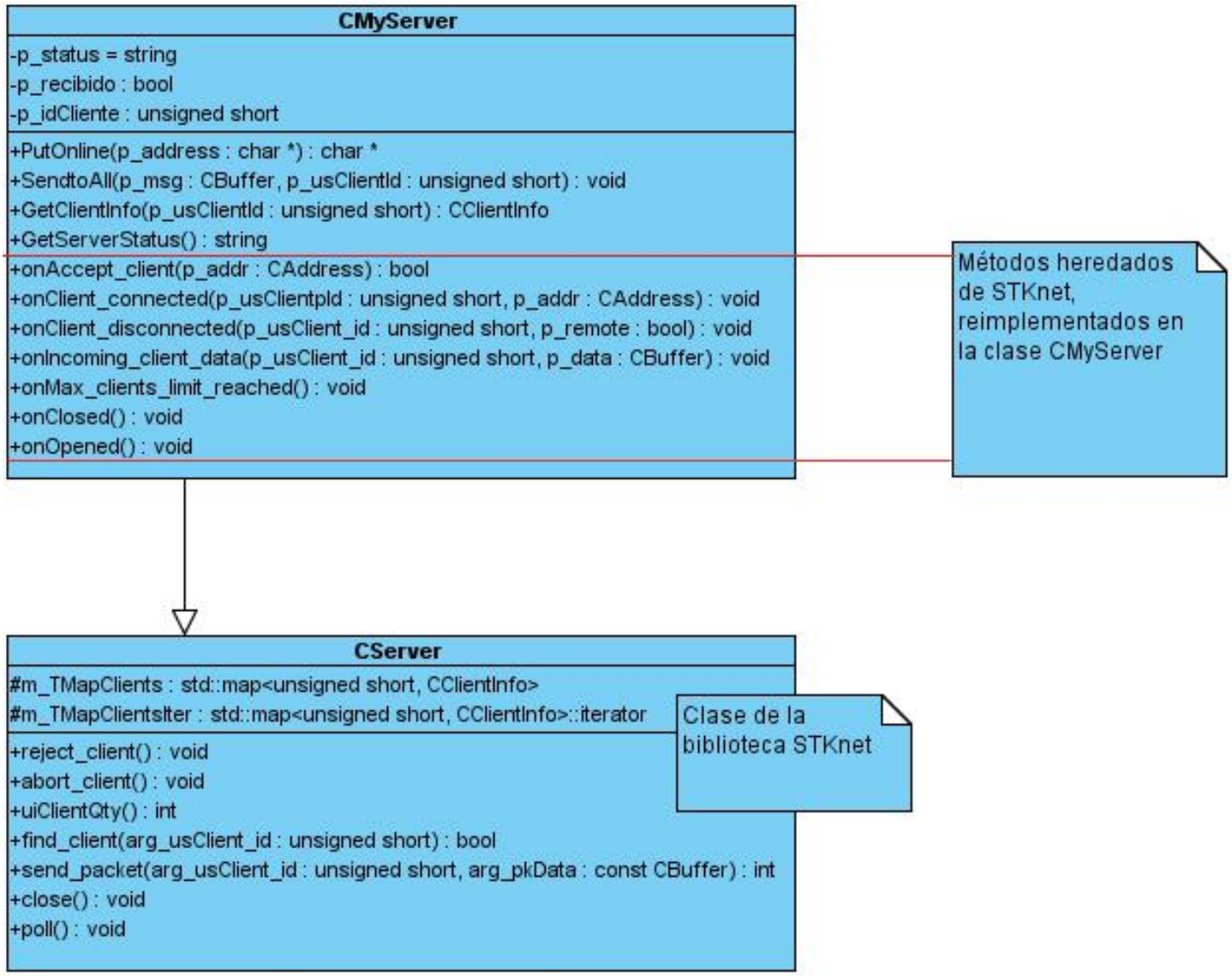

Figura 13. Diagrama de Clases del Diseño subsistema Servidor.

La clase **CMyServer** es donde se ejecuta todo el control y procesamiento de los paquetes en el servidor del módulo. Esta clase hereda directamente de la clase **CServer** de la biblioteca STKNET.

4.4 Diagrama de Clases del Diseño del subsistema Cliente

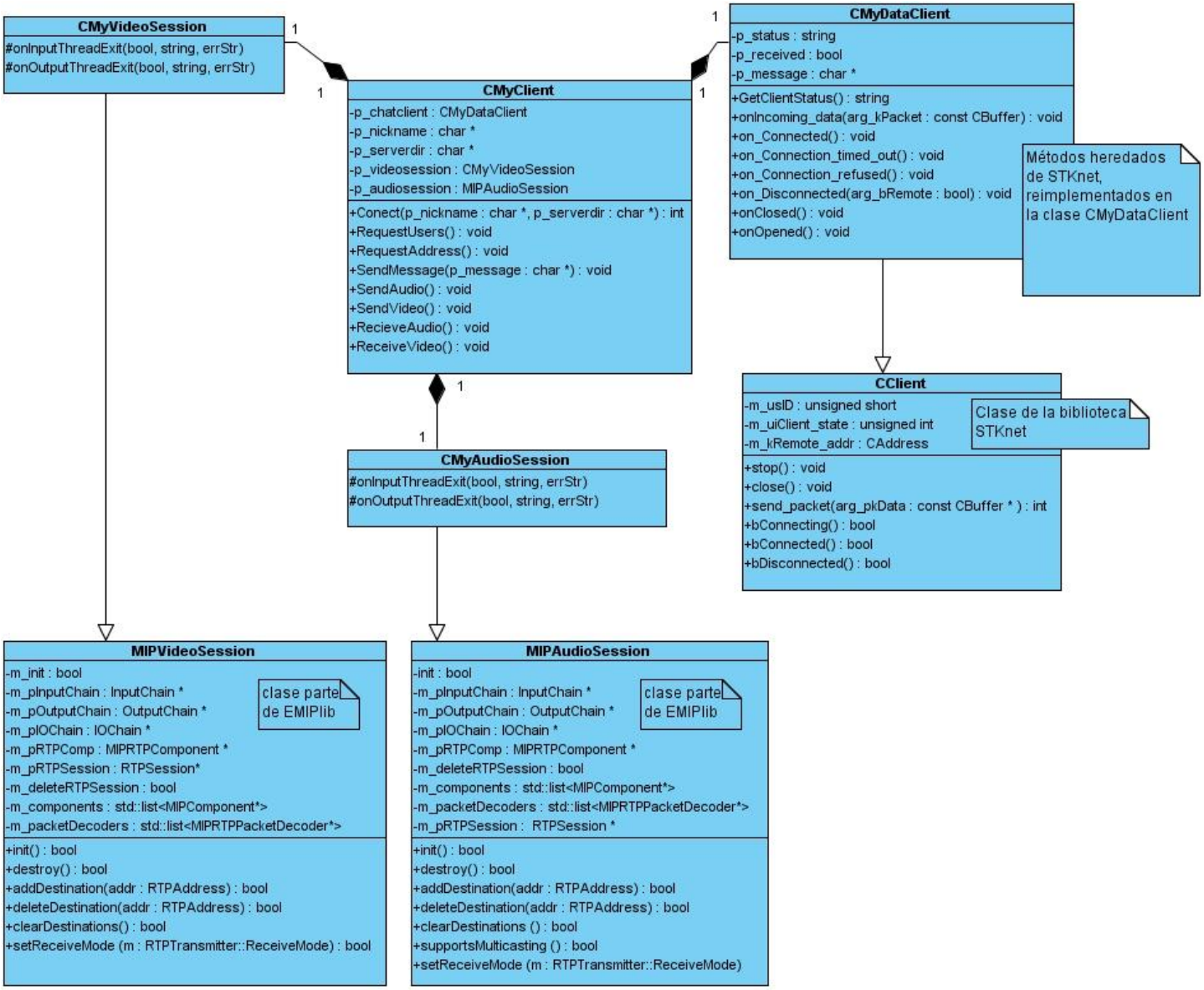

Figura 14. Diagrama de Clases del Diseño del subsistema Cliente.

La figura anterior muestra la estructura funcional del subsistema Cliente, a través de estas clases se asegura la transmisión de datos, encapsulando la lógica de la transmisión para poder brindar una interfaz al usuario que sea transparente a la estructura interna y a las acciones que se llevan a cabo para establecer la conexión y la comunicación con el Servidor. También se dividen las responsabilidades a la hora de transmitir los distintos tipos de datos.

La razón por la que algunas de las clases de las bibliotecas usadas fueron especificadas en los diagramas (figura. 13 y figura. 14), es porque la relación que tienen con las clases del módulo es de vital importancia para el funcionamiento de este y el entendimiento del modelo representado.

Las clases CMyServer y CMyDataClient heredan de CServer y CClient de la biblioteca STKNET respectivamente, las clases de la biblioteca fueron representadas para un mejor entendimiento del diagrama ya que algunos atributos y métodos funcionales que se usan en CMyServer y CMyDataClient son heredados.

Las clases CMyAudioSession y CMyVideoSession heredan de MIPAudioSession y MIPVideoSession de la biblioteca EMIPLib respectivamente. Los métodos que se implementaron en estas clases son los de controlar los errores que puedan suceder durante la transmisión de audio y video, sin embargo se usan varios métodos y atributos de estas clases, que se heredaron directamente de la biblioteca y para representar y entender cuáles son las funciones de estas clases fue necesario representarlas en el diagrama.

4.5 Descripción de las Clases del diseño

- SUBSISTEMA CLIENTE.

Nombre: CMyClient	
Tipo de clase: Controladora	
Atributo	**Tipo**
p_chatclient	CMyDataClient*
p_nickname	char*
p_serverdir	char*
p_videosession	CMyVideoSession*
p_audiossesion	CMyAudioSession*
Para cada responsabilidad:	
Nombre:	Connect(char* p_nickname, char* p_serverdir)
Descripción:	Maneja la conexión del cliente con el servidor.
Nombre:	RequestUsers()
Descripción:	Crea y envía un paquete al servidor solicitando el nombre de los usuarios en línea.
Nombre:	RequestAddress()
Descripción:	Este método crea y envía un paquete al servidor solicitando las direcciones de los usuarios en línea.
Nombre:	SendMessage(char* p_message)
Descripción:	Crea y envía un paquete al servidor con el nombre del usuario emisor y el texto del mensaje a enviar

Nombre:	SendAudio()
Descripción:	Crea y envía un paquete al servidor informándole que el usuario va a enviar sonido.
Nombre:	SendVideo()
Descripción:	Crea y envía un paquete al servidor informándole que el usuario va a enviar video.
Nombre:	RecieveAudio()
Descripción:	Inicializa los componentes necesarios para recibir audio en tiempo real.
Nombre:	ReceiveVideo()
Descripción:	Inicializa los componentes necesarios para recibir video en tiempo real.

Tabla 29 Descripción de la Clase CMyCliente

Nombre: CMyDataClient	
Tipo de clase: Controladora	
Atributo	**Tipo**
p_status	string
p_received	bool
p_message	char*
Para cada responsabilidad:	
Nombre:	GetClientStatus()
Descripción:	Devuelve el estado del cliente.
Nombre:	onIncoming_data(const CBuffer &arg_kPacket)
Descripción:	Gestiona todos los paquetes especiales que arriban del servidor.
Nombre:	on_Connected()
Descripción:	En este método inicializa las acciones relacionadas con una conexión con el server que acaba de ser establecida con éxito.
Nombre:	on_Connection_timed_out()
Descripción:	En este método se inicializan las acciones para informar a la aplicación que una solicitud de conexión con el servidor no ha tenido respuesta en un tiempo.
Nombre:	on_Connection_refused()
Descripción:	En este método se inicializan las acciones para informar a la aplicación que una solicitud de conexión con el servidor ha sido rechazada.
Nombre:	on_Disconnected(bool arg_bRemote)
Descripción:	En este método se inicializan las acciones para informar a la aplicación que la conexión activa con el servidor se acaba de perder.
Nombre:	on_Closed()
Descripción:	En este método se informa al servidor que el cliente se va a desconectar.
Nombre:	on_Opened()
Descripción:	En este método se informa a la aplicación que el cliente está listo para tratar de conectarse.

Tabla 30 Descripción de la Clase CMyDataClient

Nombre: CCliente	
Tipo de clase: Controladora	
Atributo	**Tipo**
m_usID	unsigned short
m_uiClient_state	unsigned int
m_kRemote_addr	CAddress
Para cada responsabilidad:	
Nombre:	stop()
Descripción:	Desconecta al cliente.
Nombre:	close()
Descripción:	Cierra el socket que se abrió para la transmisión de datos, desconecta al cliente y llama al evento OnClosed().
Nombre:	send_packet(const CBuffer * arg_pkData)
Descripción:	Envía el buffer que se le pasa por parámetro hacia el servidor.
Nombre:	bConnecting()
Descripción:	Retorna verdadero si el estado del cliente es “Connecting”, falso en cualquier otro caso.
Nombre:	bConnected()
Descripción:	Retorna verdadero si el estado del cliente es “Conneced”, falso en cualquier otro caso.
Nombre:	bDisconnected()
Descripción:	Retorna verdadero si el estado del cliente es “Disconnected”, falso en cualquier otro caso.
“Eventos”	
A continuación encontramos como métodos virtuales puros los que podríamos llamar eventos, son los métodos que se brinda al cliente para que controle cada uno de los posibles eventos que se pueden desarrollar en el cliente.	
Nombre:	onIncoming_data(const CBuffer& arg_kData)
Descripción:	Controlar la llegada de los datos desde el server.
Nombre:	onConnected()
Descripción:	Controlar las acciones cuando se establece la conexión.
Nombre:	onConnection_timed_out()
Descripción:	Controlar las acciones cuando se terminó el tiempo de espera para la respuesta del servidor a la solicitud de conexión.
Nombre:	onConnection_refused()
Descripción:	Controlar las acciones cuando se niega la conexión al server.
Nombre:	onDisconnected(bool arg_bRemote)
Descripción:	Controlar las acciones cuando es desconectado el cliente, si arg_bRemote es verdadero significa que fuimos desconectados por el server, sino nos desconectamos por voluntad propia.

Tabla 31 Descripción de la Clase CCliente

Nombre: MyAudioSession	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	void onInputThreadExit(bool err, const std::string &compName, const std::string &errStr)
Descripción:	Este método su usa para chequear errores en el hilo de entrada de los componentes
Nombre:	void onOutputThreadExit(bool err, const std::string &compName, const std::string &errStr)
Descripción:	Este método su usa para chequear errores en el hilo de salida de los componentes.

Tabla 32 Descripción de la Clase MyAudioSession

Nombre: MyVideoSession	
Tipo de clase: Controladora	
Para cada responsabilidad:	
Nombre:	void onInputThreadExit(bool err, const std::string &compName, const std::string &errStr)
Descripción:	Este método su usa para chequear errores en el hilo de entrada de los componentes
Nombre:	void onOutputThreadExit(bool err, const std::string &compName, const std::string &errStr)
Descripción:	Este método su usa para chequear errores en el hilo de salida de los componentes.

Tabla 33 Descripción de la Clase MyVideoSession

Esta clase pertenece a la biblioteca EMIPLIB

Nombre: MIPVideoSession	
Tipo de clase: Controladora	
Atributo	**Tipo**
m_init	bool
m_pInputChain	InputChain *
m_pOutputChain	OutputChain *
m_pIOChain	IOChain *
m_pRTPComp	MIPRTPComponent *
m_pRTPSession	MIPRTPSession *
m_deleteRTPSession	bool
m_components	std::list<MIPComponent *>
m_packetDecoders	std::list<MIPPacketDecoder *>
Para cada responsabilidad:	
Nombre:	init()
Descripción:	Inicia los parámetros de una sesión de transmisión de video.
Nombre:	destroy()
Descripción:	Llama a los destructores de cada componente para que se hagan cargo de terminar la sesión.
Nombre:	addDestination()
Descripción:	Añade un nuevo destino para la transmisión de video.
Nombre:	deleteDestination()
Descripción:	Borra un destino de la lista de destinos a los que se transmite video.
Nombre:	clearDestination()
Descripción:	Borra la lista de destinos.
Nombre:	setReceiveMode(RTPTransmitter :: ReceiveMode m)
Descripción:	En este método se inicializan las acciones para informar a la aplicación que la conexión activa con el servidor se acaba de perder.

Tabla 34 Descripción de la Clase MIPVideoSession

Esta clase pertenece a la biblioteca EMIPLIB

Nombre: MIPAudioSession	
Tipo de clase: Controladora	
Atributo	**Tipo**
m_init	bool
m_pInputChain	InputChain *
m_pOutputChain	OutputChain *
m_pIOChain	IOChain *
m_pRTPComp	MIPRTPComponent *
m_pRTPSession	MIPRTPSession *
m_deleteRTPSession	bool
m_components	std::list<MIPComponent *>
m_packetDecoders	std::list<MIPPacketDecoder *>
Para cada responsabilidad:	
Nombre:	init()
Descripción:	Inicia los parámetros de una sesión de transmisión de video.
Nombre:	destroy()
Descripción:	Llama a los destructores de cada componente para que se hagan cargo de terminar la sesión.
Nombre:	addDestination()
Descripción:	Añade un nuevo destino para la transmisión de video.
Nombre:	deleteDestination()
Descripción:	Borra un destino de la lista de destinos a los que se transmite video.
Nombre:	clearDestination()
Descripción:	Borra la lista de destinos.
Nombre:	setReceiveMode(RTPTransmitter :: ReceiveMode m)
Descripción:	En este método se inicializan las acciones para informar a la aplicación que la conexión activa con el servidor se acaba de perder.

Tabla 35 Descripción de la Clase MIPAudioSession

- SUBSISTEMA SERVIDOR.

Nombre: CMyServer	
Tipo de clase: Controladora	
Atributo	**Tipo**
p_status	string
p_recibido	bool
p_id_Client	unsigned short
Para cada responsabilidad:	
Nombre:	PutOnline(char * p_adrress)
Descripción:	Abre el puerto de conexión, pone online al servidor e informa el resultado de las acciones.
Nombre:	SendtoAll(const CBuffer msg, unsigned short arg_usClient_id)
Descripción:	Envía el buffer msg enviado por el cliente de id arg_usClient_id al resto de los clientes conectados.
Nombre:	onIncoming_client_Data(unsigned short arg_usClient_id, const CBuffer arg_kData)
Descripción:	Se procesan los mensajes que los clientes envían al servidor.
Nombre:	onMax_client_limit_reached()
Descripción:	Se inician las acciones correspondientes al evento: máximo número de clientes alcanzado
Nombre:	onClosed()
Descripción:	Se inician las acciones correspondientes al cierre del servidor.
Nombre:	onOpened()
Descripción:	Se inician las acciones correspondientes a la puesta en línea del servidor.
Nombre	GetServerStatus()
Descripción:	Retorna el estado del servidor.
Nombre:	GetClientInfo(unsigned short arg_usClient_id)
Descripción:	Retorna la información relacionada con el cliente de id arg_usClient_id.
Nombre:	onAccept_client (const CAddress adr)
Descripción:	Se inician las acciones correspondientes al evento aceptar cliente.
Nombre:	onCliente_connected(unsigned short arg_usClient_id, const CAddress adr)
Descripción:	Se inician las acciones correspondientes al evento cliente conectado
Nombre:	onClient_disconnected(unsigned short arg_usClient_id, bool arg_bRemote)
Descripción:	En este método se inician las acciones del servidor correspondientes al evento cliente desconectado, si el parámetro arg_bRemote es verdadero significa que el cliente se desconectó sino el servidor desconectó al cliente.

Tabla 36 Descripción de la Clase CMyServer

4.6 Diagramas de Secuencia

- SUBSISTEMA CLIENTE.

Los diagramas de secuencia permiten tener una visión más clara de las funcionalidades del sistema y de las secuencias de acciones realizadas en cada caso de uso. Los casos de uso del Cliente tienen acciones en común, por tanto se ha decidido mostrar en este documento tres de ellos que representen la funcionalidad del sistema y la secuencia de pasos seguidos para cumplir los pedidos de la Aplicación Usuario. El resto de los diagramas de secuencia se encuentran en el anexo “Diagramas de Secuencia”, para ser consultados en caso de interés.

Caso de Uso “Conectar Cliente”

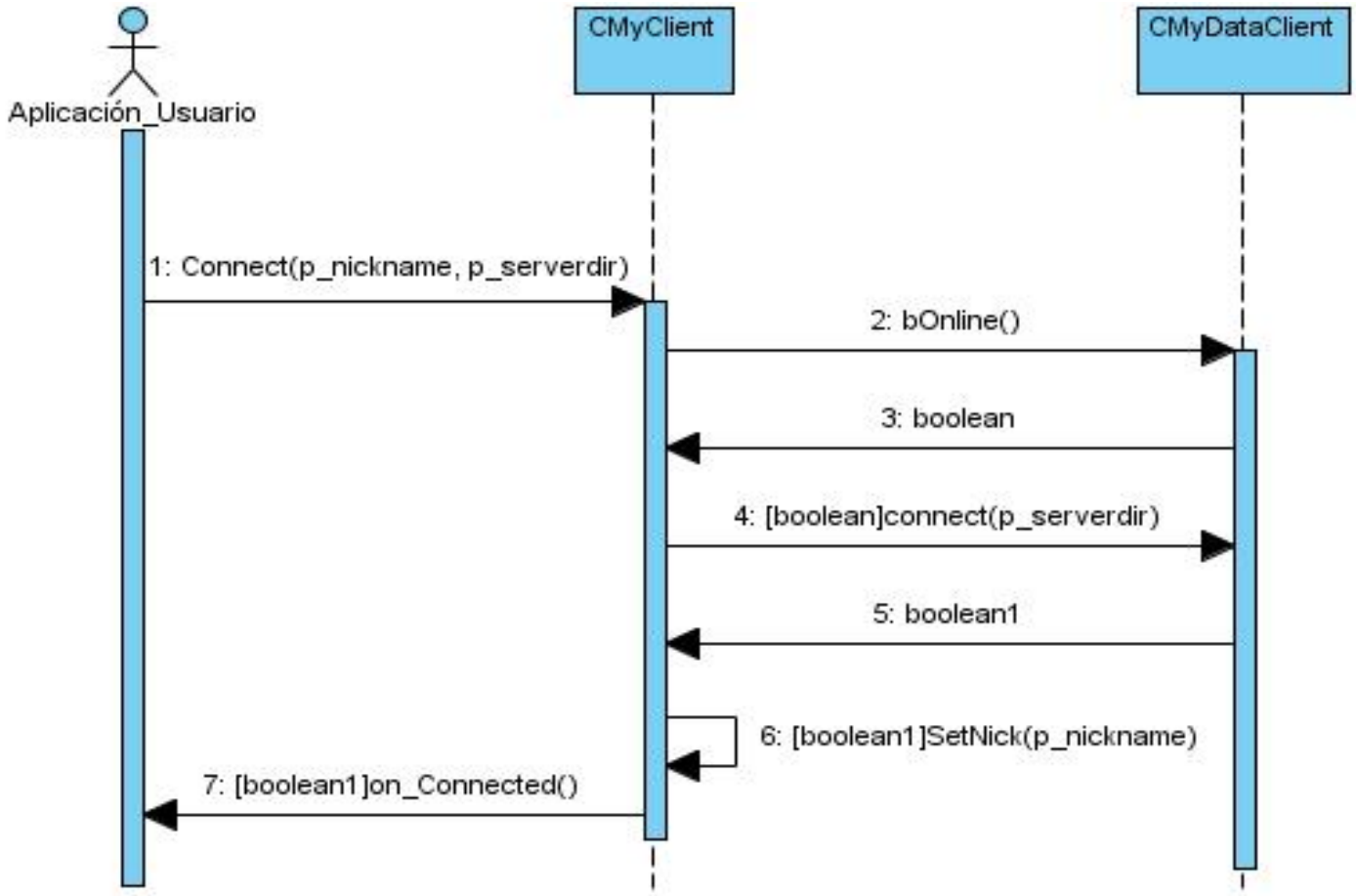

Figura 15. Diagrama de Secuencia, Caso de Uso Conectar Cliente.

Caso de Uso "Enviar Texto"

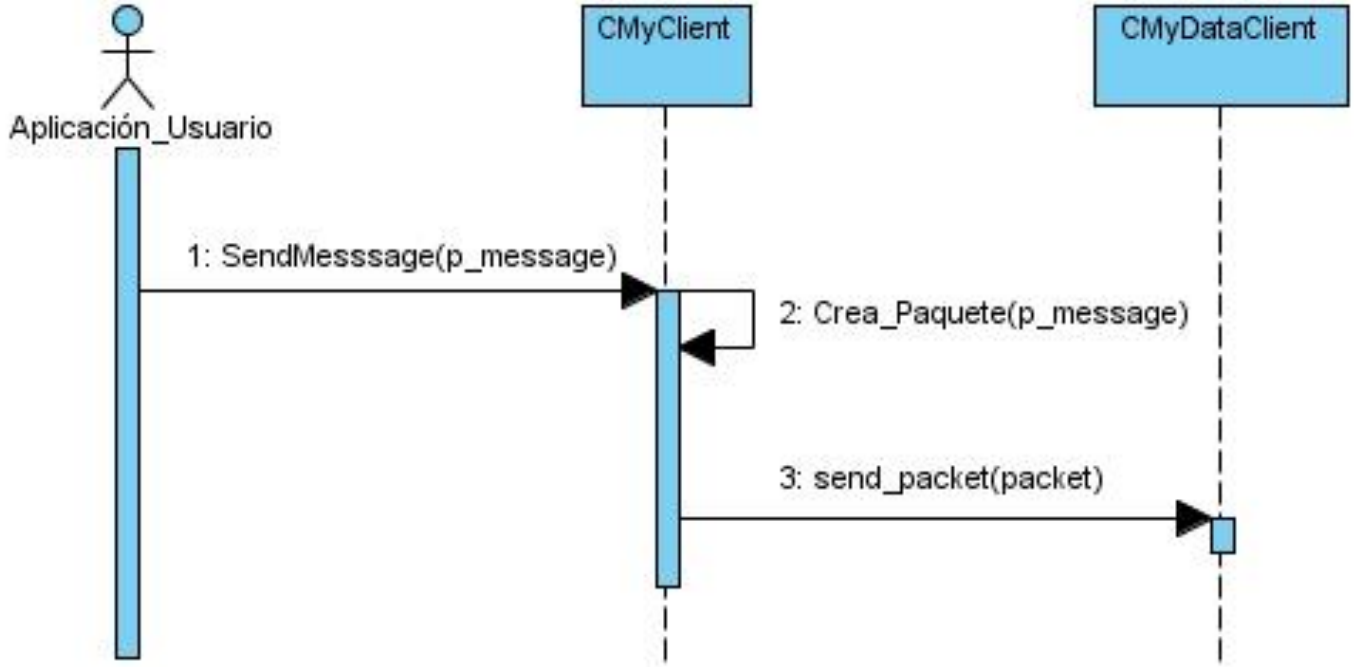

Figura 15. Diagrama de Secuencia, Caso de Uso Enviar Texto.

Caso de Uso"Enviar Audio"

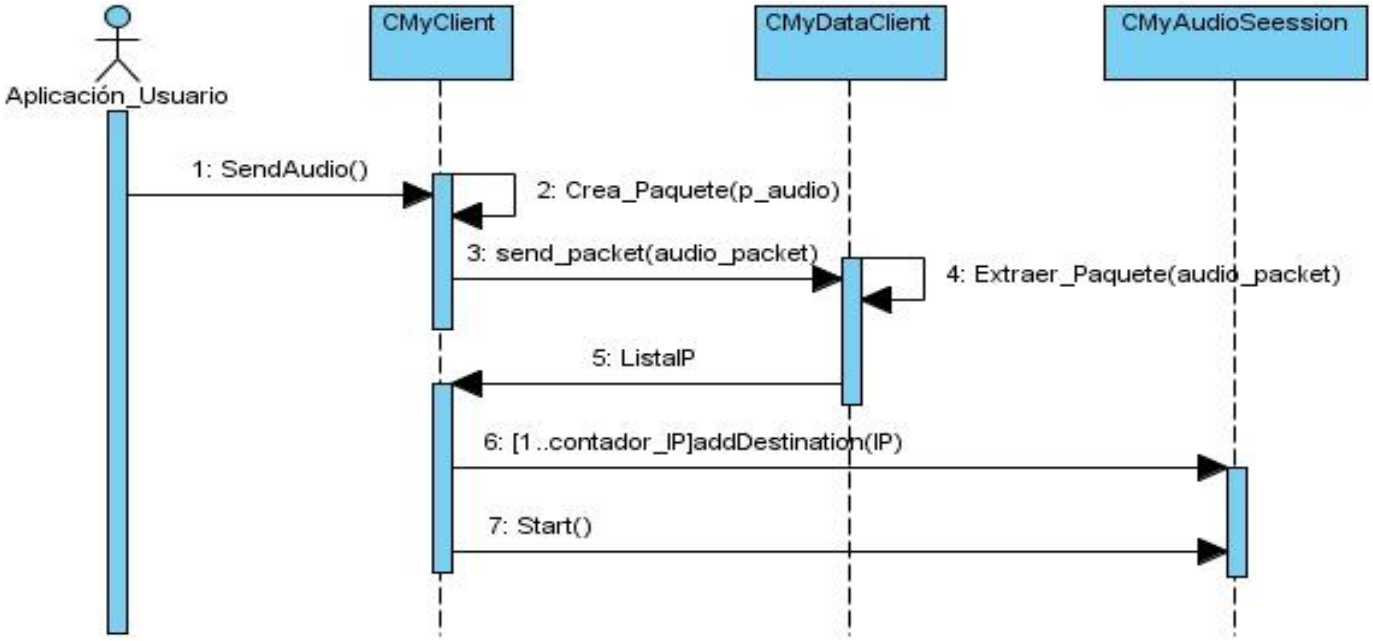

Figura 16. Diagrama de Secuencia, Caso de Uso Enviar Audio

- SUBSISTEMA SERVIDOR.

Los actores del servidor son Cliente y Administrador, sin embargo las operaciones lógicas de peso y las funcionalidades principales de este se encuentran en el procesamiento de los mensajes de los clientes. Los diagramas de secuencia ayudan a entender el flujo de acciones que se realizan en el servidor. En este trabajo se ha decido mostrar tres diagramas de secuencia donde se presentan las principales y que son comunes para el procesamiento de mensajes del cliente. El resto de los diagramas puede ser consultado en el anexo "Diagramas de Secuencia" en caso de resultar de interés.

Caso de Uso "Desconectar."

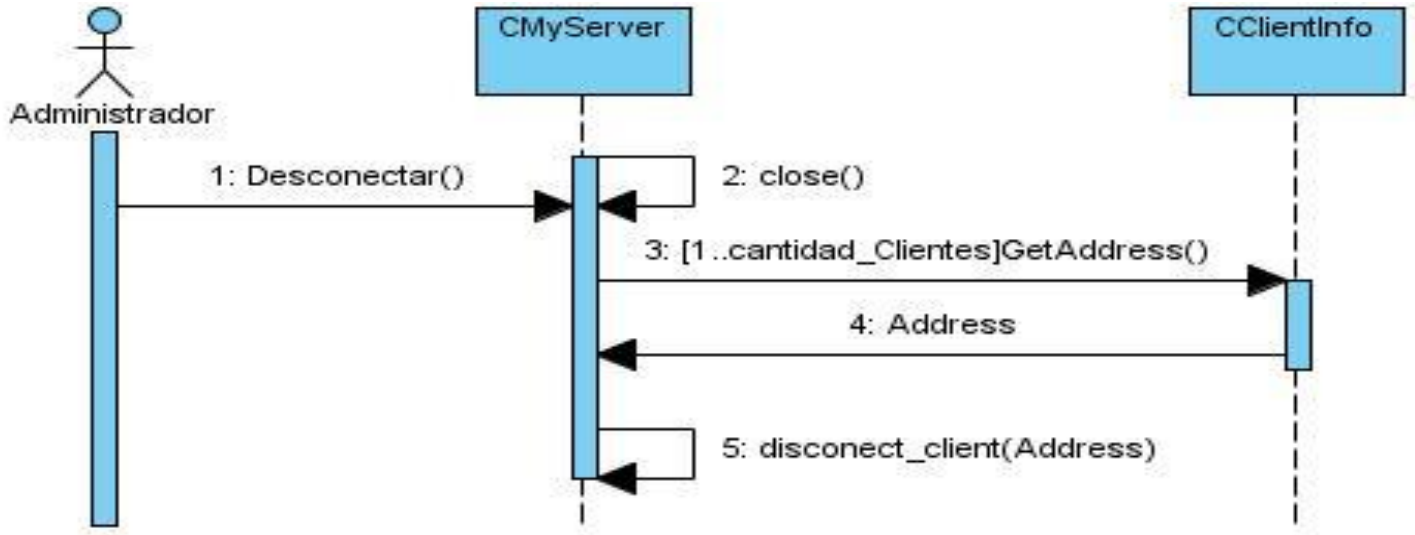

Figura 17. Diagrama de Secuencia, Caso de Uso Desconectar.

Caso de Uso "Procesamiento de paquetes" sección "Mensaje"

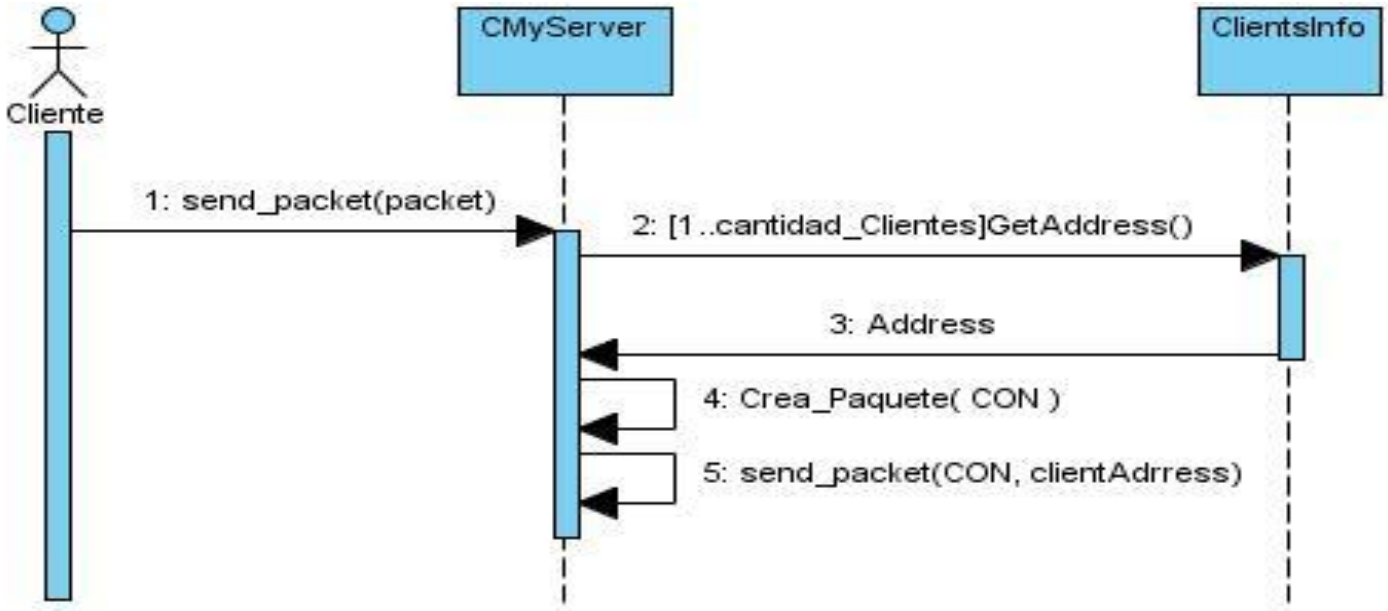

Figura 18. Diagrama de Secuencia, Caso de Uso Procesamiento de paquetes sección "Mensaje".

Caso de Uso "Procesamiento de paquetes" sección "CON"

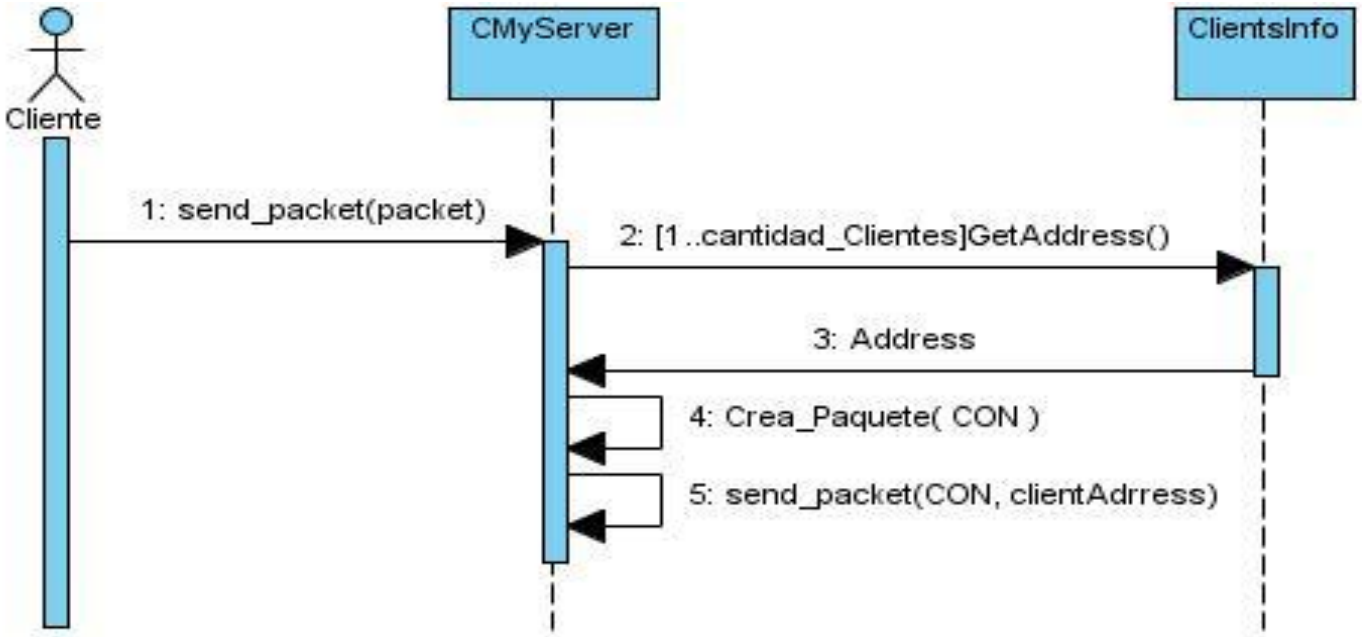

Figura 18. Diagrama de Secuencia, Caso de Uso Procesamiento de paquetes sección "CON".

4.7 Diagrama de componentes

A continuación mostramos los diagramas de componentes de los paquetes de nuestros subsistemas Cliente y Servidor, con el objetivo de tener una idea de los componentes que los conforman y las dependencias entre estos. Los diagramas mostrados a continuación incluyen las bibliotecas y los archivos de nuestras clases.

- Diagrama de componentes subsistema Cliente

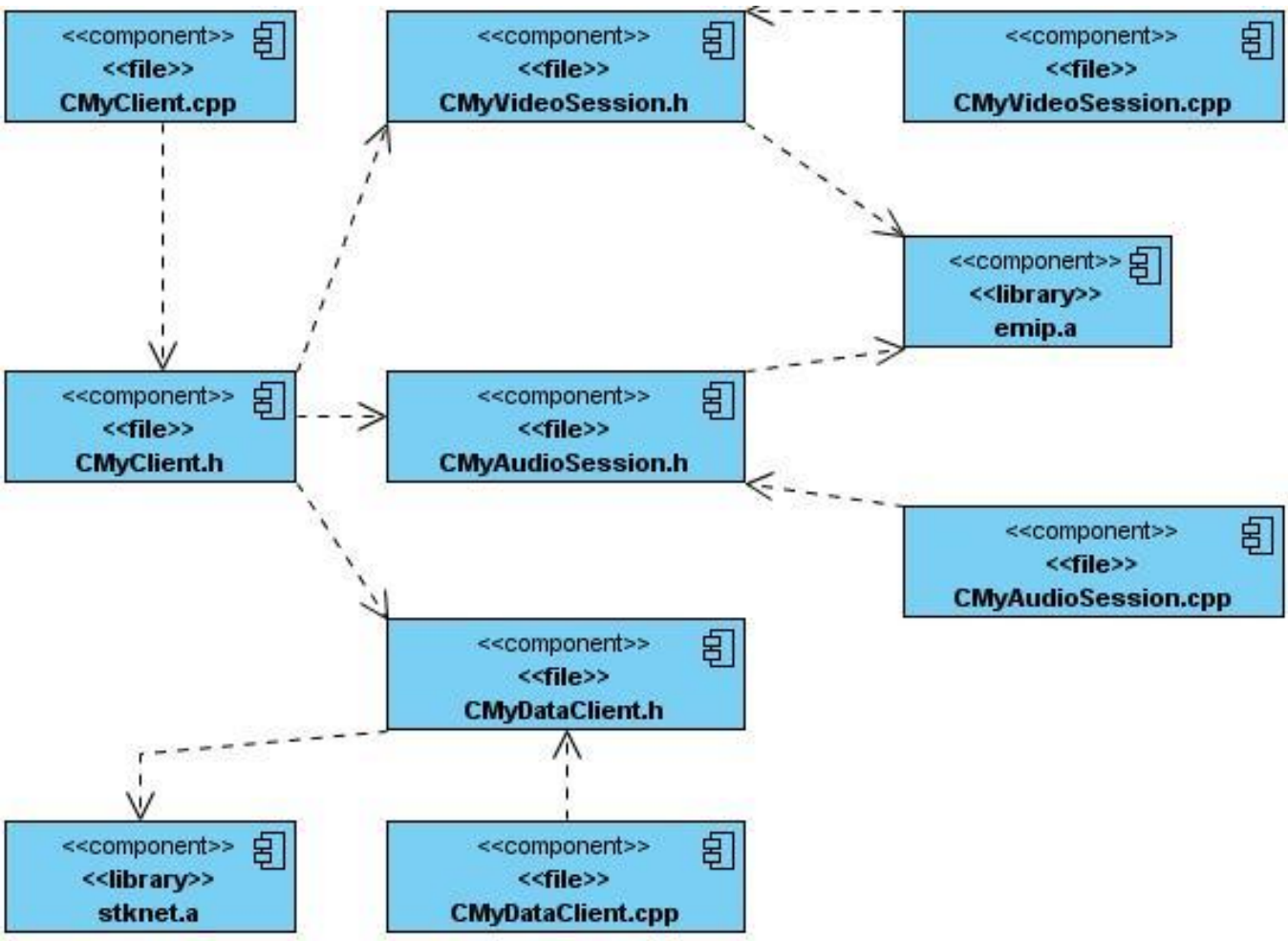

Figura 19. Diagrama de componente del subsistema Cliente.

- Diagrama de componentes subsistema Servidor

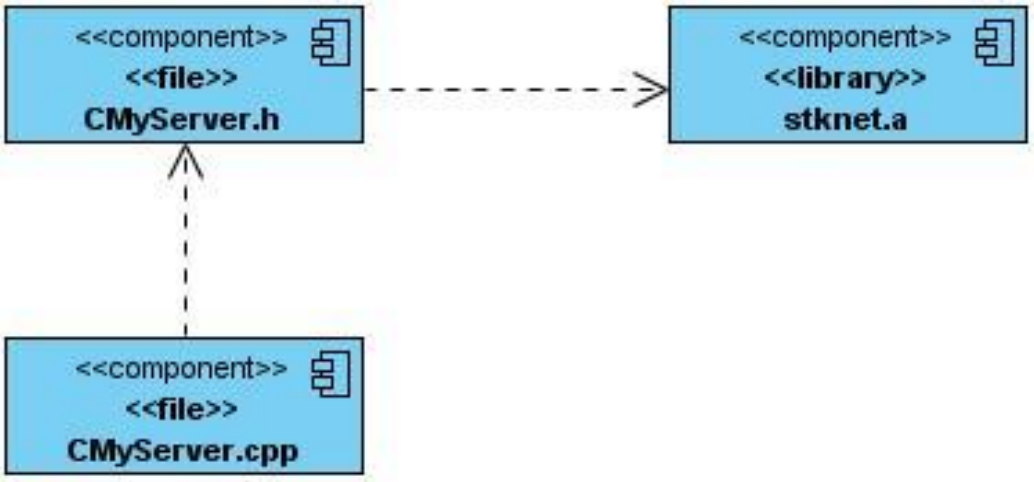

Figura 20. Diagrama de componente del subsistema Servidor.

4.8 Diagrama de despliegue

El diagrama de despliegue para este sistema incluye la representación de una PC que realiza la función de servidor y una o más PCs Cliente conectadas al Servidor por medio del protocolo UDP, los cliente también se comunican directamente entre sí (a través del protocolo UDP) cuando van a transmitir audio o video, de esta manera se evita el retraso de redireccionar el flujo de datos en el servidor, considerando que en la transmisión de audio y video en tiempo real un parámetro importante es la velocidad con que arriban los datos.

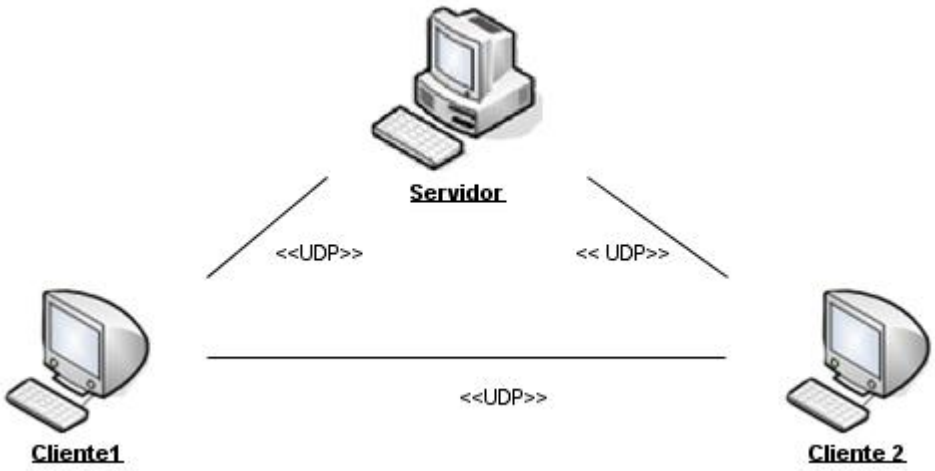

Figura 2. Diagrama de despliegue.

4.9 Reglas de codificación

A continuación se especifica el estándar de codificación a utilizar para el desarrollo del módulo. Los atributos y métodos serán escritos en inglés por ser un idioma muy difundido en el mundo informático. Además es importante especificar que los atributos y métodos usados que pertenezcan a alguna de las bibliotecas utilizadas, seguirán el mismo estándar de codificación usado por los desarrolladores de dicha biblioteca.

- **Nombre de los ficheros**

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

- **Clases**

Regla:
Las clases se nombrarán de la siguiente manera:
class CMyNameOfClase

- **Constantes**

Regla:
Los nombres de constantes se escriben con letras en mayúscula. En caso de estar compuesta por más de una palabra, éstas se separan por un carácter "_".
#define PORT 25000
#define LISTEN_PORT 60000

- **Variables**

Regla:
Las variables se escriben con minúsculas y comienzan con "p_", las variables con nombres compuestos no serán separadas por ningún carácter especial

MyDataClient* p_chatclient;

char* p_nick;

- **Parámetros (argumentos) de métodos**

Regla:
Cumplen con la misma regla de las variables.

void SendMessage(char* p_message)

- **Métodos miembros de clases**

Regla:
Comienzan con mayúscula y si son compuestos son unidos comenzando con mayúscula cada nueva palabra:

void SendMessage(char* p_message)

- **Comentarios**

Regla:

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

```
// Esto es un comentario.
```

4.10 Conclusiones del Capítulo

Al terminar este capítulo queda definido el diseño completo del módulo de red. En el diagrama de clases del diseño se describió la estructura de los subsistemas cliente y servidor que conforman el módulo. Además fue mostrada la interacción del conjunto de objetos que participan en el desarrollo de ambos subsistemas con los diagramas de secuencia correspondientes. A partir del diseño de clases se crearon los componentes físicos que se traducen en ficheros .h y .cpp correspondientes a la implementación en C++.

CONCLUSIONES

Con el desarrollo de este trabajo se obtuvo como resultado un módulo de red que se puede acoplar a las prácticas de laboratorio virtuales del proyecto Prolavi. El módulo cumple con las necesidades actuales del proyecto, asegurando los procesos de conexión y transmisión de texto, video y sonido en tiempo real entre sus usuarios. Para el desarrollo del módulo se utilizó la arquitectura Cliente-Servidor, definiendo y encapsulando las responsabilidades en dos subsistemas, Cliente y Servidor, para cada uno de los subsistemas se llevo a cabo un proceso de desarrollo obteniéndose resultados tangibles.

El resultado obtenido del desarrollo del subsistema Servidor fue un servidor funcional, el cual se ejecuta en una computadora, una vez que esté corriendo atiende las peticiones de conexión y procesa todos los paquetes de los clientes.

Al desarrollar el subsistema Cliente se obtuvieron un conjunto de clases que pueden ser utilizadas para la transmisión de datos de tipo texto, video y sonido por una aplicación. Para mostrar un ejemplo de cómo una aplicación puede utilizar las clases de este subsistema se elaboró una aplicación sencilla donde se muestra las funcionalidades que se brindan y el uso de la interfaz que provee el subsistema.

Para la transmisión de audio se utilizó la biblioteca EMIPlLIB, esta permite inicializar componentes con los cuales se pueden manipular la tarjeta de sonido de la computadora para la reproducción o entrada de audio.

Las funciones de transmisión de audio del módulo han sido probadas con programas sencillos que envían audio desde archivos (de extensión WAV) y también produciéndose la entrada del audio desde un micrófono, de una computadora a otra.

El video a transmitir se captura a través de una webcam y se transmite en tiempo real a los otros usuarios del módulo. Para esta funcionalidad también se utiliza la biblioteca EMIPlLIB, las funciones para la captura y envío del video fueron diseñadas e implementadas, sin embargo la forma en que se visualizará el video que se está recibiendo depende de la aplicación que esté usando el módulo desarrollado. La biblioteca EMIPlLIB brinda un componente (MIPQtOutput), con el cual se muestra un ejemplo para visualizar el video recibido, este componente usa QT 3, y no es compatible con versiones superiores de QT. Si no se hace uso de este componente es necesario usar en dispositivo para almacenar los frames de video que se van recibiendo, o sea el componente MIPVideoFrameStorage, en el cual se van almacenan los frames de videos recibidos y la aplicación que use el módulo decidiría como mostrar la secuencia de video.

Al acoplar el módulo desarrollado a las prácticas de laboratorio Prolavi se asegura la comunicación de los usuarios involucrados en prácticas de laboratorio conjuntas por lo que el objetivo principal de este trabajo se ha cumplido.

RECOMENDACIONES

- Se recomienda continuar con el desarrollo de este trabajo, con el objetivo de incluirle más funcionalidades orientadas al uso de este en simulaciones y prácticas de laboratorios virtuales.
- Se recomienda el estudio del componente MIPQtOutput de la biblioteca EMIPLIB, con el objetivo de analizar cómo se manejan los frames de video recibidos para la visualización, y extraer la sintaxis de los métodos que utiliza para poder aplicarlos en versiones superiores de QT.
- Se recomienda el trabajo con el componente MIPVideoFrameStorage con el objetivo de personalizar la salida del video en las aplicaciones que usen el módulo.

BIBLIOGRAFIA

CITADA:

1- **MCMAHON, Richard A.** *Introducción a las redes*. Universidad de Huston. Texas.

2- **RODRIGUES NOA, Carlos Y GÓMEZ FINALÉ, Yordanis.** *Arquitectura de red para la confección de videojuegos multijugador*. Tesis UCI, Facultad 5, 2008.

3 - **HAWKSOFT.** *Hawk Network Library*. [Online] 2005. [Cited: noviembre, 11, 2008]**.** http://www.hawksoft.com/hawknl/

4 – **JENKINS Software.** *Manual Raknet.* [Online] 2008. [Cited: noviembre, 12, 2008]. http://www.jenkinssoftware.com/raknet/manual/index.html

5 - **DATAREEL**. *DataReel Open Source Home Page*. [Online] 2008.[Cited: enero, 11, 2009]. http://www.datareel.com/

6 - **SOFTPEDIA.** *libtcp++ 0.1.2.* [Online] 2006. [Cited: noviembre, 14, 2008].

http://linux.softpedia.com/get/Programming/Libraries/libtcpplusplus-12490.shtml

7 – **ZOIDCOM.** *Zoidcom network library*. [Online] 2006. [Cited: noviembre, 14, 2008].

http://www.zoidcom.com/

8 – **Hiebert Garin.** *OpenAL Programmer's Guide*. Creative Technology Limited, 2007.

http://worldspace.berlios.de/openal/tut_0/openal00.html

9 – **Qu, Zhiyi & Guo, Yachong.** *Research on Real-Time Video Network Transmission System on Linux*. Lanzhou University, Gansu, China.

10 - **Viscaya Guarín, Pedro & Ayala Pañuela, Omar.** *Transmisión de Secuencias Codificadas de Telefonía Visual Usando RTP.* Pontificia Universidad Javeriana, Bogotá, Colombia, 2004.

11 - **Herrera Pérez, Enrique.** *Tecnologías y redes de transmisión de datos*. Editorial Limusa, 2003.

CONSULTADA:

Blanchette Jasmin. *C++ GUI Programming with Qt 4.* Editorial Prentice Hall, 2006.

Duch i Gavaldà, Jordi y Tejedor Navarro, Heliodoro. *Lógica de videojuegos.* Barcelona, 2008.

Duch i Gavaldà, Jordi y Tejedor Navarro, Heliodoro. *Sonido, interacción y redes.* Barcelona, 2008.

Jacobson Ivar & Booch Grady & Rumbaugh James. *El Proceso Unificado de Desarrollo de Software.* Santa Clara, California, 1999.

Molkentin Daniel. *The Art of Building Qt Aplications.* San Francisco, EEUU, 2007.

Negus Christopher. *Linux Bible*. Wiley Publishing. Indianapolis, Indiana, 2005.

Stallings William. *Comunicaciones y redes de Computadora. Editorial Prentice Hall, 2000.*

GLOSARIO DE TÉRMINOS

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

API: Application Programming Interface. Es una interfaz proporcionada por una librería para poder acceder a sus funciones.

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

Arquitectura cliente-servidor: Esta arquitectura consiste básicamente en un cliente que realiza peticiones a otro programa -el servidor- que le da respuesta.

Baneados: Se refiere a los IP de los usuarios que no serán admitidos por el servidor.

Broadcasting: Se refiere a la transmisión de información que será recibido por todos los dispositivos en una red.

Buffer: Estructura utilizada para almacenar datos, generalmente guarda información digital que se almacena temporalmente.

Capa de aplicación: Define los protocolos que utilizan las aplicaciones para intercambiar datos.

Capa de red: Según la normalización modelo de referencia de Interconexión de Sistemas Abiertos (OSI), es una capa que proporciona conectividad y selección de ruta entre dos sistemas de hosts.

Chat: Se refiere a cualquier tipo de comunicación sobre internet.

Chat de voz: Se refiere a la comunicación sobre internet por medio de la voz.

Cliente-servidor: Arquitectura de comunicación en la que un servidor central actúa como coordinador de la comunicación. Todos los mensajes deben pasar por él antes de llegar al destino.

Códec: Describe una especificación desarrollada en software, hardware o una combinación de ambos, capaz de transformar un archivo con un flujo de datos (stream) o una señal.

Código abierto: (en inglés open source): Es el término con el que se conoce al software distribuido y desarrollado libremente.

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

Demultiplexar: Se refiere a separar de un archivo contenedor, varias pistas, por ejemplo separar audio y video de un archivo de película.

DirectShow: Es una API de captura de video para Windows.

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

Endian: Se refiere al formato en que se almacenan en los distintos sistemas operativos los datos de más de un byte, según el orden en que se ubican en memoria los bytes se clasifican en "little endian" (byte menos significativo delante) o “big endian” (byte más significativo delante).

Frame: Se refiere a un fotograma o cuadro, una imagen particular dentro de una sucesión de imágenes que componen una animación.

Framework: Una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado.

Internet: Es un conjunto descentralizado de redes de comunicación interconectadas, que utilizan la familia de protocolos TCP/IP.

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

IPv4: Se refiere a la primera versión del protocolo IP que forma la base de Internet.

IPv6: Se refiere a la versión del protocolo IP que está actualmente en uso.

LibSDL: Biblioteca SDL, Simple DirectMedia Layer, es un conjunto de bibliotecas desarrolladas en C, con el objetivo de simplificar el trabajo con imágenes, sonido y operaciones de dibujado en dos dimensiones, muy utilizada en la creación de juegos.

Licencia GNU LGPL: (en inglés GNU Library General Public License) es una licencia de software creada por la Free Software Foundation. Los contratos de licencia de la mayor parte del software están diseñados para jugar con su libertad de compartir y modificar dicho software.

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

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

Multicasting: Se refiere al envío de información a ciertos destinatarios específicos, más de uno.

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

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

Multiplexar: Se refiere a combinar en un mismo archivo contenedor, varias pistas de dos archivos, por ejemplo de audio y vídeo, para su correcta reproducción.

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

Peer-to-peer: Arquitectura de comunicación donde todos los elementos actúan como iguales. La información se distribuye entre todos los clientes a la vez.

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

Prolavi: Nombre del proyecto Laboratorios Virtuales de la Universidad de las Ciencias Informáticas, Facultad 5, Habana, Cuba.

Protocolo: Conjunto de reglas que determinan como se realizará el intercambio de datos entre dos dispositivos.

Realidad Virtual: Se refiere a un sistema o interfaz informático que genera entornos sintéticos en tiempo real, representación de las cosas a través de medios electrónicos o representaciones de la realidad.

RFC 3550: Documento oficial que describe el protocolo de transmisión en tiempo real (RTP).

RFC 3711: Documento oficial que describe el protocolo seguro de transporte en tiempo real (SRTP).

RTP (Real-time Transport Protocol): Es un protocolo de nivel de sesión utilizado para la transmisión de información en tiempo real, como por ejemplo audio y video.

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

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

Socket: Tecnología que permite abrir una conexión entre dos ordenadores y comunicarse a través de ella.

Sonido 3D: Se refiere al uso de múltiples canales de audio para provocar efectos envolventes a la audiencia.

Speex: Es un códec libre para voz, sin restricciones de ninguna patente de software.

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

Tiempo Real: Se refiere a aquellos sistemas que interactúan activamente con un entorno con dinámica conocida en relación con sus entradas, salidas y restricciones temporales.

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

Unicasting: Se refiere al envío de información desde un único emisor a un único receptor.

VoIP: Voz sobre Protocolo de Internet, también llamado Voz sobre IP, VozIP, VoIP. Es un grupo de recursos que hacen posible que la señal de voz viaje a través de Internet empleando un protocolo IP.

Videoconferencia: Se refiere a la comunicación simultánea bidireccional de audio y vídeo, permitiendo mantener reuniones con grupos de personas situadas en lugares alejados entre sí.

Video4Linux: Es una API de captura de video para Linux. Muchas webcams USB, sintonizadoras de tv, y otros periféricos son soportados. Video4Linux está integrado con el núcleo de Linux.