Universidad de las Ciencias Informáticas
Facultad 6

DESARROLLO DEL COMPONENTE: MANEJADOR DE DATOS DEL SERVIDOR DE STREAMING DISTRIBUIDO ALLFRY'S

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN INFORMÁTICA

AUTOR: Frank Emilio Hernández Sánchez

TUTOR: Ing. Víctor Frank Molina López

CO-TUTOR: Msc. Yunet González Mulet

La Habana, 27 de junio de 2011
"Año 53 de la Revolución"

"Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber."

Albert Einstein

Dedicatoria

A mis padres por su amor y cariño.

A mi hijo por ser lo más hermoso que tengo en el mundo.

A mis abuelos por quererme mucho.

A mis hermanos por ayudarme en todo.

Al resto de mi familia.

Agradecimientos

Agradezco a todos los hombres y mujeres que han dado todo de sí, para que muchos jóvenes como yo, puedan alcanzar su sueño, en especial al Comandante en Jefe Fidel Castro Ruz.

A mi madre por siempre brindarme su amor, apoyo y confianza en todos los momentos de la vida.

A mi padre por ser mi inspiración y guía para con la sociedad y la familia.

A mi hijo que lo amo con la vida y que me hace sentir muy feliz con el cariño que me entrega.

A mi hermano Yuniol por estar siempre apoyándome y transmitiéndome confianza para llegar a ser una persona preparada en la vida.

A mis hermanos Cristian y Cristofer por el amor que ellos siempre me han dado.

A mis queridos abuelos Juan y Lidia por ser mis segundos padres.

A mi bisabuelo Emeterio y abuelos Julián y Amada, que aunque ya no están presentes siempre los llevo en mi corazón.

Agradezco a mis tíos Alexis, Sonia, Rosa, Alberto, Giraldo, Tío Rubio, Cristy y Carlos, los cuales me han dado el amor y la fortaleza para seguir adelante en mis estudios.

A mis tíos queridos que ya no están junto a mí pero siempre los tengo en mi corazón, Fernando y tío Tite.

A mis primos Jimmy, Yasel, Maikel, Yani, Ernesto, Ernestico, Sandra, Ana Flavia, Giraldito y Ernesto Ruíz.

A Manuela, Lizy, Mary, Lazarita, Elianny, Tereza, Jorge, Brian, Brianny, Yoanna y Monga por llegar a formar parte de mis seres queridos.

A las personas del barrio del Condado, específicamente a los de la esquina "La Piedra".

Doy gracias a mis amistades del IPVCE Juan Carlos, Luis Adrian, Jorge Luis, Lenia, Lizzy, Yailen, Lisandra, Silvia, Gleiny, Yuneidys, Lisbet, Maikel, Jelson, Pedro, Raquel, Betsy y Rachel, por pasar momentos muy felices juntos a ellos.

A mis amigas de todo este tiempo de la Universidad, Lisandra Lucía, Susana, Dayana, Elisa, Yaimit, Misbel, Harlenia, Raiza, Arialis, Mailin, Olga, Dailyn, Yenie, Lianne y Lilibeth.

A mis amigos David, Liuver, Nilo, Yidian y Osvaldo.

A mis compañeros de las brigadas en las que estuve en la Universidad 9204, 9306, 9403, 6511 y en especial a mi grupo de 1er año (9105).

A mis compañeros que han trabajado conmigo en la FEU: Krysna, Freddy, Jorge, Carlos Arce, José Augusto, Tan, Yenier, Idanis, Yoelito, Ernestico, Lester, Daniel, Lorenzo, Cire, Alberto, Maidelyn, Aballe, Evely, Kenier, Janet Cristina, Camue y Anel.

A Raúl de la Cruz, Alden, Pacheco, Osdanay, Alexander, Surama, Fifi, Yuliet, Yanelis, Baby, Sandy, Daniel Burgos, Alleyne, Osmel, Igor, Irina, Abel, Roexcy, Solangel e Isachi por ser profesores excelentes que han sido capaces de formarme como buen profesional y persona.

A mis amigos del proyecto DESCOMTEC Anyer, Leosmel, Leidys, Yasiel y René.

A mi tutor Víctor por todos los señalamientos y recomendaciones que me hizo, lo que posibilitó el buen desarrollo de este trabajo.

A mi co-tutora Yunet por brindarme su dedicación y su apoyo incondicional tanto en lo profesional como en lo personal.

A Yusniel, Lisbeth, Rafael y Frank por haberme exigido mucho, para que este trabajo quedara con la calidad requerida.

A mis amigos Yusimi, Elida, Silvia, Yasniel, Dulce, Migue, José Antonio y Alfredo los cuales me ayudaron mucho cuando estuve en la misión de Venezuela.

A todas las personas que de una forma u otra he tenido el privilegio de compartir en estos años de vida.

Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Autor: Frank Emilio Hernández Sánchez

_____

Tutor: Ing. Víctor Frank Molina López

_____

Resumen

En la rama de la informática ha tomado un gran auge la transmisión de archivos audiovisuales por la red, gracias a la aparición de la tecnología streaming. Para que estos archivos se puedan visualizar mediante el uso de la misma, es imprescindible contar un servidor de streaming. Este tiene como función gestionar todas las peticiones realizadas por una o varias máquinas clientes, dando así el servicio solicitado. Estos servidores cuentan diferentes problemas, uno de ellos es con el almacenamiento y la recuperación de las medias, los cuales provocan que se hagan colisiones en la red y que la cantidad de conexiones que se pueden realizar sea un número pequeño.

En este trabajo se presenta un componente llamado Manejador de Datos para un Servidor Streaming Distribuido, basado en la arquitectura ADMS (Arquitectura Adaptive Distributed Multimedia Server) que funciona sobre un clúster. El mismo es capaz de almacenar y recuperar archivos audiovisuales sin que la red se sobrecargue, ofreciendo un aumento en la cantidad de conexiones posibles por los clientes. Este almacenamiento se hace de forma distribuida por todos los componentes que están ubicados en el clúster. Para eso, tiene en cuenta diferentes características que poseen cada uno de los servidores, seleccionando uno de ellos mediante un método de búsqueda heurística. También proporciona toda la información necesaria para que puedan ser recuperados todos los datos que han sido almacenados. Este sistema esta implementado completamente con tecnologías libres.

Palabras clave: Almacenamiento, Recuperación, Servidor Streaming Distribuido, Streaming.

Abstract

In the field of computing, the transmission of audiovisual files over the network has taken an important place with the advent of streaming technology. For these files to be displayed by using this technology is essential to have a streaming server. This has the role to manage all requests made by one or more client machines, thus giving the requested service. These servers have different problems, one is the storage and retrieval of means, which can cause collisions on the network and the number of connections that can be done decreases.

This paper presents a component called Data Manager for Distributed Streaming Server, based on the ADMS architecture (Architecture Adaptive Distributed Multimedia Server) that runs on a cluster. It is capable of storing and retrieving media files without overloading the network, providing an increase in the number of connections by clients. This storage is done in a distributed way by all components that are located in the cluster. For that, it is taken into account different features that each server has, selecting one of them using a heuristic search method. It also provides all the necessary information to retrieve all the stored data. This system is fully implemented with free technology.

Keywords: Storage, Retrieval, Distributed Streaming Server, Streaming.

Índice de Figuras

Índice de Tablas

Índice General

INTRODUCCIÓN

A lo largo de la historia de la humanidad, el hombre ha realizado diferentes transformaciones en la sociedad gracias a su ingeniosidad y creatividad, trayendo consigo un mejoramiento en la civilización. Hoy día, la Internet y la televisión cobran mucha importancia a escala mundial, pues han sido gestoras de grandes transformaciones sociales, económicas y tecnológicas, logrando así, un lugar meritorio en la preferencia de la sociedad.

Durante muchos años se ha ido transformando la forma de realizar televisión. La misma ha pasado por distintas etapas hasta llegar a la más actual, la Televisión Digital (TVD), que es la transmisión de imágenes en movimiento y su sonido asociado (televisión), mediante una señal digital (codificación binaria) y a través de una red.

Con todo el desarrollo alcanzado por las Tecnologías de la Información y las Comunicaciones (TIC), surge el nuevo modo de realizar transmisiones audiovisuales, conocido como Televisión por IP (IPTV), proporcionando gran ancho de banda y así, el mejoramiento de la distribución de señales de televisión y/o vídeo. Esto fue posible gracias a la creación de la red de redes, más conocida como Internet, ya que la idea de poder contar con una red de ordenadores que permita una comunicación general entre usuarios de distintas computadoras, y el gran desarrollo tecnológico propulsado paralelamente, produjo la fusión de la infraestructura de las redes existentes con los sistemas de telecomunicaciones. A diferencia de la televisión digital por satélite, televisión digital por cable, o la televisión digital terrestre, el proveedor solo emitirá sus contenidos solo cuando el cliente los solicite explícitamente. La clave está en la personalización del contenido para cada cliente de forma individual de manera que el usuario podrá seleccionar los contenidos que desea ver o descargar para almacenar en el receptor y de esta manera poder visualizarlos tantas veces como desee. Se trata, en definitiva, de un servicio que hace posible una televisión “a la carta” en el que cada usuario puede ver el programa o película que desea y en el momento que quiera, lo que se traduce como video bajo demanda. También presenta una mayor cantidad de contenidos, pues cuenta con un sistema de almacenamiento que permite tener acumulada buena parte de la información que se le brinda al usuario.

Este sistema puede satisfacer muchos servicios, gracias a la aparición y aplicación de la tecnología streaming, permitiendo así el flujo de contenido audiovisual desde un servidor hasta los receptores a través de la red. Antes de que existiera esta tecnología, esos archivos tenían que ser necesariamente

descargados directamente al disco duro de la PC del cliente, haciendo muy engorroso el proceso de visualización, pues los ficheros de audio y/o video son en general muy grandes, por lo que esa operación se hacía muy lenta, sin embargo, con esa tecnología, el tiempo de espera es menor. Por otra parte, el servidor es el elemento principal de la tecnología streaming, condicionalmente se encarga de administrar y controlar la difusión de contenidos a los usuarios. Su valor radica en la capacidad de atender clientes simultáneamente vinculados a su visualización y reducir el grado de exposición de los elementos primarios, lo que se traduce en la cantidad de suscriptores que pueden estar haciendo uso del sistema junto a la seguridad que esta posee para las conexiones que puedan establecerse. En el mundo existen diversos servidores streaming, en su mayoría son privativos, pero existen otros que han sido desarrollados en software libre y con política de Open Source; ejemplos de servidores streaming, dígase privativos como libres se pueden encontrar: Windows Media Encoder, Darwin Streaming Server, Helix Server, Ice Cast, Real Server, VLC Media Player, entre otros.

Estos servidores en su mayoría presentan un semejante funcionamiento interno, por ende, poseen generalmente tres subsistemas, el de Control, el de Almacenamiento y el de Comunicación (Vargas Colores, 2008). El primer subsistema se encarga de recibir peticiones de clientes y ordenar las acciones para poder ser atendidas, otras de sus funciones son las de administración de estadísticas y procesos de optimización para hacer eficiente el sistema. En el caso del segundo subsistema, se encarga de trabajar en el almacenamiento y recuperación de las medias en los dispositivos de almacenamiento. El tercero, planifica la inyección de los contenidos multimedia en la red y optimiza los recursos de ancho de banda en la red y el servidor.

Algunas de las desventajas que poseen estos Servidores de Streaming, tiene que ver con el segundo subsistema que presentan, el de almacenamiento y recuperación de los datos. Una de ellas es que necesitan de gran capacidad en su sistema de almacenamiento, pues los archivos de multimedia en lo general, contienen un gran volumen de información, y esto hace que cuente con un valor significativo en su tamaño. Esto trae como dificultad, que mientras esos ficheros son almacenados en un mismo dispositivo, y al mismo tiempo, varios usuarios realizan peticiones a ese archivo, ocasiona que el número de accesos concurrentes del disco aumenten, provocando que la cantidad de usuarios que puedan acceder al mismo, se limite a la capacidad de entrega de datos del dispositivo. A este problema, varios Servidores Streaming Distribuidos consiguieron dar una solución parcial, la misma, es agregando cierta cantidad de discos para aumentar la capacidad de almacenamiento, distribuyendo las medias en estos, o

realizando varias particiones internas. Pero esta solución no es la más eficiente, puesto que cuando exista una gran concurrencia a dicho servidor, ocurrirá una congestión en la red, imposibilitando brindar ese servicio a los clientes, es por ello que surge el siguiente **problema de la investigación:** ¿Cómo mejorar el almacenamiento de medias en un servidor streaming distribuido, para obtener una recuperación más eficiente de las mismas?

Se define como **objeto de estudio:** los procesos de almacenamiento y recuperación de medias en servidores streaming, y como **campo de acción:** los procesos de almacenamiento y recuperación de medias, en Servidores Streaming Distribuidos.

Para responder al problema de investigación se precisa como **objetivo de la investigación:** Desarrollar un componente que permita almacenar y recopilar de forma eficiente, las medias que se empleen en un Servidor Streaming Distribuidos.

Se formula la siguiente **idea a defender:** Si se crea un componente, que mejore eficientemente el almacenamiento y recuperación de medias en Servidores Streaming Distribuidos, permitirá que el rendimiento en la transmisión de estas, se efectúe de modo fructífero.

Para conseguir el objetivo planteado se proponen las siguientes **tareas de investigación**:

1. Caracterizar los procesos relacionados con los Manejadores de Datos para Servidores de Streaming Distribuidos.
2. Identificar las distintas soluciones existentes de Manejadores de Datos para Servidores de Streaming Distribuidos.
3. Comparar las distintas soluciones encontradas para obtener las bases necesarias para el desarrollo de un Manejador de Datos para Servidores de Streaming Distribuidos.
4. Identificar las funcionalidades para los Manejadores de Datos para Servidores de Streaming Distribuidos.
5. Seleccionar y argumentar la Metodología de Desarrollo de Software a usar en el proceso.
6. Realizar análisis, diseño e implementación del componente Manejador de Datos para Servidores de Streaming Distribuidos.
7. Realizar pruebas al sistema.

Esperando obtener como posible resultado: Un componente capaz de hacer más eficiente el almacenamiento y recuperación de los datos en un Servidor de Streaming Distribuido.
Para la realización de las tareas propuestas se utilizaron diferentes métodos y técnicas para la búsqueda y procesamiento de la información.

Métodos teóricos:

- Analítico-Sintético: Este método permite realizar un análisis con la bibliografía adecuada sobre los Servidores Streaming Distribuidos y dentro de ellos los Manejadores de Datos, contribuyendo así, a concebir mejor la situación, para luego poder dar solución al problema científico planteado.
- Histórico - Lógico: Mediante el cual se analizará un estudio sobre los precedentes históricos de los Manejadores de Datos en Servidores Streaming Distribuidos y los diferentes tipos de almacenamientos, permitiendo conocer la trascendencia y el desarrollo alcanzado hasta la actualidad.
- Modelación: Se utiliza para la modelación de diagramas, representando los diagramas generados durante el proceso de desarrollo del software y propiciando un mejor entendimiento de la solución a implementar.

Métodos empíricos:

- Observación: Este método desempeña un papel muy importante, porque permite obtener conocimientos acerca del comportamiento de los Manejadores de Datos en los Servidores Streaming Distribuidos tal y como este se desarrolla en lo práctico; es una manera de acceder a la información directa e inmediata sobre los mismos.
- Entrevista: Se ha utilizado este método para realizar entrevistas a personas capacitadas en temas relacionados con los Servidores Streaming y en los procesos de almacenamiento de archivos audiovisuales en diferentes dispositivos.

CAPÍTULO 1. Fundamentación teórica

1.1. Introducción

En el presente capítulo se brinda una visión general de los aspectos teóricos relacionados con Manejadores de Datos en los Servidores Streaming Distribuidos, así como una descripción de los principales conceptos y tecnologías que están asociados a este componente.

1.2. Conceptos asociados al dominio del problema

En este epígrafe se estarán abordando diferentes conceptos que están relacionados con la investigación. Estos tendrán como propósito ayudar a entender y elaborar una base de conocimientos sobre el tema.

1.2.1. Streaming

Se entiende por streaming a la transferencia de un modo fluido de contenidos audiovisuales (audio/video) por la red, desde un servidor hasta llegar a sus clientes de forma eficiente. Es una tecnología que se utiliza para aligerar la descarga de archivos multimedia en la red, permitiendo escuchar y visualizar los ficheros mientras se están descargando (Torres, 2009). La no utilización de esta tecnología hace muy engorroso el proceso de mostrar algún contenido de multimedia en la red, pues se debería primeramente descargar completamente para la PC del cliente, trayendo consigo un alto costo en el tiempo y atentando contra la capacidad de memoria con que cuenta el ordenador del usuario. En la figura 1 se ilustran las formas de transferencia de archivos audiovisuales por la red.

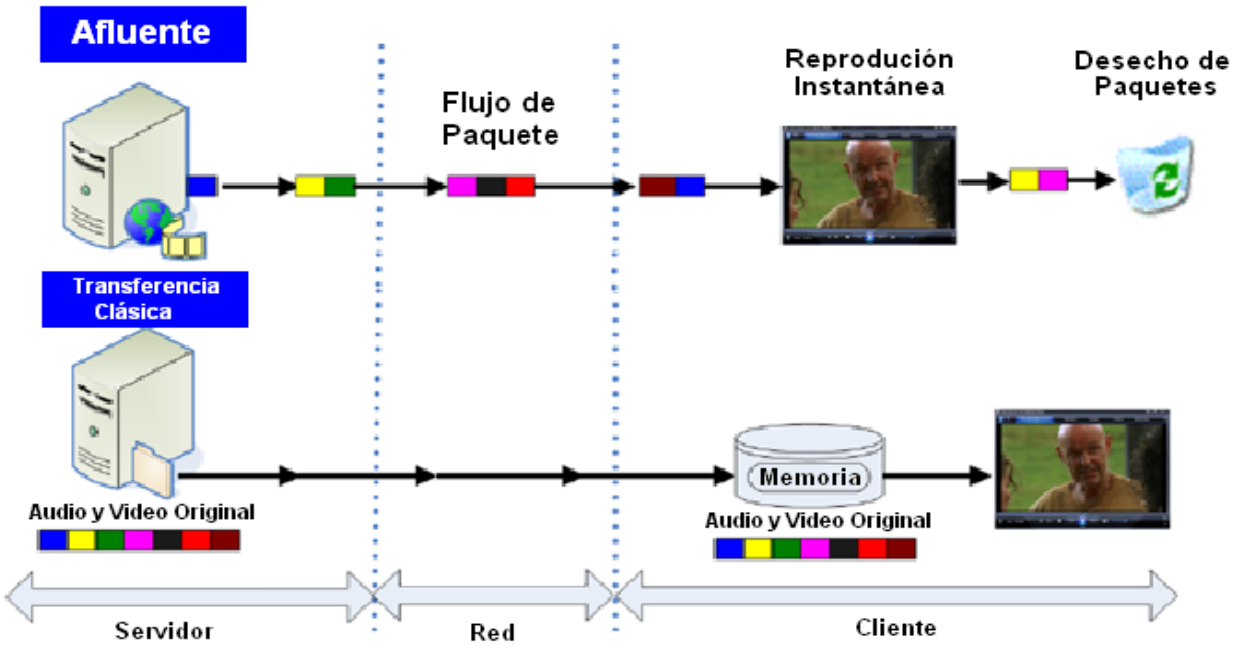

Figura 1 Modo de transferencia clásica y afluente (streaming) (Vargas Colores, 2008)

La utilización de la tecnología Streaming ha ayudado a muchas aplicaciones que utilizan materiales audiovisuales, como son: bibliotecas digitales, videos conferencias, aprendizaje a distancia, la televisión o sistemas de video sobre demanda.

Esta tecnología se basa fundamental en la mezcla de tres técnicas (Vargas Colores, 2008), las cuales se describen a continuación:

- **Afluente verdadero (True streaming):** Esta técnica hace posible que la señal de video llegue en tiempo real y sea vista inmediatamente, por ejemplo, dos minutos de video, tardan dos minutos en transmitirse.
- **Reproducción y descarga (Download & play):** Funciona enviando archivos comprimidos de audio y video, con anticipación a la reproducción ejecutada por el cliente. Así por ejemplo, dos minutos de video pueden ser descargados en un tiempo de diez segundos.
- **Reproducción progresiva y descarga (Download & play progressive):** Es una tecnología híbrida entre las técnicas antes mencionadas pues tiene los beneficios de ambas. Con esta técnica, el video es seccionado en pequeños archivos que son descargados mientras el cliente despliega los archivos que ya fueron descargados.

La tecnología Streaming posee diferentes formas de clasificarse, dependiendo fundamentalmente del servicio que se le ofrezca a los usuarios, como: la capacidad de iteración, elección del contenido, entre otros. El servicio es muy importante, pues a medida que aumenta la interactividad con los usuarios, a su vez se van complejizando los sistemas de difusión afluentes (streaming). Estos servicios son (Vargas Colores, 2008):

- **Difusión en vivo (Live):** Permite que los clientes se conecten y vean la información que se está emitiendo en ese instante, este servicio tiene una limitante en la interactividad en la reproducción, porque el cliente no puede modificar el contenido que se le transmite. Este puede clasificarse en dos cuestiones fundamentales. La primera es según el origen de las señales de audio y video, como se muestra en la figura 2.

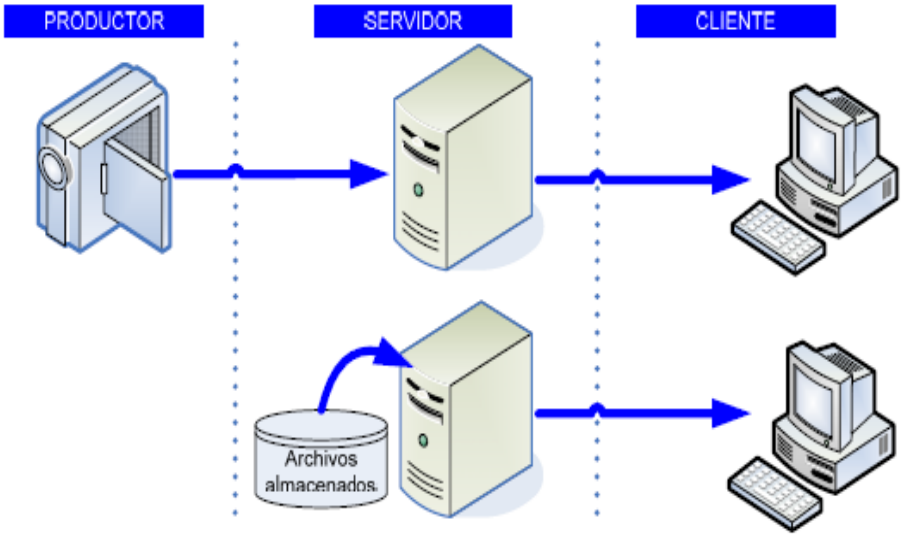

Figura 2 Clasificación de señales en vivo según su fuente (Vargas Colores, 2008).

La segunda es mediante el tipo de transmisión que utilice, las cuales son:

- ✓ **Transmisión unicast:** Consiste en la transmisión dedicada a cada cliente, es decir, a cada cliente se envían distintos archivos afluentes a cada uno, dividiendo el ancho de banda entre ellos.
- ✓ **Transmisión multicast:** Es la transmisión por igual a un grupo de clientes, enviando el fichero por la red a todos los usuarios que pertenecen al grupo y que han hecho la petición de

visualización. De esta forma, un afluente multicast siempre tiene por lo menos un destinatario y nunca se desaprovecha el ancho de banda.

- ✓ **Transmisión broadcast:** Radica en la transmisión por igual a todos los usuarios, enviando por la red el archivo afluente para todos los que pertenezcan a ella y quieran adquirirlo. Esto trae consigo que siempre se utilice un ancho de banda de red aunque el cliente no esté consumiendo de ese servicio. Es por ello que esta técnica se utiliza principalmente en la visualización de archivos con gran demanda.

- **Difusión bajo demanda (on-demand):** Este tipo de servicio permite que el usuario solicite el envió de la información cuando desee, siendo esta información personalizada para cada cliente, pues para cualquier instante de tiempo, se le puede realizar peticiones del contenido que se muestra, por ejemplo, realizarle pausas, saltos hacia adelante, saltos hacia atrás, entre otros.

1.2.2. Servidores streaming

Un servidor en la rama de la Informática, se puede denominar como un sistema informático (ordenador) que presta ciertos servicios y recursos (de comunicación, aplicaciones, ficheros, etc.) a otros ordenadores (denominados clientes), los cuales están conectados en red a él (HispaNetwork Publicidad y Servicios, S.L., 2004).

En el mundo existen diferentes tipos de servidores, dependiendo del uso que estos puedan brindar, ejemplo de ellos son los servidores web, de correos, de fax, de archivos, de streaming, entre otros.

Específicamente los Servidores de Streaming son los encargados de manejar conexiones individuales provenientes de máquinas clientes, que realicen una petición de visualización de archivos audiovisuales, el cual envía un flujo continuo de bits, permitiendo así, ser visualizado en el reproductor del cliente.

En la implementación de estos servidores se pueden utilizar diferentes tipos de arquitecturas, entre las más conocidas se encuentran las siguientes:

- Arquitecturas centralizadas.
- Arquitecturas de servidores independientes.
- Arquitecturas basadas en servidores proxy.
- Arquitecturas distribuidas.

Servidor Streaming Distribuido

Varios autores se refieren a estos sistemas distribuidos, uno de ellos expresa: "*Un sistema distribuido es una colección de computadoras autónomas que trabajan conjuntamente para dar la apariencia de un sistema coherente único*" (Tanenbaum, y otros, 2002). En otra bibliografía se plantea que: "*Un sistema distribuido es aquel en el que los componentes de hardware o software, localizados en computadores conectadas mediante red, comunican y coordinan sus acciones solo mediante paso de mensajes*" (Coulouris, y otros, 2001). También se define "*Un sistema distribuido es una colección de dispositivos individuales de computación que pueden comunicarse unos con otros*" (Attiya, y otros, 2004). Sin embargo, un concepto que resulta más acertado es el siguiente:

"Un sistema distribuido es un sistema compuesto de varias computadoras que se comunican mediante una red, almacenando procesos que utilizan un conjunto de protocolos distribuidos común para soportar la ejecución coherente de actividades distribuidas." (Veríssimo, y otros, 2001).

Las arquitecturas distribuidas, son unas de las tendencias actuales en el diseño de Servidores Streaming. El manejo de las peticiones y de los contenidos que se distribuyen entre todos los componentes del sistema, permiten que los distintos nodos de servicio tengan que colaborar entre sí para poder atender a todos los clientes.

Existen diferentes categorías de sistemas distribuidos en función de si existe o no, un nodo maestro encargado de centralizar el manejo del sistema y mantener una copia completa de los contenidos del sistema. Una propuesta en este sentido, es la política de servicio de encadenamiento (chaining), la cual utiliza los contenidos de los buffer internos de los clientes para servir peticiones de otros clientes del mismo contenido (Sheu, y otros, 1997). En la actualidad, han surgido propuestas que intentan utilizar el concepto de punto-punto (peer-to-peer) utilizándolo para la distribución de archivos en Internet, creando así, un sistema de almacenamiento distribuido de contenidos (Tran, y otros, 2004).

También existen diferentes tipos de arquitectura para estos servidores, ejemplo de ello se tiene la arquitectura ADMS (Adaptive Distributed Multimedia Server). La cual está compuesta por varios componentes, que realizan funciones distintas, pero a su vez tienen una estrecha relación entre ellos, ya que juntos forman un servidor streaming. Estos componentes son: Manejador de Clúster, Distribuidor de Datos, Colector de Datos y el Manejador de Datos (González Jiménez, 2011).

En el presente trabajo de diploma se pretende desarrollar un componente Manejador de Datos para un servidor de streaming distribuido, basado en esta arquitectura.

1.2.3. Manejador de datos

El Manejador de Datos, posee gran importancia en la arquitectura ADMS, pues es responsable de distribuir en los diferentes servidores del clúster, los archivos de media que luego se transmitirán. Esta distribución se hace de forma organizada, teniendo en cuenta una serie de parámetros que contribuyen a un mejor funcionamiento del sistema.

Dentro de cada servidor del clúster, va a existir un componente, que permita almacenar y recuperar los datos que le han sido transferidos por el Manejador de Datos Central. Esto permite hacer uso de un buen balance de carga, brindando la posibilidad de poder atender a una mayor cantidad de usuarios sin que la red y la capacidad de carga del dispositivo de almacenamiento puedan congestionarse.

Este componente, llamado Manejador de Datos, solo va a almacenar una porción del streaming, principalmente la dirección en memoria que posee este, donde lo organiza de forma tal, que a la hora de recuperarlos sea de forma eficiente (Tusch, y otros, 2004). Este módulo también debe ser capaz de poder gestionar la información de cada servidor, con los cuales va a trabajar. Este debe tener el control de cada uno de ellos para poder realizar todas las funcionalidades que debe realizar, por ejemplo, conocer la capacidad de almacenamiento, así como también su rendimiento en memoria RAM y en cantidad de conexiones.

1.3. Soluciones existentes

Los Servidores Streaming deben satisfacer una serie de requisitos derivados de su funcionalidad, dentro de ellos se encuentra que deben contar con una gran capacidad de almacenamiento, pues en la mayoría de sus aplicaciones, el sistema deberá ser capaz de ofertar al usuario una amplia gama de objetos multimedia, que serán almacenados según la arquitectura que utiliza el servidor.

Los desafíos para el diseño de un sistema de almacenamiento multimedia son cumplir con una alta capacidad de entrega y tolerancia a fallos (Rijo Sciara, 2004).

1.3.1. Darwin Streaming Server

Este servidor está basado en una arquitectura centralizada, la cual realiza su almacenamiento de forma tal, que los archivos multimedia estarán en una misma carpeta, dándole la posibilidad que los servidores vayan directamente a esa dirección y comiencen a mandar el flujo de datos a los clientes. Esta manera tiene una gran dificultad y es que como se guarda en un mismo lugar, el número de

accesos concurrentes a este archivo se limita a la capacidad de entrega de datos de ese dispositivo, limitando la cantidad de clientes que pueden hacer uso de ese servicio. Otra de las desventajas se debe, a un desbordamiento de buffer en el módulo QTSSReflector, cuando la solicitud ANNOUNCE manipulada con un valor muy grande, que es enviada, provocando un desbordamiento de entero o "integer overflow", que es cuando una variable definida como entera, sobrepasa los valores asignados.
Otra falla es provocada en la utilidad MP3Broadcaster, incluida con el servidor. Esta es provocada cuando el usuario intente ejecutar un MP3 modificado maliciosamente, recibirá un error de "Segmentation fault (core dumped)". Este error se produce cuando un proceso ha sido abortado por acceder a una dirección de memoria ilegal. Esto suele estar relacionado con un problema de uso incorrecto de algún puntero y de la memoria.

1.3.2. Servidor Streaming Distribuido Flumotion

Este servidor de streaming fue creado en el año 2006, el cual está escrito en el lenguaje de programación Python. Su implementación está basada en dos frameworks, gstreaming y twisted. Es una solución distribuida y de código abierto. Posee un sistema de administración de ficheros con un diseño amigable y accesible. Esto permite un aumento notable en cuanto a capacidad de entrega, teniendo en cuenta un equilibrio entre el balance de carga y la latencia, pues ambas son objetivos de conflicto. También proporciona gran escalabilidad con ese diseño distribuido. Además, cuenta con una forma peculiar de almacenar sus datos, pues da solución a muchos de los problemas que poseen los servidores centralizados. Su diseño distribuido permite repartir la carga del proceso de streaming entre diferentes máquinas, lo cual facilita la manipulación de los streams y la implementación de soluciones avanzadas sin comprometer la calidad de la emisión. Por tal motivo hace un almacenamiento centralizado en la PC donde se encuentra el componente que realiza el almacenamiento. En la instalación de su versión libre, se detalla que el mismo, puede soportar hasta 1000 conexiones por cada componente de almacenamiento que el software brinde, donde se puede llegar a más conexiones en dependencia del hardware que posea. Esta característica trae un problema significativo, pues cuando se sobrepase de esa cifra de conexiones posibles, hace que se cree un cuello de botella en el servidor, provocando que se retrase o no se pueda acceder al archivo que se desee transmitir.

1.3.3. VoDKA Server

Este servidor presenta una arquitectura totalmente distribuida, permitiendo un crecimiento y escalabilidad sin límite. La característica anteriormente mencionada, ofrece un gran rendimiento en cuanto a la cantidad de conexión.

Esta arquitectura VoDKA (Video on Demand Kernel Architecture), utiliza un sistema de almacenamiento por niveles, contando con tres niveles (terciario, secundario y primario) (Veríssimo, y otros, 2001), en la figura 3 se representa dicha arquitectura.

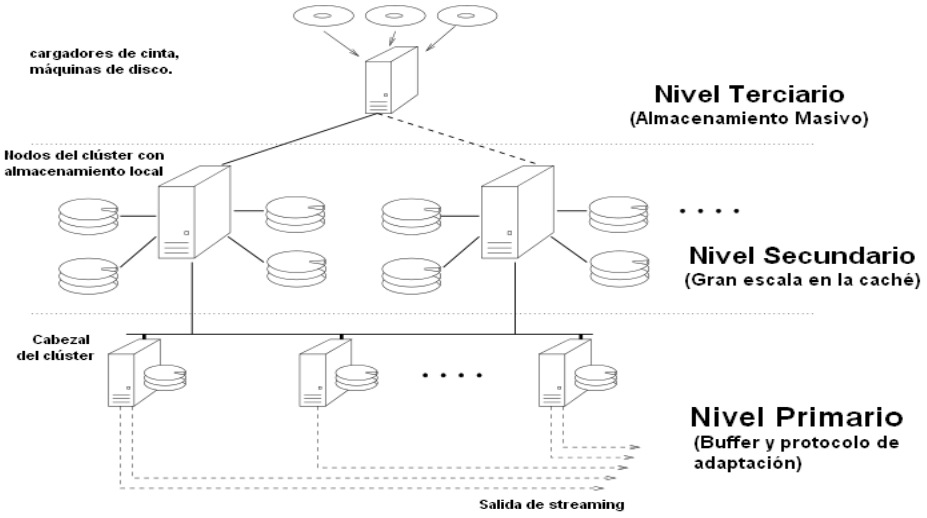

Figura 3 Estructura jerárquica de VoDKA (Veríssimo, y otros, 2001).

Otra de las características que presenta la arquitectura VoDKA son sus bajos costes de operación, ya que el sistema está altamente automatizado y es muy flexible. Esta a su vez, da la posibilidad de utilizar varias PC, lo que permite que la pérdida de calidad del servicio sea muy baja. La desventaja que posee, es que utiliza un sistema de almacenamiento interno distinto en cada nivel, trayendo consigo que en algunas ocasiones ocurran fallos en el proceso de almacenamiento y recuperación en los datos almacenados. Este problema a su vez, hace que se pierda tiempo en la transmisión de datos de un nivel a otro.

1.4. Conclusiones parciales

En este capítulo se hace un estudio de la situación actual del dominio del problema. Se realizó, una comparación entre las distintas soluciones, generando la conformación de un resumen con las diferentes formas de almacenar y recuperar información, dentro de las diversas arquitecturas de Servidores de Streaming. Se pudo comprobar que dentro de las existentes, las arquitecturas distribuidas poseen un mejor rendimiento en cuanto a accesos concurrentes entre otros parámetros. Entre ellas, la que presenta las características más adecuadas para el flujo de video por la red es la ADMS, pues hace una buena distribución de los recursos con que cuenta el clúster, permitiendo así un mejor rendimiento en la transmisión de los datos.

CAPÍTULO 2. Tendencias y tecnologías actuales a utilizar

2.1 Introducción

Para realizar una aplicación informática, es importante conocer cuáles son las mejores herramientas y tecnologías que brindan las posibilidades adecuadas para desarrollar soluciones efectivas a las necesidades.

En este capítulo se estarán abordando las diferentes tendencias y tecnologías actuales que son utilizadas para el desarrollo de componentes. También se hace un estudio de las metodologías de desarrollo de software que garantizan un mejor entendimiento del flujo de trabajo.

2.2 Metodologías de desarrollo de software

Para los desarrolladores de software es necesario saber cómo organizar las actividades, cuáles técnicas, herramientas y soporte utilizar y a su vez, poder establecer cuáles artefactos crear en cada momento del proceso. Por ello, resulta necesario contar con una herramienta que dirija las actividades y permita convertir los requisitos en un producto de software.

Las metodologías son un conjunto de métodos o reglas, que por una parte sirven de guía para realizar los trabajos que van dando forma al desarrollo de software y que por otra, obligan a la dirección del proyecto y a los componentes de los equipos a realizar ciertas comprobaciones sistemáticas de modo que el resultado final, no presente incoherencias y esté dirigido a un objetivo claro y prefijado. Por la variedad de metodologías existentes, se podrían clasificar en dos grupos, las metodologías pesadas orientadas al control de los procesos, estableciendo rigurosamente las actividades a desarrollar y las ágiles orientadas a interactuar con el cliente y al desarrollo incremental del software, mostrando versiones parcialmente funcionales al cliente en intervalos cortos de tiempo.

En el caso de las metodologías ágiles, una de las más conocidas y utilizadas es la Programación Extrema, en el caso de las pesadas, una muy reconocida es la Metodología RUP (Rational Unified Process).

2.2.1 Programación Extrema (Extreme Programming, XP)

Esta metodología ágil es una de las más utilizadas en la actualidad. Sus comienzos se remontan hacia marzo del 1996, cuando fue desarrollada por Kent Beck. Esta tiene como diferencia a las metodologías tradicionales, que principalmente ponen más énfasis en la adaptabilidad que en la previsibilidad.

Esta cuenta con diferentes características fundamentales (Pérez Vázquez, y otros, 2010):

- ✓ Desarrollo iterativo e incremental: son pequeñas mejoras, unas tras otras.
- ✓ Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
- ✓ Programación por parejas: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se supone que la mayor calidad del código escrito de esta manera -el código es revisado y discutido mientras se escribe- es más importante que la posible pérdida de productividad inmediata.
- ✓ Frecuente interacción del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- ✓ Corrección de todos los errores antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
- ✓ Propiedad del código compartida: en lugar de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores sean detectados.
- ✓ Simplicidad en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir una funcionalidad si es necesario. La programación extrema plantea que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

Lo fundamental en este tipo de metodología es:

- La comunicación, entre los usuarios y los desarrolladores.
- La simplicidad, al desarrollar y codificar los módulos del sistema.
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.

2.2.2 El Proceso Unificado de Desarrollo de Software (RUP)

El Proceso Unificado de Desarrollo de Software (en inglés Rational Unified Process, RUP), es una metodología robusta que cuenta ya con más de 30 años de experiencia. Dentro de la ingeniería de software, es un proceso planteado por Kruchten en 1996, cuyo objetivo es producir software de alta calidad, es decir, que cumpla con los requerimientos de los usuarios dentro de la planificación y presupuesto establecido.

"*RUP es un proceso de desarrollo de Software, sin embargo, es más que un simple proceso; es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto*" (BOOCH, 1999).

Esta metodología cubre el ciclo de vida y desarrollo de software el cual se caracteriza por ser:

1. Dirigido por los casos de uso: Teniendo en cuenta que la razón de ser de un sistema es brindar servicios a los usuarios, RUP define Caso de Uso como el conjunto de acciones que debe realizar un sistema para dar un resultado de valor a un determinado usuario y los utiliza tanto para especificar los requisitos funcionales del sistema, como para guiar todos los demás pasos de su desarrollo, dígase diseño, implementación y prueba.
2. Centrado en la arquitectura: La arquitectura es una vista del diseño completo, con las características más importantes, dejando a un lado los detalles. Esta no solo incluye las necesidades de los usuarios e inversores, sino también otros aspectos técnicos como el hardware, sistema operativo, sistema de gestión de base de datos, protocolos de red; con los que debe coexistir el sistema. La arquitectura representa la forma del sistema, la cual va madurando en su interacción con los casos de uso hasta llegar a un equilibrio entre funcionalidad y características técnicas.
3. Iterativo e incremental: La alta complejidad de los sistemas actuales hace que sea factible dividir el proceso de desarrollo en varios mini-proyectos. Cada uno de estos mini-proyectos se les denomina iteración y pueden o no representar un incremento en el grado de terminación del producto completo. En cada iteración los desarrolladores seleccionan un grupo de casos de uso, los cuales se diseñan, implementan y prueban. La planificación de iteraciones hace que se reduzcan los riesgos de los costes de un solo incremento, mantener la motivación del equipo pues puede ver avances claros a corto plazo y que el desarrollo pueda adaptarse a los cambios en los requisitos.

Esta metodología presenta 4 etapas muy importantes que son: Inicio, Elaboración, Construcción y Transición, donde cada una de ellas está compuesta por una o varias iteraciones. Además de las fases, contiene flujos de trabajos, los mismos se dividen en Flujos de trabajo de desarrollo y Flujos de trabajo de soporte. En la figura 4 se muestra estas etapas por las que transita esta metodología.

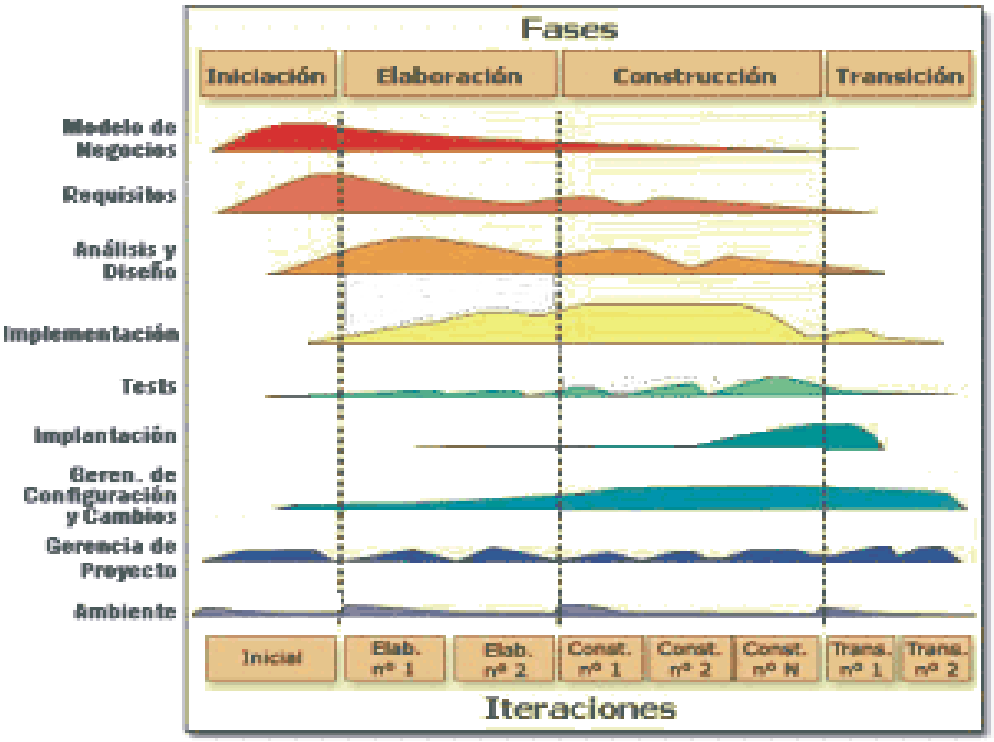

Figura 4 Ciclo de vida de RUP (Progress, 2009).

2.2.3 Fundamentación de la selección de la metodología de desarrollo de software

El Proyecto DESCOMTEC utiliza la metodología antes mencionada, llamada RUP. Esta sin dudas, se adapta a las condiciones del mismo y ofrece una mayor organización en el desarrollo del Manejador de Datos. Además, puede ser utilizada para proyectos que sean de gran envergadura, pues presenta una alta flexibilidad, lo que permite que se puedan realizar diferentes cambios en distintos ambientes de desarrollos. También es una metodología para entornos de desarrollo orientado a objetos. Se tiene en cuenta también la selección de esta metodología, pues los integrantes del proyecto tienen al menos un conocimiento básico sobre el desarrollo de aplicaciones con la misma, no siendo así con las otras existentes. Esto proporciona una ventaja respecto al tiempo de desarrollo del sistema.

2.3 El Lenguaje Unificado de Modelado (UML) como soporte de la modelación de la solución propuesta

UML, Unified Modeling Language por sus siglas en inglés, es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software (Santos Sanabria, y otros, 2010). Es utilizado en varias metodologías, presenta una estrecha relación con la que será empleada en este trabajo de diploma: el Proceso Unificado de Modelado (RUP), ya que este hace uso de todos los diagramas propuestos por este lenguaje.

Dicho lenguaje es una notación unificada con la que se permite lograr un entendimiento que propicie el intercambio entre los usuarios y los desarrolladores. Se ha convertido en un estándar de la industria del software, debido a que fue impulsado por los autores de los tres métodos más usados de orientación a objetos Grady Booch, Ivar Jacobson y Jim Rumbaugh. El UML estándar está compuesto por tres partes: bloques de construcción (tales como clases, objetos, mensajes), relaciones entre los bloques (tales como asociación, generalización) y diagramas (por ejemplo, diagrama de actividad) (H ISTCHFELD, y otros).

2.4 Herramientas de modelado visual

Las herramientas de modelado visual más conocidas como herramientas CASE (Computer Aided Software Engineering), son utilizadas para realizar la representación de un producto de software en su totalidad, mediante diagramas que se van desarrollando durante los diferentes ciclos de vida del proyecto y facilitan el desarrollo de un proceso. Existen diversos software que permiten realizar esas actividades, en la actualidad dos de las más poderosas herramientas de modelado visual para el análisis y diseños de sistemas basados en objetos son: Visual Paradigm y Rational Rose.

2.4.1 Rational Rose

Esta herramienta propone la utilización de cuatro tipos de modelos para realizar un diseño del sistema, utilizando una vista estática y otra dinámica de los modelos del sistema, uno lógico y otro físico. Permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software (Méndez Terrero, 2010). También hace posible especificar, analizar y diseñar el sistema antes de codificarlo.

Brinda la posibilidad de generar código en distintos lenguajes de programación a partir de un diseño en UML y proporciona mecanismos para realizar la denominada ingeniería inversa, o sea, la realización de los diagramas una vez conocido el código. Soporta de forma completa la especificación del UML (Pérez Vázquez, y otros, 2010).

2.4.2 Visual Paradigm 6.4 Enterprise Edition

Visual Paradigm para UML, es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML, permite una más rápida construcción de aplicaciones de calidad, mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML (Free Download Manager, 2007).

Esta herramienta presenta diversas características, una de ellas es ser multiplataforma y gratis en su edición Community, otras características que permiten que sea muy usada en la actualidad son (Valdez Altamirano, 2006):

- Se pueden realizar distintos tipos de diagramas, por ejemplo: Diagramas de clases, de Casos de Uso, de Secuencia y de Componentes.
- Integración con diversos IDE's como son: NetBeans, JDeveloper, Eclipse, JBuilder.
- Permite realizar ingeniería inversa en los lenguajes Java, C++, PHP, XML, Ada y Python.
- Exportación en imágenes de los diagramas que son elaborados, con formato jpg, png y svg (w3g estándar).
- Genera la documentación del sistema en formato PDF, HTML y Microsoft Word.
- Genera el código de aplicación según el modelado de la aplicación.
- Genera la documentación de lo que se está modelando.
- Modela todos los diagramas de UML.

2.4.3 Selección de la herramienta de modelado

Se decidió utilizar Visual Paradigm (VP) como herramienta de modelado, pues este presenta características similares que el Rational Rose, aunque posee otras que lo hacen ser mejor. Una de

ellas es que cuenta con una licencia comercial gratuita en su edición Community, mientras que Rational Rose es privada. Otra característica fundamental en que resalta, es que tiene una alta generación e inversión de código, así como una mayor integración con IDEs de desarrollo. El VP no está diseñado hacia ninguna metodología de desarrollo en específico, pues se adapta a la mayoría de las metodologías de desarrollo existentes, mientras que Rational Rose no lo hace. Luego de haber seleccionado la herramienta de modelado para el sistema se puede pasar a la selección del lenguaje con el que se pretender implementar la aplicación.

2.5 Lenguajes de programación

Los lenguajes de programación permiten crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos hardware y software existentes.

En los lenguajes de programación existen diversos paradigmas que hacen que se diferencien como lo son: Programación Estructurada, Programación Modular, Programación Orientadas a Eventos, Programación Funcional y Programación Orientada a Objetos. En este trabajo se utilizará un lenguaje que presente este último tipo de paradigma.

2.5.1 Java

Java es un lenguaje orientado a objetos, donde se agrupan en estructuras encapsuladas tanto los datos de los objetos como los métodos (o funciones) que manipulan los mismos. Además posee características como la de ser distribuido, interpretado y compilado a la vez, robusto, seguro, portable, de alto rendimiento y multi-hilo (Gonzalo Álvarez, 1999). Está basado en características que presenta C/C++, lo que este no cuenta con sobrecarga de operadores, herencia múltiple y aritmética de punteros. Posee recolector de basura y a su vez es interpretado con una pila a base de máquina virtual.

2.5.2 C#

El lenguaje C Sharp (C#) utiliza el paradigma de programación orientada a objetos, desarrollado y estandarizado por Microsoft. Tiene una sintaxis básica derivada de C/C++, aunque utiliza otros elementos del lenguaje Java.

Este lenguaje es simple, moderno y seguro. El código C # se compila como código administrado, lo que significa que utiliza los beneficios de los servicios de Common Language Runtime. Estos servicios incluyen, la recogida de basura, la mejora de la seguridad, el apoyo y la mejora de versiones (Martínez Pérez, 2010).

2.5.3 C++

C++ es una versión ampliada del lenguaje C, incluyendo todo lo que forma parte de C y añade soporte para la programación orientada a objetos. Este posee otras características que lo hacen ser un lenguaje muy utilizado y potente en el mundo como son (Santos Sanabria, y otros, 2010):

- Permite el uso de plantillas o programación genérica.
- Permite redefinir los operadores o sobrecargarlos como también se conoce.
- Permite la identificación de tipos en tiempo de ejecución.
- Permite manipular convenientemente diversas estructuras de datos y operaciones.
- Su código se puede compilar en diversas plataformas.

2.5.4 Fundamentación del lenguaje propuesto

Se seleccionó el lenguaje de C++ por ser muy rápido y potente en el desarrollo de aplicaciones. Al comparar código C++ en tiempo de ejecución con los demás lenguajes, este los supera ampliamente porque posee un tratamiento de memoria diferente que el lenguaje Java y C #, trayendo consigo numerosas ventajas en el funcionamiento de la aplicación.

Además, potencia el manejo directo de interrupciones o la manipulación directa de hardware, es considerado la solución ideal en cuanto a la obtención de un mejor rendimiento del hardware. C++ es un lenguaje de plataforma de bajo nivel, con el que los desarrolladores alcanzan acceso a recursos primarios pues los otros no alcanzan esa funcionalidad. Otras de las características por el cual fue escogido, es que es un lenguaje de propósito general que se adapta a múltiples situaciones como por el ejemplo, la ayuda de algunas librerías en el tratamiento de imágenes, sonidos y videos.

2.6 Framework de desarrollo

Después de haber seleccionado el lenguaje de programación con que se va a trabajar, se hace importante seleccionar un framework. El mismo debe de contar con varias funcionalidades que hagan que el proceso de implementación sea rápido, flexible y eficiente.

Para este trabajo se seleccionó el framework Qt. Pues este es muy apropiado para proyectos de software de larga escala, tanto comerciales como de libre distribución. Qt cuenta con un conjunto de herramientas que tiene un enfoque más eficiente para facilitar la tarea de gestión de memoria para sus programadores: cuando un objeto es suprimido, todos los objetos dependientes se eliminan automáticamente. El enfoque de Qt no es interferir con la libertad del programador para borrar manualmente cuando lo deseen, sino lo que brinda es una ayuda.

Las ventajas de usar el framework de Qt son disímiles (Méndez Terrero, 2010):

- Es multiplataforma. El mismo código que se escribe en GNU/Linux, puede ser ejecutado fácilmente en Windows XP o en Mac OSX.
- Por las amplias características con que cuenta, se hace muy fácil construir aplicaciones para distintos usos.

El API de su biblioteca cuenta con diferentes librerías que facilitan el acceso a bases de datos mediante QSqlQuery y QSqlQueryModel, así como también para uso de XML con QXml. Realiza además gestión de hilos a través de QThread y proporciona un buen tratamiento de la memoria. Igualmente, ayuda en la manipulación de archivos con las librerías QFile y QDir. Se pueden utilizar las estructuras de datos tradicionales que posee el lenguaje C++.

2.7 Entornos de Desarrollo Integrado (IDE)

Un entorno de desarrollo integrado o IDE (del inglés Integrated Development Environment) son programas compuestos por un conjunto de herramientas encaminadas a la construcción de aplicaciones. Los IDE se encargan de gestionar una estación de trabajo sencilla y amigable para los lenguajes de programación y están integrados por partes fundamentales como es: el editor de código, compilador, depurador y un constructor de interfaz gráfica de usuario.

2.7.1 Eclipse

Eclipse es de una comunidad Open Source (código abierto), cuyos proyectos se centran en la construcción de una plataforma de desarrollo abierta formada por marcos extensibles, herramientas y rutinas para crear, desplegar y gestionar software en todo el ciclo de vida. El IDE es compatible con varios de los sistemas operativos más conocidos como son, todas las versiones de Windows a partir del 98, GNU Linux y Mac OS X (Martínez Pérez, 2010).

Esta plataforma no permite desarrollar en todo tipo de lenguaje, pero sí es posible añadir nuevas funcionalidades al editor, a través de nuevos módulos ('plugins'), para programar en otros lenguajes de programación además de Java como C/C++, PHP, Python (Méndez Terrero, 2010). Pero el lenguaje fundamental para desarrollar es Java.

Características principales:

- Permite utilizar estándares en la construcción de los proyectos.
- Fácil navegabilidad por el código.
- Mediante plugins se le pueden añadir varias funcionalidades.
- Herramientas de conocimiento como tipos de jerarquía.
- Editor de código con resaltado de sintaxis.
- Autocompletado de código, plegado y exploración de hipervínculos.
- Refactorización de código fuente y generación del mismo.
- Posee herramientas de depuración visual.

Principal Desventaja:

Su principal desventaja es que consume gran cantidad de recursos del ordenador lo cual no es despreciable cuando se trata de aplicaciones que utilizarán funcionalidades del sistema operativo que ralentizan la máquina.

2.7.2 Qt Creator

Qt Creator es un excelente IDE, creado por Trolltech para desarrollar aplicaciones en C++ de manera sencilla y rápida. Como su nombre lo indica, está desarrollado por los desarrolladores de Qt, donde este obtiene toda la ayuda que brinda el framework, es un IDE multiplataforma.

Este cuenta con las siguientes características:

- Editor avanzado para C++.
- Diseñador de formularios (GUI) integrado.
- Herramientas para la administración y construcción de proyectos.
- Completado automático.
- Depurador visual.

2.7.3 Fundamentación del IDE escogido

El IDE escogido para realizar el desarrollo de esta aplicación es el Qt Creator porque este posibilita muchas ventajas con respecto a los demás. Qt Creator se centra en proporcionar características que ayudan a los nuevos usuarios de Qt a desarrollar más rápido, y también aumentar la productividad de los desarrolladores con experiencia en el lenguaje de C++, el cual fue el escogido para la implementación de este sistema. Qt Creator también permite desarrollar aplicaciones con el avanzado editor de código para C++, el mismo, proporciona características de gran alcance para completar fragmentos y refactorización de código. También se puede acceder fácilmente a la documentación mediante el uso de la ayuda contextual integrada de Qt y una de las características fundamentales es que permite el completamiento para señales y slot.

2.8 Sistema Gestor de Bases de Datos

En el desarrollo de esta aplicación se hace necesario contar con una base de datos, es por esto necesaria la selección de un Sistema Gestor de Bases de Datos (SGBD). Estos sistemas son un software que proporciona servicios para la creación, el almacenamiento, el procesamiento y la consulta de la información almacenada en base de datos de forma segura y eficiente. Un SGBD actúa como un intermediario entre las aplicaciones y los datos (López Costa, y otros, 2010).

De forma general los SGBD son los encargados de manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante para una organización. En la actualidad existe una gran variedad de SGBD, los cuales son escogidos por las necesidades del usuario y las ventajas que estas posean, dentro de ellas se puede encontrar PostgreSQL, MySQL, SQLite, entre otras.

2.8.1 PostgreSQL

PostgreSQL es un SGBD objeto-relacional, basado en el proyecto POSTGRES, de la Universidad de Berkeley. Es ampliamente considerado como una de las alternativas de sistema de bases de datos de código abierto. Soporta gran parte del SQL estándar y muchas funcionalidades modernas como son: consultas complejas, llaves foráneas, disparadores (*triggers*), vistas, integridad transaccional, así como el control de versionado concurrente (MVCC) que es una estrategia de almacenamiento que permite trabajar con grandes volúmenes de datos.

Características fundamentales que posee:

- Alta concurrencia.
- Amplia variedad de tipos nativos.
- Replicación asíncrona.
- Bloqueos de tabla y filas.
- Integridad referencial.
- Múltiples tipos de datos predefinidos.
- Conectividad.

2.8.2 SQLite

SQLite tuvo sus inicios en enero del 2000, es un proyecto de dominio público creado por Dr. Richard Hipp, que implementa un motor de base de datos SQL transaccional, autocontenido, sin servidor y sin necesidad de ser configurado.

A diferencia de los SGBD cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones (Leyva Cortina, y otros, 2010). Esto reduce la latencia en el acceso a la BD, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la BD (definiciones, tablas, índices, y los propios datos), son guardados como un solo fichero estándar en la PC donde se encuentra ejecutada el sistema. Este diseño simple se logra bloqueando todo el fichero de BD al principio de cada transacción.

También el SGBD SQLite implementa un completo motor de BD multiplataforma que no precisa configuración. Se distribuye bajo licencia de dominio público. Es muy rápido y la ventaja fundamental es que permite utilizar un amplio subconjunto del lenguaje estándar SQL. SQLite también se destaca por su versatilidad (Leyva Cortina, y otros, 2010).

Varios procesos o hilos pueden acceder a la misma BD sin problemas. Varios accesos de lectura pueden ser servidos en paralelo. Un acceso de escritura solo puede ser servido si no se está sirviendo ningún otro acceso concurrentemente. En caso contrario, el acceso de escritura falla devolviendo un código de error (o puede automáticamente reintentarse hasta que expira un timeout configurable). Esta situación de acceso concurrente podría cambiar cuando se está trabajando con tablas temporales. Sin embargo, podría producirse un deadlock debido al proceso multi-hilo (Leyva Cortina, y otros, 2010).

SQLite dispone de una completa interfaz orientada a objetos, con distintas funciones que facilitan la manipulación de datos.

2.8.3 Fundamentación del SGBD escogido

Teniendo en cuenta las características que presentan, se decidió utilizar el SQLite. Este da la posibilidad de reducir la latencia en el acceso a la BD, mejorando así el rendimiento de las PC y de conexión a la misma, pues como el PostgreSQL que tiene una conexión cliente/servidor hace que se haga más lenta. También hace posible que en lugar de trabajar por proceso, trabaje directamente con la aplicación, permitiendo así un mejor funcionamiento en cuanto a rendimiento. También posibilita que se pueda combinar el motor y la interfaz de la BD en una única biblioteca, permitiendo almacenar los datos en un único archivo de texto plano. El hecho de almacenar toda la BD en un único archivo, facilita la portabilidad de los datos.

2.9 Tecnología de comunicación

Para el desarrollo de sistemas distribuidos heterogéneos es de vital importancia la tecnología que se utiliza para la comunicación entre los componentes. En este caso son utilizados los middlewares de comunicaciones. Estos son plataformas que facilitan la implementación y despliegue de aplicaciones distribuidas. Teniendo como objetivo principal, ofrecer un acuerdo entre las interfaces y los mecanismos de interoperabilidad, como contrapartida de los distintos desacuerdos en hardware, sistemas operativos, protocolos de red, y lenguajes de programación.

ICE (Internet Communications Engine) es un middleware orientado a objetos, el cual proporciona herramientas, APIs, y soporte de bibliotecas para construir aplicaciones (cliente-servidor) orientadas a objetos. Una aplicación ICE se puede usar en entornos heterogéneos: los clientes y los servidores pueden escribirse en diferentes lenguajes de programación, pueden ejecutarse en distintos sistemas operativos y en distintas arquitecturas, y pueden comunicarse empleando diferentes tecnologías de red. Además, el código fuente de estas aplicaciones puede portarse de manera independiente al entorno de desarrollo (Vallejo Fernández, 2006).

Para la implementación de este componente, se propone utilizar ICE, pues sus objetivos hacen que se asemejen a las características que necesita. Estos objetivos son los que a continuación se describen (Henning, y otros, 2009):

- Proporcionar una plataforma de middleware orientado a objetos adecuados para su uso en sistemas heterogéneos.
- Proporcionar un conjunto completo de características que apoyan el desarrollo de aplicaciones distribuidas reales para una amplia variedad de dominios.
- Evitar la complejidad innecesaria, por lo que la plataforma sea fácil de aprender y de usar.
- Proporcionar una implementación que sea eficiente en el ancho de banda, en el uso de la memoria, y la sobrecarga de la CPU.
- Proporcionar una implementación asentada en la seguridad, por lo que es conveniente para el uso sobre las de redes públicas no seguras.

Esta tecnología posee diferentes terminologías, las cuales con importantes conocer para poder trabajar con ella. De ellas se destacan las siguientes:

- Clientes: Son entidades activas, es decir, emiten solicitudes de servicio a un servidor.
- Servidores: Son entidades pasivas, es decir, proporcionan un servicio en respuesta a las solicitudes de los clientes.
- Invocación de métodos síncrona: Es la operación que se comporta como una llamada local a un procedimiento, es decir, el hilo del cliente se suspende mientras dure la llamada y se vuelve activar cuando la llamada se ha completado con la respuesta del servidor.
- Invocación de métodos asíncrona: Es cuando un cliente utiliza un proxy de la manera habitual para invocar una operación pero, además de pasar los parámetros necesarios, también pasa un objeto de retro-llamada y retorna de la invocación inmediatamente. Una vez que la operación se ha completado,

el núcleo de ejecución de la parte del cliente invoca a un método en el objeto de retro-llamada pasado inicialmente, proporcionando los resultados de la operación a dicho objeto (Vallejo Fernández, 2006).

- Tratamiento de métodos asíncrono: Es el equivalente a la invocación de métodos asíncrona del cliente pero esté funciona en el lado del servidor. Con este tratamiento se informa de la llegada de la invocación de una operación al código de la aplicación de la parte del servidor. Sin embargo, en lugar de forzar al proceso a atender la petición inmediatamente, la aplicación de la parte del servidor puede retrasar el procesamiento de dicha petición y liberar el hilo de ejecución asociado a la misma (Vallejo Fernández, 2006). El código de aplicación de la parte del servidor es ahora libre de hacer lo que quiera. Una vez que los resultados de la aplicación están disponibles, el código de aplicación de la parte del servidor realiza una llamada a la API del cliente para informar al núcleo de ejecución de la parte del servidor, de que la petición que se realizó con anterioridad ha sido completada.

ICE proporciona un protocolo RPC que puede utilizar tanto TCP/IP como UPD como capa de transporte subyacente. Además, ICE permite utilizar SSL, de forma que todas las comunicaciones entre el cliente y el servidor estén encriptadas.

El protocolo ICE define:

- Un determinado número de tipos de mensajes, como por ejemplo de solicitud y respuesta.
- Una máquina de estados que determina qué secuencia siguen los distintos tipos de mensajes que se intercambian el cliente y el servidor, junto con el establecimiento de conexión asociado.
- Reglas de codificación que determinan cómo se representa cada tipo de datos en la red.
- Una cabecera para cada tipo de mensaje que contiene detalles como el tipo del mensaje el tamaño del mensaje, y el protocolo y la versión de codificación empleados.

2.10 Búsqueda heurística

Para el desarrollo de este componente existe la necesidad de aplicar algún método heurístico, pues es importante determinar un servidor eficiente, teniendo diferentes características que contribuirán en el proceso de almacenamiento de los paquetes de medias.

Estos métodos de búsqueda heurística están orientados a reducir la cantidad de búsqueda requerida para encontrar una solución. En esencia una heurística es simplemente un conjunto de reglas que evalúan la posibilidad de que una búsqueda va en la dirección correcta. Generalmente los métodos de búsqueda

heurística se basan en maximizar o minimizar algunos aspectos del problema (Bello Pérez, 1998). Estos métodos de búsqueda heurística presentan diferentes características, que hacen que se seleccione uno de ellos para poder dar solución a dicho problema, estas son:

- Algunos no garantizan que se encuentre una solución, aunque existan soluciones.
- Si encuentran una solución, no se asegura que esta tenga las mejores propiedades (que sea de longitud mínima o de coste óptimo).
- En algunas ocasiones (que, en general, no se podrán determinar a priori), encontrarán una solución lo más aceptable posible en un tiempo razonable.

2.10.1 Función heurística y métodos heurísticos

El profesor Rafael Bello Pérez del Grupo de Investigaciones en Inteligencia Artificial de la Universidad Central de las Villas, en el año 1998 definió que una función heurística es: "…una función que hace corresponder situaciones del problema con números. Es decir, da una medida conceptual de la distancia entre un estado dado y el estado objetivo. Estos valores son usados para determinar cuál operación ejecutar a continuación, típicamente seleccionando la operación que conduce a la situación con máxima o mínima evaluación."

Estas funciones deben de aportar gran información sobre un estado en particular el cual debe ser ventajoso y objetivo, lo que proporciona una mejor búsqueda. Otra característica que deben presentar estas funciones es que su evaluación no puede requerir de un gran cálculo, pues la misma consumiría mucho recurso en cuanto a tiempo y memoria. Existen diferentes formas de conformar una buena heurística, una de las que más se utiliza es seleccionando rasgos de un estado, lo que contribuye a su evaluación, la cual es construida con una combinación lineal de estos rasgos. Estos rasgos pueden tener un peso, donde el mismo indica la importancia que este rasgo posee.

Método Hill climbing o Escalador de colinas

Este método consiste en considerar todos los posibles movimientos a partir del estado actual y elegir el mejor de ellos como nuevo estado. Este algoritmo puede o no encontrar una solución, o sea, puede acabar sin encontrar un estado objetivo, y en cambio encontrar un estado del que no sea posible generar nuevos estados mejores que él. Esto ocurre si el programa se topa con un máximo local, una meseta o una cresta (Departamento Inteligencia Artificial, 2008).

Este método no es muy eficaz, especialmente es inadecuado, para problemas en los que el valor de la función heurística cambia bruscamente al alejarse de una solución. Esto ocurre frecuentemente cuando aparece algún tipo de efecto umbral. Como es un método local, decide cual va a ser el siguiente movimiento atendiendo únicamente a las consecuencias inmediatas que va a tener esa elección, en lugar de explorar exhaustivamente todas las consecuencias.

Método Best-First

La búsqueda por el mejor nodo es una forma de combinar las ventajas de las búsquedas en profundidad y a lo ancho en un único método. En cada paso del proceso de búsqueda se selecciona el más prometedor de aquellos nodos que se han generado hasta el momento (Bello Pérez, 1998).

Este método presenta diversas características, una de ella es que tiene un control sobre todos los estados terminales y sus heurísticas. También tiene toma como raíz el estado de valor óptimo entre todos los estados terminales, desde el cual aplica este método. Una vez que aplica este método se generan en amplitud todos los hijos del nodo raíz. La aplicación de este método es que siempre va a expandir el nodo no expandido más idóneo, si uno de ellos es la solución, el proceso termina. En caso de no ser, se selecciona el otro más idóneo de la lista.

Si no se encuentra una solución, la rama empezará a parecer menos prometedora, lo que hará que comience su exploración. La vieja rama no se olvida, pues su último nodo se almacena en el conjunto de nodos aún sin expandir. Esto permite que se pueda utilizar este, si las soluciones anteriores son menos eficientes que esta (Rich, y otros, 1994).

Método A*

El método A* es una generalización del método de búsqueda del primero o Best-First. Tiene como objetivo evitar expandir caminos que ya acumulan un costo elevado, teniendo en cuenta el costo que tiene el camino desde el nodo raíz hasta el actual.

Para esto utiliza una función (**f**), la cual es la suma de dos, que se las llamará ***g*** y ***h'***. La función ***g*** es una medida del costo de llegar desde el nodo inicial al nodo actual. La función ***h'*** es una estimación del costo adicional para llegar desde el nodo actual al estado objetivo. Aquí es donde se explota el conocimiento que se dispone sobre el dominio del problema (Departamento Inteligencia Artificial, 2008). Donde, si el nodo actual ha generado más de un sendero, este método solo deberá dejar registrado el mejor.

En este trabajo se selecciona el método A*, pues el Manejador de Datos deberá atender **N** peticiones y seleccionar un servidor que en un instante ***i*** que cuente con las mejores condiciones para realizar el almacenamiento. Este algoritmo se evidencia cuando se tiene que atender una petición en el instante ***i*** + 1, pues tiene en cuenta el cambio realizado en el servidor en la petición anterior. Cuenta además para la selección de una función heurística que la misma analiza diferentes parámetros como son la capacidad de almacenamiento, carga del CPU, cantidad de hilos disponibles, entre otros. Estos establecen una relación dando la medida del rendimiento de cada servidor que se busque, proporcionando así, el servidor idóneo.

2.11 Conclusiones parciales

En este capítulo se encuentra recogido un resumen del estudio realizado de las diferentes tecnologías y herramientas para el desarrollo del software, permitiendo así tener una base teórica y tecnológica sólida que será de gran ayuda en la construcción de esta aplicación. Se determinó utilizar a RUP como metodología de desarrollo de software, SQLite como SGBD. También se decidió utilizar UML como lenguaje de modelado, teniendo en cuenta esto, se utilizará a Visual Paradigm como herramienta de modelado. Se estableció como IDE de desarrollo Qt Creator el cual brinda muchas facilidades de uso para desarrollar con el lenguaje C++. Se decidió utilizar como tecnología de comunicación a ICE, la cual permitirá realizar de forma eficiente la conexión entre los componentes que tienen que interactuar con el Manejador de Datos. Se seleccionó el método heurístico A*, para determinar un servidor eficiente, donde el mismo tendrá como responsabilidad el almacenamiento de paquetes de medias, teniendo en cuenta diferentes parámetros relacionados con el rendimiento que estos poseen.

CAPITULO 3. Presentación de la solución propuesta

3.1 Introducción

En este capítulo se hace una descripción breve sobre la solución propuesta a través del modelo de dominio. Se hace también un análisis sobre los requisitos funcionales y no funcionales, detallando los actores que intervienen en la misma. Por otra parte, se presenta el diagrama de casos de uso del sistema generados por los requisitos funcionales y una explicación de los mismos.

3.2 Modelo de dominio

El Modelo de Dominio permite representar los conceptos más importantes de los objetos que se muestran en el dominio del problema. Esto hace posible que se haga una visualización de las clases conceptuales del mundo real, no de componentes de software. Este modelo, no se trata de un conjunto de diagramas que describen clases de software u objetos de software con responsabilidades, sino que puede considerarse como un diccionario visual de las abstracciones relevantes, vocabulario e información del dominio. En la figura 5 se representa el diagrama del modelo de dominio de esta investigación.

3.2.1 Diagrama del modelo de dominio

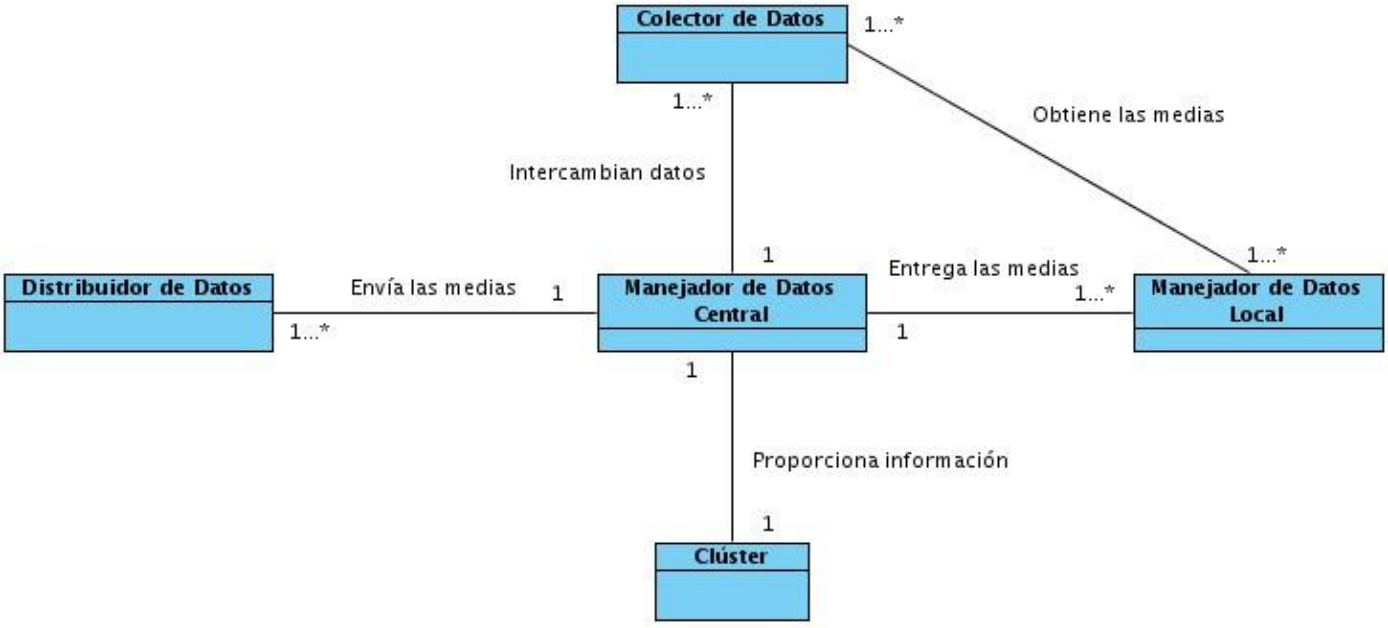

Figura 5 Diagrama del Modelo de Dominio

Conceptos principales del modelo de dominio:

Distribuidor de Datos: componente que se encarga de enviar paquetes de medias, para ser almacenados.

Colector de Datos: componente que necesita los datos que están almacenados para realizar su función.

Manejador de Datos Central: es el encargado de entregar las medias para ser almacenadas, así como el de brindar información de la ubicación de los paquetes de medias.

Manejador de Datos Local: se encarga de almacenar y brindar información de las medias.

Clúster: concepto que abarca toda la información relacionada con los servidores del sistema.

De los conceptos anteriormente mencionados, se derivan varias relaciones entre ellos. Estas relaciones, deben de verse en dos procedimientos distintos, almacenamiento y recuperación de los paquetes de medias.

En el primer caso, el Distribuidor de Datos es el encargado de enviar los paquetes de media al Manejador de Datos Central. Este se encarga de obtener información de los servidores dentro del Clúster, permitiendo así, la entrega del paquete de media al Manejador de Datos Local que se encarga de almacenar ese paquete. En el segundo, el Colector de Datos necesita información proporcionada por el Manejador de Datos Central, para poder obtener los paquetes de media que le proporciona el Manejador de Datos Local.

3.3 Requerimientos

El componente debe de contar con diferentes requerimientos que ayudan a la creación y el buen funcionamiento del mismo. Pues los requerimientos no son más que las circunstancias o capacidades que necesita el usuario para resolver un problema o conseguir un objetivo determinado. Los mismos se pueden clasificar en funcionales y no funcionales.

3.3.1 Requisitos funcionales

Los requerimientos funcionales son las capacidades o las condiciones que el sistema debe cumplir, se mantienen invariables, sin importar con que propiedades o cualidades se relacionen por lo que no alteran la funcionalidad del producto (Pérez Vázquez, y otros, 2010).

Después de realizar un estudio exhaustivo del modelo de dominio, se pudo establecer cuatro funcionalidades importantes para realizar dicho componente.

A continuación les relacionamos esas funcionalidades:

RF 1 - Distribuir paquetes de media en servidores.

RF 2 - Obtener información en los servidores del clúster.

RF 3 - Gestionar distribución de paquetes de media.

RF 3.1 - Adicionar ubicación de los paquetes de media en el clúster.

RF 3.2 - Eliminar ubicación de los paquetes de media en el clúster.

RF 4 - Almacenar paquetes de media.

RF 5 - Obtener dirección de la media en el servidor.

RF 6 - Obtener información de los paquetes de media en el clúster.

RF 7 - Eliminar media en el servidor.

3.3.2 Requisitos no funcionales

Los requerimientos no funcionales son aquellos requisitos que hacen que el sistema sea usable, rápido, confiable y agradable para los usuarios (Pérez Vázquez, y otros, 2010). En este caso, se puede decir de forma general, que son las cualidades que el sistema debe de tener, para que pueda funcionar de una manera eficiente.

- **Usabilidad**
  - ✓ Se concibe que el sistema sea utilizado por personas con conocimientos básicos de informática, en el trabajo con aplicaciones de escritorio y con medias.

- **Soporte**
  - ✓ El sistema será modificable, pudiendo agregarle módulos cuando sea necesario, para así logar mayor extensibilidad y mejores prestaciones.

- **Eficiencia**
  - ✓ La velocidad de procesamiento de la información, la actualización y la recuperación dependerán de la cantidad de información que tenga que procesar el sistema y la infraestructura tecnológica

- **Requerimientos de hardware**

  Manejador de Datos Central:

  ✓ Se requiere tarjeta de red de 100 Mbps o superior.
  ✓ Memoria RAM de 4GB o superior.
  ✓ Disco duro de 80GB o superior.
  ✓ Procesador 3 GHz o superior.

  Manejador de Datos Local:

  ✓ Se requiere tarjeta de red de 10 Mbps o superior.
  ✓ Memoria RAM de 2GB o superior.
  ✓ Disco duro de 1TB o superior.
  ✓ Procesador 3 GHz o superior.

- **Requerimientos en el diseño y la implementación**

  ✓ Lenguaje de programación C++.
  ✓ QtCreator como IDE de desarrollo.
  ✓ Visual Paradigm como herramienta CASE.
  ✓ SQLite como gestor de bases de datos.

3.4 Modelo de sistema

Siguiendo las actividades que propone RUP para el modelado del sistema, referente al flujo de trabajo de requerimientos, se dio la tarea de agrupar lo requerimientos funcionales en casos de uso del sistema, también se identifican los actores del sistema y así como la descripción de los CUS.

3.4.1 Actores del sistema

Los actores del sistema se definen como algo con comportamiento, como una persona (identificada por un rol), o sistema informatizado u organización, cuando solicita los servicios de otros sistemas (Larman, 2003). Estos están relacionados con los diferentes casos de uso con que cuanta el sistema para satisfacer un objetivo.

Tabla 1 Actores del Sistema

Actor	Descripción
Distribuidor de datos	Es un sistema que interactúa con el sub-componte manejador de datos central, inicializando el caso de uso distribuir paquetes de media.
Colector de datos	Es un sistema encargado de interactuar con el componente manejador de datos, iniciando los casos de uso obtener dirección de media y obtención de información de los paquetes de media.
Manejador de datos central	Es un sistema encargado de gestionar distintas informaciones, el cual inicializa diferentes casos de uso como almacenar media y el gestionar paquetes de media.
Manejador de datos local	Es un sistema que realiza diversas funcionalidades dentro de cada servidor como inicializar el caso de uso eliminar media.

3.4.2 Diagrama de casos de uso del sistema

Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. A continuación se muestra el diagrama de casos de uso de este sistema (ver figura 6):

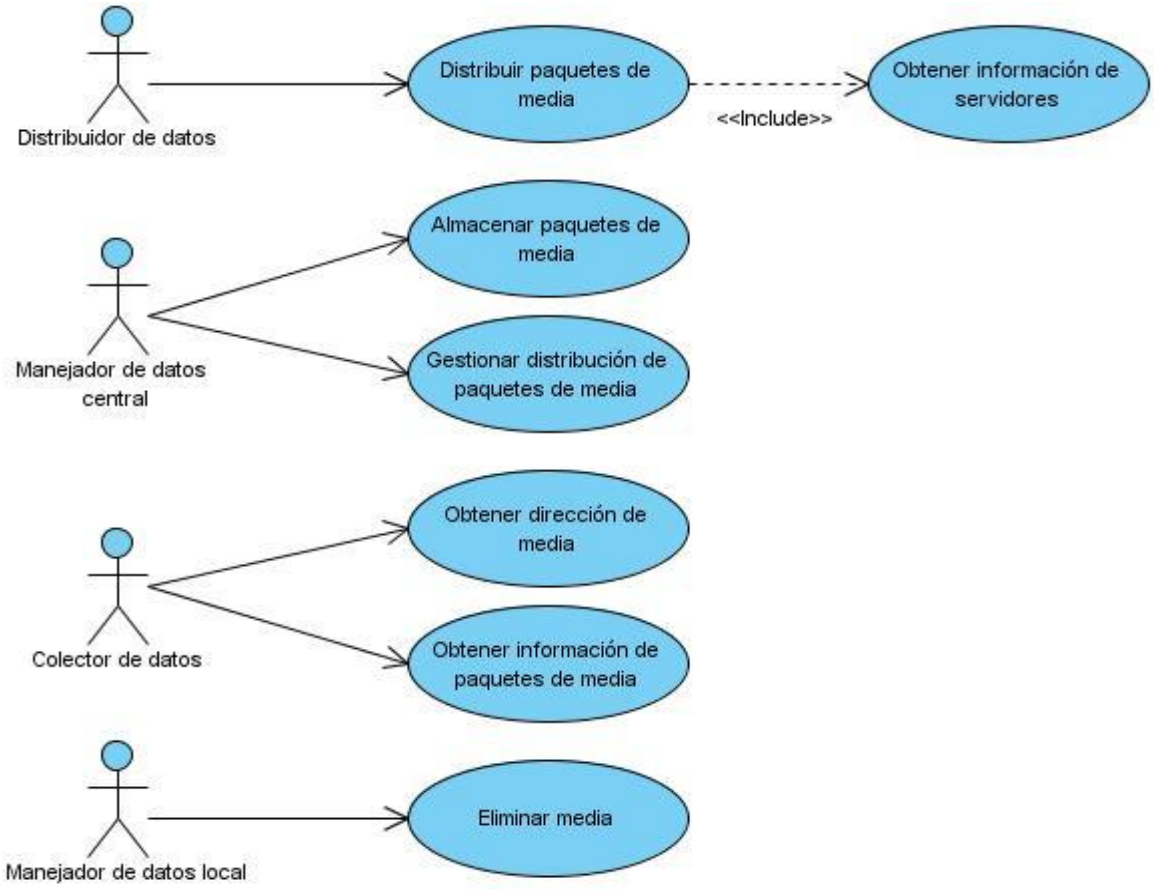

Figura 6 Diagrama de Casos de Uso del Sistema

3.4.3 Descripción de los casos de uso del sistema

En esta sección se describen todos los casos de uso identificados. En los mismos se da una breve explicación de las funcionalidades con que debe contar el sistema.

Tabla 2 Descripción del caso de uso distribuir paquetes de media en servidores

Caso de Uso:	Distribuir paquetes de media en servidores.
Actores:	Distribuidor de datos.
Propósito:	Su funcionalidad es la de distribuir los paquetes de media en los Servidores.
Resumen:	Este caso de uso inicia cuando se desea efectuar un

	almacenamiento de algún archivo de media.
Precondiciones:	Ninguna.
Referencias:	RF 1, RF 2, RF 3.1
Prioridad:	Crítico.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El actor envía flujo de paquetes de media.	**2.** El sistema obtiene el flujo de paquetes de media que ha sido transmitido. **3.** Selecciona un servidor eficiente para realizar la distribución (ver caso de uso Obtener información en los servidores del clúster). **4.** Envía el paquete de media al servidor seleccionado
Flujo Alterno	
	3.1 El sistema muestra un mensaje de error si no tiene algún servidor disponible. **4.1** El sistema muestra un mensaje de error sino se pudo enviar el paquete de media.
Postcondiciones:	Queda distribuido el paquete de media enviado por el actor.

Tabla 3 Descripción del caso de uso obtener información en los servidores del clúster

Caso de Uso:	Obtener información en los servidores del clúster.
Actores:	CUS Distribuir paquetes de media.
Propósito:	Se basa en obtener distintos datos que poseen los servidores en el clúster.

Resumen:	Se realiza para obtener la información de los servidores.
Precondiciones:	Debe haberse inicializado por el CUS Distribuir paquetes de media.
Referencias:	RF 2
Prioridad:	Crítico.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El actor solicita la información de varios parámetros que influyen en el rendimiento del servidor.	**2.** Realiza la búsqueda de la información solicitada. **3.** Envía información encontrada al actor.
Flujo Alterno	
Postcondiciones:	Se notifica la información de los parámetros pedidos.

Tabla 4 Descripción del caso de uso gestionar distribución de paquetes de media

Caso de Uso:	Gestionar distribución de paquetes de media.
Actores:	Manejador de datos central.
Propósito:	Gestiona toda la información referente a la distribución de paquetes de media hacia los servidores.
Resumen:	En él se puede adicionar, eliminar toda la información referente a las distribuciones realizadas.
Precondiciones:	
Referencias:	RF 3, RF 3.1, RF 3.2, RF 7
Prioridad:	Crítico.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El actor selecciona una de las siguientes opciones:	**2.** El sistema ejecuta una de las siguientes secciones:

- Adicionar. - Eliminar.	- Si seleccionó la opción Adicionar, ir a la sección Adicionar. - Si seleccionó la opción Eliminar, ir a la sección Eliminar.
Sesión "Adicionar"	
Acción del Actor	**Respuesta del Sistema**
1. El actor solicita adicionar pasando como datos los parámetros de la distribución.	**2.** El sistema adiciona la ubicación de los datos distribuidos.
Flujo Alterno	
	2.1 El sistema muestra un mensaje de error, si los datos que son pasado no son válidos.
Postcondiciones:	Quedan adicionados los datos de los paquetes de media que han sido distribuidos.
Sección "Eliminar"	
Acción del Actor	**Respuesta del Sistema**
1. El actor selecciona el nombre del archivo que se desea eliminar.	**2.** Elimina la información que corresponde al nombre del archivo que seleccionó el actor. **3.** Envía solicitud de eliminar media en los servidores donde se encuentra cada paquete de la media (ver caso de uso Eliminar media en el servidor).
Flujo Alterno	
	2.1. El sistema muestra un mensaje de error si no se pudo eliminar la media. **3.1.** El sistema muestra un mensaje

	de error si no puede eliminar los paquetes de media en el servidor que se encuentra.
Postcondiciones:	Se elimina todos los paquetes de media, del archivo que se quiere eliminar.

Tabla 5 Descripción del caso de uso almacenar paquete de media

Caso de Uso:	Almacenar paquete de media.	
Actores:	Manejador de datos central.	
Propósito:	Su función fundamental es el almacenamiento de paquetes de medias en un servidor del clúster.	
Resumen:	Su función es de almacenar el paquete de media en cada servidor.	
Precondiciones:	Debe haberse inicializado el CUS Distribuir paquetes de media.	
Referencias:	RF 4	
Prioridad:	Crítico.	
Flujo Normal de Eventos		
Acción del Actor		**Respuesta del Sistema**
1. El actor envía flujo del paquete de media.		**2.** Recibe el flujo de paquetes de media. **3.** Crea un sistema de archivos para almacenar la media.
Flujo Alterno		
Postcondiciones:	Queda almacenado el paquete de media.	

Tabla 6 Descripción del caso de uso obtener dirección de la media en el servidor

Caso de Uso:	Obtener dirección de la media en el servidor
Actores:	Colector de datos.
Propósito:	Su función es devolver la ubicación física de donde se encuentra el paquete.
Resumen:	Es encargado de devolver la dirección física del paquete de la media dada.
Precondiciones:	Ninguna.
Referencias:	RF 5
Prioridad:	Crítico.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El actor envía los datos del nombre y el número del paquete de la media que quiere obtener la dirección.	**2.** Busca la dirección física de donde se encuentra almacenado el paquete en el servidor por el nombre de la media y el número de paquete pasados. **3.** Devuelve la dirección donde se encuentra la media.
Flujo Alterno	
	2.1. Envía un mensaje de error sino encuentra el paquete que se quiere.
Postcondiciones:	Obtiene la dirección física de la media.

Tabla 7 Descripción del caso de uso obtener información de los paquetes de media en el clúster.

Caso de Uso:	Obtener información de los paquetes de media en el clúster.
Actores:	Colector de datos.
Propósito:	Su función es devolver la ubicación de todos los paquetes de una media dada.

Resumen:	Es encargado de devolver la dirección de los paquetes de media.
Precondiciones:	Ninguna.
Referencias:	RF 6.
Prioridad:	Crítico.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El actor envía el nombre de la media que quiere obtener la ubicación de todos los paquetes.	**2.** Busca todas las ubicaciones de los paquetes donde se encuentra el nombre de la media pasada. **3.** Devuelve todas las direcciones de los paquetes de donde se encuentra la media.
Flujo Alterno	
	2.1. Envía un mensaje de error sino encuentra la media que se quiere.
Postcondiciones:	Obtiene todas las direcciones de la media solicitada que están en el clúster.

Tabla 8 Descripción del caso de uso eliminar media en el servidor

Caso de Uso:	Eliminar media en el servidor.
Actores:	Manejador de datos local.
Propósito:	Su función es eliminar una media almacenada en un servidor.
Resumen:	Es encargado de eliminar una media en un servidor del clúster.
Precondiciones:	Ninguna.
Referencias:	RF 7.
Prioridad:	Crítico.

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El actor envía el nombre de la media que quiere eliminar.	**2.** Busca la media que se quiere eliminar por el nombre enviado por el actor. **3.** Elimina la media físicamente y de la base de datos.
Flujo Alterno	
	2.1. Envía un mensaje de error sino encuentra la media que se quiere eliminar.
Postcondiciones:	Se elimina la media deseada por el actor.

3.5 Patrones

En el desarrollo de un software, se hace necesario seleccionar diferentes patrones los cuales ayudan a que esté presente una buena estructura y organización que hace eficiente su funcionamiento. Estos patrones no son más que una guía para cometer alguna acción. Especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones, para organizar los distintos componentes. Esto hace posible un mejor entendimiento de la situación que se encuentre algún problema que se le quiera dar solución.

3.5.1 Patrones de arquitectura

Existen diversos patrones de arquitectura, el escogido para resolver esta situación fue el de arquitectura por capa. Esta elección fue realizada por el arquitecto del proyecto DESCOMTEC, ya que brinda muchas funcionalidades para la solución de este componente.

Este patrón tiene dentro de sus características fundamentales, que permite dividir sistemas de software complicados, para así hacerlo más entendible y fácil de realizar. En este esquema la capa más alta utiliza varios servicios definidos por la inferior, pero la última es inconsciente de la superior. Además, normalmente cada capa, oculta a las capas inferiores a esta.

Este patrón de arquitectura brinda los siguientes beneficios:

- Se puede entender una capa como un todo, sin considerar las otras.
- Las capas se pueden sustituir con implementaciones alternativas de los mismos servicios básicos
- Se minimizan dependencias entre capas.
- Las capas posibilitan la estandarización de servicios
- Luego de tener una capa construida, puede ser utilizada por muchos servicios de mayor nivel.

3.5.2 Patrones de diseño

Para la solución de esta aplicación se tuvo en cuenta diversos patrones de diseño, los cuales permiten tener una guía en la creación del sistema.

Patrones GRASP:

- **Experto:** Este patrón, está diseñado para que la responsabilidad de realizar una labor sea de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. Si se realiza bien la aplicación de este, los sistemas tienden a ser más fáciles de entender, mantener y ampliar, y existen más oportunidades para reutilizar componentes en futuras aplicaciones (Larman, 2003). Este patrón es utilizado en la aplicación cuando se está dando todas las funcionalidades a las clases que se crean. Ejemplo de esto lo demuestra en la clase **CServidor** que tiene un método que es calcular heurística, el cual lo realiza porque cuanta con todos los parámetros para dar solución a esto.
- **Creador:** Guía la asignación de responsabilidades relacionadas con la creación de objetos, una tarea muy común. La intención básica del patrón Creador es encontrar un creador que necesite conectarse al objeto creado en alguna situación. Eligiéndolo como el creador se favorece el bajo acoplamiento (Larman, 2003). Este se utiliza cuando se crea algún objeto que sea o no de la clase en la que se quiere crear, pues va a contar con todos los valores posibles para crear dicho objeto, así como para poder utilizarlo.

- **Controlador:** Está diseñado para asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades. Dicho patrón es utilizado en el sistema pues cuenta con una clase controladora encargada de interactuar con las clases de interfaz, así como las de acceso a datos y a las de tipo de entidad.
- **Alta cohesión:** Propone asignar la responsabilidad de manera que la complejidad se mantenga dentro de límites manejables asumiendo solamente las responsabilidades que deben manejar, evadiendo un trabajo excesivo. Este permitió que se mejore mejora la claridad y facilidad con que se entiende el diseño del sistema, por lo cual soporta una mayor capacidad de reutilización de las clases.
- **Bajo Acoplamiento:** Debe haber pocas dependencias entre las clases. Si todas las clases dependen de todas no se puede extraer software de forma independiente y reutilizarlo. Este patrón es un principio que asigna la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades**.**

Patrones GoF:

- **Singleton:** Este patrón posibilita mantener la visibilidad global o un único punto de acceso a una única instancia de una clase, en lugar de cualquier otra forma de visibilidad, haciéndolo con una referencia permanente a la misma (Larman, 2003). Este se utiliza en la inicialización del sistema, pues crea una única instancia de la clase de conexión a la base de datos, permitiendo acceder a esta instancia en cualquier clase ajena a ella, también se aplica en crear una sola instancia de la clase controladora.
- **Observador:** Permite a los objetos captar dinámicamente las dependencias entre objetos, de tal forma que un objeto notificará a los objetos dependientes de él cuando cambia su estado, siendo actualizados automáticamente. Se utiliza pues hay clases de la capa del negocio que dependen de la clase publicadora de ICE en la capa superior (Presentación) y es necesario que esta notifique en caso de algún cambio.

- **Fachada:** Se hace uso de este patrón pues se utiliza una clase interfaz publicadora de ICE que encapsula todas las funcionalidades del componente y a través de esta se realiza toda la interacción con los otros componentes.

3.6 Diagramas de clases del diseño

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema; también muestra sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, cuando se crea el diseño conceptual de la información que se manejará en el sistema, conjuntamente con los componentes que se encargarán del funcionamiento y las relaciones entre uno y otros. A continuación se representa los diagramas de clases de los casos de uso.

3.6.1 Diagrama de Clase del Caso de Uso eliminar media en el servidor

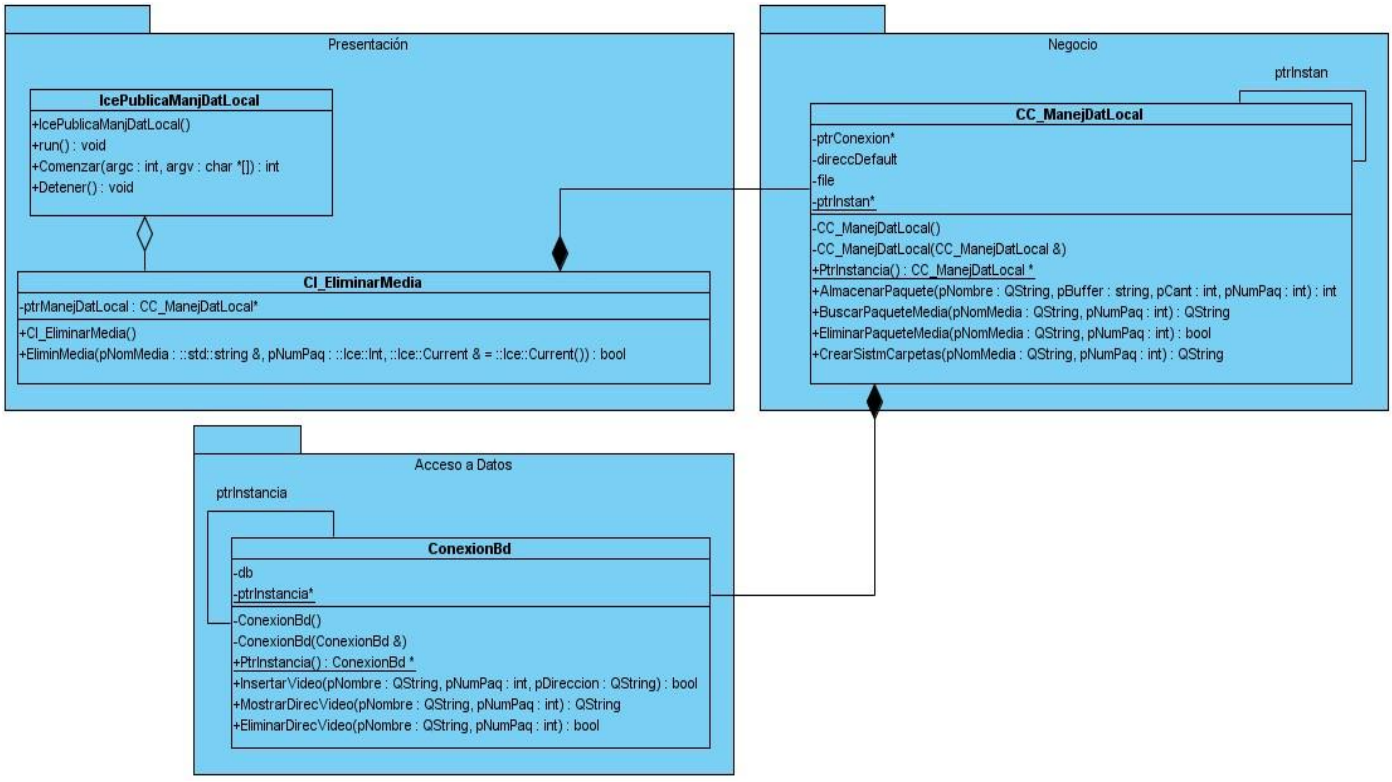

Figura 7 Diagrama de Clase del Caso de Uso eliminar media en el servidor

3.6.2 Diagrama de Clase del Caso de Uso distribuir paquetes de media en servidores

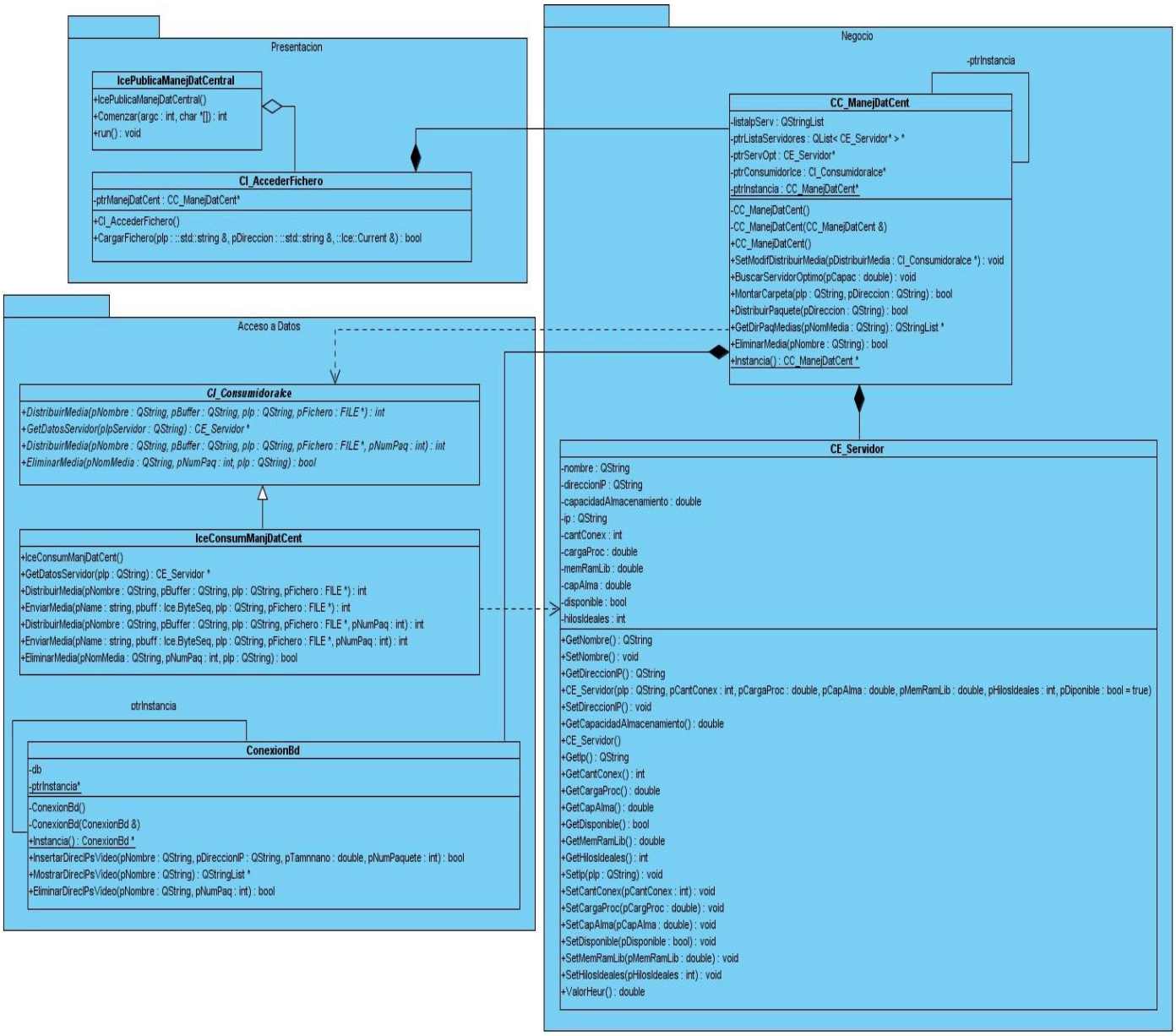

Figura 8 Diagrama de Clase del Caso de Uso distribuir paquetes de media en servidores

3.6.3 Diagrama de Clase del Caso de Uso gestionar distribución de paquetes de media

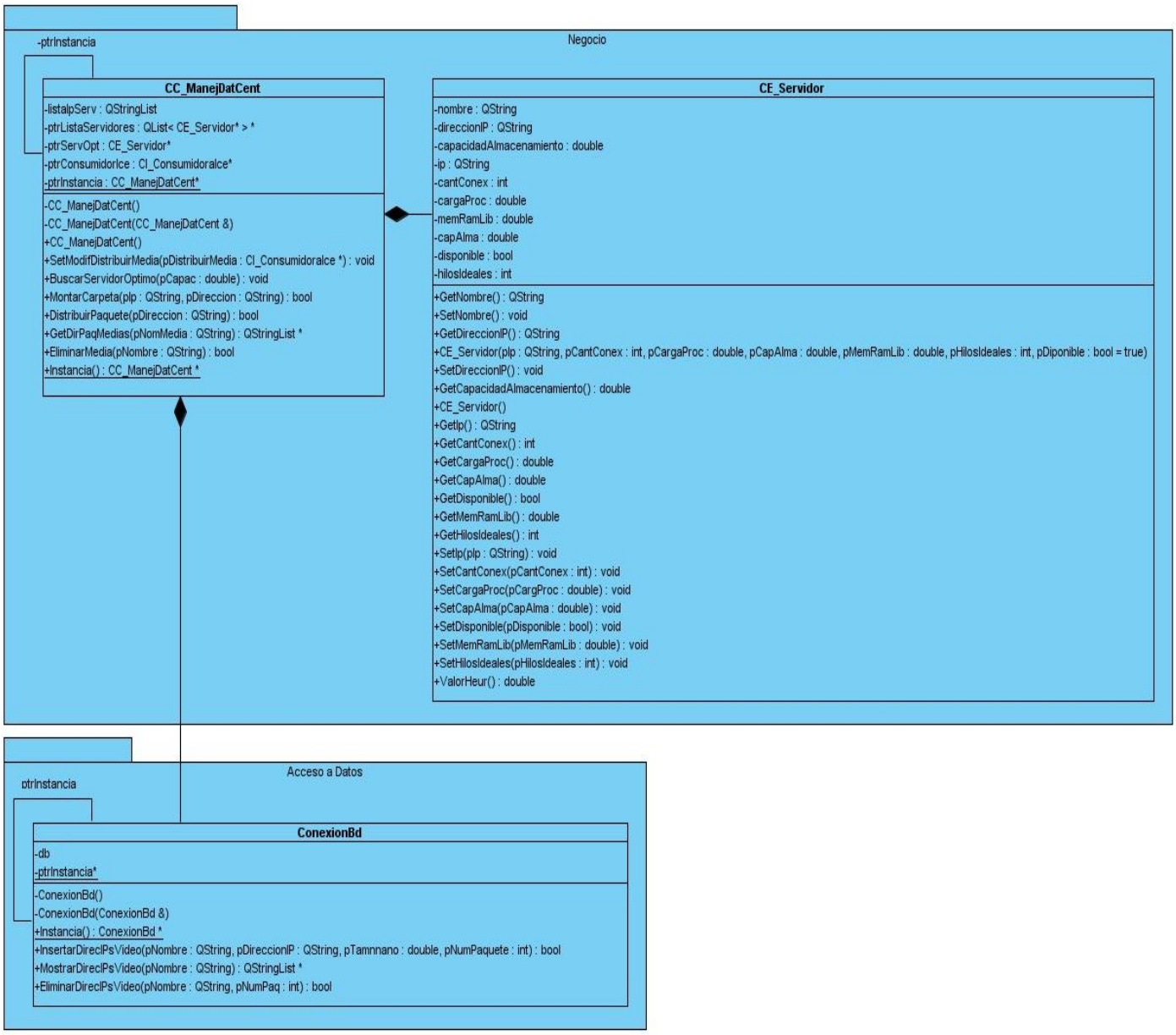

Figura 9 Diagrama de Clase del Caso de Uso gestionar distribución de paquetes de media

3.6.4 Diagrama de Clase del Caso de Uso almacenar paquete de media

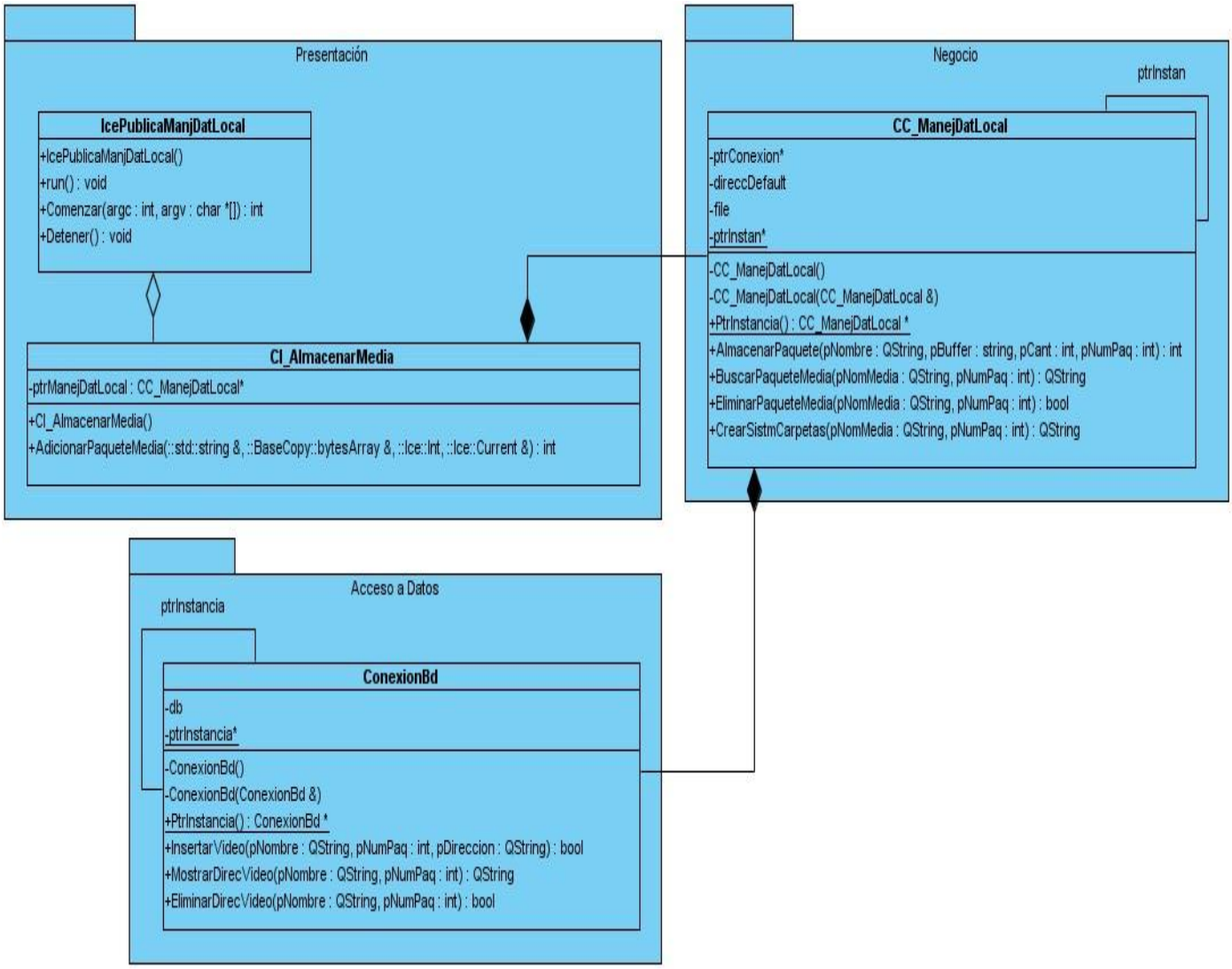

Figura 10 Diagrama de Clase del Caso de Uso almacenar paquete de media

3.6.5 Diagrama de Clase del Caso de Uso obtener dirección de la media en el servidor

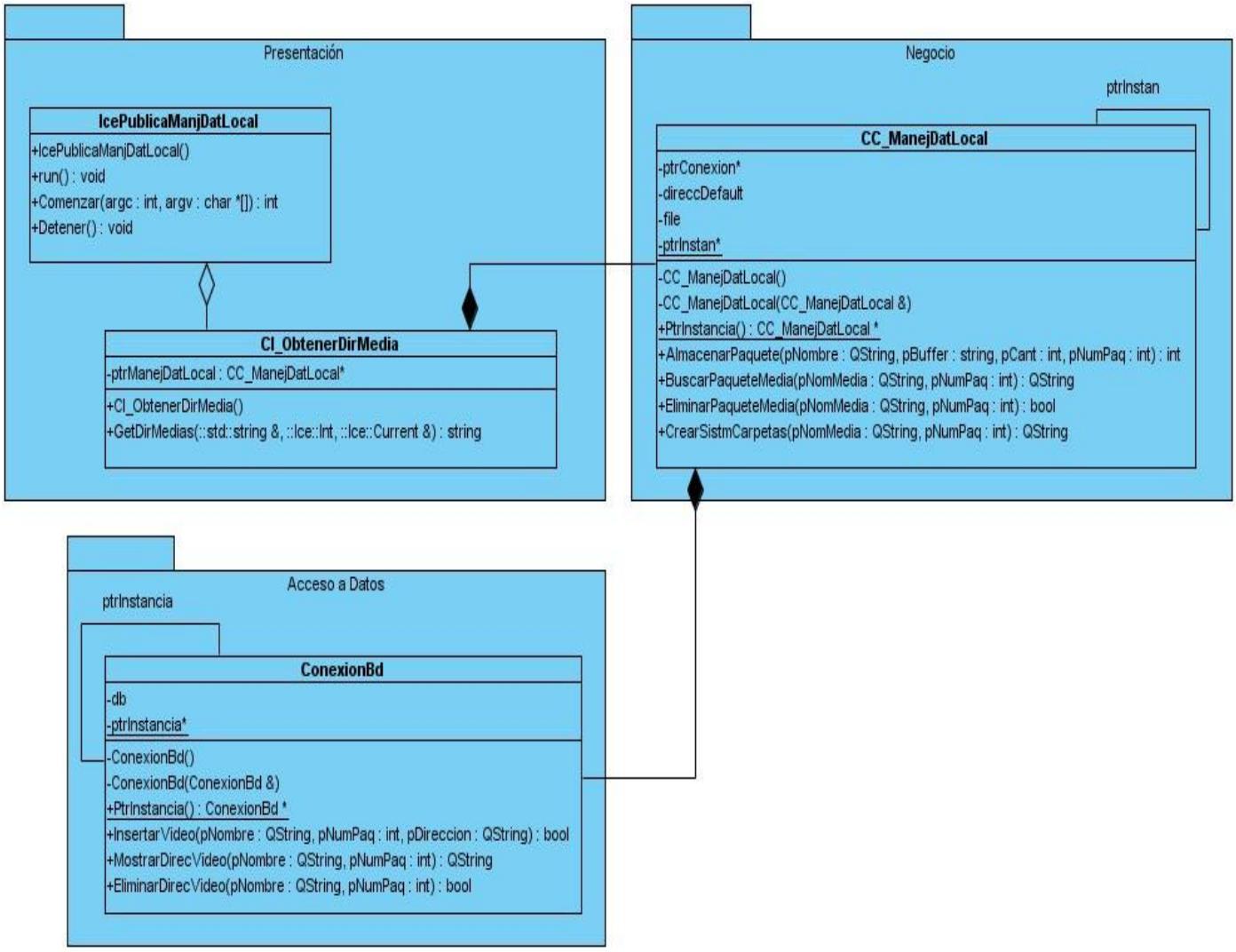

Figura 11 Diagrama de Clase del Caso de Uso obtener dirección de la media en el servidor

3.6.6 Diagrama de Clase del Caso de Uso obtener información de paquetes de media en el clúster

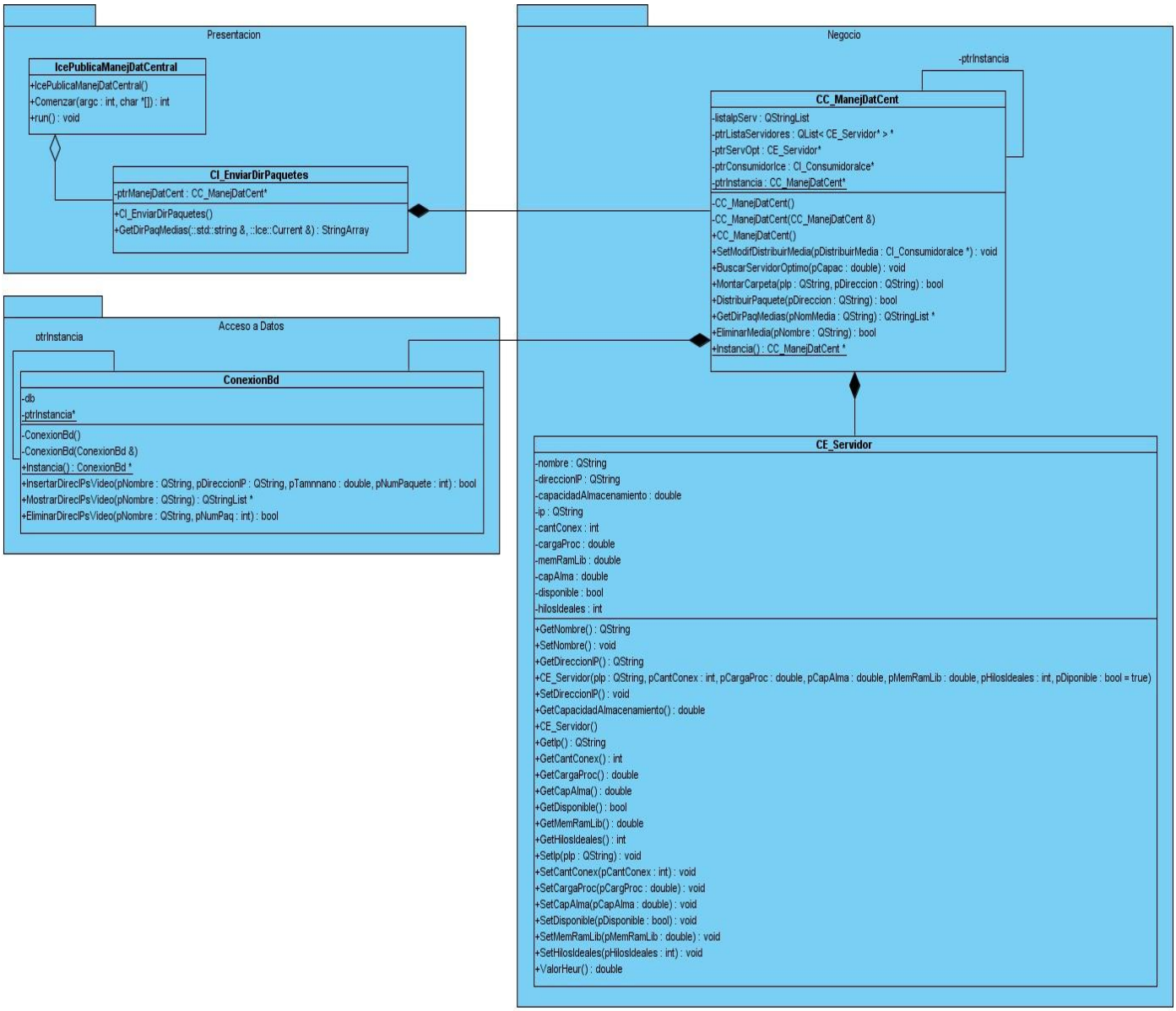

Figura 12 Diagrama de Clase del Caso de Uso obtener información de paquetes de media en el clúster

3.7 Diseño de la Base de Datos

Para que el diseño del sistema esté completamente terminado, es necesario, realizar el diseño de la base de datos con que este cuenta. El mismo contará con un diseño por separado de cada uno de los subcomponentes.

En la realización de un buen diseño, posibilita que exista una mejor calidad en el funcionamiento del componente. Es por ello que en este epígrafe se abordarán los Diagramas de Entidad Relación (DER) y el de Clases Persistentes (DCP) propios de cada base de datos.

El DER representa la realidad de la problemática identificada a través de las entidades y de los enlaces que rigen la unión de las mismas y que constituyen la relación del modelo (Suárez Pérez, y otros, 2008).

En la figura 13 se muestra el Diagrama Entidad Manejador de Datos Central y en la figura 14 el de Diagrama Entidad Manejador de Datos Local.

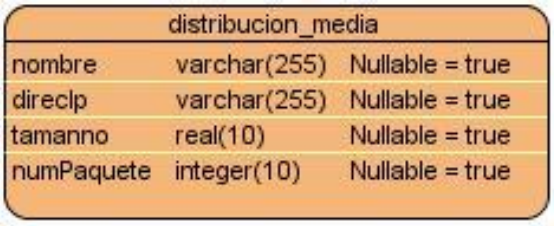

Figura 13 Diagrama Entidad Manejador de Datos Central

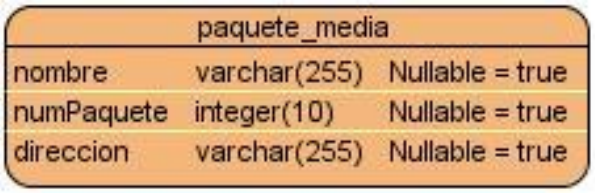

Figura 14 Diagrama Entidad Manejador de Datos Local

El diagrama de clases persistentes muestra las relaciones existentes entre aquellas clases que implementan las entidades del problema de negocio, manteniendo su valor en un espacio y tiempo determinado (Suárez Pérez, y otros, 2008).

En la figura 15 se muestra el Diagrama de Clases Persistente del Manejador de Datos Central y en la figura 16 el Diagrama de Clases Persistente del Manejador de Datos Local.

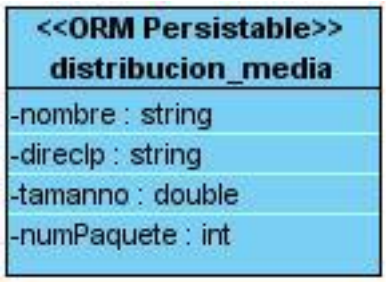

Figura 15 Diagrama de Clases Persistente del Manejador de Datos Central

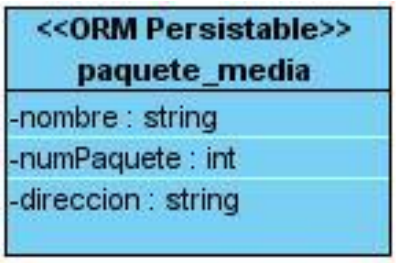

Figura 16 Diagrama de Clases Persistente del Manejador de Datos Local

3.8 Conclusiones parciales

En este capítulo se trataron los temas necesarios para definir las características con las que contará el componente que se desarrollará. Se definieron siete requerimientos que debe tener el sistema, apoyado por el modelo de dominio. También se pudo distinguir los actores y los distintos casos de usos, terminando con la descripción de los mismos.

Igualmente se expuso el patrón de arquitectura que se emplea, el cual está dirigido por la arquitectura en capas. Se presentaron los patrones de diseño que serán utilizados en el desarrollo, facilitando la representación de los diagramas de clase del diseño, lo cual servirá de apoyo para la fase de implementación. También quedaron diseñadas las diferentes bases de datos correspondientes a cada uno de los subcomponentes que contará el sistema, haciendo así más funcional el componente.

CAPITULO 4. Construcción de la solución propuesta

4.1. Introducción

En este capítulo se describe la implementación de sistema, después de haber realizado en el capítulo anterior el análisis y el diseño. Se muestra los diagramas de componentes del Manejador de Datos los cuales han sido desarrollados. También se presenta todo lo relacionado con las pruebas realizadas al sistema.

4.2. Modelo de implementación

El modelo de implementación describe cómo los elementos del modelo de diseño, como las clases, se implementan en términos de componentes, como ficheros de código fuente, ejecutables, entre otros. El modelo de implementación describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y cómo dependen los componentes unos de otros (López Amado, 2010).

En el modelo de implementación del sistema manejador de datos, representado en la figura 17. Se observan los dos subsistemas y componentes externos, los cuales en su mayoría son librerías. Dentro de ellos se pueden encontrar sql, la cual es una librería que se utiliza para interactuar con gestores de base de datos. Otra que se puede encontrar es lice, la cual permite hacer las conexiones con la tecnología de ICE.

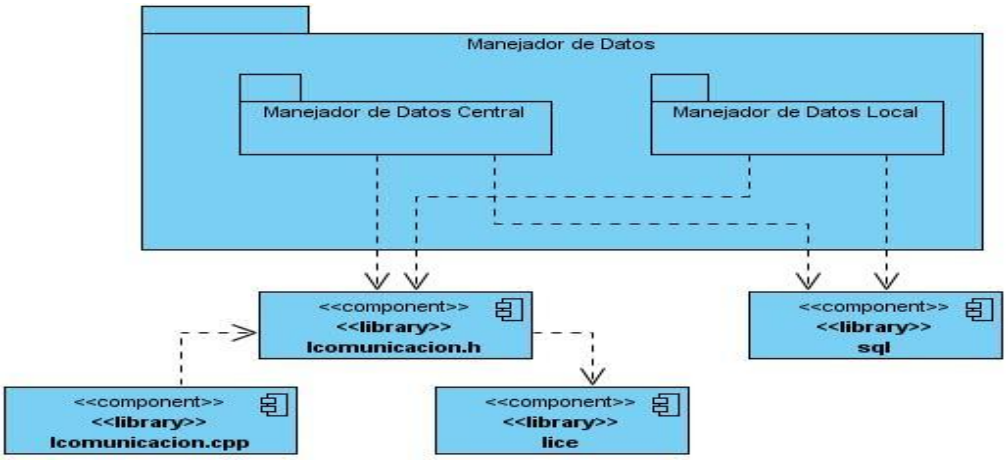

Figura 17 Modelo de implementación

Diagrama de componentes:

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes software, sean estos componentes de código fuente, binarios o ejecutables. Desde el punto de vista del diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización, y las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo. A continuación se muestra en la figura 18 el diagrama de componente del subsistema Manejador de Datos Central el cual interactúa con otros dos subsistemas y en la figura 19 el de Manejador de Datos Local que al igual se relaciona con otros dos subsistemas:

- Subsistema Manejador de Datos Central

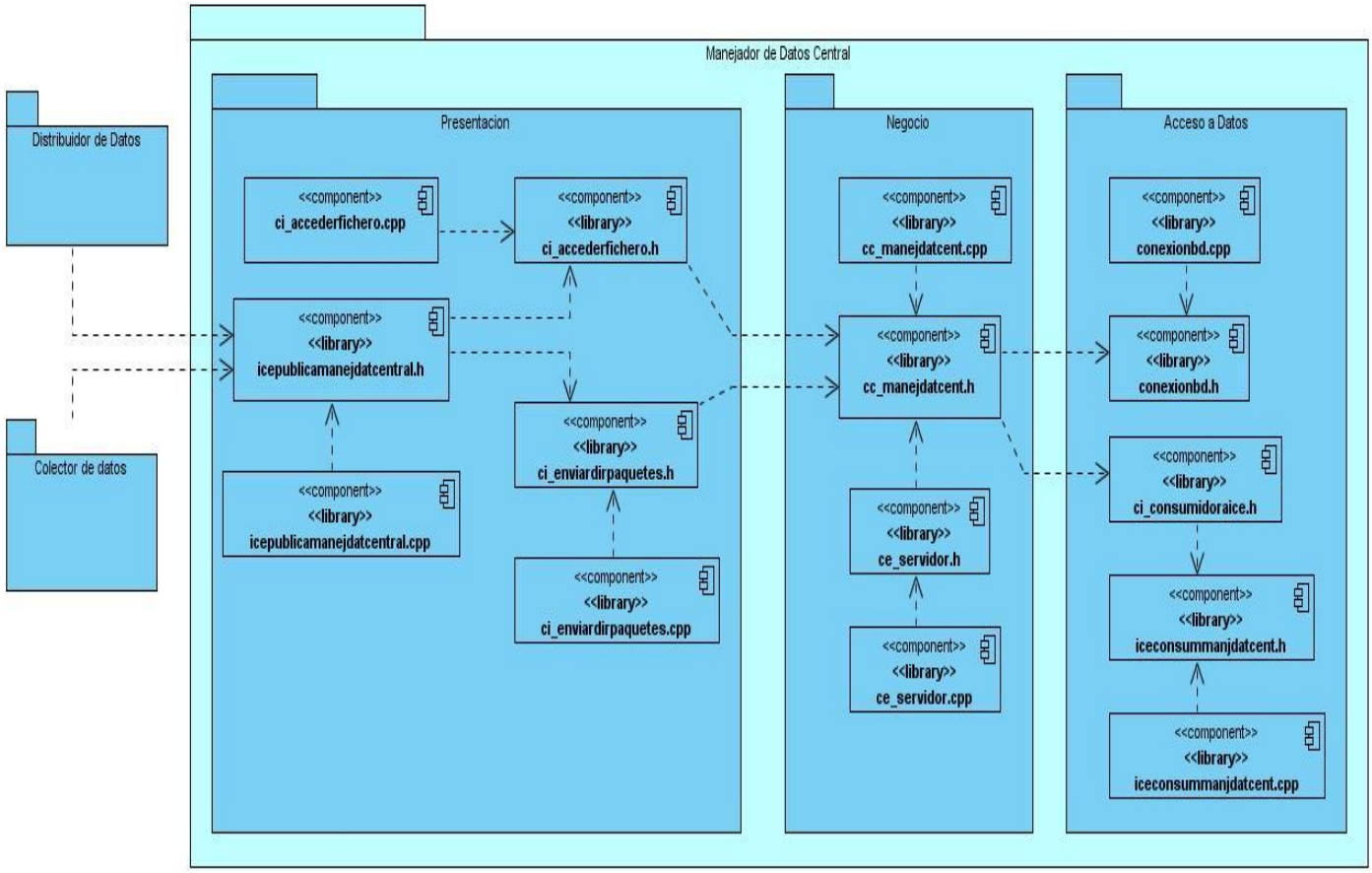

Figura 18 Diagrama de componentes del subsistema manejador de datos central

- Subsistema Manejador de Datos Central

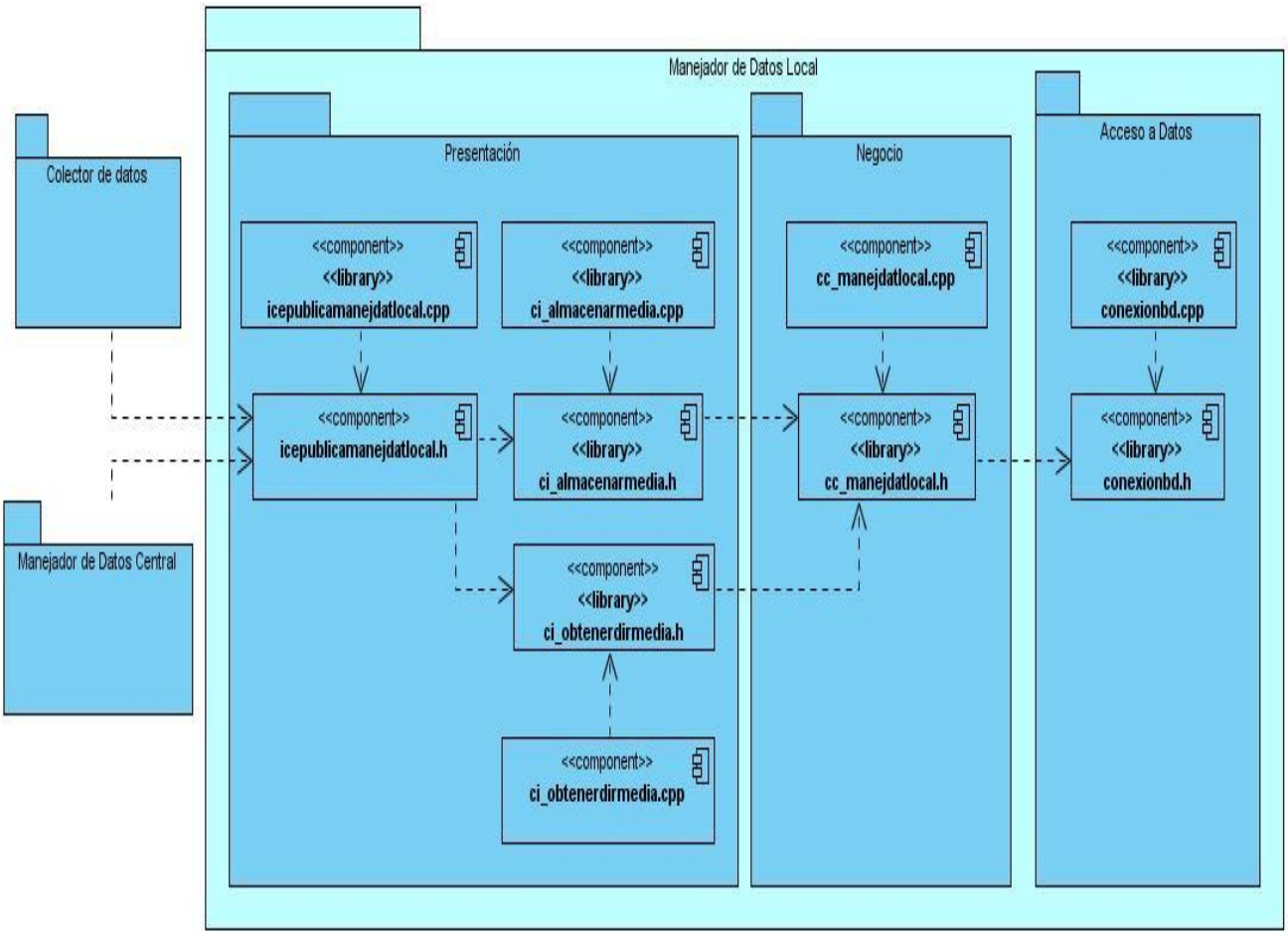

Figura 19 Diagrama de componentes del subsistema manejador de datos local

4.3. Pruebas del sistema

La creciente inclusión del software como un elemento más de muchos sistemas y la importancia de los costos asociados a un fallo del mismo, han motivado la creación de pruebas más minuciosas y bien planificadas.

Las pruebas son una actividad en la cual un sistema o componente, son ejecutados bajo unas condiciones o requerimientos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. La prueba de software es un elemento crítico para la

garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación (Departamento de Ingeniería de Software, 2010).

Métodos de pruebas:

Caja negra: Son las pruebas que se llevan a cabo sobre la interfaz del software. O sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene (Departamento de Ingeniería de Software, 2010).

Estas pruebas permiten encontrar:

- Funciones que estén incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Caja blanca: Son las pruebas que comprueban los caminos lógicos del software proponiendo casos de pruebas que se ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos, para determinar si el estado real coincide con el esperado o mencionado (Departamento de Ingeniería de Software, 2010). Por tanto, estas requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código.

Conociendo los métodos antes descritos se puede definir que para realizar las pruebas a este componente, se utilizará el método de caja blanca. Este proporcionará la calidad de las funcionalidades internas del sistema, ya que el mismo no presenta interfaz visual.

Dentro del método seleccionado se utilizará la prueba de camino básico, seleccionando al caso de uso distribuir paquetes de media, que es el más significativo que posee el sistema. Este método contiene un conjunto de pasos definidos que logran que se ejecute al menos una vez cada sentencia del código.

4.3.1. Pruebas al caso de uso distribuir paquete de media

En la figura 20 se representa el código que posee una de las funcionalidades de este caso de uso. Esta funcionalidad es la de distribuir paquete.

```
//-----Esto hace la distribucion------------------
bool CC_ManejDatCent::DistribuirPaquete(QString pDireccion)
{
    bool copio = false;
    QFile* ptrFichero = new QFile();
    ptrFichero->setFileName(pDireccion);
    ptrFichero->open(QIODevice::ReadWrite);
    //esto es para verificar si ese fichero existe.
    if(!ptrFichero->exists())
        return false;

    QString buff(2048);

    float tamanno = 0;
    tamanno = (ptrFichero->size()*1.0 / 1024) / 1024;
//------Devuelve el servidor optimo para hacer el almacenamiento
    BuscarServidorOptimo(tamanno);

    if(ptrServOpt != NULL)
    {
        QString name = pDireccion.mid(pDireccion.lastIndexOf('/')+1,pDireccion.length());

    //-----Sacar el numero de paquete---------
        int numPaq;
        QString nombPa = name.mid(name.lastIndexOf('p')+1,name.length());
        nombPa = nombPa.left(nombPa.lastIndexOf('.'));
        numPaq = nombPa.toInt();
    //----------------------------------------

        FILE* ptrFile = fopen(ptrFichero->fileName().toStdString().c_str(), "rb");

        int num = ptrConsumidorIce->DistribuirMedia(name,buff,ptrServOpt->GetIp(),ptrFile);
        if(num == 1)
        {
            ConexionBd * bd = ConexionBd::Instancia();
            bd->InsertarDirecIPsVideo(name,ptrServOpt->GetIp(),tamanno,numPaq);
            copio = true;
        }
        else
            copio = false;
    }
    ptrFichero->close();
    return copio;
}
```

1 2 3 4 5 6 7 8 9 10 11

Figura 20 Código de la funcionalidad distribuir paquete

A continuación, se representa en la figura 21 el grafo de flujo del método distribuir paquete, el cual se deriva del código representado en la figura 20:

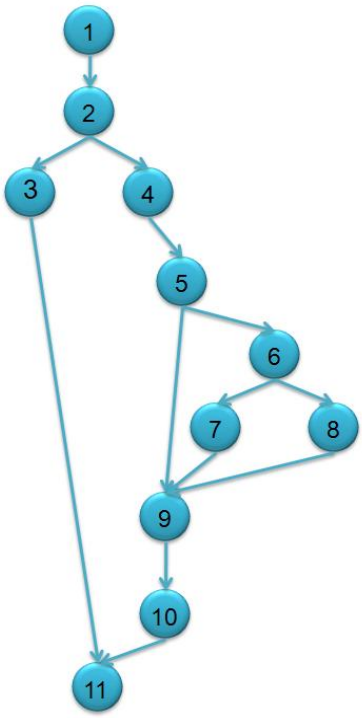

Figura 21 Grafo de flujo del método distribuir paquete

Complejidad ciclomática: es una métrica de software extremadamente útil pues proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez (Departamento de Ingeniería de Software, 2010).

De acuerdo con el concepto abordado anteriormente se define la siguiente fórmula para calcular dicha complejidad:

Complejidad ciclomática [V (G)] = Cantidad de Aristas [A] – Cantidad de nodos [N] + 2.

V (G) = A – N + 2

V (G) = 13 – 11 + 2

V (G) = 4

Después de haber realizado el cálculo de la complejidad ciclomática, se obtiene como resultado que existen cuatro caminos, los cuales se exponen en la siguiente tabla:

Tabla 9 Caminos básicos del flujo

Número	Caminos
1	1,2,3,11
2	1,2,4,5,9,10,11
3	1,2,4,5,6,7,9,10,11
4	1,2,4,5,6,8,9,10,11

Otra de las fases que proponen la prueba del camino básico, es que se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico. Por tal motivo, a continuación se realiza los casos de prueba para cada unos de los caminos encontrados.

Tabla 10 Caso de prueba para el camino básico 1

Caso de prueba para el camino básico 1	
Camino 1:	1-2-3-11
Descripción:	Los datos de entrada a esta función son los siguientes: Se debe pasar como parámetro una cadena indicando la dirección física del paquete de media que se quiere distribuir, el cual se carga y en su verificación el fichero no existe, devolviendo falso.
Entrada:	pDireccion = /home/Frank/Escritorio/Servidor/naruto_p4.wmv existe = false
Resultados esperados:	Devuelve falso, pues en esa dirección no existe ningún archivo con ese nombre.

Tabla 11 Caso de prueba para el camino básico 2

Caso de prueba para el camino básico 2	
Camino 2:	1,2,4,5,9,10,11
Descripción:	Los datos de entrada a esta función son los siguientes: Se debe pasar como parámetro una cadena indicando la dirección física del paquete de media que se quiere distribuir. Se tomaran todos los datos necesarios para buscar un servidor disponible. Se verifica si hay algún servidor disponible.
Entrada:	pDireccion = /home/Frank/Escritorio/Servidor/naruto_p4.wmv existe = true ptrServidor = NULL
Resultados esperados:	Devuelve falso, pues no hay un servidor disponible para distribuir el paquete.

Tabla 12 Caso de prueba para el camino básico 3

Caso de prueba para el camino básico 3	
Camino 3:	1,2,4,5,6,7,9,10,11
Descripción:	Los datos de entrada a esta función son los siguientes: Se debe pasar como parámetro una cadena indicando la dirección física del paquete de media que se quiere distribuir. Se tomarán todos los datos necesarios para buscar un servidor disponible. Se verifica si hay algún servidor disponible. Se comprueba si el archivo fue distribuido y copiado en el servidor escogido.
Entrada:	pDireccion = /home/Frank/Escritorio/Servidor/naruto_p4.wmv existe = true ptrServidor = 10.34.13.30 num = 0

Resultados esperados:	Devuelve falso, pues no se pudo copiar o distribuir en el servidor especificado.

Tabla 13 Caso de prueba para el camino básico 4

Caso de prueba para el camino básico 4	
Camino 4:	1,2,4,5,6,8,9,10,11
Descripción:	Los datos de entrada a esta función son los siguientes: Se debe pasar como parámetro una cadena indicando la dirección física del paquete de media que se quiere distribuir. Se tomarán todos los datos necesarios para buscar un servidor disponible. Se verifica si hay algún servidor disponible. Se comprueba si el archivo fue distribuido y copiado en el servidor escogido.
Entrada:	pDireccion = /home/Frank/Escritorio/Servidor/naruto_p4.wmv existe = true ptrServidor = 10.34.13.30 num = 1
Resultados esperados:	Devuelve verdadero, pues se pudo copiar o distribuir en el servidor especificado.

4.4. Conclusiones parciales

En este capítulo se presentó la descripción del sistema propuesto, también se construyó el modelo de implementación el cual refleja las relaciones del sistema manejador de datos con otros componentes. También se modelaron los diagramas del sistema, donde se establecen las relaciones internas de este.

Además, en este capítulo se hicieron las pruebas necesarias para garantizar la calidad del mismo. Para estas pruebas fue escogido el método de caja blanca y dentro de este, el de camino básico por ser el que más asocia con las características que presenta este componente. Esta prueba permitió realizar un examen minucioso sobre el algoritmo que se utilizó para distribuir los paquetes de media, teniendo en

cuenta las decisiones lógicas en las vertientes verdadera y falsa que este posee. También se tuvieron en cuenta las características que poseen las estructuras internas de datos utilizadas, garantizando que al menos una vez, todos los caminos independientes del algoritmo fueran visitados.

CONCLUSIONES

Los resultados obtenidos con la realización del presente trabajo han dado cumplimiento al objetivo propuesto en esta investigación:

- Se establecieron las bases teóricas para dar solución al problema de la investigación, partiendo de los diferentes métodos de almacenamiento en los Servidores Streaming existentes, permitiendo tener los conocimientos básicos para realizar el desarrollo del componente.
- Se realizó el análisis, diseño e implementación de los requisitos definidos para el componente Manejador de Datos, logrando crear el sistema propuesto.
- Se aplicó el método de pruebas de caja blanca, específicamente el de camino básico. Este se le realizó al CU distribuir paquetes de media que es el más importante que posee el sistema, obteniendo resultados satisfactorios.
- Con el desarrollo de este sistema, se logró mejorar la distribución del almacenamiento de las medias y a su vez, la eficiencia en la recuperación de las mismas.

RECOMENDACIONES

Después de concluir con un estudio profundo y de haber realizado el componente Manejador de Datos, se tienen las siguientes recomendaciones:

- Continuar el desarrollo de la investigación con el objetivo de incluir otras funcionalidades al sistema.
- Crear un módulo, que sea capaz de garantizar la seguridad del componente.
- Mejorar la implementación del proceso de comunicación con los demás componentes que se comunican con él, haciendo uso de la tecnología ICE, mediante el método asíncrono.
- Hacer una búsqueda de métodos efectivos para que el subsistema Manejador de Datos Central pueda tener réplicas en otros servidores, lo cual contribuirá al mejoramiento del sistema.

BIBLIOGRAFÍA

1. **Attiya, Hagit y Welchr, Jennife. 2004.** *Distributed Computing: Fundamentals,Simulations and Advance Topics.* New Jersey : John Wiley & Sons, 2004.
2. **Bello Pérez, Dr. Rafael. 1998.** *Métodos de Solución de Problemas para la Inteligencia Artificial.* Grupo de Investigaciones en Inteligencia Artificial, Universidad Central de Las Villas "Marta Abreu". Santa Clara : s.n., 1998.
3. **BOOCH, IVAR JACOBSON. 1999.** *El Proceso Unificado de Desarrollo de Software.* 1999.
4. **Coulouris, George, Dollimore, Jeam y Kindberg, Tim. 2001.** *Sistemas Distribuidos: Conceptos y Diseño.* Madrid : Pearson Educación S.A, 2001.
5. **Departamento de Ingeniería de Software. 2010.** *Material de caja blanca y caja negra.* Ciudad Habana : s.n., 2010.
6. **—. 2010.** *Sobre_la_disciplina_de_Prueba.* Ciudad Habana : s.n., 2010.
7. **Departamento Inteligencia Artificial. 2008.** *Métodos informados heurísticamente.* Departamente de técnicas de programación, Universidad de las Ciencias Informáticas. Ciudad habana : s.n., 2008. Conferencia.
8. **Free Download Manager. 2007.** Free Down Load Manager. [En línea] Mayo de 2007. [Citado el: 20 de Febrero de 2011.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p.
9. **González Jiménez, Yasiel. 2011.** *Arquitectura del servidor de streaming distribuido ALLFRY´S.* Universidad de las Ciencias Informaticas. La Habana : s.n., 2011. Trabajo de Diploma.
10. **Gonzalo Álvarez, Marañón. 1999.** Java. [En línea] 1999. [Citado el: 19 de Febrero de 2011.] http://www.iec.csic.es/criptonomicon/java/quesjava.html.
11. **H ISTCHFELD, Nancy y CARO, Patricio S.** Departamente de Ciencias de la Computacion. *Departamente de Ciencias de la Computacion.* [En línea] [Citado el: 18 de Febrero de 2011.] http://www.dcc.uchile.cl/ psalinas/uml.
12. **Henning, Michi y Spruiell, Mark. 2009.** *Distributed Programming with Ice.* 2009.

13. **HispaNetwork Publicidad y Servicios, S.L. 2004.** Interbusca. *Interbusca.* [En línea] HispaNetwork Publicidad y Servicios, S.L., 2004. [Citado el: 1 de Diciembre de 2010.] http://antivirus.interbusca.com/glosario/SERVIDOR.html.
14. **Larman, Craig. 2003.** *UML y Patrones.* Segunda edición. s.l. : Prentice Hall, 2003.
15. **Leyva Cortina, Aniuska y Díaz Benítez, Dairelys. 2010.** *Subsistema genérico de gestión y archivo de datos para videojuegos.* Ciudad de la Habana : s.n., 2010.
16. **López Amado, José Augusto. 2010.** *Subsistemas de Administración y Catalogación del Sistema de Captura y Catalogación de Medias (CCM).* Universidad de las Ciencias informáticas. Ciudad Habana : s.n., 2010. TRABAJO DE DIPLOMA.
17. **López Costa, Aliana y Garcia Padrón, Liset. 2010.** *Módulo para la gestión de metadatos geográficos de LiberMaps.* Ciudad de la Habana : s.n., 2010.
18. **Martínez Pérez, Reynier. 2010.** *Subsistema de Almacenamiento y Gestión de Archivos para el producto Captura y Catalogación de Medias.* Ciudad de la Habana : s.n., 2010.
19. **Méndez Terrero, Yaniley. 2010.** *Conceptualización del subsistema de transmisión de noticias como aplicación de escritorio para la Plataforma de Televisión Informativa.* Ciudad de la Habana : s.n., 2010.
20. **Pérez Vázquez, Maridalia y Rodríguez Hidalgo, Guillermo William. 2010.** *Desarrollo de componentes de interfaz de usuario para la representación de información mediante gráficas en el Polo PetroSoft.* Ciudad de La Habana : s.n., 2010.
21. **Progress. 2009.** Progress, Software y Servicio. [En línea] 2009. [Citado el: 15 de Febrero de 2011.] http://www.progress.com.py/rup.php.
22. **Rich, Elaine y Knight, Kevin. 1994.** *Inteligencia Artificial.* Segunda edición. Madrid : McGraw-Hill/Interamericana de España, S.A, 1994. 84-481-1858-8.
23. **Rijo Sciara, Daniel. 2004.** *Fundamentos de Video Streaming.* Montevideo, Uruguay : Instituto de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de la República Montevideo, 2004.
24. **Santos Sanabria, Cesar y Colomina Curbelo, Luisdey. 2010.** *Componente Evaluador de Expresiones Matemáticas.* Ciudad de la Habana : s.n., 2010.
25. **Sheu, Simon, Hua, Kien A. y Tavanapong, Wallapak. 1997.** *Chaining: A generalized batching technique for video-on-demand system.* s.l. : Proceeding, IEEE International Conference on Multimedia Computing and Systems, 1997.

26. **Suárez Pérez, Jean Michael y Estrada Velazco, Aylin. 2008.** *Sistema de Captura e Indexación de video.* Universidad de las Ciencias Informáticas. Ciudad Habana : s.n., 2008.
27. **Tanenbaum, Andrew y Steem Maarten, Van. 2002.** *Distributed Systems: Principles and Paradigms.* [Documento] Amsterdam : VU Amsterdam, Dept. Computer Science, 2002.
28. **Torres, Angel Laguna. 2009.** *Propuesta de Servidor Streaming de software libre para la captura y transmisión de video y sonido digital.* [Documento] Ciudad Habana : s.n., 2009.
29. **Tran, D.A., Hua, Kien A. y Do, T.T. 2004.** *A Peer-to-Peer, Arquitecture for Media Streaming.* s.l. : Dept. of Comput. Sci., Univ. of Dayton, 2004. 0733-8716.
30. **Tusch, Roland, y otros. 2004.** *Offensive and Defensive Adaptation in Distributed Multimedia Systems.* Klagenfurt, Austria : Institute of Information Technology, Klagenfurt University,, 2004.
31. **Valdez Altamirano, Alfonso. 2006.** *Comparativo de Entornos de Desarrollo Integrados.* 2006.
32. **Vallejo Fernández, David. 2006.** *Documentación de ZeroC ICE.* UNIVERSIDAD DE CASTILLA-LA MANCHA. Castilla-La Mancha : s.n., 2006.
33. **Vargas Colores, Juan Miguel. 2008.** *Estudio comparativo de sistemas de difusión de video afluente.* [Documento] Tijuana, México : s.n., 2008. B061184.
34. **Veríssimo, Paulo y Rodrigues, Luís. 2001.** *Distributed Systems for System Architects.* s.l. : Kluwer Academic Publishers, 2001.

GLOSARIO DE TÉRMINOS.

API: Application Programming Interface - Interfaz de Programación de Aplicaciones es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Aplicación: Es una clase de programa informático creado para facilitar al usuario un determinado tipo de trabajo. Esto lo caracteriza frente a otros programas como los sistemas operativos, las utilidades y los lenguajes de programación.

Cliente: Es una aplicación informática cuya función es acceder a los servicios que ofrece un servidor, haciendo uso generalmente de una red de telecomunicaciones.

Clúster: Grupo de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio.

CPU: Central Processing Unit (unidad de proceso central). La CPU es el cerebro del ordenador, es donde se producen la mayoría de los cálculos. En términos de potencia del ordenador, la CPU es el elemento más importante de un sistema informático.

Deadlock: Se denomina a la acción de interbloqueo en una base de datos. Este proceso ocurre si cada proceso del conjunto está esperando un evento que solo otro proceso del conjunto puede causar. Puesto que todos los procesos están esperando, ninguno de ellos puede causar ninguno de los eventos que podrían despertar a cualquiera de los demás miembros del conjunto, y todos los procesos continúan esperando indefinidamente.

Frameworks: Estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

GEYSED: Centro Geoinformática y Señales Digitales.

GOF: Gang-of-Four “Pandilla de los cuatro”.

GRASP: Patrones de Software para la asignación General de Responsabilidad.

IDE: Entorno de Desarrollo Integrado.

Licencia GNU/GPL: Está diseñada para asegurar la libertad de distribuir copias de Software Libre (y cobrar por ese servicio), asegurar que recibirá el código fuente del programa o bien podrá conseguirlo si quiere, asegurar que puede modificar el programa o modificar algunas de sus piezas para un nuevo programa y para garantizar que puede hacer todas estas cosas.

Multimedia: Se aplica a cualquier objeto que utilice simultáneamente diferentes formas de contenido informativo como texto, sonido, imágenes y video para informar o entretener al usuario.

Multiplataforma: Se utiliza el término para denominar a los programas, lenguajes de programación u otra clase de software que pueden brindar sus prestaciones funcionando sobre diversas combinaciones de hardware y software.

Protocolo: En informática es un método estándar que permite la comunicación entre procesos. Comprende un conjunto de reglas y procedimientos que deben respetarse para el envío y la recepción de datos a través de una red.

RAM: Random Access Memory (Memoria de acceso aleatorio). Un tipo de memoria de ordenador a la que se puede acceder aleatoriamente; es decir, se puede acceder a cualquier byte de memoria sin acceder a los bytes precedentes.

RUP: Proceso Unificado de Rational, o en sus siglas en inglés: Rational Unified Process.

Servidor: Software u ordenador que provee servicios a otros programas o equipos denominados clientes.
SGBD: Sistema de Gestor de Base de Datos.

SSD: Servidor de Streaming Distribuido.

TDT: Televisión Digital Terrestre.

TIC: Tecnologías de la Información y las Comunicaciones.

UCI: Universidad de las Ciencias Informáticas.

UML: Lenguaje Unificado de Modelado.