Universidad de las Ciencias Informáticas

Título:"Desarrollo de la arquitectura del proyecto Captura y Catalogación de Medias".

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor:

Yoandri Quintana Rondón

Tutor:

Ing. Karlen Trimiño Pérez

Ciudad de la Habana, Mayo, 2009

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 2009.

_____	_____
Firma del Autor	Firma del Tutor

DATOS DE CONTACTO

Síntesis del Tutor:

Nombre y Apellidos: Karlen Trimiño Pérez

Especialidad: Ingeniería en Ciencias Informáticas.

Años de experiencia: 2

Teléfono particular: +53 (42) 22 – 2016, +53 (52) 93 -- 0320

Dirección electrónica para correspondencia: ktrimino@uci.cu, karlen1983@gmail.com

PENSAMIENTO

"Una máquina puede hacer el trabajo de 50 hombres corrientes. Pero no existe ninguna máquina que pueda hacer el trabajo de un hombre extraordinario."

Elbert Green Hubbard

AGRADECIMIENTOS

A mi madre por su amor, su cariño y la confianza que depositó en mí durante todo este tiempo, enseñándome siempre que lo más importante en los hombres se lleva en el corazón.

A mis amigos, por ser mi familia y poder contar con ellos en todo momento, dejando claro que haberlos conocidos ha sido uno de los grandes regalos que me ha dado la vida.

A mi tutor por darme fuerzas para seguir adelante y por confiar en mí, por su apoyo y dedicación, a los profesores que me impartieron clases, al equipo de trabajo del proyecto Captura y Catalogación de Medias que todos de una forma u otra hicieron posible la realización de este trabajo.

A mi oponente y los miembros del jurado por sus oportunas críticas.

A la revolución y a Fidel por darnos la oportunidad de estudiar en esta maravillosa Universidad.

En fin a todos aquellos que siempre confiaron en mí.

DEDICATORIA

Dedico el presente trabajo al ser más dulce de mi vida, mi madre: Luz María Rondón Mora, luz, faro y guía de cada uno de mis actos.

A mis abuelos, mis hermanos, mi padre en fin a toda mi familia que es bastante grande pero no por eso se les quiere menos.

En especial a mis amigos, quienes desde un principio, cuando prácticamente me encontraba solo en este centro me brindaron su apoyo y confianza en todo momento, los cuales más que amigos para mí son mis hermanos.

A todos muchas gracias….

Yoandri

Resumen

En los últimos años el avance de las Tecnologías de la Información y las Comunicaciones (TIC) han despertado un gran interés en las comunidades de desarrollo del mundo donde Cuba no se encuentra ausente. Las necesidades actuales que tiene toda organización para el logro de sus objetivos, demandan la construcción de grandes y complejos sistemas de software que requieren de la combinación de diferentes tecnologías y plataformas de hardware y software para alcanzar un funcionamiento acorde con dichas necesidades. Lo anterior, exige de los profesionales dedicados al desarrollo de software poner especial atención y cuidado al diseño de la arquitectura, bajo la cual estará soportado el funcionamiento de sus sistemas.

Hoy día muchos sistemas fracasan o no alcanzan el total de los requerimientos establecidos porque muchas veces la arquitectura de software se encuentra deficiente en su concepto o diseño, o quizás no cuentan con la arquitectura del software adecuada para el sistema que se desea desarrollar. Este trabajo se centra hacia el desarrollo de una arquitectura robusta y confiable que cumpla con todos los requisitos necesarios para el desarrollo del proyecto Captura y Catalogación de Madias (CCM). El mismo debe permitir organizar el desarrollo, fomentar la reutilización, comprender y hacer evolucionar el sistema.

Índice de Contenido

INTRODUCCIÓN **1**

CAPÍTULO 1 ESTADO DEL ARTE Y FUNDAMENTACIÓN TEÓRICA **5**

1.1 Introducción 5
1.2 Características del Sistema 5
1.3 La Arquitectura de Software 6
1.4 Patrones 7
1.4.1 Clasificación de Patrones 7
1.5 Estilos Arquitectónicos 8
1.5.2 Patrón en Capas 9
1.5.3 Patrón Modelo-Vista-Controlador (MVC) 11
1.5.4 Arquitecturas Orientadas a Objetos (AOO) 11
1.5.5 Arquitecturas Orientadas a Servicios (SOA) 12
1.7 Análisis de soluciones existentes 13
1.7.1 MEDIABOX 13
1.7.2 Videoma 14
1.8 Conclusiones del Capítulo 16

CAPÍTULO 2 HERRAMIENTAS PARA LA SOLUCIÓN **17**

Introducción 2.1 17
2.2 ¿Qué metodología utilizar? 17
2.2.1 Rational Unified Process (RUP) 17
2.2.1.1 Descripción general 17
2.2.1.2 Flujos de trabajo de RUP 18
2.2.1.3 Características de RUP 18
2.2.1.4 Fases de RUP 18
2.2.2 Extreme Programing (XP). 18
2.2.2.1 Fases de Desarrollo de XP 18
2.3 Lenguaje Unificado de Modelado (UML) 20
2.4 Herramientas CASE 20
2.4.1 Visual Paradigm 21
2.4.2 Rational Rose Enterprise 21
2.5 Lenguaje de Programación 22
2.5.1 C++ 22
2.5.2 Java 23
2.6 Frameworks 25
2.6.1 Framework Swing 25
2.6.2 Framework Spring 27
2.6.3 Framework Hibernate 28

2.7 Entornos de Desarrollo Integrado (IDE) 30
2.7.1 Eclipse 30
2.7.2 NetBeans 30
2.7.3 QT Creator 31
2.8 Sistemas Gestores de Base de datos 33
2.8.1 MySQL 34
2.8.2 Oracle 34
2.8.3 PostgreSQL 35
2.9 Software para el Control de Versiones 37
2.10 Salvas Automáticas 38
2.10.1 Bacula 38
2.11 Conclusiones del Capítulo 39
CAPÍTULO 3 PROPUESTA DE LA ARQUITECTURA DEL SISTEMA 40
3.1 Introducción 40
3.2 Estructura del equipo de Desarrollo 40
3.2.1 Herramientas de Desarrollo 40
3.2.2 Configuración de los puestos de trabajos 41
3.3 Organigrama de la arquitectura 42
3.4 Módulos del Sistema 47
3.5 Descripción de la Arquitectura 49
3.5.1 Vista de Caso de Uso 49
3.5.2 Vista Lógica 54
3.5.3 Vista del Modelo de Análisis 60
3.5.4 Vista del Modelo de Diseño 60
3.5.5 Vista de Procesos 61
3.5.6 Vista de Implementación 62
3.5.7 Vista de Despliegue 66
3.5.7.1 Diagrama de Despliegue 66
3.6 Requerimientos no Funcionales 73
3.6.1 Requerimientos de Seguridad 74
3.6.2 Restricciones de acuerdo a la estrategia de diseño 74
3.6.3 Portabilidad, escalabilidad, reusabilidad 75
3.6.4 Redes 75
3.6.5 Soporte 75
3.6.6 Usabilidad 75
3.6.7 Apariencia o Interfaz Externa 76
3.6.8 Rendimiento 76
3.6.9 Requerimientos de Software 76
3.6.10 Requerimientos de Hardware 76
3.7 Conclusiones del Capítulo 78

CAPÍTULO 4 EVALUACIÓN DE LA ARQUITECTURA DEL SISTEMA 79

4.1 INTRODUCCIÓN 79
4.2 ANÁLISIS DE ARQUITECTURAS DE SOFTWARE 79
4.3 METODOLOGÍA PROPUESTA PARA EVALUAR LA AS 81
4.4 ESTRATEGIAS DE PRUEBA DE LA AS 84
4.5 VALORACIÓN DE PROPUESTAS ARQUITECTÓNICAS SIMILARES 91
4.5 CONCLUSIONES DEL CAPÍTULO 96

CONCLUSIONES GENERALES 97

RECOMENDACIONES 98

GLOSARIO DE TÉRMINOS 99

REFERENCIAS BIBLIOGRÁFICAS 100

BIBLIOGRAFÍA 103

ANEXOS 106

Índice de Figuras

Figura 1: Patrón Modelo Vista Controlador (MVC) 11
Figura 2: Resultados para Consultas Complejas 36
Figura 3: Estructura de capas del patrón en Capas aplicada al sistema 42
Figura 4: Aspectos de la directiva de seguridad 44
Figura 5: Vista de casos de uso del módulo Administración 51
Figura 6: Vista de casos de uso del módulo Captura e Indexación de Video 52
Figura 7: Vista de casos de uso del módulo Captura y Transcripción de Audio 53
Figura 8: Vista de casos de uso del módulo Catalogación de Medias 53
Figura 9: Vista general de casos de uso del sistema 54
Figura 10: Vista Lógica 55
Figura 11: Paquetes de la Capa de Presentación 56
Figura 12: Paquetes de la Capa de Negocio 57
Figura 13: Paquetes de la Capa de Acceso a Datos 59
Figura 14: Paquetes de la Capa Seguridad 60
Figura 15: Diagrama de Procesos 61
Figura 16: Vista general del modelo de implementación 62
Figura 17: Componentes de la capa Presentación 63
Figura 18: Componentes de la capa Negocio 64
Figura 19: Componentes de la capa Acceso a Datos 64
Figura 20: Componentes de la capa Seguridad 65
Figura 21: Diagrama de Despliegue Recomendado 67
Figura 22: Diagrama de Despliegue Propuesto 68
Figura 23: Diagrama de Despliegue Optimo 69
Figura 24: Nodo Consola de Audio 70
Figura 25: Nodo Grabación de Audio 70
Figura 26: Nodo Dispositivo de Almacenamiento 70
Figura 27: Nodo Antena 71
Figura 28: Nodo Cámara 71
Figura 30: Nodo Captura de Video 71
Figura 30: Nodo Servidor de BD PostgreSQL 72
Figura 31: Nodo Servidor de Análisis, Streaming y Medias 72
Figura 32: Nodo de Catalogación, Transcripción y Presentación 73
Figura 33: Nodo de Administración y Planificación 73
Figura 34: Arquitectura del sistema Videoma (37) 95
Figura 35: Flujos de Trabajo, Fases y esfuerzo por Iteración 106
Figura 36: Flujos de Trabajo de la Metodología XP 107
Figura 37: Representación del framework Spring 108
Figura 38: Representación del framework Hibernate 108

Índice de Tablas

Tabla 1: Interfaces del framework Hibernate 28
Tabla 2: Equipo de Desarrollo 41
Tabla 3: Tecnologías utilizadas por capas 47
Tabla 4: Cantidad de Casos de Uso del sistema por módulos 50
Tabla 5: Atributos de calidad 84
Tabla 6: Estrategia para Evaluar la AS 88
Tabla 7: Resultados de la Evaluación 88
Tabla 8: Fases de la Metodología RUP 107

Introducción

En el mundo la demanda de equipos y sistemas de comunicación ha evolucionado a gran velocidad dando un vuelco hacia la era de las tecnologías. Uno de los factores más desarrollado en este ámbito es el trabajo con medias porque estas son indispensables para las personas y de suma importancia para las empresas, comúnmente se usan para dar capacitación o motivar al personal hacia alguna actividad específica. Es tan grande la demanda de las medias que los procesos de captura y catalogación de medias se realizan de forma automatizada en varios países, casi todos potencias en informática, pero en su gran mayoría lo hacen utilizando herramientas propietarias.

Cuba no está alejada al desarrollo de las TIC y actualmente se trabaja en la informatización de la sociedad, pero el trabajo con medias aun es desarrollado de forma manual y existen personas encargadas de realizar las grabaciones de los diferentes programas de noche, donde los archivos de audio y video son almacenados sin hacer una descripción previa de los mismos, lo cual implica retardo en la ejecución de los procesos y un margen de error relacionado con los fallos que pueda cometer la persona que los realiza por demoras o descuidos. A estas personas se les debe pagar por ese horario nocturno, y eso con la automatización de los procesos se evita, siendo más económico para el país y evitando mayores esfuerzos en los trabajadores. Una de las fuentes promotoras del desarrollo de la informática en Cuba es La Universidad de las Ciencias Informáticas (UCI). Este centro cuenta con 10 facultades para el desarrollo de software donde trabajan estudiantes, profesores y otros profesionales. La facultad 9 entre uno sus polos presenta el de Video y Sonido Digital encargado del trabajo con medias. En la UCI también los procesos de captura y catalogación de medias se hacen de forma manual y esto ha provocado que en algunos momentos las transmisiones no funcionen de manera eficiente. A partir de esta situación es que surge el proyecto Captura y Catalogación de Medias en el polo de Video y Sonido Digital con el objetivo de automatizar los procesos de captura y catalogación de medias haciendo uso de herramientas libres.

En un primer estudio se analizó que no todos los sistemas logran el éxito y una de las causas para su fracaso es el mal diseño de la arquitectura. El desarrollo de la arquitectura de software es una de las etapas fundamentales y, en muchos casos, la más importante en el desarrollo de software, pues es aquí donde se aportan todos los conocimientos, creatividad y experiencia para crear la mejor propuesta de

solución que se le dará al cliente, que cumpla con los requerimientos establecidos para el sistema en desarrollo, así como sus preocupaciones principales de lo que esperan del sistema. Según estudios realizados para lograr el desarrollo del proyecto Captura y Catalogación de Medias es necesario diseñar una arquitectura de software robusta que cumpla con todas las necesidades para lograr una integración uniforme del mismo y garantizar que el producto resultante tenga la calidad requerida.

De la situación anterior se identificó el siguiente **problema científico**: ¿Cómo diseñar un mecanismo que permita la gestión rápida, robusta y confiable de las medias en el proyecto CCM del polo Video y Sonido Digital de la facultad nueve?

Atendiendo las necesidades del proyecto el **objeto de estudio es:** Las arquitecturas usadas en el proceso de desarrollo de software y el **campo de acción:** La arquitectura de software para el proyecto Captura y Catalogación de Medias.

Como **Idea a defender** se tiene que el diseño de una arquitectura robusta que cumpla con los requerimientos y atributos de calidad establecidos para el sistema de CCM con el uso de herramientas libres garantizará un esquema para la realización del producto.

El **objetivo general** de este trabajo es diseñar una arquitectura de software para el proyecto CCM que permita que el sistema sea modificable, interoperable, seguro, funcional y disponible, y cumpla con los requerimientos apoyándose en el uso de herramientas libres.

Para lograr este objetivo se desarrollan las siguientes **Tareas de la investigación:**

1. Evaluar los sistemas de captura y la catalogación de medias a través del estudio de los mismos.
2. Valorar el estado del arte de varios estilos y patrones de la arquitectura de software y tomar posición en cuanto al más idóneo para este tipo de aplicación.
3. Definir las herramientas y tecnologías a utilizar para el desarrollo.
4. Plantear la propuesta arquitectónica del sistema CCM.
5. Valorar la metodología a seguir para realizar las estrategias de prueba enfocada al sistema de Captura y Catalogación de Medias
6. Diseñar al menos 2 estrategias de prueba para la arquitectura de software seleccionada.

7. Valorar al menos 2 casos donde se hayan utilizado propuestas arquitectónicas en sistemas de similar propósito a partir de consultar la bibliografía especializada.

Para la realización de estas tareas se emplearon los diferentes **métodos de la investigación.**

Métodos Teóricos

- "Histórico – Lógico": Para determinar las tendencias actuales, en este caso se empleó para realizar el estudio de la trayectoria real de determinados elementos que servirán de guía para la construcción de una buena arquitectura de software.

- "Analítico - Sintético": Permite procesar toda la información en particiones más pequeñas y fácil de comprender, y después compactar esas partes que fueron analizadas formándolas más simples y con ideas más claras y concisas.

- "Modelación": En este trabajo es muy importante la modelación para desarrollar una buena arquitectura, es por eso que se utilizó este método para crear modelos con vista a investigar la realidad.

Métodos Empíricos

- "Observación": Mediante este método se realizó la investigación para conocer los detalles fundamentales sobre el desarrollo de la arquitectura para el proyecto CCM.

El documento está constituido por cuatro capítulos.

Capítulo 1.Estado del Arte y Fundamentación Teórica

En este capítulo se abordará los elementos relacionados con el estudio del estado del arte y la fundamentación teórica. Se realiza un estudio de las arquitecturas más comunes para determinar cuál es la adecuada para desarrollar el proyecto antes de iniciar el trabajo, apoyando la investigación en el estudio de los sistemas de capturas y catalogación de medias existentes y en las características fundamentales del sistema.

Capítulo 2. Analizar las herramientas y tecnologías a utilizar para el desarrollo.

Se dispone de un análisis sobre las principales herramientas, metodologías de desarrollo, lenguajes de programación, ambientes de desarrollo entre otras definiendo finalmente cuáles son las apropiadas para el desarrollo del sistema.

Capítulo 3.Propuesta de la Arquitectura del Software

En esta sección se reflejará la propuesta de la Arquitectura para el proyecto CCM basada en el estudio realizado en los capítulos anteriores.

Capítulo 4.Evaluación de la Arquitectura

En este capítulo se realizará la evaluación de la arquitectura desarrollada mediante el diseño de dos estrategias de pruebas. Además se realiza una comparación con dos sistemas de similar propósito.

Capítulo 1
Estado del Arte y Fundamentación Teórica

1.1 Introducción

En este capítulo se aborda sobre el análisis del estado del arte de la Arquitectura de Software, partiendo del estudio de los sistemas de Captura y Catalogación de Medias ya existentes y basados en los principales conceptos de arquitectura, los diferentes estilos arquitectónicos y los patrones de la arquitectura. Además se definen las principales características que debe tener el sistema para a partir de estas desarrollar la arquitectura.

1.2 Características del Sistema

El sistema a desarrollar debe ser seguro y funcional. El mismo debe permitir:

- Captura de señales de video según planificaciones definidas.
- Fácil cambio de la señal que se está capturando en cada momento.
- Almacenamiento de los ficheros de video capturados en un archivo.
- Grabación de señales de audio o radio FM.
- Visualización de los materiales audiovisuales almacenados.
- Catalogación de los materiales audiovisuales almacenados.
- Trascripción de ficheros de audio.

Para lograr con éxito todas las funcionalidades antes mencionadas se debe tener en cuenta que el sistema:

- Debe ser desarrollado con herramientas libres.
- Las herramientas seleccionas deben ser optimas para cualquier plataforma permitiéndole al sistema que en futuras versiones sea multiplataforma.
- Utilizar un sistema gestor de base de datos lo más seguro posible.
- Dar respuesta rápida ante cada uno de los procesos (captura, transcripción, catalogación y otros) que debe brindar el sistema.

A partir de los aspectos mencionados sobre el sistema se realiza un estudio para el desarrollo de la arquitectura del sistema CCM.

1.3 La Arquitectura de Software

Los primeros antecedentes de la arquitectura de software surgen en la década de 1960 pero estos no tuvieron un seguimiento continuo y no es hasta 1992 que se conoce la expresión AS por Perry y Wolf. Hoy existen varias definiciones entre las que se encuentran las siguientes:

…"Arquitectura de software es el estudio de la estructura a gran escala y el rendimiento de los sistemas de software. Aspectos importantes de la arquitectura de un sistema incluye la división de funciones entre los módulos del sistema, los medios de comunicación entre módulos, y la representación de la información compartida." (1)

"La AS es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones". (2)

Según la IEEE std. 14_71-2000: "la Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución."(3)

Es decir la AS establece los fundamentos necesarios para que analistas, diseñadores, programadores, y los demás integrantes del equipo de desarrollo trabajen en una línea común que permita alcanzar los objetivos del sistema logrando cubrir todas las necesidades.

Es importante resaltar que la arquitectura tiene una relación directa con los requerimientos no funcionales de un sistema, los que permiten definir escenarios para la descripción arquitectónica. Los requerimientos funcionales se satisfacen mediante modelado y diseño de la aplicación.

La Arquitectura de Software a grandes rasgos es:

- Una vista estructural de alto nivel.
- Define estilos o combinación de estilos para dar una solución.
- Se concentra en los requerimientos no funcionales.

1.4 Patrones

Según Christopher Alexander en el libro *"The Timeless Way of Building"* define al patrón en la siguiente manera:

"Cada patrón es una regla de 3 partes, que expresa una relación entre un contexto, un problema y una solución. Como un elemento en el mundo, cada patrón es una relación entre un contexto, un sistema de fuerzas que ocurren repetidamente en ese contexto y una configuración espacial que permite que esas fuerzas se resuelvan entre sí." (4)

"Como elemento de un lenguaje, un patrón es una instrucción que muestra como puede ser usada esta configuración espacial una y otra vez para resolver el sistema de fuerzas, siempre que el contexto lo haga relevante." (4)

"Cada patrón describe un problema que ocurre una y otra vez en el entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo siquiera dos veces de la misma forma". (4)

A partir de estas definiciones un patrón se puede ver de una manera más simple como un par problema/solución con nombre, que se puede aplicar en nuevos contextos; con consejos acerca de cómo aplicarlo en nuevas situaciones y discusiones sobre sus compromisos. Además conducen a arquitecturas más pequeñas, más simples y más comprensibles.

1.4.1 Clasificación de Patrones

En el libro *Pattern Oriented Software Architecture* de Buschmann se definen tres tipos de patrones:

- **Patrones arquitectónicos** sobre aspectos fundamentales de la estructura de un sistema software. Especifican un conjunto predefinido de módulos con sus responsabilidades y una serie de

recomendaciones para organizar los distintos componentes. Es decir son aquellos que organizan la estructura del software.

- **Patrones de diseño** sobre aspectos relacionados con el diseño de los subsistemas. Por tanto se centran en aspectos más específicos.
- Un ***Idiom*** que es un patrón de bajo nivel específico de un lenguaje de programación o entorno de desarrollo. (5)

1.5 Estilos Arquitectónicos

Según Carlos Reynoso y Nicolás Kiccillof plantean que un estilo arquitectónico provee una colección de elementos edificadores del diseño en bloque, reglas y restricciones para componer los bloques constructivos, y las herramientas para analizar y manipular los diseños creados en el estilo. Los estilos generalmente proveen guía y análisis para crear una clase amplia de arquitecturas en un dominio específico donde los patrones se enfocan en solucionar los problemas más pequeños, más específicos dentro de un estilo dado. (6)

Un estilo arquitectónico define a una familia de sistemas en términos de un patrón de organización estructural. Específicamente, un estilo arquitectónico determina el vocabulario de componentes y conectores que pueden ser usado, así como un conjunto de restricciones de cómo pueden ser combinados.

Se puede considerar un estilo arquitectónico como un conjunto coordinado de restricciones arquitectónicas que restringe los rasgos de los elementos arquitectónicos y las relaciones permitidas entre esos elementos dentro de la arquitectura que se conforma a ese estilo. Cuando se habla de estilo arquitectónico se deben tener presente las famosas cuatro C, componentes, conectores, configuración y c*onstraint* (restricciones).

1.5.1 Ejemplos de Estilos Arquitectónicos

- **Estilos de Flujo de Datos**
  - Tubería y filtros.
- **Estilos Centrados en Datos**
  - Arquitecturas de Pizarra o Repositorio.

- **Estilos de Código Móvil**
  - Arquitectura de Máquinas Virtuales.
- **Estilos Peer-to-Peer**
  - Arquitecturas Basadas en Eventos.
  - Arquitecturas Orientadas a Servicios (SOA).
  - Arquitecturas Basadas en Recursos.
- **Estilos de Llamada y Retorno**
  - *Model-View-Controller* (MVC).
  - Arquitecturas en Capas.
  - Arquitecturas Orientadas a Objetos.
  - Arquitecturas Basadas en Componentes.
- **Estilos Derivados**
  - C2
  - *GenVoca*
  - *Rest*
- **Estilos Heterogéneos**
  - Sistema de Control de Procesos
  - Arquitecturas Basadas en Atributos (6)

1.5.2 Patrón en Capas

El patrón en capas es definido como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. (7)

En un patrón en capas, los conectores se definen mediante los protocolos que determinan las formas de la interacción. Los diagramas de sistemas clásicos dibujan las capas en adyacencia, sin conectores, flechas ni interfaces; en algunos casos se suele representar la naturaleza jerárquica del sistema en forma de círculos concéntricos.

Las restricciones topológicas del patrón pueden incluir una limitación, más o menos rigurosa, que exige a cada capa operar sólo con capas adyacentes, y a los elementos de una capa entenderse sólo con otros

elementos de la misma; se supone que si esta exigencia se relaja, el patrón deja de ser puro y pierde algo de su capacidad heurística**.** (8)

Para la realización de este sistema es muy importante el uso de este patrón porque la arquitectura en capa define el patrón en capas como una organización jerárquica, lo que posibilita un diseño basado en niveles de abstracción creciente, posibilitando a los implementadores, fraccionar un problema en una secuencia de pasos incrementales. Este estilo de desarrollo en varios niveles, facilita que en caso que ocurra algún cambio, sólo se tendrían que realizar las correcciones necesarias en el nivel requerido sin tener que revisar código de otros niveles. Los sistemas o arquitecturas en capas constituyen uno de los patrones que aparecen con mayor frecuencia.

El objetivo primordial del patrón en capas es dividir, fraccionar y llegar a separar la Presentación (donde muestras y obtienes datos), de la Lógica de Negocio (donde realizas operaciones) y con todo esto se obtendría independencia, por lo que si hay cambios de arquitectura se puede acceder a los datos o cambiar el negocio o modificar la presentación y solo sería en esa fracción de la capa.

- **La Capa de la Presentación:** Esta capa reúne todos los aspectos del software que tiene que ver con las interfaces y la interacción con los diferentes tipos de usuarios. Estos aspectos típicamente incluyen el manejo y aspecto de las ventanas, el formato de los reportes, menús y gráficos en general.

- **La Capa de Lógica de Negocio:** Esta capa reúne todos los aspectos del software que tienen que automatizar o apoyan los procesos de negocio que llevan a cabo los usuarios. Estos aspectos típicamente incluyen las tareas que forman parte de los procesos, las reglas y restricciones que aplican. Esta capa también recibe el nombre de la capa de la Lógica de la Aplicación.

- **La Capa de Acceso a Datos:** La capa de Acceso a Datos es el puente entre la capa de Lógica de Negocio y el sistema de base de datos. Encapsula la lógica de acceso a datos. Aquí se encuentran componentes que hacen transparente el acceso a la base de datos. Este es el lugar idóneo para implementar los objetos de acceso a datos, permitiendo los mismos ingresar, obtener, actualizar y eliminar información del sistema de bases de datos.

- **Capa de Seguridad:** Esta es una capa especial y su objetivo es lograr que el sistema alcance una seguridad total. Esta capa está relacionada con la administración del sistema porque es aquí donde se definen los permisos que puede tener cada usuario para acceder a la aplicación. Además se definen las variables de sesión para cada uno de los usuarios.

1.5.3 Patrón Modelo-Vista-Controlador (MVC)

El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página html y el código que provee de datos dinámicos a la página, el modelo son los datos provenientes de la lógica de negocio recogidos en el controlador y el controlador invoca a la lógica de negocio, utilizando los objetos de negocio. La vista y el controlador constituyen la interfaz del usuario.

En otras palabras MVC es un patrón de arquitectura de software que divide una aplicación en tres componentes:

- **Modelo:** El modelo contiene la información central y los datos.
- **Vista:** Despliegan la información a los usuarios.
- **Controlador:** Captura las entradas del usuario.

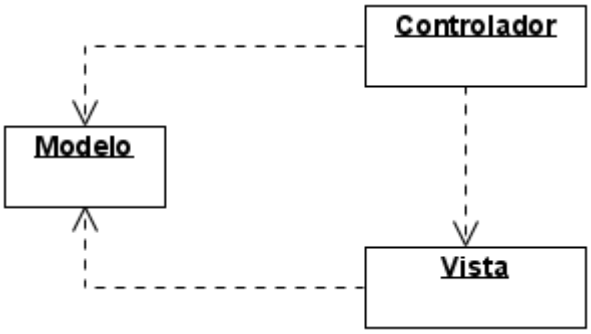

Figura 1: Patrón Modelo Vista Controlador (MVC)

1.5.4 Arquitecturas Orientadas a Objetos (AOO)

Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. En la caracterización clásica de David Garlan y Mary Shaw, los objetos representan una clase de componentes que ellos llaman *managers*, debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto

no es accesible desde otros objetos. En la semblanza de estos autores curiosamente no se establece como cuestión definitoria el principio de herencia. Ellos piensan que, a pesar de que la relación de herencia es un mecanismo organizador importante para definir los tipos de objeto en un sistema concreto, ella no posee una función arquitectónica directa. En particular, en dicha concepción la relación de herencia no puede concebirse como un conector, puesto que no define la interacción entre los componentes de un sistema. Además, en un escenario arquitectónico la herencia de propiedades no se restringe a los tipos de objeto, sino que puede incluir conectores e incluso estilos arquitectónicos enteros. (6)

Se puede decir que en las AOO los componentes del estilo se basan en principios orientados a objetos donde se puede realizar encapsulamiento, herencia y polimorfismo. Las unidades de modelado, diseño e implementación, los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación. En cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases. En tantos componentes, los objetos interactúan a través de invocaciones de funciones y procedimientos.

1.5.5 Arquitecturas Orientadas a Servicios (SOA)

La arquitectura orientada a servicios no se trata de software, ni de una herramienta o de un lenguaje de programación**,** SOA es una estrategia tecnológica para la cual las aplicaciones hacen uso de los servicios disponibles en la red. Implementar SOA comprende el desarrollo de aplicaciones que usen los servicios, aplicaciones disponibles como servicios para otras o ambas situaciones.

Principales Características

- **Reutilización:** El factor fundamental en el cambio a SOA es la reutilización de los servicios de negocios.
- **Interoperabilidad:** El objetivo de una arquitectura débilmente acoplada es que los clientes y servicios se comuniquen independientemente de la plataforma en que radican.
- **Escalabilidad:** Como los servicios de SOA están débilmente acoplados, las aplicaciones que usan esos servicios escalan fácilmente. Este es debidamente a que existe poca dependencia entre las aplicaciones clientes y los servicios que usan.
- **Flexibilidad:** Cualquier cambio en la implementación no afectaría siempre que se mantenga la interfaz.

SOA es un término que en la actualidad se está utilizando masivamente y, como suele ocurrir cuando esto sucede, en muchos casos se utiliza de forma errónea. SOA en realidad es una estrategia que pretende la construcción de todos los componentes software de la compañía utilizando una metodología de desarrollo orientada a servicios y un soporte tecnológico que lo haga posible.

SOA no es algo que pueda estar al alcance de todas las empresas. En primer lugar, únicamente tiene sentido en caso de que el tamaño del departamento de Tecnologías de la Información sea considerable y esté formado, al menos, por más de un sistema primario. Es decir, SOA aporta su mayor beneficio dentro de sistemas complejos y heterogéneos, por eso, para pequeñas empresas es difícil que pueda aportar un valor directo. Dentro de las grandes corporaciones, con sistemas tecnológicos complejos y heterogéneos, SOA no va a proporcionar los mismos beneficios a cada una de ellas.

1.7 Análisis de soluciones existentes

En el mundo los sistemas de capturas y catalogación de medias han avanzado a una gran velocidad. Cada día se hacen más eficientes y con una mayor seguridad, estos sistemas son diseñado generalmente para ambientes de tipo parlamentario pero también son utilizados por universidades, corporaciones, hospitales, centros de investigación entre otros. El objetivo de estos es grabar las sesiones de audio y video en formato digital, permitiendo la indexación de la misma y facilitando su reproducción para realizar la trascripción y permitir realizar consultas selectivas. En el mundo son varios los sistemas que realizan funciones de captura y catalogación de medias. A continuación se muestran algunos de ellos.

1.7.1 MEDIABOX

MEDIABOX es el producto de XTREAM que facilita la catalogación y archivo multimedia (vídeo, audio, documentos electrónicos) desde diferentes tipos de fuentes y distintos formatos. El sistema permite asociar documentos según criterio del usuario (por autor, materia, fechas y otros) digitalizando o transcodificando para guardarlos con la calidad deseada. Trabaja en red y almacena los datos en un Servidor Central. Está indicado para edificios con múltiples puntos de captura y que pueden requerir interconexión con otros centros.

Principales Ventajas

- Gestión de la aplicación desde interfaz gráfico.
- Digitalización desde cinta de video o señal en vivo.
- Posibilidad de captura en continuo (24 x 7).
- Se puede realizar marcas manuales durante la grabación (ej.: intervinientes).
- Admite diferentes formatos de compresión de entrada (MPG1, MPG2, WM8, WM9, MP3, DV25, DV50) y calidad media o alta de almacenamiento.
- Módulo de transcripción para pasar audio a texto.
- Búsqueda flexible empleando caracteres booleanos (*and, or, not*) para consultas.
- Permite volcado a DVD, VCD, CD y disco local del video y metadatos asociados.(9)

Después de esta caracterización es importante resaltar algunos aspectos que son factibles para la realización del proyecto. En estos sistemas es muy importante la seguridad, porque así garantizan la confianza del cliente y ese es un factor que tiene bien definido este sistema por que el acceso se realiza mediante usuario y contraseña con diferentes tipos de usuarios y permisos posibles. Otra característica que es necesaria es la flexibilidad y aquí se observa claramente que permite almacenamiento de alta y media calidad, posibilita búsqueda de palabra clave o varias, captura de forma manual o programada entre otras.

1.7.2 Videoma

Videoma almacena, cataloga, recupera y distribuye imágenes, video y audio de manera eficiente. La solución está diseñada para trabajar en red, a través de múltiples puestos, desde los que se pueden realizar consultas del material almacenado en el servidor central.

Ventajas

- Su arquitectura WEB mejora los flujos de trabajo.
- Funcionamiento estable 24x7.
- Administración personalizada de usuarios y grupos de usuarios.
- Sistema de documentación jerarquizado basado en metadatos.
- Gestión completa de vídeo, audio e imagen.
- Asociación de otros archivos PDF, Word, Excel, etc. al contenido.

- Posibilidad de integración con otras tecnologías del mercado.
- Sistema abierto conformado por módulos con funcionalidades complementarias. Posibilidad de adicción de nuevos módulos desarrollados bajo la plataforma Videoma.
- Aplicación configurable a cualquier idioma. (10)

Este sistema presenta la misma técnica de acceso para la seguridad que MEDIABOX, pero además presenta un sistema de respaldo ante posibles caídas de red y realiza la configuración de borrado de archivos de forma automatizada características que permiten la eficiencia en la administración. Posee un buscador de contenido muy completo permitiendo realizar la búsqueda de manera fácil e intuitiva a través de metadato, previamente definido por el usuario, genera reportes procedentes de búsquedas de video y audio y además facilita búsquedas simples o combinadas de video, audio e imágenes. Estas particularidades antes mencionadas de Videoma son aspectos importantes para el desarrollo de un producto sólido y fiable.

1.7.3 XTAMP

Xtamp es un producto que permite la captura, digitalización, compresión, almacenamiento y consulta de audio y vídeo. Responde a la necesidad de tratamiento de grandes volúmenes de información en forma de audio y vídeo, que ha de ser grabada y archivada para su posterior consulta. El sistema está orientado a sectores como administración e instituciones públicas, banca, universidad, empresa, servicios, industria, construcción, audiovisual y publicidad entre otros.

Principales ventajas

- Calidad de grabación configurable por el usuario.
- Posibilidad de planificación de la grabación.
- Manejo sencillo gracias al Interfaz gráfico.
- Eficiente captura y grabación de contenidos.
- Eficaz reutilización de contenidos.
- Funcionalidad disponible en red. (11)

Este sistema posee un interfaz gráfico que facilita su configuración, control, monitorización y consulta. Permite la gestión remota del sistema desde cualquier estación de trabajo con acceso a la Intranet dando

muestra de su operatividad. Xtamp admite la grabación planificada, captura hasta 16 canales de audio por estación de ingesta con Xtamp audio y un canal de video por estación de video con Xtamp video. Posibilita trabajar con múltiples canales de vídeo o un número de canales de audio superior a 16, sin más que añadir equipos de captura adicionales.

Los sistemas analizados anteriormente son muy utilizados en el mundo. De los mismos solo se tuvieron en cuenta sus principales características para la realización del sistema, es decir se agrupaba lo mejor de cada uno para obtener un mejor resultado aunque estos tienen como inconveniente que todos están realizados con herramientas propietarias. En estos sistemas es muy importante la seguridad pero este aspecto se descuida en los antes mencionados porque están diseñados para ser modificables según los avances de la tecnología y para que un sistema sea seguro no deben existir muchas modificaciones. Entre las empresas que más se destacan en la realización de sistemas de captura y catalogación de medias se encuentran Tedial, Xtream, Vitelsa entre otras. En Cuba todavía no se ha desarrollado ningún sistema capaz de realizar los procesos de captura y catalogación de madias de forma automática, es decir que estos procesos se realizan de forma manual.

1.8 Conclusiones del Capítulo

Al terminar este capítulo se abordaron temas relacionados con la AS, entre ellos los estilos y patrones arquitectónicos, clasificación y caracterización de los más utilizados para tomar posición de los más idóneos para el desarrollo del sistema. Además se analizaron algunos sistemas de captura y catalogación de medias existentes con el objetivo de lograr juntar sus principales características y alcanzar un resultado mejor. A partir del estudio realizado se ganó en conocimiento, aspecto que es de gran importancia para realizar una mejor propuesta arquitectónica.

Capítulo 2
Herramientas para la solución

Introducción 2.1

Este capítulo es el resultado del estudio realizado sobre las tecnologías adecuadas para el desarrollo del sistema. Aquí se analizan la metodología de desarrollo, la herramienta CASE, el entorno de desarrollo (IDE), lenguaje de programación, el gestor de bases de datos y otras herramientas necesarias para lograr la evolución del proyecto.

2.2 ¿Qué metodología utilizar?

Es una pregunta que requiere de mucha importancia, pues el arquitecto debe tener un plano en que apoyarse. Todo desarrollo de software es riesgoso y difícil de controlar, pero si no se lleva una metodología de por medio, no se obtiene el resultado esperado y el cliente no queda satisfecho. A continuación se expondrán las principales características de dos metodologías desarrollo, *Rational Unifield Process y Extreme Programming*.

2.2.1 Rational Unified Process (RUP)

RUP es un proceso que define claramente quien, cómo, cuándo y qué debe hacerse; y, como su enfoque está basado en modelos y utiliza un lenguaje bien definido para tal fin, UML. Éste aporta herramientas como los casos de uso, que definen los requerimientos. Permite la ejecución iterativa del proyecto y del control de riesgos.

2.2.1.1 Descripción general

- El RUP es un proceso de ingeniería de software.
- Utiliza el paradigma de orientación a objetos para su descripción.
- Es un marco de proceso configurable para satisfacer necesidades específicas.
- Implementa las mejores prácticas de desarrollo de software.

2.2.1.2 Flujos de trabajo de RUP

Ver Anexo 1

2.2.1.3 Características de RUP

1. Guiado por los Casos de Uso: Los casos de uso capturan requerimientos funcionales y representan piezas de funcionalidad que brindan un resultado de valor al usuario.
2. Centrado en la Arquitectura: Comprende los aspectos estáticos y dinámicos más importantes del sistema.
3. Iterativo e Incremental: El trabajo se divide en piezas pequeñas o mini proyectos; cada uno proveyendo un subproducto incremental.

2.2.1.4 Fases de RUP

Ver Anexo 2

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.

2.2.2 Extreme Programing (XP).

Es una de las metodologías de desarrollo de software más exitosas en la actualidad utilizada para proyectos de corto plazo y corto equipo. La misma consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

2.2.2.1 Fases de Desarrollo de XP

Ver Anexo 3

¿Qué propone XP?

- Empieza en pequeño y añade funcionalidad con retroalimentación continua
- El manejo del cambio se convierte en parte sustantiva del proceso
- El costo del cambio no depende de la fase o etapa

- No introduce funcionalidades antes que sean necesarias
- El cliente o el usuario se convierte en miembro del equipo

Derechos del Cliente

- Decidir que se implementa
- Saber el estado real y el progreso del proyecto
- Añadir, cambiar o quitar requerimientos en cualquier momento
- Obtener lo máximo de cada semana de trabajo
- Obtener un sistema funcionando cada 3 o 4 meses

Derechos del Desarrollador

- Decidir cómo se implementan los procesos
- Crear el sistema con la mejor calidad posible
- Pedir al cliente en cualquier momento aclaraciones de los requerimientos
- Estimar el esfuerzo para implementar el sistema

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

Después de este análisis la metodología seleccionada es RUP porque sus características son factibles para la realización del sistema, primero porque el Proceso Unificado es una propuesta de proceso para el desarrollo de software orientado a objetos que utiliza *Unified Model Language (UML)* como único lenguaje para describir el proceso. Está basado en componentes, lo cual quiere decir que el sistema software en construcción está formado por componentes software interconectados a través de interfaces bien definidas. Además el sistema se va desarrollando y documentando al mismo tiempo por si algún miembro del equipo no puede seguir con el trabajo el que ocupe su lugar tenga por donde guiarse y conozca que fue lo que se hizo.

2.3 Lenguaje Unificado de Modelado (UML)

UML es un lenguaje para visualizar, especificar, construir, y documentar los artefactos que se crean durante el proceso de desarrollo. Es un lenguaje que permite la modelación de sistemas con tecnología orientada a objetos. No es una guía para realizar el análisis y diseño orientado a objetos, es decir, no es un proceso. UML no es método, ni una metodología

La representación en UML de un software está formada por las 4+1 vistas o modelos parciales separados, relacionados entre sí, estas vistas son:

- ✓ Vista de casos de uso.
- ✓ Vista lógica.
- ✓ Vista de procesos.
- ✓ Viste de implementación.
- ✓ Vista de despliegue.

En favor de UML se puede señalar que es un lenguaje gráfico con sintaxis y semántica bien definidas. La sintaxis de la notación gráfica se especifica mediante su correspondencia con los elementos del modelo semántico subyacente, cuya semántica se define por medio de un meta-modelo, textos descriptivos y restricciones.

Según diversos autores UML no es, en sí, un lenguaje de descripción arquitectónica pues su forma de expresar ciertas características, sobre todo dinámicas de las estructuras no es suficiente para los arquitectos. (12)

2.4 Herramientas CASE

Las **herramientas CASE** (***C****omputer* ***A****ided* ***S****oftware* ***E****ngineering o* Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. (13)

Estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, calculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Desde que se crearon éstas herramientas (1984) hasta la actualidad, las CASE cuentan con una credibilidad y exactitud que tienen un reconocimiento universal, siendo usadas por cualquier desarrollador o programador que busca un resultado óptimo y eficiente, pero sobre todo que busca esa minuciosidad necesaria de los procesos y entre los procesos.

2.4.1 Visual Paradigm

Esta herramienta CASE ha sorprendido gratamente por el trabajo que se puede llegar a desarrollar básicamente. Para gestionar la persistencia y el mapeo de estas clases con la base de datos utiliza Hibernate para Java y NHibernate en el caso de un proyecto .Net. El Visual Paradigm es colaborativa, es decir, soporta múltiples usuarios trabajando sobre el mismo proyecto, genera la documentación del proyecto automáticamente en varios formatos como Web o Pdf (*Portable Document Format),* además de permitir el control de versiones. (14)

Entre sus características se destacan que es robusto y portable. Genera código y realiza ingeniería inversa para diez lenguajes de programación. Se integra con el Visio para importar imágenes del mismo para realizar los diagramas de despliegue. Además exporta e importa los diagramas con estándar XML y como imágenes. Es multiplataforma y gratis en su edición *Community*. Es muy utilizado para desarrollar proyectos importantes.

2.4.2 Rational Rose Enterprise

Rational Rose es la herramienta *CASE* desarrollada por los creadores de UML (Booch, Rumbaugh y Jacobson), que cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases y entregables. El navegador UML de Rational Rose permite establecer una trazabilidad real entre el modelo (análisis y diseño) y el código ejecutable. Facilita el desarrollo de un proceso cooperativo en el que todos los agentes tienen sus propias vistas de información (vista de casos de uso, lógica, de componentes y de despliegue), pero utilizan un lenguaje común para comprender y comunicar la estructura y la funcionalidad del sistema en construcción. (15)

Rose no se integra con varios IDE como Visual Paradigm, solo lo hace con Borland JBuilder de la versión 7.0 en lo adelante y Microsoft Visual Studio de la versión 2003 en adelante. Además no es multiplataforma y es aconsejable utilizar Windows 2000, Windows NT y Windows XP.

Para el desarrollo de este sistema se seleccionó la herramienta Visual Paradigm ya que es una poderosa herramienta CASE que al igual que el Rational Rose utiliza UML para el modelado, es la herramienta por excelencia para ser utilizada en un ambiente de software libre. Permite crear tipos diferentes de diagramas en un ambiente totalmente visual. Es muy sencillo de usar, fácil de instalar y actualizar. Genera código para varios lenguajes. Tiene integrado el MS Visio y es compatible con otras ediciones. Posibilita la representación gráfica de los diagramas permitiendo ver el sistema desde diferentes perspectivas, como el de componentes, despliegue, secuencia, casos de uso, clase, actividad, estado, entre otros.

2.5 Lenguaje de Programación

2.5.1 C++

El C++ es un lenguaje de programación, diseñado a mediados de los años 1980, como extensión del lenguaje de programación C. Se puede decir que C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos. Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. (16)

Existen también algunos intérpretes como *ROOT* (enlace externo). Las principales características del C++ son las facilidades que proporciona para la programación orientada a objetos y para el uso de plantillas o programación genérica. Además posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel:

- Posibilidad de redefinir los operadores (sobrecarga de operadores)
- Identificación de tipos en tiempo de ejecución (*RTTI*)

C++ está considerado como un lenguaje potente, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos automatismos traen, obliga a hacerlo casi todo manualmente al igual que C lo que "dificulta" mucho su aprendizaje.

Ventajas:

- Herencia múltiple, genericidad, plantillas.
- Funciones virtuales, excepciones, etcétera.

Características

- Programación orientada a objetos: La posibilidad de orientar la programación a objetos permite al programador diseñar aplicaciones desde un punto de vista más cercano a la vida real. Además permite la reutilización del código de una manera más lógica y productiva.
- Portabilidad: Un código escrito en C++ puede ser compilado en diferentes ordenadores y sistemas operativos sin hacer apenas cambios.
- Brevedad: El código escrito en C++ es muy corto en comparación con otros lenguajes, sobretodo porque en este lenguaje es preferible el uso de caracteres especiales que las "palabras clave".
- Programación modular: Un cuerpo de aplicación en C++ puede estar hecho con varios ficheros de código fuente que son compilados por separado y después unidos. Esta característica permite unir código en C++ con código producido en otros lenguajes.
- Velocidad: El código resultante de una compilación en C++ es muy eficiente, por su capacidad de actuar como lenguaje de alto y bajo nivel. (16)

2.5.2 Java

El objetivo principal de Java es conseguir un entorno de desarrollo de software que sea independiente de la plataforma de ejecución. Necesita 128 MB de memoria por sesión de usuario. Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. C++ es un lenguaje que adolece de falta de seguridad, pero C y C++ son lenguajes más difundidos, por ello Java se diseñó para ser similar a C++ y así facilitar un rápido y fácil aprendizaje. (17)

Java elimina muchas de las características de otros lenguajes como C++, para mantener reducidas las especificaciones del lenguaje y añadir características muy útiles como el G*arbage Collector* (reciclador de memoria dinámica).

El reciclador se encarga de liberar memoria y como es un *thread* de baja prioridad, cuando entra en acción, permite liberar bloques de memoria muy grandes, lo que reduce la fragmentación de la memoria. (17)

Java reduce un 50% los errores más comunes de programación con lenguajes como C y C++ al eliminar muchas de las características de éstos, entre las que destacan:

- Aritmética de punteros.
- No existen referencias.
- Registros (*struct*)
- Definición de tipos (*typedef*)
- Necesidad de liberar memoria (*free*)

Seguridad de Java

Existe una preocupación lógica en Internet por el tema de la seguridad: virus, caballos de Troya, y programas similares navegan de forma usual por la red, constituyendo una amenaza palpable. Java ha sido diseñado poniendo un énfasis especial en el tema de la seguridad, y se ha conseguido lograr cierta inmunidad en el aspecto de que un programa realizado en Java no puede realizar llamadas a funciones globales ni acceder a recursos arbitrarios del sistema, por lo que el control sobre los programas ejecutables no es equiparable a otros lenguajes.
Los niveles de seguridad que presenta son:

- Fuertes restricciones al acceso a memoria, como son la eliminación de punteros aritméticos y de operadores ilegales de transmisión.
- Rutina de verificación de los códigos de byte que asegura que no se viole ninguna construcción del lenguaje.
- Verificación del nombre de clase y de restricciones de acceso durante la carga.
- Sistema de seguridad de la interfaz que refuerza las medidas de seguridad en muchos niveles.(17)

Para la realización de este sistema se decidió utilizar el lenguaje Java para el módulo de Administración con la utilización de varios *frameworks (marcos de trabajo)* entre ellos: Swing, Spring e Hibernate, se utiliza este lenguaje por las ventajas que presenta para lograr la seguridad del sistema. También se opto por trabajar con el lenguaje C++ con la librería gráfica QT4 para los restantes módulos debido que C++

es un lenguaje basado en componentes y las ventajas que presenta la librería QT4 para el trabajo con medias.

2.6 Frameworks

Conjunto de clases que cooperan y forman un diseño reutilizable para un tipo específico de software. Un *framework* ofrece una guía arquitectónica partiendo el diseño en clases abstractas y definiendo sus responsabilidades y sus colaboraciones. Un desarrollador personaliza el *framework* para una aplicación particular mediante herencia y composición de instancias de las clases del *framework.* (18)

Se puede decir que los *frameworks* son arquitecturas definidas para un determinado dominio de la aplicación que contienen un conjunto de componentes implementados y sus interfaces bien definidas, donde estos componentes se pueden utilizar, redefinir y crear nuevos componentes.

A partir de la organización del sistema en Capas se propone la utilización de tres *frameworks* para el lenguaje Java, estos distribuidos en cada capa de la aplicación con funcionalidades bien definidas y componentes implementados que permiten desarrollar sus propios componentes para cada capa de la aplicación:

- Capa de Presentación: **Framework Swing, librería QT**
- Capa de Negocio: **Framework Spring**
- Capa de Acceso a Datos: **Framework Hibernate**

2.6.1 Framework Swing

Swing es un *framework* compuesto por un amplio conjunto componentes y presenta arquitectura **Modelo-Vista-Controlador**: Esta arquitectura da lugar a todo un enfoque de desarrollo muy arraigado en los entornos gráficos de usuario realizados con técnicas orientadas a objetos. Cada componente tiene asociado una clase de modelo de datos y una interfaz que utiliza. Entre sus principales características se destacan:

- Gestión mejorada de la entrada del usuario: Se pueden gestionar combinaciones de teclas en un objeto *KeyStroke* y registrarlo como componente. El evento se activará cuando se pulse dicha

combinación si está siendo utilizado el componente, la ventana en que se encuentra o algún hijo del componente.

- Objetos de acción (*action objects*): Estos objetos cuando están activados (*enabled*) controlan las acciones de varios objetos componentes de la interfaz.
- Contenedores anidados: Cualquier componente puede estar anidado en otro. Por ejemplo, un gráfico se puede anidar en una lista.
- Escritorios virtuales: Se pueden crear escritorios virtuales o "interfaz de múltiples documentos" mediante las clases *JDesktopPane* y *JInternalFrame*.
- Bordes complejos: Los componentes pueden presentar nuevos tipos de bordes. Además el usuario puede crear tipos de bordes personalizados.
- Diálogos personalizados: Se pueden crear multitud de formas de mensajes y opciones de diálogo con el usuario, mediante la clase *JOptionPane*.
- Clases para diálogos habituales: Se puede utilizar *JFileChooser* para elegir un fichero, y *JColorChooser* para elegir un color.
- Componentes para tablas y árboles de datos: Mediante las clases *JTable* y *JTree*.
- Potentes manipuladores de texto: Además de campos y áreas de texto, se presentan campos de sintaxis oculta *JPassword*, y texto con múltiples fuentes *JTextPane*. Además hay paquetes para utilizar ficheros en formato HTML o RTF.
- Capacidad para "deshacer": En gran variedad de situaciones se pueden deshacer las modificaciones que se realizaron.
- Soporte a la accesibilidad: Se facilita la generación de interfaces que ayuden a la accesibilidad de discapacitados, por ejemplo en Braille.(19)

Sobre este *framework* se realizó un estudio y el mismo no cumplió con las expectativas para la realización de este sistema pues el mismo desde finales 2007 no presenta ningún aspecto relevante en cuanto a progreso y además Sun ya ha avisado que debido a la actual situación económica a medio plazo no se pretende realizar contribuciones significativas a Swing, sino que simplemente se seguirá manteniendo la base de código y corrigiendo *bugs*. Este aspecto no presenta un problema para la realización del sistema debido a que con la utilización de la librería QT se puede realizar toda la parte de la interfaz gráfica tanto para el lenguaje C++ como para Java. En este caso se usará el QtJambi para Java que el mismo provee

toda la funcionalidad que QT C++ y el código es mucho más simple que otros frameworks como SWT/Swing.

2.6.2 Framework Spring

Spring es un *framework* que facilita la creación de diversas aplicaciones. Diseñado en módulos, con funcionalidades específicas y consistentes con otros módulos, te facilita el desarrollo de nuevas funcionalidades y hace que la curva de aprendizaje sea favorable para el desarrollador, al ser fácil de asimilar. (20)

El objetivo central de Spring es permitir que objetos de negocio y de acceso a datos sean reusables, no atados a servicios J2EE específicos. Estos objetos pueden ser reutilizados tanto en entornos J2EE (web o EJB), aplicaciones independientes y entornos de pruebas.

Filosofía de Spring Framework

- Proveer un Framework no invasivo.
- Siempre que se pueda rehusar código.
- *Plug* & *Play* de componentes.

Dentro de las ventajas que ofrece Spring, se encuentran que facilita la manipulación de los objetos, se usen *EJBs* o no, reduce la proliferación de *Singletons* (Patrón de Diseño *Singleton*), elimina la necesidad de usar distintos y variados tipos de ficheros de configuración, mejora la práctica de programación, permite el uso o no de *EJBs*, realizando el mismo tipo de funciones sin ellos.

Spring proporciona:

- Una potente configuración de la aplicación basada en JavaBeans, aplicando los principios de Inversión de Control (IoC). Esto hace que la configuración de aplicaciones sea rápida y sencilla. Ya no es necesario tener *singletons* ni ficheros de configuración, una aproximación consistente y elegante. Esta factoría de beans puede ser usada en cualquier entorno, desde el contexto de la aplicación.
- Una capa genérica de abstracción para la gestión de transacciones, permitiendo gestores de transacción enchufables (*pluggables*), y haciendo sencilla la demarcación de transacciones sin tratarlas a bajo nivel. Se incluyen estrategias genéricas y un único JDBC *DataSource*. En contraste

con EJB (Enterprise Java Bean), el soporte de transacciones de Spring no está atado a entornos J2EE.

- Manejo de transacciones desde el contexto de la aplicación, cada uno de los “Servicios” de la aplicación deben de ser instanciados usando un manejador de transacción. Se integra con Hibernate, JDO e *iBatis* SQL *Maps* en términos de soporte a implementaciones DAO y estrategias con transacciones. Especial soporte a Hibernate añadiendo convenientes características de IoC, y solucionando muchos de los comunes problemas de integración de Hibernate. Todo ello cumpliendo con las transacciones genéricas de Spring y la jerarquía de excepciones DAO.
- Funcionalidad AOP es decir Programación Orientada a Aspectos, está totalmente integrada en el framework Spring. Se puede aplicar AOP a cualquier objeto gestionado por Spring, añadiendo aspectos como gestión de transacciones declarativa.(20)

2.6.3 Framework Hibernate

Hibernate es un framework que constituye un motor de persistencia que implementa múltiples funcionalidades. El centro de la arquitectura de Hibernate lo constituyen una serie de interfaces que realizan el grueso de las funcionalidades del framework, dentro de ellas están:

Interfaces	**Descripción**
Session	Es la más usada en las aplicaciones, es la manejadora de la persistencia, a través de ella se puede guardar y cargar objetos de la base de datos.
SessionFactory	Mediante ella se obtiene la *Session.*
Configuration	Es usada para configurar Hibernate se utiliza para especificar la ruta de los documentos de mapeo.
Transaction	Se utiliza para el control de las transacciones.
Query y Criteria	Permiten la ejecución de consultas a la base de datos.

Tabla 1: Interfaces del framework Hibernate

Además de estas interfaces Hibernate brinda otros aspectos muy importantes y que son de gran utilidad para cualquier aplicación, por ejemplo:

- La interface *Type*, es un elemento fundamental y muy poderoso, permite el mapeo de tipos “java” a columnas en bases de datos incluso a varias columnas, incluye tipos como Calendar, byte [], y permite además definir tipo de datos propios (Persona, Dirección, Nombre).
- Las interfaces *Callback* gestionan el ciclo de vida de los objetos (*Lifecycle, Validatable*).
- Un dialecto orientado a objetos (HQL) fácil de manejar y familiar a SQL que aunque no es un lenguaje de manipulación de datos como este, puesto que es usado solamente para extraer datos no para borrar, insertar o actualizar, permite realizar complejas consultas.(21)

Define dos tipos de configuración:

- Entorno manejado, entornos que proveen reservas de recursos como conexiones a base de datos (*connections pool*), transacciones y seguridad declarativa, en este caso se encuentran los servidores de aplicación como *JBoss*, *BEA WebLogic* entre otros.
- Entorno no manejado, es el caso de los contenedores de *servlet* como Tomcat, las aplicaciones de escritorio también son incluidas aquí; este tipo de entorno no provee transacciones automáticas ni manejos de recursos, la propia aplicación realiza el manejo de las conexiones a base de datos.

En los entornos no manejados Hibernate se ocupa de las transacciones y las conexiones JDBC (*Java Database Connectivity*) o lo deja para que el desarrollador se ocupe de esto, en el caso de los entornos manejados Hibernate se integra con el contenedor para ocuparse de los *DataSources* y de las transacciones. Salvo estas diferencias lo demás es común para cualquier entorno. (21)

Otros aspectos que son esenciales en el desarrollo de aplicaciones con Hibernate lo constituyen los ficheros de mapeo y configuración. Para cada clase persistente se le hace corresponder un fichero de mapeo que es el lugar donde se le especifica al framework como va a persistir dicha clase en la base de datos pero además se trata todo lo referente a las relaciones de esta clase con otras incluyendo el tratamiento de herencia y polimorfismo. Ahora el archivo de configuración es el que le dice a Hibernate todo lo referente a la conexión que se puede alcanzar vía JNDI (*Java Naming and Directory Interface o* La Interfaz de Nombrado y Directorio Java) en el caso de los entornos manejados o cargando la conexión con el driver correspondiente al gestor haciendo uso si se quiere de alguna herramienta que maneje la reserva de conexiones todo esto en el mismo archivo.

2.7 Entornos de Desarrollo Integrado (IDE)

2.7.1 Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto independiente de una plataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. (22)

La versión actual de Eclipse dispone de las siguientes características:

- Editor de texto.
- Resaltado de sintaxis.
- Compilación en tiempo real.
- Pruebas unitarias con *JUnit*.
- Control de versiones con CVS.
- Integración con Ant.
- Asistentes (*wizards*): Para creación de proyectos, clases, *tests*, etc.
- Refactorización.

Asimismo, a través de "*plugins*" libremente disponibles es posible añadir:

- Control de versiones con Subversion.
- Integración con Spring.
- Integración con Hibernate
- Eclipse es un completo IDE (*Integrated Development Environment)* de programación, fue desarrollado originalmente para construir proyectos Java aunque dada su gran evolución y creciente popularidad ahora se puede crear todo tipo de proyectos en numerosos lenguajes de programación. Este entorno es uno de los más populares en el mundo y además muy usado por la facilidad de integración con diferentes herramientas.

2.7.2 NetBeans

Se refiere a una plataforma para el desarrollo de aplicaciones de escritorio usando Java y a un Entorno Integrado de Desarrollo (IDE) desarrollado usando la Plataforma NetBeans en su versión 6.0.

El soporte de *Java Enterprise Edition* es de importancia, en especial para los desarrolladores de JBuilder. Dado que cuenta con el mejor soporte a estándares industriales de la tecnología Java, el proyecto NetBeans ha hecho que el desarrollo de aplicaciones Java de tipo empresarial sea rápido y sencillo, su facilidad de uso, su cumplimiento de regulaciones, sus perfiles de rendimiento, además de su flexibilidad entre plataformas. (23)

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios y una comunidad en constante crecimiento. Los desarrolladores acostumbrados a trabajar con herramientas basadas en la tecnología Java, tales como el conjunto de herramientas *Borland JBuilder*, descubren que la migración a NetBeans puede acelerar en forma significativa los esfuerzos de desarrollo. (20)

El IDE NetBeans es un producto libre y gratuito sin restricciones de uso. Les sirve a los programadores para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Además es un IDE de código abierto escrito completamente en Java usando la plataforma NetBeans. La versión actual es NetBeans IDE 6.5 y presenta las características existentes del Java EE incluyendo Soporte a Persistencia, EJB 3 y JAX-WS.

2.7.3 QT Creator

Qt Creator es el nuevo, liviano y multiplataforma Entorno Integrado de Desarrollo (IDE) de Trolltech, diseñado para hacer que el desarrollo en C++ de la aplicación Qt sea más rápido y fácil. No quiere ser un reemplazo de Eclipse ni Visual Studio, sino un IDE ligero pensado especialmente para el desarrollo en múltiples plataformas: Windows XP y Vista, Linux (desde la versión 2.6) y Mac OS X (desde 10.4 en adelante). (24)

Como características tiene el reconocimiento de métodos, la facilidad de creación de formularios y amplia documentación On-line, así como un sin fin de opciones que facilitan el desarrollo de cualquier aplicación. (24)

Este nuevo IDE tiene el futuro garantizado, y es precisamente por la utilización de la librería QT la cual ha sido utilizada para la realización de grandes proyectos como son: *Google Earth* que pone la información geográfica del mundo al alcance de tu mano y esto lo hace multiplataforma gracias a QT **,** otro ejemplo es

Guión de navegación donde QT es utilizado para crear el *Dash Express*, el primer dispositivo de navegación GPS con dos vías de conexión a Internet. Esta biblioteca es multiplataforma para el desarrollo de aplicaciones gráficas. Utiliza el lenguaje de programación C++ de forma nativa y además existen *bindings* para C, Python (PyQt), Java (Qt Jambi), Perl (*PerlQt*) y Ruby (*QtRuby*).

Este IDE está en desventaja con NetBeans y Eclipse porque solo se emplea para el desarrollo de aplicaciones de escritorio pero esto no es problema para el desarrollo del proyecto y aún así para el trabajo con medias está en ventaja por la utilización de QT, una muestra es:

- **Amarok**: Reproductor de audio con grandes capacidades de integración.
- **Dragon Player**: Reproductor de video.
- **K3b**: Suite de grabación de discos compactos y disco versátil digital.
- **JuK**: Reproductor de audio.
- **K9copy**: Software para hacer copias de seguridad disco versátil digital.
- **Kaffeine**: Reproductor de medios (similar a Totem, pero más completo).
- **KAudioCreator**: Extractor digital de discos compactos.
- **Kdetv**: Visualizador de TV analógica.
- **Klear**: Visualizador y grabador de TV digital.
- **KMid**: Repdroductor de MIDI/Karaoke.
- **KMix**: Mezclador de audio.
- **KMPlayer**: Frontispicio para GStreamer, MPlayer o Xine.
- **Konverter**: Conversor de video, GUI para MEncoder.
- **KPlayer**: Reproductor y biblioteca multimedia, GUI para MPlayer.
- **KRadio**: Sintonizador de radio.
- **KsCD**: Reproductor de discos compactos.
- **LMMS**: Secuenciador con interfaz similar a FL Studio.
- **MusE**: Secuenciador con interfaz similar a *Cubase/Logic Audio*.
- **Noatun**: Reproductor multimedia.
- **Qtractor**: Secuenciador con interfaz similar a *Cubase/Logic Audio*.
- **Rosegarden**: Secuenciador con interfaz similar a *Cubase/Logic Audio.*

- **soundKonverter**: Conversor de ficheros de audio.
- **SMPlayer**: Frontispicio completo para MPlayer.

Todos estos programas están muy relacionados con medias y son basados en QT.

Después del análisis sobre los entornos de desarrollo se decide utilizar QT Creator con la librería gráfica QT utilizando como lenguaje de programación C++ y Eclipse con lenguaje Java para la parte de seguridad pero además con la integración del plugins QtJambi para la realización de las interfaces. En este sistema se seleccionaron dos IDEs porque a Eclipse se le puede integrar fácilmente los plugins para lo que se desee realizar, no así en el NetBeans que es un paquete único, es decir se emplea Eclipse para el desarrollo del módulo de Administración. El otro entorno de desarrollo es QT Creator que con la utilización de la librería gráfica QT4 le es muy factible a los módulos de Captura e Indexación de Videos, Captura y Transcripción de Audio y Catalogación de Medias por las características que presenta esta librería para el trabajo con medias.

2.8 Sistemas Gestores de Base de datos

Un Sistema Gestor de Base de Datos (SGBD) es un conjunto de programas que permiten crear y mantener una base de datos, asegurando su integridad, confidencialidad y seguridad. Presenta una interfaz, mediante la cual el usuario puede comunicarse con el sistema físico y realizar operaciones como almacenar o recuperar datos de ella. Las principales funciones que debe cumplir un SGBD son la creación y mantenimiento de la base de datos, el control de accesos, la manipulación de datos de acuerdo con las necesidades del usuario, el cumplimiento de las normas de tratamiento de datos, evitar redundancias e inconsistencias. (25)

Algunos ejemplos de SGBD son Oracle, DB2, PostgreSQL, MySQL, MS SQL Server, etc.

Un SGBD debe permitir:

- Definir una base de datos: especificar tipos, estructuras y restricciones de datos.
- Construir la base de datos: guardar los datos en algún medio controlado por el mismo SGBD.
- Manipular la base de datos: realizar consultas, actualizarla, generar informes.

2.8.1 MySQL

MySQL es un sistema de gestión de bases de datos relacionales, multi–hilo y multi-usuario. Una base de datos relacional almacena datos en tablas separadas en lugar de poner todos los datos en un gran almacén, lo que añade velocidad y flexibilidad. La parte SQL de "MySQL" se refiere a "*Structured Query Language*". SQL es el lenguaje estandarizado más común para acceder a bases de datos. MySQL trabaja en entornos cliente/servidor o incrustados. Además, funciona en diferentes plataformas y fue escrito en C y en C++. Tiene como una de sus principales ventajas la velocidad en la lectura de datos, pero a costa de eliminar un conjunto de facilidades que presentan otros SGBD: integridad referencial, bloqueo de registros, procedimientos almacenados.

MySQL es muy popular en aplicaciones web y actúa como un componente de bases de datos para las plataformas LAMP, MAMP y WAMP (Linux/MAC/Windows-Apache-MySQL-PHP/Perl/Python), también trabaja en numerosas plataformas como AIX, HP-UX, GNU/Linux, Mac OS X, Novell NetWare, OpenBSD, OS/2, Solaris, SunOS, y todas las versiones de Windows. Su mayor desempeño se logra cuando se combina con el lenguaje de programación PHP.

2.8.2 Oracle

Oracle es un sistema de gestión de base de datos relacional (*Relational Data Base Management System*), fabricado por Oracle *Corporation*. Se considera uno de los sistemas de bases de datos más completos. Es un sistema gestor de base de datos robusto, tiene muchas características que garantizan la seguridad e integridad de los datos. Las transacciones se ejecuten de forma correcta, sin causar inconsistencias. Ayuda a administrar y almacenar grandes volúmenes de datos. Presenta estabilidad, escalabilidad además de ser multiplataforma.

La tecnología Oracle se encuentra prácticamente en todas las industrias alrededor del mundo. Oracle es la primera compañía de software que desarrolla e implementa software para empresas 100 por ciento activado por Internet a través de toda su línea de productos: base de datos, aplicaciones comerciales y herramientas de desarrollo de aplicaciones y soporte de decisiones. Garantiza el funcionamiento de sus bases de datos, que en caso de caídas del servidor compensa económicamente con cifras cercanas a las 7 cifras.

Las últimas versiones de Oracle han sido certificadas para poder trabajar bajo Linux. (26)

Aunque su dominio en el mercado de servidores empresariales ha sido casi total, recientemente sufrió la competencia de gestores de bases de datos comerciales y de la oferta de otros con licencia Software Libre como PostgreSQL.

2.8.3 PostgreSQL

PostgreSQL es un sistema gestor de base de datos de objetos relacionales basado en software libre. Ofrece una alternativa ante otros sistemas gestores de bases de datos comerciales. Similar a proyectos de software libres como Apache y GNU/Linux, PostgreSQL no es controlado por ninguna compañía, pero responde al esfuerzo de una comunidad global de desarrolladores. (27)

Cada usuario obtiene una visión consistente de lo último que se le hizo al *commit*. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando el uso de bloqueos explícitos.

Ofrece una potencia adicional sustancial al incorporar los siguientes cuatro conceptos adicionales básicos en una vía en la que los usuarios pueden extender fácilmente el sistema: clases, herencia, tipos y funciones. Además aportan potencia y flexibilidad adicional como son las restricciones (*constraints*), disparadores (*triggers*), reglas (*rules*) e integridad transaccional. (27)

Uno de los problemas al cual las personas que realizan desarrollo de sistemas se tiene que enfrentar es elegir el sistema de gestión de base de datos que muestre mejor rendimiento y ofrezca menor tiempo de respuesta a la hora de ejecutar las consultas.

A continuación se muestra un estudio realizado por Ramon Aliendre, Juan Paz y Josefina Juaniquina en cuanto al rendimiento entre PostgreSQL, Oracle y SQL Server donde realizaron pruebas con diferentes cantidades de registros (100,000 , 500,000 y 1,000,000 registros), y las pruebas presentaban diferentes tipos de consulta (simple, media y compleja), emitiendo un análisis comparativo de los tiempos obtenidos.

Para esta prueba se creó una base de datos con cinco tablas y se fueron analizando los resultados obtenidos a partir del tipo de consulta (simple, media y compleja) y la cantidad de registros (100,000 , 500,000 y 1,000,000).

En la siguiente figura se muestra el resultado obtenido para la consulta compleja en la cual se observa que el gestor de BD Oracle es muy robusto y rápido, pero ademas es visible que PostgreSQL cuanto mayor es el numero de registro y más compleja es la consulta sus resultados son más rápido.

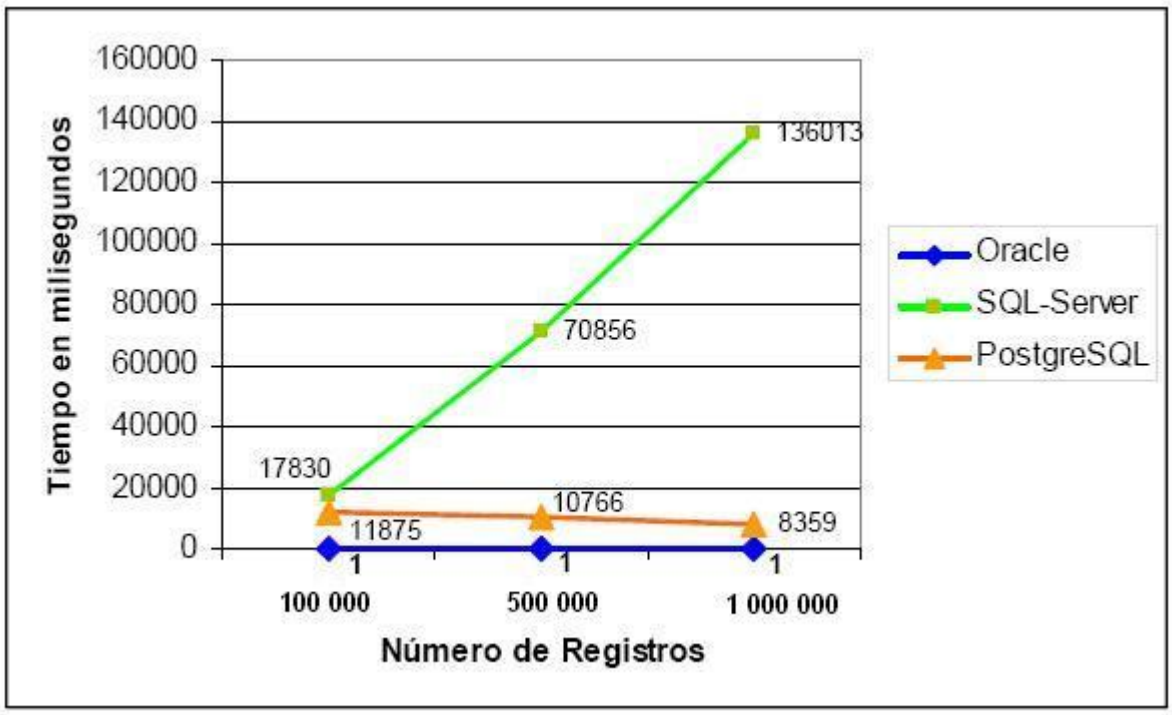

Figura 2: Resultados para Consultas Complejas

Mediante los criterios anteriores se observa que PostgreSQL es un SGBD que la hace competencia a los demás, es caracterizado como un motor de bases de datos avanzado y de código abierto. Mediante un sistema denominado MVCC (Control de Versiones Concurrente) permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Además soporta Vistas (*Views*), Integridad Transaccional (ACID) y consultas (*querys*) complejas, incluyendo sub-selecciones (*subselects*). Debido a estas características es seleccionado PostgreSQL como gestor de bases de datos donde su principal característica es que es código abierto, además de permitir soporte con la mayoría de los lenguajes de programación y se considera como una de los SGBD más potentes.

2.9 Software para el Control de Versiones

El control de versiones es el arte de manejar cambios en la información. Ha sido desde siempre una herramienta crítica para los programadores, quienes típicamente emplean su tiempo haciendo pequeños cambios al software y luego deshaciendo o comprobando esos cambios al día siguiente.

2.9.1 Subversion con cliente TortoiseSVN

Subversion es un de sistema de control de versiones libre y de código fuente abierto. Soporta el manejo de ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto que recuerda todos los cambios hechos a sus ficheros y directorios, permitiendo recuperar versiones antiguas de datos, o examinar el historial de cambios de los mismos. Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. Brinda la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas. Existen varias interfaces a Subversion, ya sea programas individuales como interfaces que lo integran en entornos de desarrollo, donde se encuentra TortoiseSVN, el cual provee integración con el explorador de Windows, siendo este la interfaz más popular en este sistema operativo.

TortoiseSVN es un cliente gratuito de código abierto para el sistema de control de versiones Subversion. Integrado en la shell de Windows, por ejemplo el “explorador”. Y ni siquiera está obligado a usar el Explorador de Windows. Los menús contextuales de TortoiseSVN también funcionan en otros administradores de archivos, y en la ventana Fichero/Abrir que es común a la mayoría de aplicaciones estándar de Windows. Todos los comandos de Subversion están disponibles desde el menú contextual del explorador. TortoiseSVN añade su propio submenú allí. (28)

¿Qué hace de TortoiseSVN tan buen cliente de Subversion?

A continuación se muestran las principales características por lo que se decidió utilizar TortoiseSVN como cliente para Subversion:

- Integración con el pantalla de comandos de Windows
- Íconos sobre-impresionados

- Fácil acceso a los comandos de Subversion
- Versionado de carpetas
- Confirmaciones atómicas
- Metadatos versionados
- Elección de capas de red
- Manejo de datos consistente
- Etiquetado y creación de ramas eficiente
- Extensibilidad

2.10 Salvas Automáticas

2.10.1 Bacula

Bacula es una colección de herramientas de respaldo muy amplia, capaces de cubrir eficientemente las necesidades de respaldo de equipos bajo redes IP. Se basa en una arquitectura cliente/servidor que resulta muy eficaz y fácil de manejar, dada la amplia gama de funciones y características que brinda; copiar y restaurar ficheros dañados o perdidos. Además, debido a su desarrollo y estructura modular, Bacula se adapta tanto al uso personal como profesional, para parques de ordenadores muy grandes. (29)

Algunas de las características de Bacula:

- Planificación para múltiples tareas al mismo tiempo
- Ejecución de una tarea o múltiples al mismo tiempo
- Secuencias de tareas usando prioridades.
- Independencia del sistema operativo en el formato de los volúmenes. Linux, Solaris, y Windows clientes pueden realizarse en el mismo volumen
- Verificación de catálogos "*Tripwire like capability (system break-in detection)"*
- Soporte para múltiples "*drive autochangers*".
- ACL's para restricción de acceso a datos
- Soporte para save/restore de archivos de más de dos 2GB.
- Soporte para ordenadores de 64 bits
- Encriptación de las comunicaciones por stunnel.

- Salvas consistentes de ficheros abiertos sobre Win32 *systems* (WinXP, Win2003) (26)

Después de esta caracterización se aprecia que Bacula es un software libre para realizar copias de seguridad muy potente que apenas tiene nada que envidiar a sus equivalentes en software propietario. No es una aplicación monolítica sino que es un conjunto de varios servicios junto con una interfaz de usuario. Los servicios tienen responsabilidades establecidas y utilizan la red para comunicarse. Bacula es una herramienta muy útil, extremadamente flexible y con características profesionales. Contiene una buena documentación y se integra fácilmente en entornos con sistemas heterogéneos.

2.11 Conclusiones del Capítulo

En este capítulo después de un estudio realizado quedaron definidas las principales herramientas tanto de modelado como de desarrollo para darle solución al sistema que se desea realizar, teniendo presente para esta selección que dichas herramientas debían ser libres, fáciles de emplear para el desarrollo de aplicaciones de escritorio y compatible en varias plataformas para que el sistema en futuras versiones puede ser multiplataforma.

Capítulo 3
Propuesta de la Arquitectura del Sistema

3.1 Introducción

En este capítulo se tiene como objetivo lograr una mejor visión del sistema, organizar el desarrollo, posibilitando esto que el desarrollo del sistema sea de forma eficiente. Aquí se obtienen dos artefactos fundamentales (Línea Base de la Arquitectura y el Documento de Descripción de la Arquitectura) donde se describe la solución arquitectónica del sistema, además otros artefactos definidos por la metodología RUP como son las vistas arquitectónicas.

3.2 Estructura del equipo de Desarrollo

3.2.1 Herramientas de Desarrollo

A continuación se mencionarán las herramientas tanto de modelación como de desarrollo que se definieron en el proyecto CCM.

Herramientas de Modelado.

- Lenguaje de Modelado : UML
- Herramienta CASE: Visual Paradigm Enterprise Edition

Herramientas de Desarrollo.

- Lenguajes de Programación: C++ con QT y Java
- IDEs QT Creator y Eclipse
- Gestor de Base de Datos: PostgreSQL
- Control de Versiones: Subversion con cliente TortoiseSVN.
- Salvas Automáticas: Bacula

Distribución del equipo de trabajo

El equipo de desarrollo está conformado por:

Miembros	
Gerente de Proyecto	Arquitecto
Gerente de Audio	Jefes de Módulos
Gerente de Video	Analistas
Gerente de Calidad	Diseñador de Base de Datos
Arquitecto Principal	Programadores
Gestión de Configuración y Configuración	Grupo de Calidad

Tabla 2: Equipo de Desarrollo

3.2.2 Configuración de los puestos de trabajos

- Analista
    1. PC, con mouse (Periférico de ordenador utilizado para señalar elementos en la pantalla) y teclado.
    2. Instalación de Visual Paradigm.
    3. Instalación del paquete Office.

- Implementador
    1. PC, con mouse y teclado
    2. Instalación del IDE QT Creator o Eclipse con sus *frameworks*

- Diseñador de BD
    1. PC, con mouse y teclado
    2. Instalación del Visual Paradigm
    3. Instalación de PostgreSQL

3.3 Organigrama de la arquitectura

La siguiente figura muestra la representación del estilo en Capas para el proyecto Captura y Catalogación de Medias.

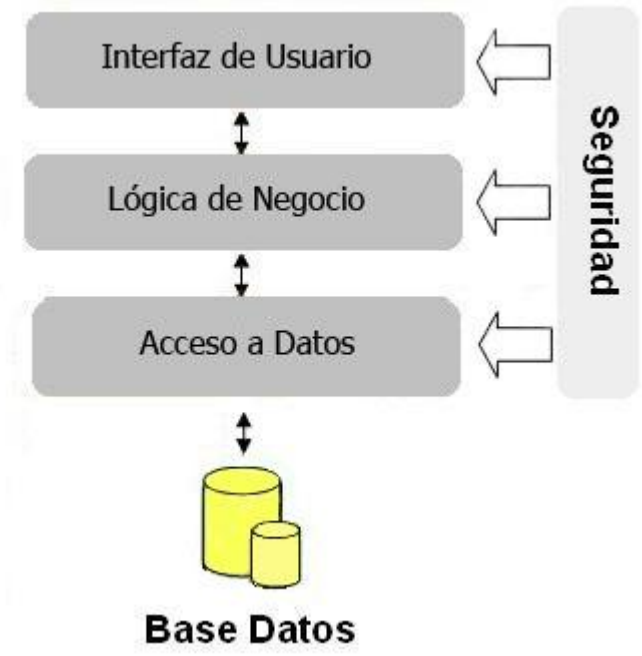

Figura 3: Estructura de capas del patrón en Capas aplicada al sistema

Descripción de cada capa

Interfaz de Usuario (IU) o Presentación: Esta capa contiene la funcionalidad necesaria para permitir que los clientes interactúen e intercambien información con la aplicación. Tiene la funcionalidad de encapsular la lógica para la presentación de la información al usuario, es la encargada de gestionar los eventos y la principal relación con el “modelo” de los datos que se deben mostrar. Aquí se hacen las principales validaciones de los datos a entrar en el sistema.

En esta capa se muestran las interfaces con las que interactúan los usuarios, las cuales son realizadas con la utilización de la librería QT para los dos lenguajes seleccionados.

Lógica de Negocio (LN) o Empresariales: Esta capa almacena la lógica del negocio, en la implementación de cada una de las clases que la integran se manejan todas las transacciones de la aplicación y se aplican cada una de las reglas del negocio.

Los Servicios son los componentes principales de esta capa, son las clases que encapsulan las funcionalidades del negocio de la aplicación y que ejecutan las transacciones y devuelven la información que se necesita.

En esta capa el *framework* Spring desarrolla un papel importante debido a que con el uso del mismo se pueden obtener varios beneficios por ejemplo: se puede integrar con diferentes tecnologías de acuerdo a lo que se desea realizar en este caso se integra con Acegi para la parte de la seguridad del sistema donde determina identidad del usuario por usuario y contraseña, administra accesos a recursos restringidos y permite acceder según la información de autenticación y los atributos asociados al recurso. También autoriza acceso a sub-recursos según niveles determinados por autenticación y credenciales.

Otro aspecto importante para utilizar a Spring en el desarrollo de este sistema es que se basa en lo que se conoce como el Principio de Inversión de Dependencia, *Inversion of Control* (IoC) donde presenta un contenedor que maneja objetos por ti. También resalta la utilización de AOP, son las siglas en ingles de Programación orientada al aspecto (*Aspect Oriented Programming*), en otras palabras se puede ver como una manera de eliminar código duplicado. Además elimina la baja reusabilidad y permite la separación de funcionalidad por módulos.

Acceso a Datos (AD): Esta capa contiene la funcionalidad de acceder al repositorio de los datos, es la encargada de ejecutar la lógica de acceso a los datos; En esta capa se generan algunos ficheros para lograr la comunicación con la base de datos.

- XML Mapping (Ficheros de Mapeo)

Los ficheros de mapeo, son ficheros xml que contienen la información de la base de datos a la que hace referencia, contiene los campos de cada una de las tablas y sus relaciones.

- Objetos de Negocio, BaseObject

Los Objetos de Negocio que se generan a partir del mapeo de las clases de las tablas de la base de datos son clases entidades de Java que contiene los atributos y métodos para acceder a ellos, estos constituyen los objetos con que se trabaja en la aplicación.

- DAO (Objetos de Acceso a Datos)

Los Objetos de Acceso a Datos son las clases que contienen la funcionalidad necesaria para acceder a la base de datos y realizar un conjunto de operaciones definidas para poder realizar las transacciones.

Para desarrollar las funcionalidades de esta capa se emplea el uso del framework Hibernate para los módulos en Java y para los de C++ se emplea Qtsql que es una librería del mismo QT que permite realizar el proceso del mapeo similar al Hibernate.

Seguridad: En esta capa además de otorgar los permisos a cada usuario para acceder a la aplicación se definen las variables de sesión para el control de acceso por roles y se implementa una funcionalidad específica para aplicar a la BD con el objetivo de evitar las inyecciones sql. Esta capa tiene como objetivo cumplir con la siguiente directiva de seguridad que se ocupa de la autenticación, autorización, comunicación segura y administración de perfiles.

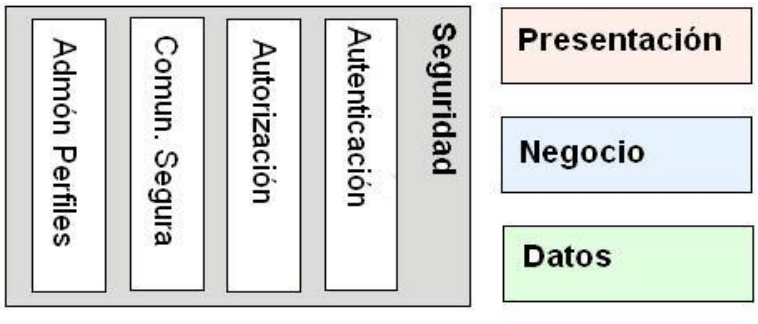

Figura 4: Aspectos de la directiva de seguridad

Principios generales sobre seguridad

Para lograr una mayor eficiencia en cuanto a seguridad se tendrán en cuenta los siguientes principios:

- Nunca confiar en las aportaciones externas. Deberá validar todos los datos que introduzcan los usuarios o envíen otros servicios.

- Considerar por principio que los sistemas externos no son seguros. Si la aplicación recibe datos confidenciales sin cifrar desde un sistema externo, se asume que dicha información no es segura.
- Aplicar el principio del menor privilegio. No habilitar más atributos en las cuentas de servicios que los que resulten estrictamente necesarios para la aplicación. Obtener acceso a los recursos con cuentas que tengan los mínimos permisos necesarios.
- Reducir el área de superficie. El riesgo se incrementa según aumenta el número de componentes y datos que haya expuesto a través de la aplicación y, por lo tanto, se deberá exponer únicamente la funcionalidad que se crea que otros van a utilizar.
- Establecer como predeterminado un modo seguro. No habilitar servicios, tecnologías y derechos de cuenta que no sean absolutamente necesarios. Cuando se implemente la aplicación en equipos cliente o servidor, la configuración predeterminada de esta deberá ser segura.
- No confiar en la seguridad a través del ocultamiento. El cifrado de los datos implica disponer de claves y de un algoritmo de cifrado demostrado. El almacenamiento de los datos seguros evitará el acceso a ésta en cualquier circunstancia. No se puede considerar seguridad la mezcla de diversas cadenas, el almacenamiento de la información en rutas de archivo inesperadas y demás técnicas similares.
- Realizar la comprobación desde la misma puerta. No permitir que los procesos vayan más allá del lugar para el que los usuarios están autorizados.
- Bloquear su sistema interna y externamente: los usuarios y operadores internos pueden representar un riesgo igual que los intrusos externos.

Conectores y Configuraciones

Los conectores son las formas de comunicación entre los distintos componentes o elementos definidos en el estilo arquitectónico, los conectores que se utilizaron para unir las distintas capas son:

- Presentación – Lógica de Negocio, *ServiceBeans*

Los Servicios que se crean en la capa de Negocio son referenciados por los *ActionBeans* creados en la capa de Presentación.
La referencia a estos *Beans* puede hacerse mediante el Constructor o mediante una propiedad.

De esta forma los *ActionBeans* pueden acceder a las funcionalidades que les brindan los servicios referenciados.
El esquema utilizado para los xml donde se referencian los *beans* es el propuesto por el framework Spring:
Ver anexo 4

- Lógica de Negocio – Acceso a Datos, *ServiceBeans*

Los DAO que se crean en la capa de Acceso a Datos son referenciados por los *bean* que se crean en la capa de Negocio (clases Servicios).
La referencia a estos *beans* puede hacerse mediante el constructor o mediante una propiedad. De esta forma los *ServiceBeans* pueden acceder a las funcionalidades que les brindan los DAO referenciados.

- Acceso a Datos – Base Datos, *SessionFactory*

El *SessionFactory* es un *bean* del framework Hibernate que proporciona la conexión al repositorio de datos. El *SessionFactory* es la fábrica de las sesiones (conexiones) que se crean a la base de datos, contiene los datos necesarios para realizar la conexión los que se le pasan como propiedad al *bean*, por ejemplo: el driver que identifica a qué tipo de base de datos se va a conectar, el usuario y la contraseña para conectarse y el nombre del servidor.

El esquema utilizado para los xml donde se referencian los *beans* es el propuesto por el *framework* Hibernate:
Ver anexo 5

La principal restricción que se le impone a los componentes de este estilo arquitectónico es:
Los componentes que pertenecen a las capas superiores solo pueden hacer uso de los componentes de sus capas inmediatas inferiores.

Con la estructura presentada de la arquitectura se definió las interfaces con las van a interactuar los usuarios, las cuales son desarrolladas con QT para C++ para el módulos de Captura e Indexación de Videos y QtJambi para Java para los restantes módulos. También para la lógica del negocio se emplea el *framework* Spring que le aporta seguridad y eficiencia al sistema; y como los sistemas que trabajan con medias se vuelven un poco tedioso por las demoras de los procesos, con el uso de este *framework* que

elimina las dependencias y posibilita que cada uno de los procesos sea independiente este aspecto se mejora. Además para el acceso a datos se emplea Hibernate para los módulos en Java y la librería Qtsql para el módulo en C++.

En la siguiente tabla se muestran las herramientas a utilizar para cada capa:

Capas	C++/ QT	Java	Spring	Hibernate	PostgreSQL
Presentación	X	X			
Negocio	X	X	X		
Datos				X	X
Seguridad	X	X			

Tabla 3: Tecnologías utilizadas por capas

3.4 Módulos del Sistema

- Módulo de Administración
- Módulo de Captura y Transcripción de Audio.
- Módulo de Captura e indexación de Video
- Módulo de Catalogación de Medias

Responsabilidad de cada módulo

Módulo de Captura e Indexación de Video:

Es el encargado de realizar la captura e indexación de videos. Permite leer de la base de datos la planificación y según los tiempos indicados realizar las grabaciones, y una vez terminadas subirá los ficheros al servidor de medias. Además guardará en la base de datos varias características y metadatos iniciales del fichero (hora, canal, tamaño, formato, dirección física en el servidor).

Módulo de Captura y Transcripción de Audio:

En cada una de las comisiones existirá un juego de micrófonos conectados a una consola que estará conectada a una PC (existirá una en cada comisión). En estas PC estará la aplicación de grabación de audio, encargada de grabar el audio recibido de la consola y salvarlo en el servidor central de medias o audio, incluyéndole a los ficheros varios metadatos. El sistema permitirá guardar varios datos asociados a

la grabación como por ejemplo: tema de la reunión, presentes, además de otros datos como fecha, hora, y dirección física en el servidor.

Transcripción: De forma similar a la catalogación las transcriptoras tendrán acceso a los materiales grabados, podrán escucharlos y el sistema ofrecerá una interfaz para la edición del texto transcrito el cual será guardado como un metadato más del material. Esta funcionalidad requiere para una mejor usabilidad la utilización de un periférico (pedales) u otro hardware para la reproducción del audio.

Módulo de Catalogación de Medias

Agrupará varios módulos: Catalogación, Búsquedas, Reportes y Presentación.

Catalogación: A través de búsquedas predefinidas o personalizados por el usuario se mostrarán los materiales grabados a través de una interfaz que permita la catalogación, es decir la inclusión y edición de características y datos descriptivos de las materiales.

Búsquedas: Permite la realización de búsquedas parametrizadas a partir de metadatos asociados a las medias desde su captura o catalogación.

Reportes: Permite la generación e impresión de reportes en formato definido por el cliente. Los reportes pueden ser diseñados desde el módulo de Administración o también pueden ser confeccionados a partir de criterios de búsquedas.

Presentación: Permite la visualización de los materiales de varias formas, por ejemplo online, a través de la salida de video de la PC (hacia un TV o *DataShow*) y en el caso de audio por la salida de audio de la PC (por ejemplo hacia un amplificador).

Módulo de Administración:

Es el responsable de la seguridad y la configuración del sistema.

Seguridad: Se encargará de la gestión de usuario, roles y permisos. Debe tener funcionalidades para registrar la traceabilidad de los usuarios en el sistema.

Configuración: Se encarga de la configuración y personalización de todos los elementos configurables en el sistema, estando presente en todas las aplicaciones de la solución.

Planificación: Realiza la planificación de la grabación de los canales de interés. Esta planificación será registrada en la base de datos a la cual accederá la aplicación de Captura de Video.

3.5 Descripción de la Arquitectura

Para proporcionar una mejor comprensión arquitectónica del sistema se utilizan las vistas arquitectónicas definidas por la metodología RUP. Serán modeladas utilizando Visual Paradigm Enterprise Edition.

- Vista de casos de uso.
- Vista lógica.
- Vista de Procesos
- Vista de implementación.
- Vista de despliegue

3.5.1 Vista de Caso de Uso

A partir de la vista de Casos de Uso, se puede definir los escenarios o los casos de uso que serán de interés para cada iteración del ciclo de desarrollo. Esta describe los escenarios o casos de uso que tienen significación y que encapsulan la funcionalidad central del sistema.

Los casos de uso arquitectónicamente significativos, son aquellos que describen funcionalidades imprescindibles para el sistema, y que a través de estos se valida la arquitectura propuesta para el mismo.

El sistema cuenta con 36 casos de uso, de los cuales 29 son arquitectónicamente significativos; Están representados por módulo de la siguiente forma:

Módulo	**Cantidad de Casos de Uso**
Captura e indexación de Video	9
Captura y Transcripción de Audio	8

Catalogación de Medias	7
Administración	12

Tabla 4: Cantidad de casos de uso por módulos

A continuación se presentan los diagramas que representan los Casos de Uso arquitectónicamente significativos para cada módulo:

Módulo de Administración

- Insertar Usuario
- Gestionar Usuario
- Gestionar Reportes
- Instalar y Configurar Subsistema
- Administrar Procesos
- Realizar Planificación
- Gestionar Servidor
- Generar Reportes
- Visualizar Reportes
- Insertar Planificación por Necesidad de Permisos

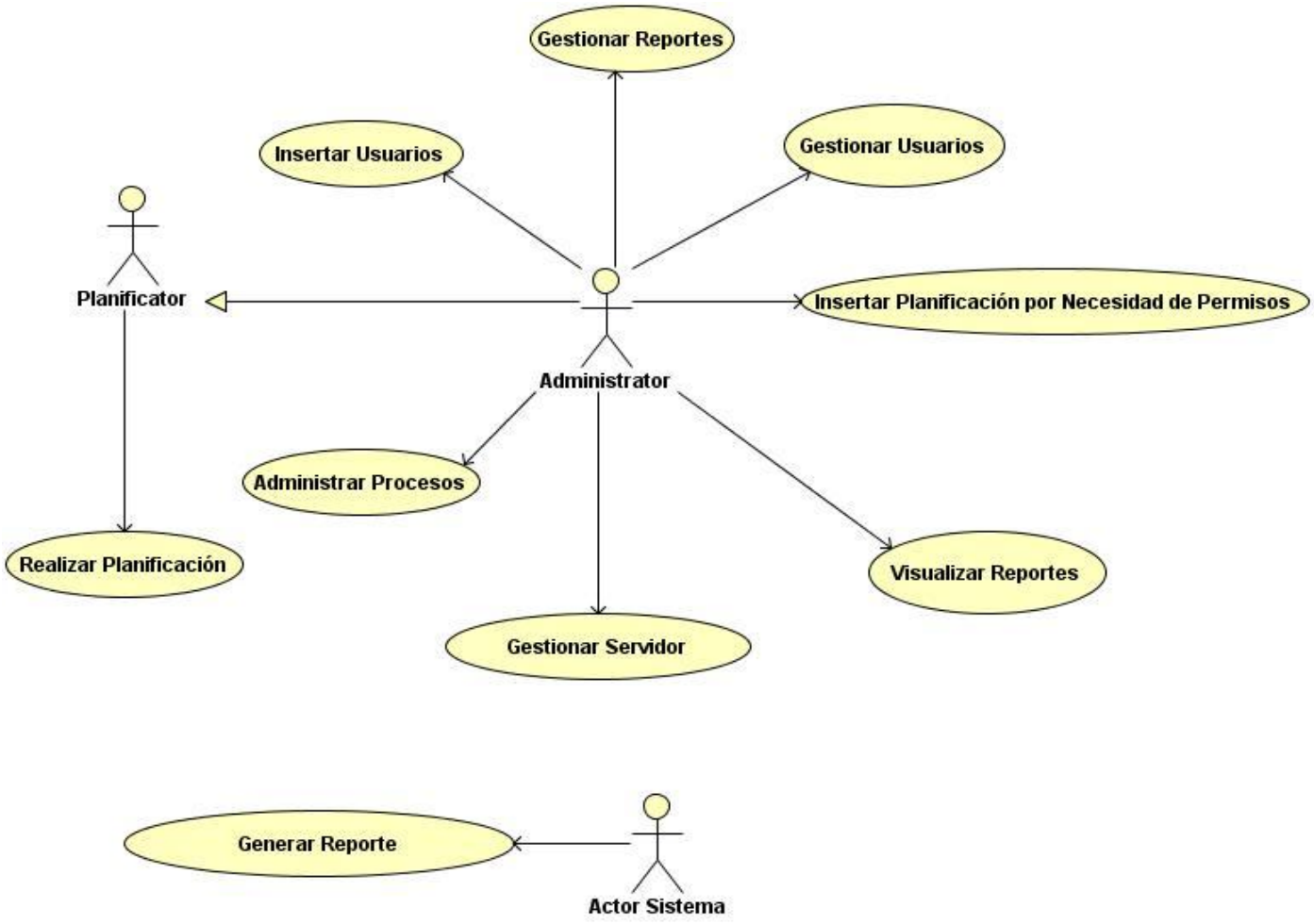

Figura 5: Vista de casos de uso del módulo Administración

Estos casos de uso constituyen la base para mantener la administración del sistema, haciendo énfasis la seguridad del mismo.

Módulo de Captura e Indexación de Video:

- Realizar Captura de Video
- Extraer Fotogramas
- Generar Índices
- Transferir Material
- Gestionar Video
- Digitalizar
- Registrar Datos
- Realizar Transcodificación

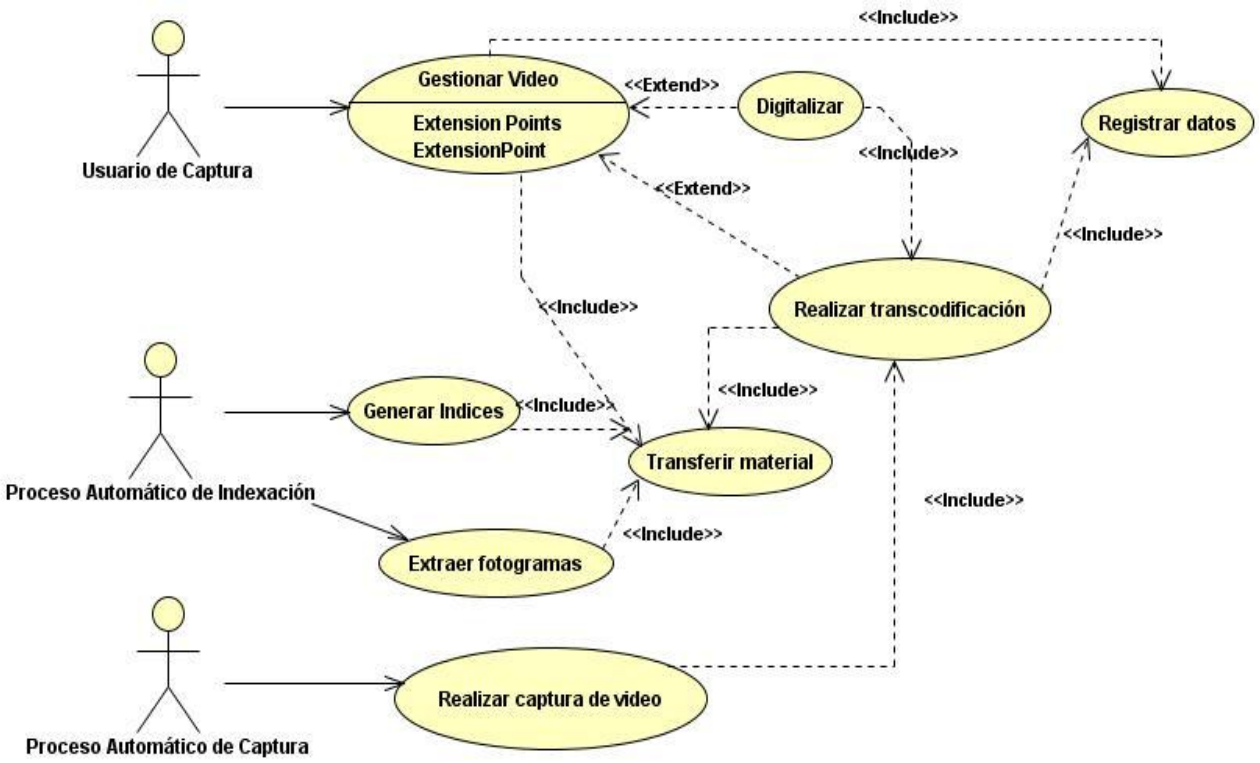

Figura 6: Vista de casos de uso del módulo Captura e Indexación de Video

Estos casos de uso son la base para el trabajo con videos, los mismos son indispensables para poder realizar la catalogación.

Módulo de Captura y Transcripción de Audio:

- Grabar Audio
- Transcribir Audio
- Transferir Media
- Registrar Datos
- Consultar Archivo
- Visualizar Reportes
- Gestionar Archivo
- Realizar Transcodificación

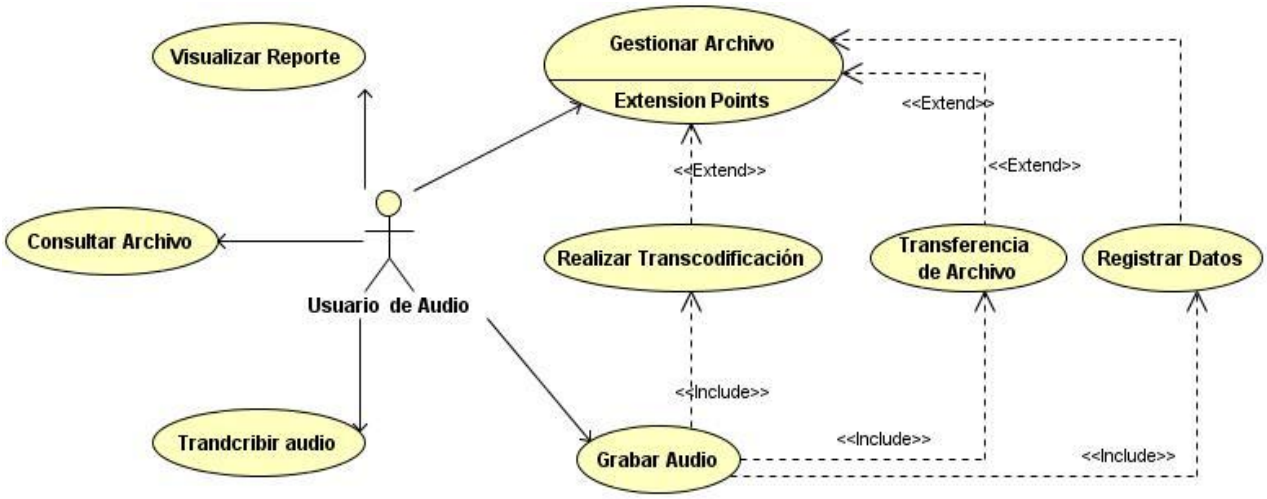

Figura 7: Vista de casos de uso del módulo Captura y Transcripción de Audio

Estos casos de uso describen las principales operaciones a realizar con audio, los mismo son la base para que la parte del audio funcione correctamente en el sistema.

Módulo de Catalogación de Medias

- Reproducir medias
- Catalogar Medias
- Editar Medias
- Visualizar Reportes

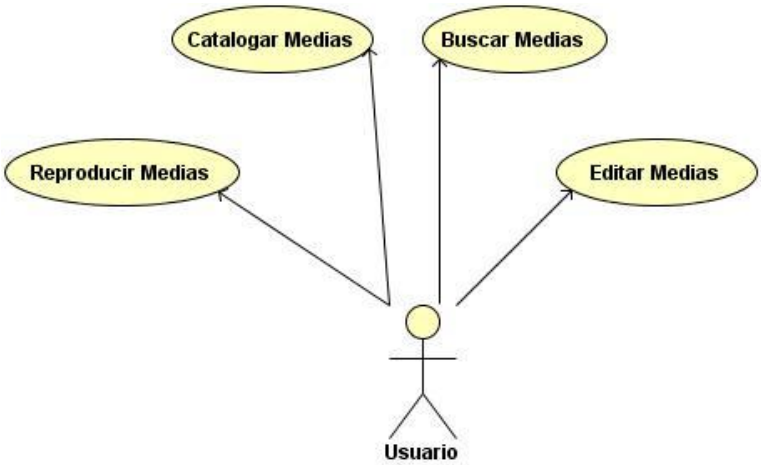

Figura 8: Vista de casos de uso del módulo Catalogación de Medias

Estos casos de uso en gran parte dependen de la realización de los casos de uso de los módulos Captura e Indexación de Videos y Captura y Transcripción de Audio aunque pueden realizarse también de forma independiente.

A continuación se muestra la visión general sobre el sistema CCM:

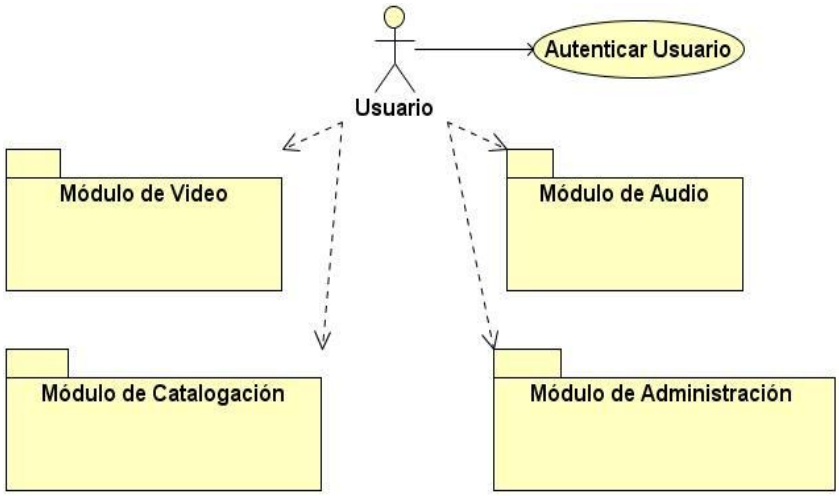

Figura 9: Vista general de casos de uso del sistema

En la vista general del sistema se especifica un solo caso de uso que es autenticar y su relación con los cuatro módulos, este caso de uso es arquitectónicamente significativo para todos los módulos del proyecto y es por esto que no se representa en ningún módulo específico.

3.5.2 Vista Lógica

En la descripción de la arquitectura, la vista lógica describe las clases más importantes que formarán parte del ciclo de desarrollo. Se describe los paquetes más abstractos del sistema y las relaciones que entre ellos existen ya sea de dependencia o de uso. La siguiente imagen muestra la distribución de los paquetes más significativos dentro de cada una de las capas definidas para la aplicación y la dependencia entre ellos.

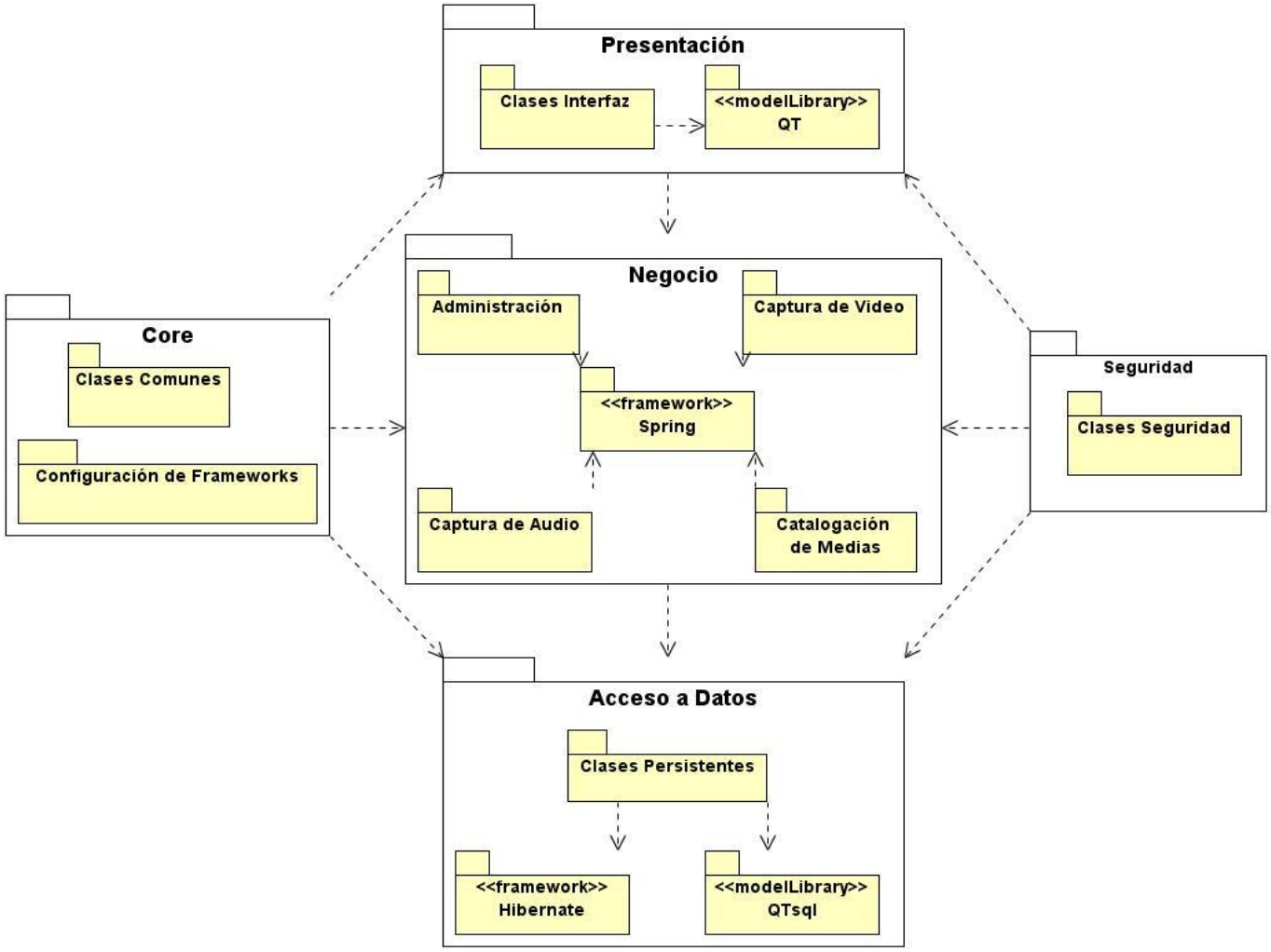

Figura 10: Vista Lógica

Distribución en Capas

La vista está formada por cinco paquetes principales:

Presentación

En este paquete están las clases y componentes de la librería QT. Está formado por componentes de interfaz, como son los formularios *Windows Forms*, los componentes de esta capa administran la interactuación con el usuario, muestran los datos al mismo, obtienen sus datos personales e interpretan los eventos generados para actuar en los datos del negocio, cambiar el estado de la interfaz o facilitar la tarea al cliente.

La estructura de la capa de Presentación es la siguiente:

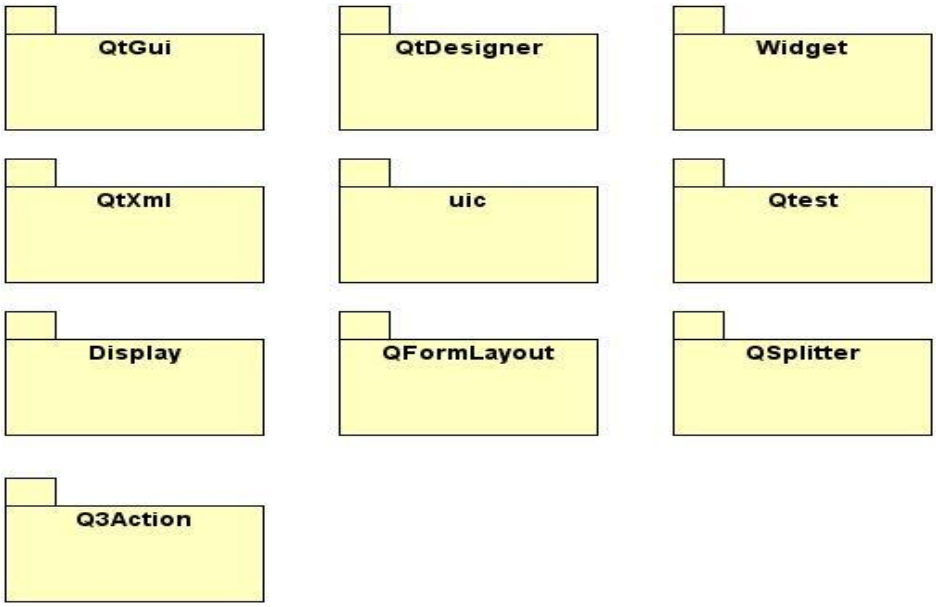

Figura 11: Paquetes de la Capa de Presentación

- Paquete **QtXml**: Contiene las clases para el manejo de XML.
- Paquete **QtDesigner**: Contiene las clases del diseño para la aplicación de QT
- La clase **Widget** funciona como la clase base de los componentes.
- El paquete **QtGui** contiene un conjunto de clases para el desarrollo de interfaz.
- La clase **QSpliter** implementa un separador *widget*, un separador permite al usuario controlar el tamaño de los componentes *widgets* arrastrando el límite entre los componentes.
- Paquete **Qtest**: Contiene todas las funciones y las declaraciones que están relacionadas con la herramienta qtestLib (herramienta para las pruebas de unidad de aplicaciones basadas en Qt).
- Paquete **UIC** (*User interface compiler*): Contiene la funcionalidad de implementar la clase del modelo realizado con QtDesigner a partir del archivo .ui, es decir obtener los archivos .cpp y .h.
- La clase **Display** tiene las secciones que se muestran en el momento de editar.
- La clase **QFormLayout** gestiona las formas de entrada de reproductores y sus etiquetas. QFormLayout es una clase que establece la disposición de sus hijos en una forma de dos columnas. La columna de la izquierda consta de las etiquetas y la columna de la derecha se compone de "campo" *widgets* (línea de editores, spin cajas, etc.)

- La clase **Q3Action** proporciona una interfaz de usuario resumen de acción que pueden aparecer tanto en los menús y barras de herramientas

Negocio

Contiene los componentes del framework Spring y las clases controladoras que encapsulan la lógica del negocio separadas en los subpaquetes Administración, Captura e Indexación de Videos, Captura y Transcripción de Audio y Catalogación de Medias.

La estructura de la capa de Negocio es la siguiente:

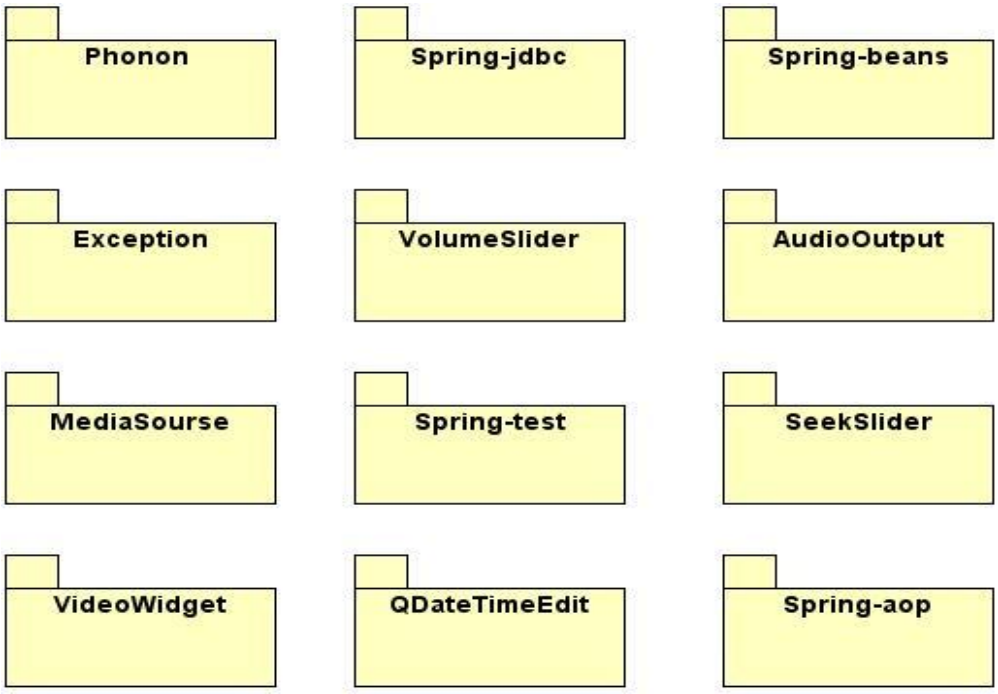

Figura 12: Paquetes de la Capa de Negocio

- La clase **Excepción:** Proporciona una clase base para excepciones que pueden transferirse a través de los hilos.

- Paquete **Phonon**: Contiene las clases y funciones para aplicaciones multimedia.
- El paquete **Spring-jdbc**: A partir de este se pueden realizar los métodos de insertar, modificar y eliminar, pero además se puede integrar con Hibernate para brindar funcionalidades más potentes.

- Paquete **Spring-aop**: Contiene los aspectos definidos, los que pueden ser ejecutado en varios métodos o clases.
- Clase **Spring-test**: Puede extender de la clase AbstractDependencyInjectionSpringContextTests o AbstractTransactionalDataSourceSpringContextTests, la primera para el soporte de inyección de dependencias y la segunda para realizar transacción por cada uno de los métodos de la base de datos.
- Paquete **Spring-beans**: Contiene los beans que tienen la particularidad de ser reutilizable y así evitar la tediosa tarea de programar los distintos componentes uno a uno.
- El *widget* **VolumeSlider** ofrece un control deslizante que se utiliza para controlar el volumen de un dispositivo de salida de audio. El control deslizante también muestra un ícono que indica si el volumen de la AudioOutput está conectado a está silenciado.
- La clase **MediaSource** proporciona datos multimedia para los objetos de las medias.
- La clase **SeekSlider** ofrece un control deslizante para buscar posiciones en el *streaming* de la media.
- La clase **VideoWidget** proporciona un *widget* que se usa para mostrar video.
- La clase **QDateTimeEdit** proporciona un *widget* para la edición de las fechas y las horas.

Acceso a Datos

Clases que controlan la lógica transaccional y la interacción con la base de datos, componentes del framework Hibernate. El paquete Map proporciona la lógica para el mapeo de las tablas de la base de dato y el paquete DAO que contiene clases correspondientes a cada entidad persistente, las cuales se apoyan en las clases del paquete Map para obtener los datos de la BD y crear los objetos que van a pasar a la capa del negocio.

La estructura de la capa de Acceso a Datos es la siguiente:

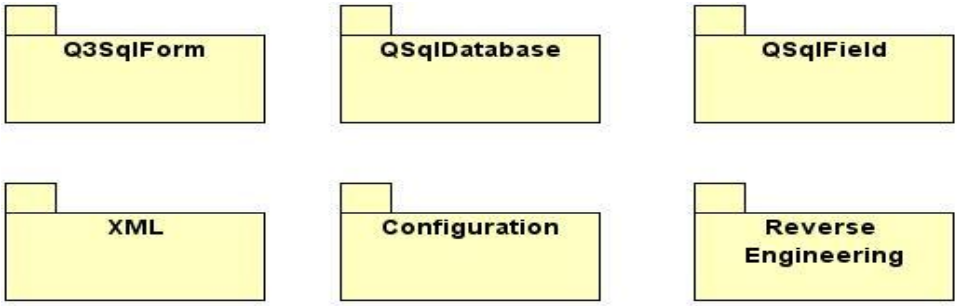

Figura 13: Paquetes de la Capa de Acceso a Datos

- Paquete **XML**: Contiene los ficheros XML para la configuración de los *Beans* de cada uno de los servicios.
- La clase **Q3SqlForm** crea y gestiona los formularios de entrada de datos vinculados a bases de datos SQL
- La clase **QSqlDatabase** representa una conexión a una base de datos
- La clase **QSqlField** manipula los campos de base de datos SQL, dígase tablas y vistas.
- El archivo **Configuration** es un fichero de Hibernate donde se describe la conexión de la BD para realizar el mapeo.
- El archivo **Reverse Engineering** es el fichero que permite realizar el proceso de ingeniería inversa en el framework Hibernate para el mapeo.

Core

Paquete que encierra clases comunes, componentes comunes para todas las capas de la aplicación, contiene la configuración de los frameworks.

Clases comunes: clases que se utilicen por diferentes módulos con el mismo objetivo.

Componentes comunes: Componentes que se utilicen por diferentes módulos con el mismo objetivo.

Configuración de frameworks: En este caso solo se tiene la configuración del framework Hibernate.

Seguridad:

Paquete que encierra clases y componentes para lograr la seguridad del sistema, aquí se programa todo lo que tiene ver con seguridad del sistema, es decir desde la parte que debe llevar cada capa para lograr su eficacia hasta la trasmisión de información hacia cada capa.

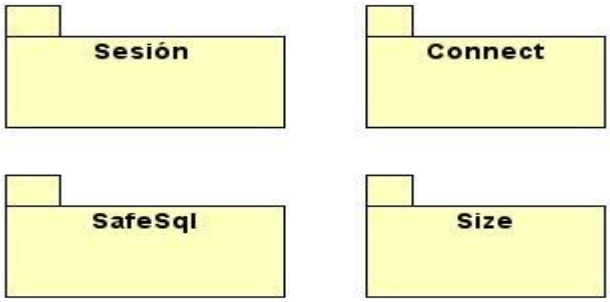

Figura 14: Paquetes de la Capa Seguridad

- El paquete **Sesión** contiene la implementación de las variables se sesión para cada usuario.
- El paquete **Connect** contiene la configuración de los conectores para acceder a cada capa.
- El paquete **SafeSql** contiene la declaración de la sentencia a utilizar para evitar inyección sql.
- El paquete **Size** contiene las clases para llevar el control del tamaño de la base de datos y el de media.

3.5.3 Vista del Modelo de Análisis

Esta vista muestra los artefactos del modelo de análisis que son significativos para la arquitectura. Según Jacobson, Booch y Rumbaugh (31) estos son:

- Clases de análisis que son generales, centrales y que tienen muchas relaciones con otras clases.
- Clases entidad que encapsulan un fenómeno importante del dominio del problema.
- Clases interfaz que encapsulan interfaces de comunicación importantes y mecanismos de interfaz de usuario.
- Clases de control que encapsulan importantes secuencias con una amplia cobertura, o sea, que coordinan realizaciones de casos de uso significativos.

La descripción de las clases se realiza a través del Modelo de Análisis, se presenta un documento para cada módulo.

3.5.4 Vista del Modelo de Diseño

Esta vista muestra los artefactos del modelo de diseño que son significativos para la arquitectura. Según Jacobson, Booch y Rumbaugh en su libro “El Proceso Unificado de Desarrollo de Software” (31) estos son:

- La descomposición del modelo de diseño en módulos, sus interfaces y las dependencias entre ellos.
- Clases de diseño que posean una traza con clases de análisis significativas.
- Clases de diseño generales y centrales que representen mecanismos de diseño genérico o que tengan muchas relaciones con otras clases de diseño.

La descripción de las clases se realiza a través del Modelo de Diseño, se presenta un documento para cada módulo.

3.5.5 Vista de Procesos

Esta vista suministra una base para la comprensión de la organización de los procesos de un sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Solo suele usarse cuando el sistema presenta procesos concurrentes o hilos.

A continuación se muestra como queda la representación de esta vista:

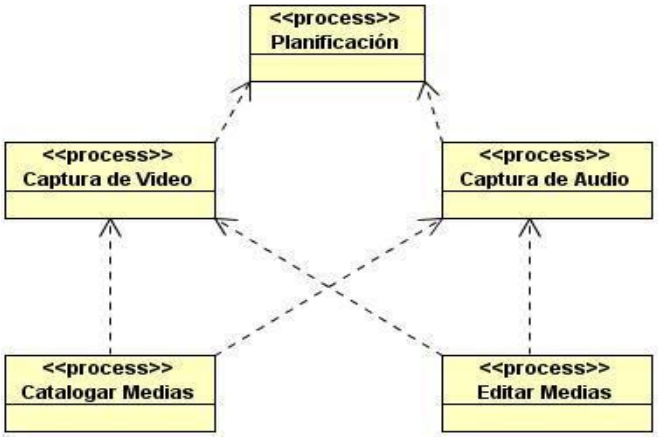

Figura 15: Diagrama de Procesos

Los procesos de captura de audio y video se pueden realizar de forma concurrente en cualquier momento pero ambos depende del proceso planificación. También pueden ejecutarse de forma concurrente los procesos de catalogación de medias y edición de medias pero los mismos dependen de la captura de audio y video.

3.5.6 Vista de Implementación

La vista de implementación proporciona una descripción de las principales capas y módulos de componentes de la aplicación. Los paquetes principales de la aplicación por cada uno de los módulos son:

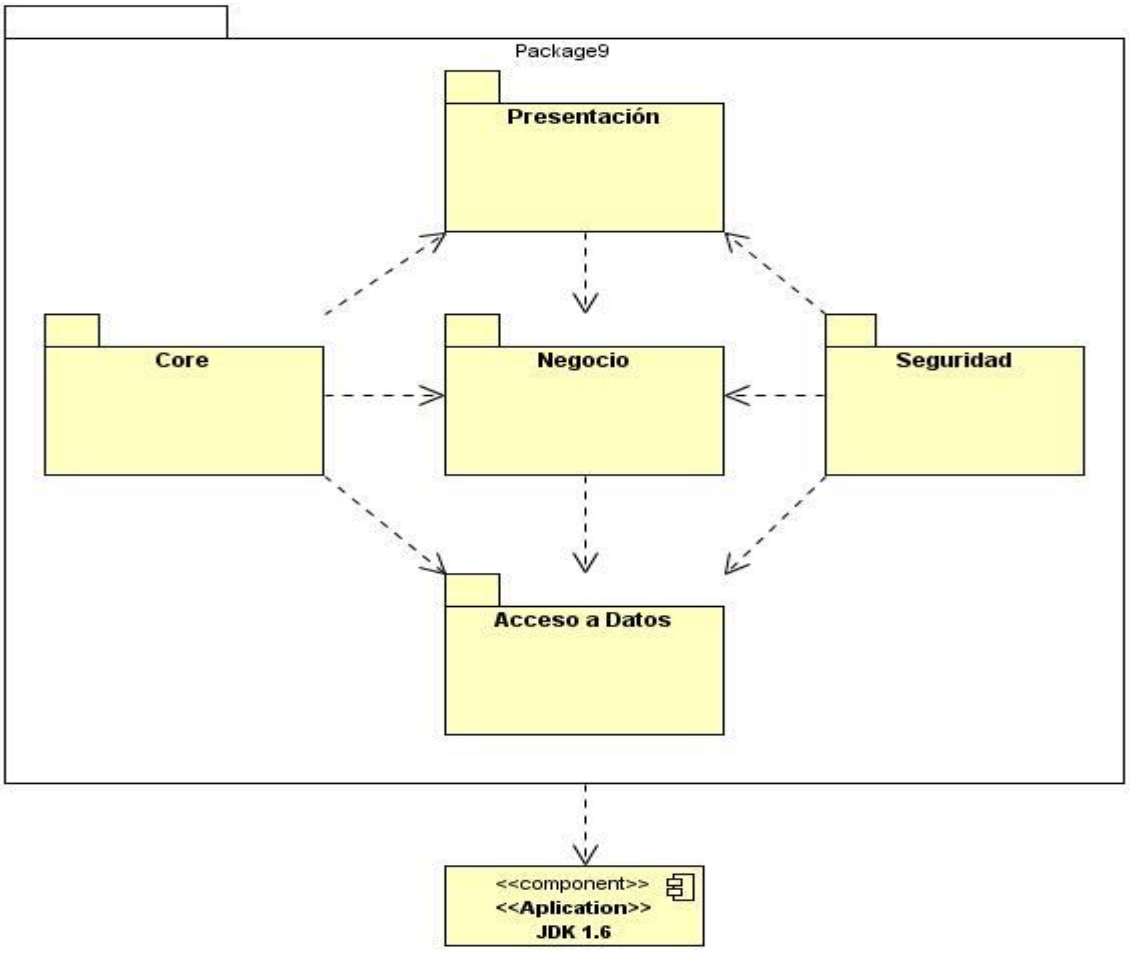

Figura 16: Vista general del modelo de implementación

Cada uno de dichos paquetes encapsulan uno o más componentes que se interrelacionan entre ellos para darle solución a la aplicación y todos depende de la JDK que debe de estar instalada en la estación de trabajo que decida ser instalada la aplicación.

Los componentes están distribuidos en las capas y paquetes de la aplicación de la siguiente forma:

Presentación

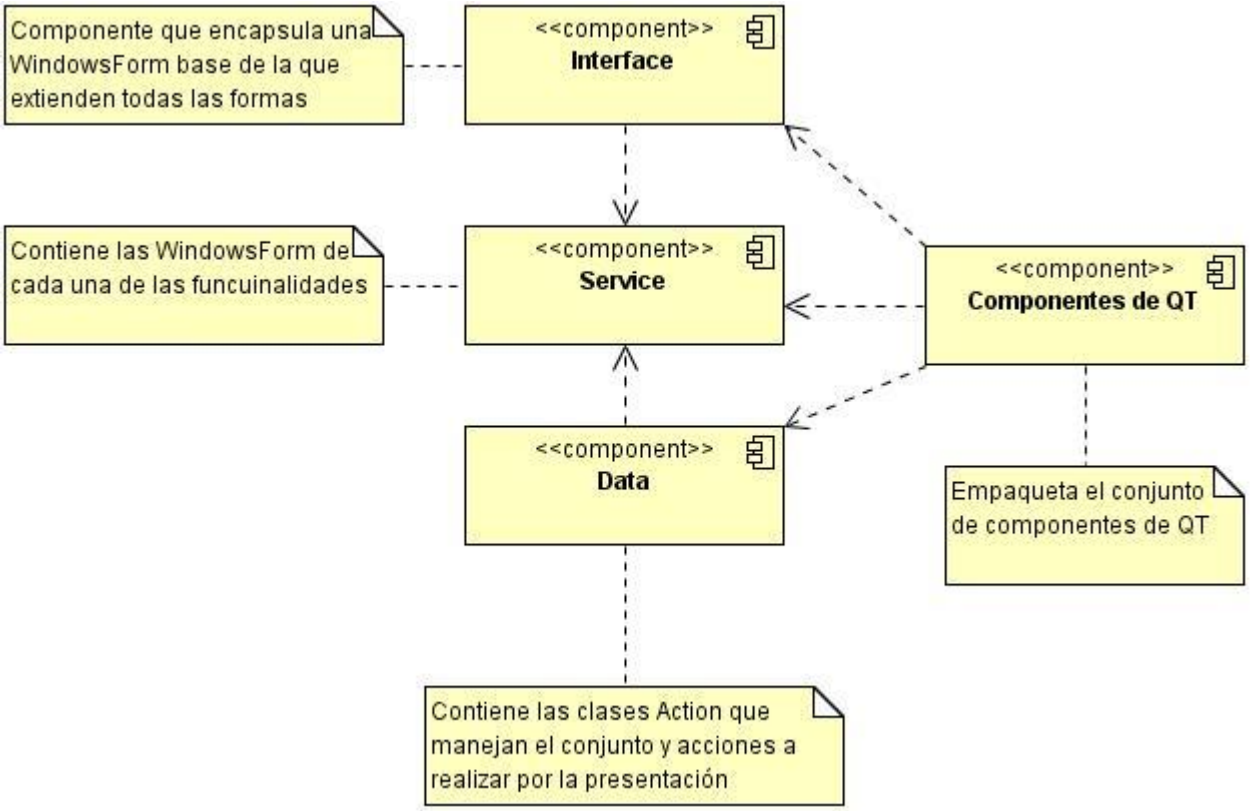

Figura 17: Componentes de la capa Presentación

Negocio

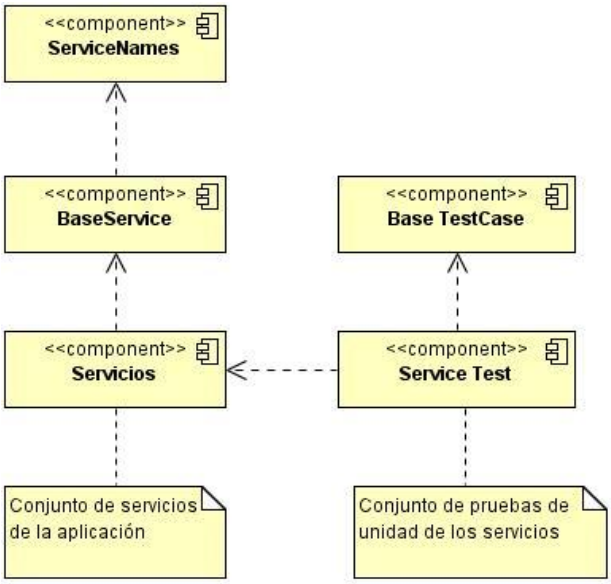

Figura 18: Componentes de la capa Negocio

Acceso a Datos

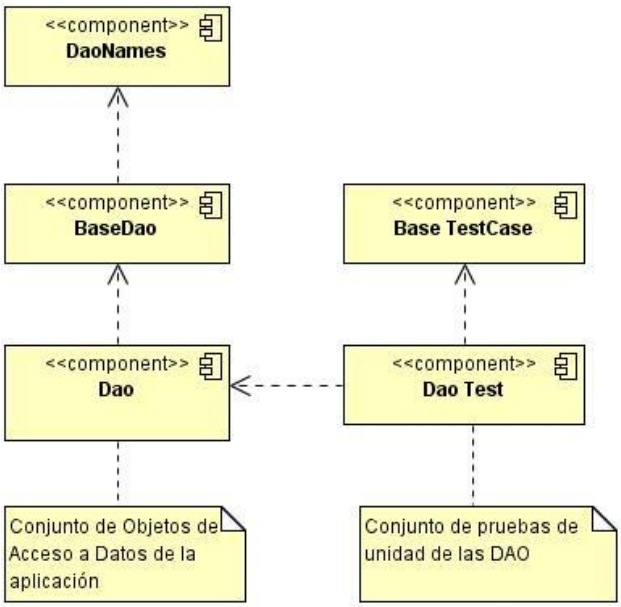

Figura 19: Componentes de la capa Acceso a Datos

Seguridad:

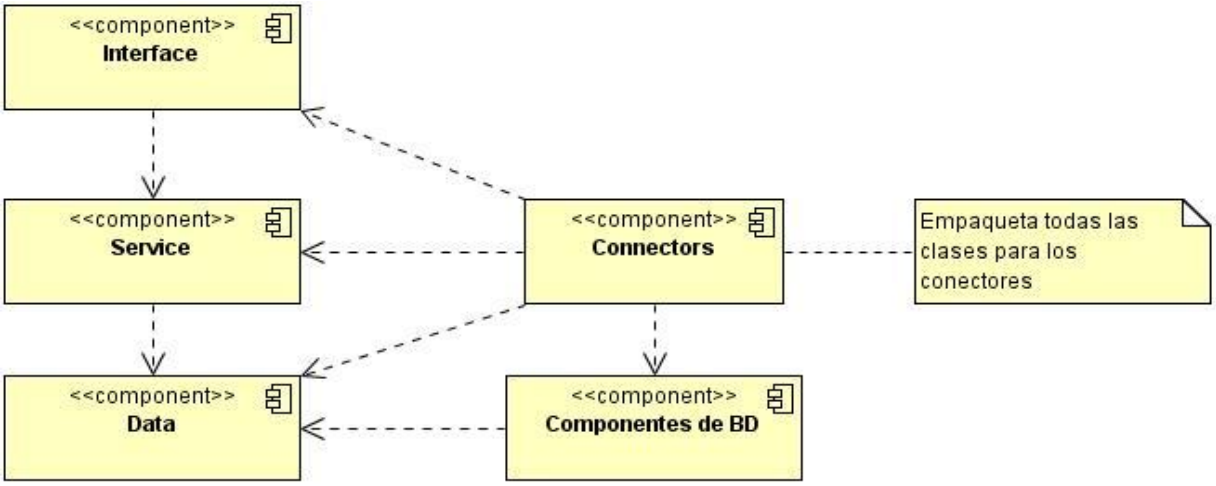

Figura 20: Componentes de la capa Seguridad

Core:

Contiene los componentes fundamentales para el trabajo en cada una de las capas:

- **BaseForm**: Componente del cual heredan las Forms de la aplicación.
- **BaseAction**: Componente que contiene las funcionalidades básicas de las acciones para todas las *Forms* de la aplicación.
- **BaseService**: Componente que contiene las funcionalidades básicas de todos los servicios de la aplicación.
- **BaseDao**: Componente que contiene las funcionalidades básicas que maneja el acceso a los datos de la aplicación.
- **FormNames**: Componente que maneja cada una de las *WindowsForm* creadas en la aplicación.
- **ActionNames**: Componente que maneja cada una de las clases *Action* de la subcapa de acciones creadas en la aplicación.
- **ServiceNames**: Componente que maneja cada uno los servicios de la aplicación.
- **DaoNames**: Componente que maneja cada uno de los DAO (Objetos de Acceso a Datos) creados en la aplicación.

- **Paquete init**: Conjunto de componentes que manejan el arranque de la aplicación, la configuración del contexto de la aplicación.
- **Paquete VisualUtil**: Contiene componentes de utilidad para las distintas capas por ejemplo: ColumnNames y SearchParameters.
- **SearchParameter:** Contiene los componentes que permiten obtener los parámetros de búsquedas para la capa de presentación

El conjunto de paquetes identificados permiten que cualquiera de ellos pueda ser reutilizado, en esta y en futuras aplicaciones, permite además el mantenimiento de la aplicación.

3.5.7 Vista de Despliegue

La vista de despliegue propone la distribución física del sistema y como estarán distribuidos los componentes de la aplicación en ellos. Suele utilizarse cuando el sistema es distribuido. Hay una traza directa del modelo de implementación, puesto que cada componente físico debe estar almacenado en un nodo, esto incluye también la asignación de tareas provenientes de la vista de procesos en los nodos.

3.5.7.1 Diagrama de Despliegue

Un diagrama de despliegue muestra las relaciones físicas entre los componentes hardware y software en el sistema. Es un conjunto de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes software, objetos y procesos.

El diagrama que se muestra en la figura 21 es la mejor solución para que el sistema funcione de manera eficiente. Para el desarrollo de este sistema se propone el que se muestra en la figura 22 debido a que el país no cuenta con los recursos necesarios, aunque no brinda las mismas comodidades que el anterior. En caso que se desee realizar un sistema similar y no se cuente con los recursos necesarios o que el trabajo que se realice con las medias no sea grande, es decir que se capture audio o video escasas veces y que igual se cataloguen los mismos raras veces entonces la captura del audio y el video se pudieran realizar en una misma computadora, además en este caso el servidor de base de datos y el servidor de medias estarían en una misma PC, esto se muestra en el diagrama 23.

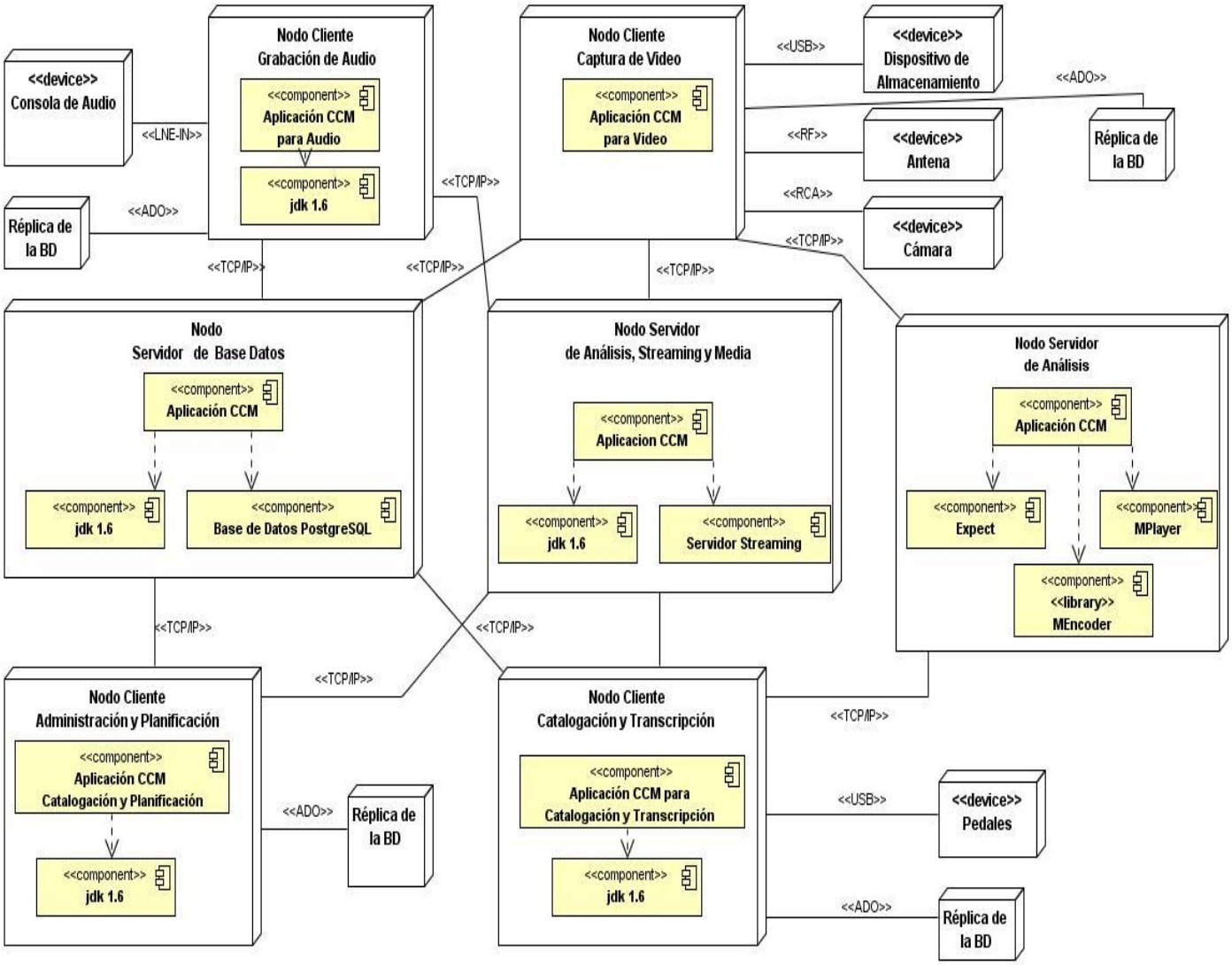

Figura 21: Diagrama de Despliegue Recomendado

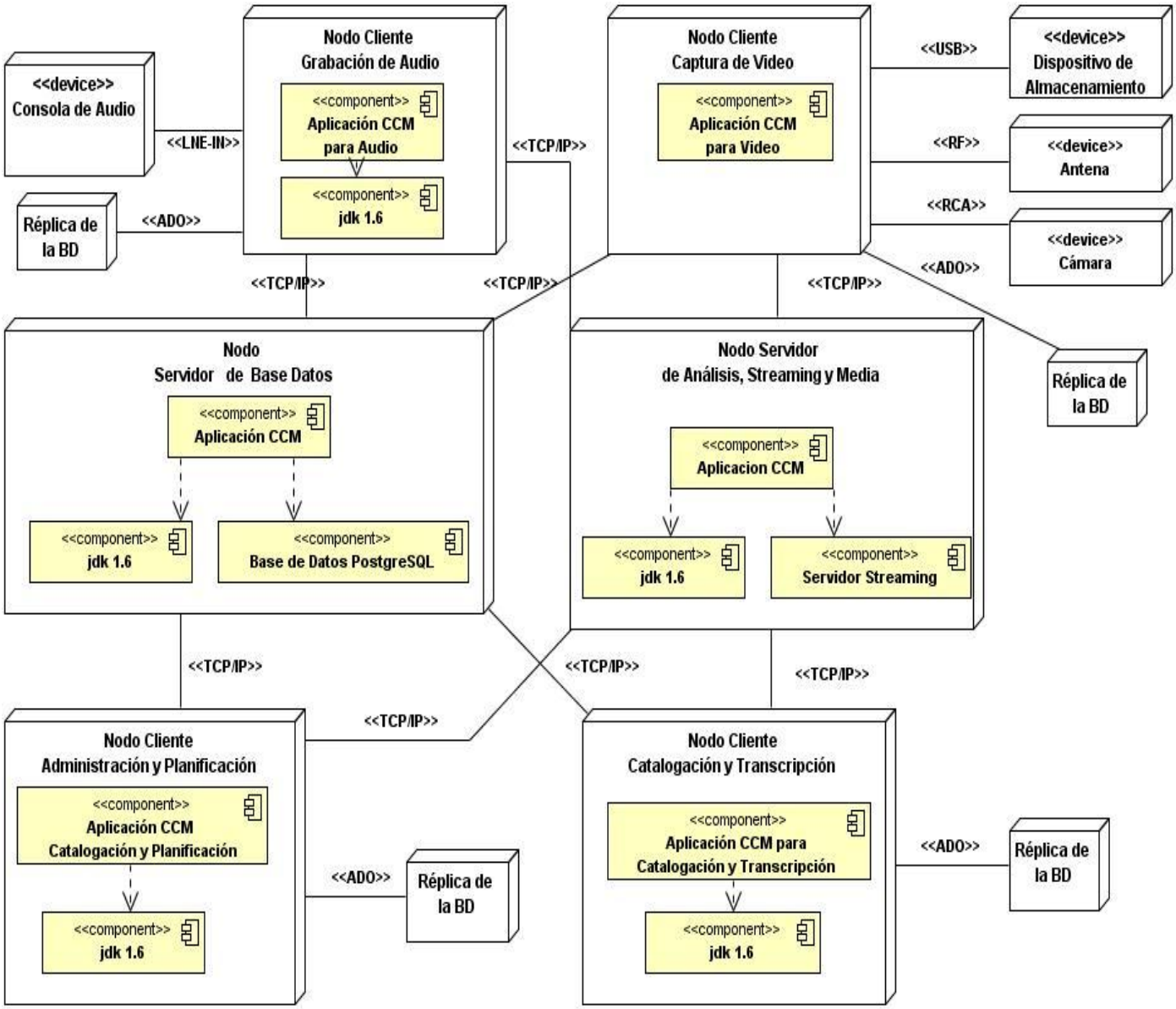

Figura 22: Diagrama de Despliegue Propuesto

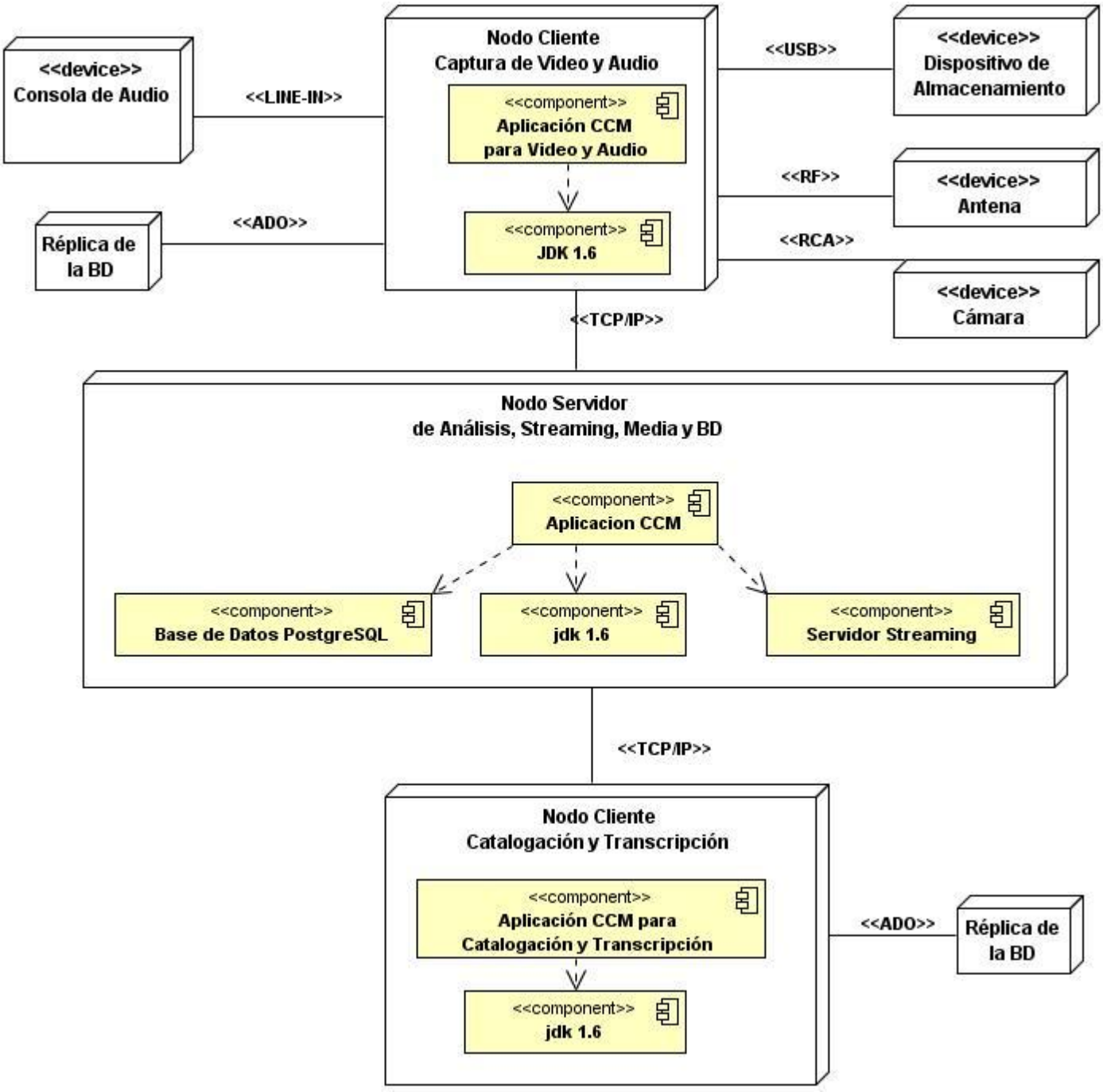

Figura 23: Diagrama de Despliegue Optimo

A continuación se describirán los nodos físicos representados en el diagrama de despliegue:

Consola de Audio

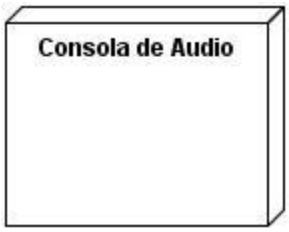

Figura 24: Nodo Consola de Audio

Representa el nodo consola de audio cuya función es hacer llegar el audio a las maquinas donde se va a capturar el audio.

Grabación de Audio

Figura 25: Nodo Grabación de Audio

Representa el nodo grabación de audio, cuyo objetivo es capturar la entrada de audio para grabar el mismo y enviarlo al servidor de medias.

Dispositivo de Almacenamiento

Figura 26: Nodo Dispositivo de Almacenamiento

Representa el nodo dispositivo de almacenamiento (memoria y disco duro) el cual contiene el video que se desea capturar.

Antena

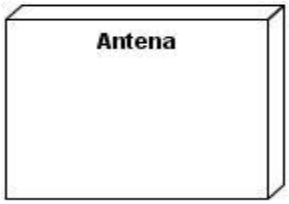

Figura 27: Nodo Antena

Representa el nodo antena que permite realizar la captura de video a través de sí mismo.

Cámara

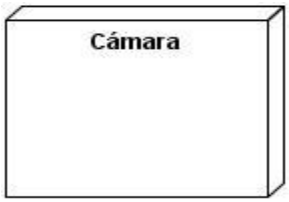

Figura 28: Nodo Cámara

Representa el nodo cámara en el cual se tiene el video que se desea capturar.

Captura de Video

Figura 29: Nodo Captura de Video

Representa el nodo captura de video, encargado de capturar a través de una tarjeta especifica el video…

Servidor de Base de Datos

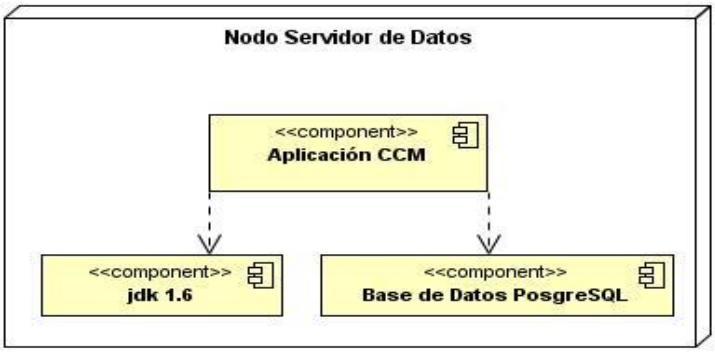

Figura 30: Nodo Servidor de BD PostgreSQL

En este nodo se estará ejecutando el servidor PostgreSQL. La lógica del tratamiento de los datos no se implementará aquí sino en la misma aplicación en la capa de Acceso de a Datos.

Servidor de Análisis, Streaming y Medias

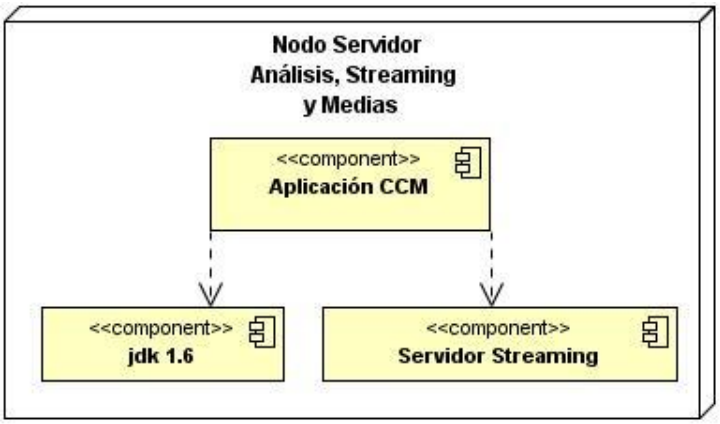

Figura 31: Nodo Servidor de Análisis, Streaming y Medias

Constituye el nodo servidor de análisis, streaming y medias cuyo propósito es proveer los datos de modo que otras máquinas puedan utilizar esos datos.

Catalogación, Transcripción y Presentación

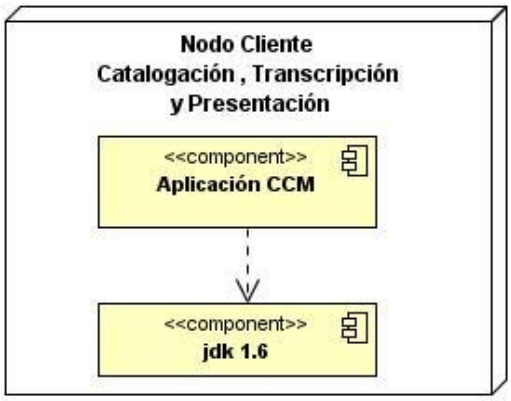

Figura 32: Nodo de Catalogación, Transcripción y Presentación

Representa el nodo que permite realizar varias operaciones como catalogar materiales grabados, transcribir materiales grabados y presentar que no es más que visualizar los materiales en varias formas.

Administración y Planificación

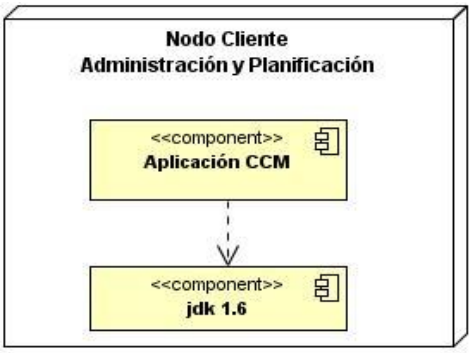

Figura 33: Nodo de Administración y Planificación

Representa el nodo de Administración y Planificación que permite lograr la administración del sistema y la planificación de la grabación de los canales de interés.

3.6 Requerimientos no Funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. Estos se representan en varias categorías.

3.6.1 Requerimientos de Seguridad

- La seguridad se trata desde las primeras fases de desarrollo del sistema.
- Parte de la seguridad corre por parte de la tecnología Java utilizada para el desarrollo de la aplicación.
- La asignación de usuarios y sus opciones sobre el sistema se garantizan desde el módulo de Administración.
- Debe quedar constancia de quién, desde donde, y cuando se realizó una operación determinada en el sistema.
- Los mecanismos de seguridad no deben impedir que los usuarios autorizados accedan a la información, la misma debe estar disponible en todo momento.
- La base de datos deberá estar disponible las 24 horas, pudiendo realizar operaciones sobre la misma en cualquier momento que se necesite. En caso de una falla en el servidor (interrupción del servicio eléctrico, desconexión momentánea o caída de la red) se deberá contar con un respaldo para cada módulo que le permita seguir con sus actividades.

3.6.2 Restricciones de acuerdo a la estrategia de diseño

- El diseño de las aplicaciones se hará utilizando la Programación Orientada a Objetos (POO).Encapsulación de la lógica por clases.
- El sistema deberá ser desarrollado como una aplicación de escritorio. Será multiplataforma y libre.
- Se utilizarán las tecnologías que brindan los *frameworks* definidos para cada una de las capas de la aplicación:
  1. Para la capa de presentación: La librería QT, aplicación de escritorio.
  2. Para la capa de lógica del negocio: los objetos del negocio, framework Spring, la IoC y la programación orientada a aspectos.
  3. Para la capa de Acceso a Datos: framework Hibérnate.

3.6.3 Portabilidad, escalabilidad, reusabilidad

- El sistema deberá hacer un uso racional de los recursos de hardware de la máquina, sobre todo en las PC clientes.
- Se desarrollará cada pieza del sistema en forma de componentes (elementos) con el fin de ser utilizados en futuras versiones del sistema.
- La documentación de la arquitectura deberá ser reutilizable para poder documentarla como una familia de productos.

3.6.4 Redes

- La red existente en las instalaciones debe de soportar la transacción de paquetes de información de al menos unas 10 máquinas a la vez.
- Para hacer más fiable la aplicación debe de estar protegida contra fallos de corriente y de conectividad, para lo que se deberá parametrizar los tiempos para realizar copias de seguridad. Implementar las transacciones de paquetes en la red con el protocolo TCP/ IP que permite la recuperación de los datos.

3.6.5 Soporte

- El sistema permitirá la modificación o agregarle módulos cuando sea necesario, asegurando su extensibilidad y lograr mejores prestaciones.

3.6.6 Usabilidad

- La aplicación debe ser concebida para ser utilizada por personas que tengan conocimientos básicos de informática en el trabajo con aplicaciones de escritorio y en el trabajo con medias.

- La aplicación garantizará una conexión segura y eficiente con la base de datos, que tendrá almacenada toda la información del sistema, mejorando esto la gestión de la información y los datos.

3.6.7 Apariencia o Interfaz Externa

- Las interfaces de la aplicación tendrán los componentes visuales necesarios para las operaciones correspondientes, evitando la sobrecarga de imágenes.
- El sistema debe mostrar un ambiente profesional, sin información repetida o en exceso.
- El diseño será sugerente y permitirá al usuario navegar con facilidad e intuición por la aplicación.
- La aplicación debe ser rápida, flexible y adaptable al entorno.

3.6.8 Rendimiento

- El sistema debe responder en un tiempo relativamente rápido a las peticiones del usuario (menos de 5 segundos)
- La BD deberá resistir una alta concurrencia, así como un gran número de operaciones simultáneas sobre la misma.

3.6.9 Requerimientos de Software

En las PCs Clientes

- Sistema Operativo: Ubuntu 8.10
- Driver de sonido: ALSA 1.0.19 (Para audio)
- Reproductor: MPlayer (Para video)
- Aplicación MEncoder (Para video)

En las PCs Servidor:

- Sistema Operativo: Ubuntu 8.10
- Gestor de Base de Datos: PostgreSQL 8.3

3.6.10 Requerimientos de Hardware

PCs Captura de Video.

- Tarjeta : Hauppauge WinTV-PVR-350 modelo 00990 acoplada

- Memoria RAM: 2 GB
- Microprocesador: Quad Core.
- Capacidad de almacenamiento en disco duro : 250 GB

PCs Captura de Audio

- Motherboard: Asus P5LD2-VM
- Memoria RAM: 512 MB DDR2-533 SDRAM
- Procesador: Intel Pentium 4 CPU 3.00 GHz (simulación 2 procesadores)
- Capacidad de almacenamiento en disco duro: 80 GB

PCs Catalogación

- Memoria R.A.M: 512 Mb DDR2-533 SDRAM
- Espacio libre en H.D: 500 MB
- Procesador: Intel Pentium 4 CPU 3.00 GHz

PCs Administración

- Memoria R.A.M: 512 Mb DDR2-533 SDRAM
- Capacidad de almacenamiento en disco duro: 80 GB
- Procesador: Intel Pentium 4 CPU 3.00 GHz

Servidor de Análisis:

- Memoria RAM: 4 GB
- Microprocesador: Quad Core.
- 1 TB de disco duro en RAID 5.

Servidor de Medias:

- Memoria RAM: 2 GB
- Microprocesador: Quad Core.
- 5 TB de disco duro en RAID 5.
- Tarjeta de Red: Ethernet a 100 Mbps

Servidor de Base de Datos:

- Memoria RAM: 1 GB.
- Capacidad en disco duro: 80 GB
- Procesador: Intel Pentium 4 CPU 3.00 GHz

3.7 Conclusiones del Capítulo

En este capítulo se realiza la propuesta de la arquitectura del proyecto Captura y Catalogación de Medias, definiendo las herramientas para modelado y desarrollo, la estructura del equipo de desarrollo, los módulos que conforman al proyecto, las vistas arquitectónicas y los requisitos no funcionales. A partir de los resultados obtenidos ya está disponible todo para la implementación

Capítulo 4
Evaluación de la Arquitectura del Sistema

4.1 Introducción

La arquitectura es un artefacto definitivo para la calidad del software que se desarrolla. Durante la evaluación de la misma se puede conocer los diferentes riesgos asociados con el desarrollo del sistema y como la arquitectura es un producto temprano son muchos los errores que se pueden corregir durante el flujo de requerimiento o diseño siendo estos menos costosos que si tuvieran que aclararse en implementación o prueba. En este capítulo se analizan los resultados obtenidos a través de dos estrategias de prueba y se valoran dos sistemas con arquitecturas similares a la del proyecto que se desarrolla para comparar los resultados obtenidos.

4.2 Análisis de Arquitecturas de Software

La arquitectura de software posee un gran impacto sobre la calidad de un sistema, por lo que es muy importante estar en capacidad de tomar decisiones acertadas sobre ella, en diversos tipos de situaciones, entre las cuales destacan: (32)

- Comparación de alternativas similares.
- Comparación de la arquitectura original y la modificada.
- Comparación de la arquitectura de software con respecto a los requerimientos del sistema.
- Comparación de una arquitectura de software con una propuesta teórica.
- Valoración de una arquitectura en base a escalas específicas.

Evaluar una arquitectura, consiste en evaluar el potencial de la arquitectura diseñada, para alcanzar los atributos de calidad requeridos. Además, mediante la arquitectura de software es posible determinar la estructura del proyecto de desarrollo del sistema, sobre elementos como el control de configuración, calendarios, control de recursos, metas de desempeño, estructura del equipo de desarrollo y otras actividades que se realizan con la arquitectura del sistema como apoyo principal.

El tema de evaluar una arquitectura, estimar el comportamiento de los atributos ante cambios arquitectónicos, lograr hacer predicciones de diseño en etapas temprana, siempre han sido actividades de vital interés para la industria, debido a que todas estas predicciones aseguran la calidad del software, la disminución de los tiempos de desarrollo, y por lo tanto el esfuerzo y los costos de cada producto. (33)

La Universidad Técnica de Eindhoven, en Holanda, la cual ha tenido un desarrollo vertiginoso en el tema de estimación y evaluación de los atributos de calidad, se interesó además en la evaluación de la arquitectura. En esta universidad para el año 2001, se creó el grupo de Revisión y Evaluación de Arquitectura de Software (Software *Architecture Reviews and Assessment,* SARA). El SARA ha dedicado su trabajo a potenciar la aplicación de métodos de evaluación a través de casos de estudio, y diseñar nuevas herramientas que tributen a esta actividad. (SARA) (34)

Por otra parte la industria cubana del software está en una etapa de desarrollo. La madurez alcanzada por este país en desarrollos desde una perspectiva industrial está dando sus primeros pasos, tanto productivos como investigativos. Es por esto que la evaluación de arquitecturas de software todavía no es una práctica arraigada en la comunidad de desarrollo.

Para la evaluación de arquitecturas existen varias técnicas y métodos que son utilizados por la mayoría de las empresas productoras de software.

Técnicas

Las técnicas utilizadas para la evaluación de los atributos de calidad requieren grandes esfuerzos por parte de los ingenieros de software para crear especificaciones y predicciones. Estas técnicas requieren información del sistema a desarrollar que no está disponible durante el diseño arquitectónico, sino al principio del diseño detallado del sistema. (35)

Entre las técnicas de evaluación de arquitecturas de software propuestas por (Bosch) se encuentran: la evaluación basada en escenarios, evaluación basada en simulación, evaluación basada en modelos matemáticos y evaluación basada en experiencia.

En vista de que el interés es tomar decisiones de tipo arquitectónico en las fases tempranas del desarrollo, son necesarias técnicas que requieran poca información detallada y puedan conducir a resultados relativamente precisos. (35)

Hasta ahora las técnicas existentes para evaluar arquitecturas permiten hacer una evaluación cuantitativa sobre los atributos de calidad a nivel arquitectónico. Sin embargo, debido al costo de realizar este tipo de evaluación, en muchos casos los arquitectos de software evalúan cualitativamente, para decidir entre las alternativas de diseño.

Métodos

Por otra parte se reconocen los métodos propuestos por Kazman: el Método de Análisis de Arquitecturas de Software (*Software Architecture Analysis Method*, SAAM), el Método de Análisis de Acuerdos de Arquitectura (*Architecture Trade-off Analysis Method,* ATAM) y el (*Active Reviews for Intermediate Designs ARID*), entre otros.

Estos métodos son empleados para un alto nivel de análisis de toda arquitectura. Realizan solamente el análisis cualitativo de los atributos de calidad. Indican lo que se debe mejorar en la arquitectura para la satisfacción de los atributos de calidad esperados. Son métodos bastante generales, por lo que es grande el esfuerzo que se realiza para su aplicación en una arquitectura en particular. Para sus evaluaciones, se basan en la técnica de escenarios. De manera independiente, no logran abarcar una evaluación integral de los atributos de calidad.

En la UCI, fuente promotora de la industria de software cubana, un equipo de autores han definido un procedimiento de evaluación de arquitectura basado en el método ATAM, que aunque no ha sido generalizado hacia todos los proyectos de la universidad, y abarca un conjunto de elementos que no han logrado concretarse, es una realidad que ha sido el motor impulsor para muchas investigaciones y prácticas que se está llevando en el seno de cada equipos de desarrollo. (33)

4.3 Metodología Propuesta Para Evaluar la AS

Otro aspecto importante para la evaluación de la arquitectura es la metodología que se escoge para realizar las pruebas de arquitectura, en este caso en la UCI se realizó un trabajo donde se propuso una metodología, la misma cuenta con dos áreas de procesos fundamentales, el proceso de evaluación (base) y el proceso de documentación. A continuación se exponen las fases y actividades propias de cada proceso.

Proceso de evaluación.

Este proceso está conformado por cuatro fases, concepción, presentación, desarrollo y exposición de resultados, las cuales se desglosan en actividades. Este conjunto de acciones son la que impactan de forma directa sobre la evaluación de la arquitectura de software. Los artefactos que generan cada unas de las actividades son detallados en el proceso de documentación que se efectúa de manera paralela al mismo. (33)

A continuación se enumeran las fases del proceso y las actividades de cada una, en orden de sucesión.

- **Fase de concepción.**

1. Selección de los momentos en que se realizará la evaluación arquitectónica a lo largo del ciclo de desarrollo del software.
2. Realización del cronograma de evaluación.
3. Selección de los involucrados.
4. Selección de los integrantes del equipo de evaluación.
5. Preparación de la información.

- **Fase de presentación.**

6. Descripción del desarrollo del proceso evaluativo.
7. Presentación de los objetivos del negocio.
8. Presentación de la arquitectura de software.

- **Fase de desarrollo.**

9. Selección de los atributos de calidad.
10. Descripción del comportamiento esperado de cada atributo.
11. Determinación del grado de prioridad de cada atributo.
12. Identificación de los escenarios.
13. Descripción de los escenarios.

14. Establecimiento de la prioridad de los escenarios.
15. Determinar puntos críticos.

- **Fase de exposición de resultados.**

16. Valorar impacto de los posibles cambios arquitectónicos en los atributos de calidad.

Proceso de documentación.

El proceso de documentación es de suma importancia para la metodología propuesta, pues es quien permite la reutilización de toda la información que genera cada una de las evaluaciones, y permite crear una base de conocimientos que asegure la calidad de los proceso evaluativos en presencia de personal experto o no. (33)

Para esto de definieron un conjunto de artefactos documentales que se enumeran a continuación:

- Definición de evaluaciones de arquitectura de software.
- Cronograma de evaluaciones de arquitectura.
- Definición de los involucrados a participar en el proceso de evaluación
- Definición del grupo de evaluación.
- Definición de las metas del negocio
- Descripción de la arquitectura
- Cronograma de sesiones de la evaluación.
- Valoraciones de los objetivos del negocio.
- Valoraciones de la arquitectura presentada.
- Definición de los atributos de calidad.
- Comportamiento esperado de los atributos de calidad.
- Grado de prioridad los atributos de calidad.
- Identificación de los escenarios.

- Descripción detallada de los escenarios.
- Grado de prioridad de los escenarios.
- Puntos críticos de la arquitectura.
- Identificación de posibles cambios.
- Memorias del proceso de evaluación.

A partir de la metodología propuesta se realizan las estrategias de pruebas con el objetivo de verificar cuan factible es la arquitectura.

4.4 Estrategias de Prueba de la AS

Para realizar las estrategias de pruebas se analizaron los atributos de calidad más importantes para el cliente y se le concedió un orden de prioridad a los mismos para cubrir todas las necesidades que debe brindar el sistema. El orden se realizó entre los valores del [1-5], con prioridad baja para los valores [1-2], media [3] y alta para [4-5].

Atributos de Calidad	Prioridad		
	Alta	Media	Baja
Funcionalidad	X-5		
Integrabilidad	X-5		
Seguridad	X-4		
Eficiencia	X-4		
Usabilidad		X-3	
Portabilidad		X-3	
Mantenibilidad		X-3	
Reusabilidad			X-2

Tabla 5: Atributos de calidad

Funcionalidad: Habilidad del sistema para realizar el trabajo para el cual fue concebido.

Integrabilidad: Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados independientemente al ser integrados.

Seguridad: Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información y la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos.

Eficiencia: Capacidad del producto software para proporcionar tiempos de respuesta, tiempos de proceso y potencia apropiados, bajo condiciones determinadas.

Usabilidad: La facilidad con la cual las personas interactúan con la aplicación en forma efectiva, y sobre como el sistema provee soporte al usuario en este sentido

Portabilidad: Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.

Mantenibilidad: Es la capacidad de someter a un sistema a reparaciones y evolución. Capacidad de modificar el sistema de manera rápida y a bajo costo.

Reusabilidad: Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.

Primera Estrategia:

Consiste en evaluar la arquitectura a través de la misma metodología propuesta es decir se van obtener como resultados los 18 artefactos documentales del proceso de documentación de la metodología pero solo se describe el artefacto Memorias del Proceso de Evaluación que no es más que un breve resumen del proceso evaluativo. Esta estrategia fue desarrollada por el grupo de desarrollo de arquitectura de la facultad nueve y los líderes del proyecto.

Para desarrollar esta estrategia se siguió un cronograma en el cual se presenta como se desarrolla la misma:

Nº	Participantes	Actividades que abarca	Resultados
1	-Líderes del Proyecto. - Arquitectos. - Evaluador.	- Definir las Evaluaciones de Arquitectura. -Definir el Cronograma de Evaluaciones de Arquitectura. - Definir los Involucrados en el Proceso de Evaluación. - Definir el Grupo de Evaluación. - Definir las Metas del Negocio. - Describir la Arquitectura.	-Definición de las Evaluaciones de Arquitectura. - Cronograma de Evaluaciones de Arquitectura. - Involucrados en el Proceso de Evaluación. - Grupo de Evaluación. .- Metas del Negocio. - Descripción de la Arquitectura.
2	- Evaluador. -Líderes del Proyecto.	-Realizar Cronograma de Sesiones de Trabajo. -Realizar Valoraciones de las Metas del Negocio. -Realizar Valoraciones de la Arquitectura Presentada. - Definir los Atributos de Calidad. -Definir el Comportamiento Esperado de los Atributos de Calidad. - Definir el Grado de Prioridad de los Atributos de Calidad.	- Cronograma de Sesiones de Trabajo. - Valoraciones de las Metas del Negocio. - Valoraciones de la Arquitectura Presentada. - Atributos de Calidad. - Comportamiento Esperado de los Atributos de Calidad. - Grado de Prioridad de los Atributos de Calidad.
3	- Lideres del Proyecto. - Evaluador. -Equipo de Evaluación.	- Presentar el Cronograma de las Sesiones de Trabajo. -Presentar las Valoraciones sobre las Metas del Negocio. -Presentar las Valoraciones sobre la Arquitectura Presentada.	- Presentación del Cronograma de las Sesiones de Trabajo. -Presentación de las Valoraciones sobre las Metas del Negocio. -Presentación de las Valoraciones sobre la Arquitectura Presentada.

		-Presentar los Atributos de Calidad del Sistema. -Presentar el Comportamiento Esperado de los Atributos de Calidad. - Presentar el Grado de Prioridad de los Atributos de Calidad.	- Presentación de los Atributos de Calidad del Sistema. -Presentación del Comportamiento Esperado de los Atributos de Calidad. - Presentación del Grado de Prioridad de los Atributos de Calidad.
4	- Lideres del Proyecto. - Evaluador.	- Definir Escenarios.	- Definición de Escenarios.
5	- Lideres del Proyecto. - Evaluador. -Equipo de Evaluación.	-Presentar los Escenarios Identificados.	- Presentación de los Escenarios Identificados.
6	-Equipo de Evaluación.	-Documentar los Escenarios Identificados. -Describir Detallada de los Escenarios. - Definir el Grado de Prioridad de los Escenarios. - Definir los Puntos Críticos de la Arquitectura. - Definir los Posibles Cambios en la Arquitectura.	-Documentación de los Escenarios Identificados. - Descripción Detallada de los Escenarios. -Definición del Grado de Prioridad de los Escenarios. - Definición de los Puntos Críticos de la Arquitectura. -Definición de los Posibles Cambios en la Arquitectura.
7	- Evaluador. -Equipo de Evaluación.	- Chequear de Puntos Críticos. -Chequear de Posibles Cambios. -Actualizar la Definición de Evaluación.	- Chequeo de Puntos Críticos. - Chequeo de Posibles Cambios. - Actualización de la Definición de Evaluación.
8	- Evaluador. -Equipo de Evaluación.	- Presentar los Resultados de la Evaluación. - Realizar las Memorias del	- Presentación de los Resultados de la Evaluación. - Realización de las Memorias del

	- Lideres del Proyecto.	Proceso de Evaluación.	Proceso de Evaluación.

Tabla 6: Estrategia para Evaluar la AS

Resultado: Se realizaron 18 escenarios para el proceso de evaluación, a partir de estos se analizó hasta qué punto se cumplía con los atributos de calidad establecidos. Con esta evaluación se detectaron como puntos sensibles de la arquitectura el servidor de BD, servidor de medias, sistema de administración y el servidor *streaming*. También se tienen como puntos críticos de la arquitectura los procesos de Catalogación de Medias, Edición de Medias y Transcripción de Audio debido a que dependen del servidor de *Streaming* el cual se detecto como posible cambio por la necesidad de que este servidor cuando falla la conexión no comienza donde estaba realizando el *streaming*.

En los resultados los atributos de funcionalidad y eficiencia se comportaron por debajo de lo esperado porque si falla el servidor *Streaming* ya el sistema no sería completamente funcional y además tampoco sería eficiente, por otro lado la usabilidad, portabilidad, Mantenibilidad y reusabilidad por encima de lo esperado y la seguridad e integridad de la forma establecida.

Atributos de Calidad	Comportamiento		
	Alto	Medio	Bajo
Funcionalidad		X	
Integrabilidad	X		
Seguridad	X		
Eficiencia		X	
Usabilidad	X		
Portabilidad	X		
Mantenibilidad	X		
Reusabilidad	X		

Tabla 7: Resultados de la Evaluación

Con esta estrategia queda demostrado que los resultados obtenidos fueron favorables pero se pueden mejorar. Se recomienda realizar los cambios necesarios en el servidor de *Streaming* VLC (VideoLAN

Client) o analizar otro servidor de *streaming* que cumpla con las características necesarias para lograr una total funcionalidad del sistema.

Segunda Estrategia:

Para evaluar la calidad de la arquitectura se emplean los mismos atributos seleccionados y se establece una escala de valores de 1 a 10, la cual expresa de forma cuantitativa como se cumple cada uno de los atributos y se brinda una valoración final calculando el promedio de valores de cada uno de los parámetros de calidad.

- [1 a 4] mal cumplimiento
- [5 a 7] cumplimiento regular
- [8 a 10] buen cumplimiento

Funcionamiento:

La propuesta de arquitectura cumple con este atributo de calidad puesto que la misma centra su desarrollo en dos características fundamentales.

Se basa en un proceso que considera los casos de uso como un elemento de vital importancia. Esto da una medida de cómo se concibe un sistema que realmente responda a las necesidades del cliente, las cuales quedan expresadas en los casos de uso. Sigue en un proceso iterativo e incremental y esto muestra cómo la arquitectura propone un prototipo del sistema funcional escogiendo las funcionalidades básicas o casos de uso críticos y los prioriza en las primeras iteraciones dentro del ciclo de vida. A través de estas dos características queda demostrado que la arquitectura propuesta va dirigida a obtener un sistema funcional.

Se considera que tiene un valor de 10 dentro de la escala.

Integrabilidad:

La arquitectura cumple con este atributo debido a que sus componentes se van comunicar a través de la base de datos. La relación entre un componente y otro está dada por la BD, la cual sirve de puente entre los componentes.

Se considera que tiene un valor de 9 dentro de la escala.

Seguridad:

La arquitectura cumple con este atributo debido a que se emplea el uso del Spring con el Acegi para resistir a intentos de usos no autorizados, negar servicios a quien no posea los permisos necesarios y seguir sirviendo a usuarios legítimos. Además se implementa una funcionalidad en la capa de Seguridad con el objetivo de evitar inyección sql en la BD y se crean las variables de sesión para guardar información de cada usuario.

Se considera que tiene un valor de 9 dentro de la escala.

Eficiencia:

La arquitectura considera la posibilidad de que la aplicación garantiza una conexión segura y eficiente con la base de datos, que tendrá almacenada toda la información del sistema, mejorando esto la gestión de la información y los datos. Esta conexión segura y eficiente con la BD posibilita que los tiempos de respuestas de cualquier proceso sean factibles.

Se considera que tiene un valor de 9 dentro de la escala.

Usabilidad:

La arquitectura cumple con este atributo porque las interfaces con las que interactúa el usuario son sencillas, pero para un mejor manejo del sistema se necesita personas que tengan conocimientos básicos de informática en aplicaciones de escritorio y en el trabajo con medias.

Se considera que tiene un valor de 6 dentro de la escala.

Portabilidad:

La propuesta de arquitectura cumple con este atributo porque el sistema está analizado para desarrollarse con herramientas libres, lo que le proporciona la característica de poder ejecutarse en diferentes plataformas.

Se le considera un valor de 8 dentro de la escala.

Mantenibilidad:

La arquitectura considera la Mantenibilidad. La misma se evidencia dentro de la arquitectura en capas propuesta, las cuales ofrecen interfaces de comunicación que agrupan funcionalidades bien definidas, esto trae consigo que modificar una capa no tenga consecuencias sobre el funcionamiento de otra.

Se le considera un valor de 8 dentro de la escala.

Reusabilidad:

La propuesta de arquitectura cumple con este atributo pues el sistema está desarrollado por cuatro módulos de manera independiente, los cuales se comunican a través de la BD, estos módulos pueden ser utilizados por otros proyectos con características similares en futuras aplicaciones.
Se le considera un valor de 7 dentro de la escala.

Análisis de los Resultados:

Después del análisis del valor que alcanza cada uno de los atributos dentro de la escala propuesta se obtuvo el promedio de la calidad de la arquitectura dando como resultado 8.25 por lo que se considera que cumple con los parámetros de calidad analizados.

4.5 Valoración de Propuestas Arquitectónicas Similares

Esta valoración consiste en analizar propuestas arquitectónicas en sistemas de similar propósitos, a partir de consultar la bibliografía especializada.

Estructure

Como primera propuesta se tiene Estructure que es una plataforma avanzada sobre la que se construyen y configuran sistemas informáticos de gestión y explotación de contenidos audiovisuales, integrando todas las funcionalidades y tareas necesarias de forma flexible y modular, permitiendo una correcta y ágil organización, explotación y flujo del trabajo.

Características Técnicas de Estructure

Los sistemas Estructure se componen de diversas tecnologías tanto de software como de hardware, superpuestas en capas de funcionamiento e interrelacionadas y comunicadas entre sí. Las tecnologías incorporadas están probadas y testeadas por el equipo de desarrollo, y en pleno funcionamiento en los sistemas instalados hasta la fecha. No obstante, debido a la constante evolución de la tecnología, sus mejoras, versiones y actualizaciones se encuentran en la vanguardia de las nuevas tecnologías digitales, así como a la gran diversidad de capas y elementos que componen los sistemas (software, hardware,

redes, almacenamientos, dispositivos) resulta inviable la inclusión y descripción de todas las tecnologías incorporadas. A continuación se muestran algunas características de la plataforma.

- Soporte de múltiples formatos de entrada y salida: H264, MXF, DV, DVCPRO, MPEG2, MJPEG1, MPEG-4, WMW, FLV, 3G, 3G2.
- Captura simultánea en alta y baja calidad.
- Entradas y salidas en SDI, SDTI y QSDI (opcionales) conforme a estándares SMPTE 259M y 305M.
- Audio Digital Estéreo 48 KHz y analógico.
- Componentes analógicos, Y/C (S-Video) y compuesto.
- Soporte IEEE 1394.
- Arquitectura escalable para múltiples canales en captura y en salida.
- Acceso simultáneo a toda la información del sistema: desde múltiples estaciones de trabajo.
- Configuraciones hasta 150 puestos aseguradas.
- Configuración de permisos por módulos y niveles de acceso; usuarios y grupos de usuarios.
- Uso de redes Ethernet, Gigabit y FC.
- Metadata Controller (MDC) independiente o redundante.
- Sincronización automática de todo el sistema de trabajo.
- Guardado automático del trabajo en módulos críticos.
- Almacenamiento seguro mediante sistemas RAID 0, 1, 3, 5
- Detección automática de cambios de plano en las estaciones de captura.
- Bases de datos seguras SQL Server, MySQL u Oracle.
- Programación automática de salvas.
- Plataformas estándares y líderes: Windows 2000, XP y 2003.
- Combinación de redes hasta alta velocidad mediante FC.
- Lector y Generador de referencia de tiempos incorporado.
- Codificación a *Windows Media File* (WMF).
- Asociación de etiquetas XML.
- Administración de usuarios, grupos y funciones de cada módulo.
- Componentes hardware compatibles certificados.

- Estándares de tecnología abierta.
- Soporte de protocolos VDCP Louth.
- Conexión con robóticas para almacenamiento.
- Conexión con sistemas de gestión de radio.
- Conexión con sistema de noticias EDF. (36)

A partir de estas características se pueden realizar algunas valoraciones entre la arquitectura de Estructure y la que se propone para el sistema de captura y catalogación de medias. Estructure presenta una arquitectura ya probada y empleada en varios sistemas relacionados con el tema de las medias que utilizan herramientas propietarias. Estas dos arquitecturas tienen en común que se basen fundamentalmente en el patrón en capas lo que facilita que en caso de realizar alguna modificación, sólo se tuvieran que realizar las correcciones necesarias en el nivel requerido sin tener que revisar código de otros niveles. También emplean el uso de acceso a la aplicación de manera similar, con permisos definidos para cada usuario. Además Estructure utiliza programación automática de salvas mientras que en este sistema se propone Bacula para salvas automáticas.

Estructure emplea bases de datos seguras como SQL Server, MySQL y Oracle, el sistema propuesto emplea PostgreSQL que en análisis realizados se ha demostrado que cuanto mayor es el volumen de información más robusto resulta. Otra aspecto es que Estructure es factible para Windows (2000, 2003, XP) no así para el sistema propuesto, es decir es multiplataforma. Estructure presenta un buen funcionamiento pero además es muy escalable y esto provoca en varias ocasiones que se descuide la seguridad, y es debido a que en sistema donde ocurren muchos cambios la seguridad es difícil de mantener, no ocurre así en el sistema de captura y catalogación de medias que luego de la funcionalidad lo que más importa es la seguridad.

Videoma

Otro sistema importante en la gestión de medias es Videoma, que es un producto de la empresa española ISID la cual es líder en el desarrollo de soluciones software de gestión multimedia.

Videoma es un gestor de contenido completo, robusto, multi-formato y modular. Está especialmente diseñado para dar solución a un gran número de aplicaciones relacionadas con el mundo del vídeo, audio e imagen.

Principales características de Videoma:

- Plataforma con arquitectura modular para la captura de vídeo analógico o digital, en tiempo real o diferido. Permite el análisis, codificación, clasificación y publicación de un archivo digital.
- Cataloga fácilmente el material audiovisual y lo vincula con otros activos digitales como imágenes, documentos, de forma que su recuperación y localización permite la reutilización de dichos materiales.
- Vincula las versiones digitales de estos contenidos de forma que se puedan obtener, reproducir y reutilizar directamente. Su arquitectura Internet/Intranet permite una integración sencilla en redes corporativas que soporten TCP/IP.
- Realiza no sólo la captura de la información, sino también la segmentación automática con inserción de metadata asociado al material a digitalizar, facilitando la necesaria documentación de grandes archivos de información.
- Funcionamiento estable y continuo 24x7.
- Flexible e integrable con otros sistemas utilizados en el entorno de trabajo. (37)

Arquitectura de Videoma

La arquitectura del sistema varía en función de las fuentes de entrada que se tengan, de la cantidad de almacenamiento que se quieran mantener en línea, de los formatos a los que se hayan digitalizado los contenidos y de la cantidad de información que se vayan a mover a través de la red local.
Videoma utiliza aplicaciones estándar para la programación de sus módulos, PHP y C++. Como base de datos soporta Oracle 9i y MySQL y como Servidor Web IIS y Apache. El resto de necesidades para poner en marcha el sistema pueden ser aquellas disponibles en el mercado, como diversos sistemas de

almacenamiento, diversos servidores de datos, diversos codificadores de vídeo, etc. A continuación se muestra la representación del sistema Videoma.

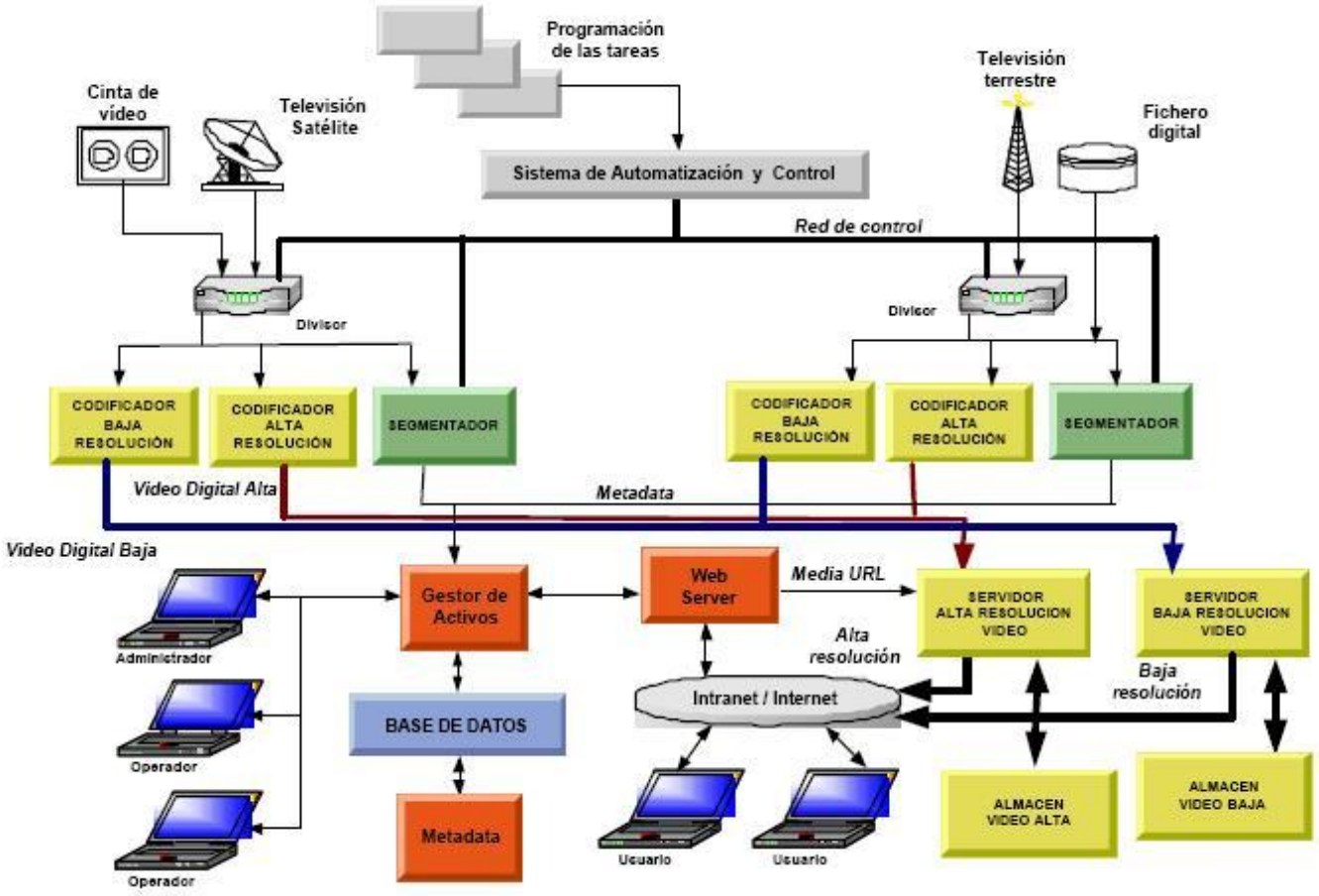

Figura 34: Arquitectura del sistema Videoma (37)

Aspectos comunes y diferentes entre Videoma y CCM

Videoma es una aplicación Web desarrollada con los lenguajes PHP y C++ mientras CCM es una aplicación de escritorio con Java y C++. Videoma utiliza como gestores de base de datos Oracle o MySQL, no así el sistema propuesto que utiliza PostgreSQL, ambos sistemas realizan replica en cada una de las PCs para trabajar con o sin conexión a la BD. En cuanto a la administración, los sistemas emplean la misma forma de acceso a la aplicación es decir mediante usuario y contraseña y con sus

respectivos permisos para interactuar con el sistema. Ambos sistemas presentan sistema de salva automático de los datos. Los dos sistemas consideran la seguridad como un aspecto importante pero en el caso de Videoma se considera muy escalable y este aspecto posibilita que el sistema sea poco seguro por la cantidad de modificaciones. En la selección de herramientas existen algunas diferencias, casi todas las empleadas por Videoma son para dar soporte en Windows no así en el sistema CCM que esta selección se realizó para dar soporte en cualquier plataforma.

4.5 Conclusiones del Capítulo

En este capítulo se realiza la evaluación de la arquitectura del sistema Captura y Catalogación de Medias, en el que se presentaron dos estrategias de pruebas, y una de ella basada en la metodología propuesta a seguir por la UCI para este tipo de evaluación donde los resultados obtenidos fueron aceptables. Además se realizó una valoración con dos sistemas de similar arquitectura y propósito donde los resultados obtenidos en el sistema propuesto son aceptables

Conclusiones Generales

En el presente trabajo se analizaron los principales aspectos para el desarrollo de la arquitectura. La selección del estilo arquitectónico, las herramientas de modelado, las herramientas de desarrollo y los requisitos no funcionales se realizó a partir de las características o funcionalidades que debía brindar el Sistema de Captura y Catalogación de Medias.

La selección del estilo arquitectónico en Capas permitió que el sistema se estructurara de forma tal que permitiera la descomposición de la complejidad estructural y que el mantenimiento del sistema no sea complejo.

A través de la descripción de la arquitectura mediante las vistas definidas por RUP garantizaron que se tuviera una visión de todos los modelos del sistema y de los elementos arquitectónicamente significativos durante el Proceso de Desarrollo.

Con el desarrollo del trabajo se generaron los artefactos necesarios que permiten la implementación del sistema a desarrollar.

Recomendaciones

- El refinamiento constante de la arquitectura propuesta, durante el ciclo de desarrollo.
- Se recomienda eliminar el uso de herramientas específicas de una plataforma para hacer el sistema más portable hacia otras plataformas de desarrollo.
- Se recomienda el uso de Bacula solo para los ficheros de programación
- Realizar algunos cambios específicos en el hardware que permitan que el sistema sea multiplataforma.

Glosario de Términos

IEEE: Institute of Electrical and Electronics Engineers (Instituto de Ingenieros Eléctricos y Electrónicos).

RUP: Metodología de desarrollo de software basada en UML. Organiza el desarrollo de software en 4 fases.

XP: Es una metodología del software que persigue simplificar los procesos de desarrollo del software. Fue diseñada para ser usado con equipos de desarrollo pequeños que necesiten desarrollos ágiles y requerimientos cambiantes.

Herramientas CASE (Computer Aided Software Engineering): Se incluyen una serie de herramientas, lenguajes y técnicas de programación que permiten la generación de aplicaciones de manera semiautomática.

Capas: Es uno de los patrones de diseño más utilizado para cualquier tipo aplicaciones, en este, básicamente se divide los elementos de diseño en paquetes. Se describen las diversas capas del sistema y su contenido, detallan los límites entre las capas y las reglas preestablecidas para cada una.

Framework: Marco de trabajo, solución reutilizable y extensible.

Triggers (disparador): Se defina así a una subrutina que es ejecutada de manera automática cuando se produce algún tipo de transacción (inserción, borrado o actualización) en la tabla de una base de datos.

Multiusuario: Es un tipo de configuración que permite soportar a varios usuarios o puestos de trabajo al mismo tiempo.

DAO: Siglas en inglés de Data Access Object (objeto de acceso a datos). Se trata de un objeto que realiza una labor de interfaz entre la aplicación y los datos de la misma.

Referencias Bibliográficas

1. ***Departamento de Ciencias de la Computación y Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pensilvania****. Sistemas de gran escala requieren de más alto nivel abstracciones. s.l. : ACM New York, NY, EE.UU, 1989. Vol. 14. 0163-5948.*

2. ***Bass, Len, Clement, Paul y Kazman, Rick****. Software Architecture in Practice . s.l. : Addison-Wesley Professional; 2 edition , 2003. 0321154959 .*

3. ***Hilliard, Rich****. IEEE-Std-1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems. [Pdf publicado en http://www.enterprisearchitecture.info/Images/Documents/IEEE%201471-2000.pdf] 2000.*

4. ***Alexander, Christopher****. The Timeless Way of Building. s.l. : Oxford University Press . 0195024028.*

5. ***Buschmann, Frank****. Pattern Oriented Software Architecture . s.l. : Wiley; 1 edition (August 8, 1996), 1996. 0471958697.*

6. ***Kiccillof, Carlos Reynoso – Nicolás****. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. BUENOS AIRES : s.n., 2004.*

7. ***Shaw, David Garlan y Mary****. An Introduction to Software Architecture. New Jersey : V.Ambriola and G.Tortora, World Scientific, 1994.*

8. ***Microsoft Corporation****. Integration Patterns (Patterns & Practices). s.l. : Microsoft Press, 2004. 073561850X.*

9. ***Rey, Fernando****. MEDIABOX. Madrid : XTREAM.*

10. ***Pozuelo., Aurelio Hernández y Amado José****. Soluciones Videoma. Madrid : ISID.*

11. ***XTAMP, Sistema de grabacion de Emision****. Madrid : XTREAM.*

12. ***Domains of Concern in Software Architectures and Architecture Description Languages****. Rosenblum, Nenad Medvidovic y David S. California : s.n., 1997.*

13. ***Gómez, Ruth Priscila Landeros****. Herramientas Case. s.l. : Universidad Veracruzana, 2007.*

14. ***Company Headquarters****. Visual Paradigm. [En línea] 2004. [Citado el: 20 de 11 de 08.] http://www.visual-paradigm.com.*

15. ***S.A, Grupo Soluciones Innova****. GSI. [En línea] 2007. [Citado el: 10 de diciembre de 2008.] http://www.rational.com.ar/apertura/acerca.html.*

16. ***Allison, Chuck****. Fundamentos para Java y C++. s.l. : MindView, Inc, 2000.*

17. ***Kopylenko, Johnson/ Hoeller/ Arendsen/ Rsiberg/.*** *Professional Java Development With The Spring Framework. Texas : John Wiley & Sons, 2004. 764574833.*

18. ***SEMATECH, Austin****. Computer Integred Manufacturing(CIM) Framework Specification- Version 2.0. 1998 : s.n. 93061697j.*

19. ***Ribelles, José Luis Gómez. Introducción al Swing****. España : Universidad Politécnica de Madrid, 2006.*

20. ***WALLS, CRAIG y BREIDENBACH, RYAN****. Spring in Action. United States of America : Manning Publications Co., 2005. 1-932394-35-4.*

21. ***Bauer,Christian y King Gavin****. Hibernate in Action. United States of America : Manning Publications Co, 2005. 1932394-15-X.*

22. ***Eclipse Foundation, Inc****. Eclipse. [En línea] Enero de 2004. [Citado el: 11 de diciembre de 2008.] http://www.eclipse.org/.*

23. ***Sun Microsystems*** *. Sun. [En línea] 2000. [Citado el: 20 de noviembre de 2008.] http://www.sun.com/emrkt/innercircle/newsletter/latam/0207latam_feature.html.*

24. ***VivvaLinux****. [En línea] vivab0rg, 19 de diciembre de 2008. [Citado el: 22 de diciembre de 2008.] http://www.vivalinux.com.ar/soft/qt-creator-0.9.1-beta.html.*

25. ***CAVSI. CAVSI****. [En línea] [Citado el: 22 de noviembre de 2008.] ¿ Qué es un Sistema Gestor de Bases de Datos o SGBD ?.*

26. ***Guiarte Multimedia S.L****. desarrolloweb.com. [En línea] 19 de 7 de 2002. [Citado el: 20 de 11 de 2008.] http://www.desarrolloweb.com/articulos/840.php.*

27. ***Lockhart, Thomas****. The PostgreSQL Development Team. s.l. : Thomas Lockhart, 1996.*

28. ***Serradilla, Juan Luis****. Control de Versiones con Subversion y TortoiseSVN. s.l. : ATICA, 2007.*

29. ***Rusio, Rafael****. Comunidad Debian Mexico. [En línea] 2008. [Citado el: 15 de 01 de 09.] http://www.debian-mx.com/2008/06/05/how-to-bacula-cliente-servidor/.*

30. ***Dominguez, Disnier Alberto Camejo****. PROPUESTA DE LA ARQUITECTURA PARA EL SISTEMA INTEGRAL DEL CALCULO DE INDICES AL CONSUMIDOR. Ciudad de la Habana : s.n., 2007.*

31. ***Booch, Grady, Jacobson, Ivar y Rumbaugh, James.*** *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2000. 8478290362.

32. ***Bass, Len, Clements, Paul, Kazman, Rick y otros****. Software Engineering Institute. [En línea] Carnegie Mellon. [Citado el: 05 de 03 de 2009.] http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.025.html.*

33. ***Regalado, Yamila Vigil.*** *Metodología de evaluación de arquitecturas de software. [Tesis de Maestria] Cuidad de la Habana, Habana : s.n., 2008.*

*34.****Ionita, Mugurel T., Hammer, Dieter K. y Obbink, Henk. 2006****. Scenario-Based Software Architecture Evaluation Methods: An Overview. Eindhoven, The Netherlands, : SARA, 2006.*

35. ***Bosch, Jan****. Design and Use of Software Architectures. s.l. : Addison-Wesley Professional , 2000. 0201674947.*

36. **Estructure Media Systems**. *Estructure. [En línea] 2001. [Citado el: 25 de marzo de 2009.]* http://www.estructuretv.com/.

37. **García, Antonio Albacete**. *GESTIÓN COMPLETA DE VÍDEO. [PDF publicado en http://www.unideco.net/docs/VIDEOMA.pdf] Madrid : s.n., 2002.*

Bibliografía

1. *Foundations for the Study of Software Architecture.* **Dewayne E. Perry, Alexander L. Wolf.** 4, New York : ACM SIGSOFT , 1992, Vol. 17. 0163-5948.

2. *An Introduction to Software Architecture.* **David Garlan, Mary Shaw.** New Jersey : V.Ambriola and G.Tortora, 1994, Vol. 1. 15213-3890.

3. **Buschmann, Frank, Meunier, Regine, Rohnert , Hans, Sommerlad , Peter and Stal, Michael.** *Pattern-Oriented Software Architecture .* England : s.n., 1996. 0 417 95869 7.

4. **Larman, Craig.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* s.l. : Prentice Hall. 84-205-3438-2.

5. **Krafzig, Dirk, Banke, Karl and Slama, Dirk.** *Enterprise Soa.* s.l. : Prentice Hall, 2005. 0131465759.

6. **Evans, Eric.** *Domain-Driven Design: Tackling Complexity in the Heart of Software.* s.l. : Addison-Wesley. 0-321-12521-5.

7. **Rey ,Fernando.** *XTREAM Sistemas de Informacion Global. [Online] Madrid. [Cited: diciembre 10, 2008.] http://www.xtreamsig.com.*

8. *ISID Gestión Inteligente de Activos Multimedia. [Online] ISID, 1996. [Cited: diciembre 10, 2008.] http://www.isid.es/.*

9. **Montalvo, David Peg.** Seguridad en la transmisión de Datos. [Documento] Santiago de Compostela : s.n., 2005.

10. **Medvidovic ,Nenad and Rosenblum ,David S..** *Domains of Concern in Software Architectures and Architecture Description Languages.* [Documento] California : s.n., 1997. 92697-3425.

11. **Mendoza ,María A. Sanchez.** *informatizate. [Online] junio 7, 2004. [Cited: enero 20, 2009.] http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html.*

12. **Zarzuela, Jorge Ferrer.** Metodologías Ágiles. [Documento publicado en http://libresoft.dat.escet.urjc.es/html/downloads/ferrer-20030312.pdf] Madrid : s.n.

13. **Company Headquarters.** *Visual Paradigm. [Online] 2002. [Cited: enero 20, 2008.] http://www.visual-paradigm.com/.*

14. **Grupo Soluciones Innova S.A.** *G.S.I Grupo Soluciones Innova . [Online] 2007. [Cited: enero 21, 2009.] http://www.rational.com.ar/herramientas/roseenterprise.html.*

15. **Sierra, Javier Ceballos.** Enciclopedia Del Lenguaje C++. s.l. : Ra-ma, 2003. 8478975845..

16. **Álvarez ,Gonzalo Marañón.** *Criptonomicón. [Online] 1997. http://www.iec.csic.es/CRIPTONOMICON/java/quesjava.html.*

17. **Daum, Berthold.** Eclipse 3 Para Desarrolladores Java. s.l. : Anaya Multimedia, 2005. 8441518815.

18. **Cerda, Felipe.** *NetBeans 6.5 El único IDE que necesitas.* [documento publicado en http://www.bloog.cl/talks/netbeans65es_cl.pdf]

19. **Microsystems, Sun.** *MySQL. [Online] Sun Microsystems, Inc. [Cited: enero 30, 2009.] http://www.mysql.com/.*

20. **PostgreSQL Global Development Group .** *PostgreSQL. [Online] tinysofa, 1996. http://www.postgresql.org/.*

21. **ORACLE CORPORATION.** *Oracle. [Online] [Cited: enero 25, 2009.] http://www.oracle.com/global/es/index.html.*

22. **Walls, Craig and Breidenbach, Ryan** *SPRING IN ACTION.* United States of America : Manning Publications Co., 2005. 1-932394-35-4.

23. **CRAIG WALLS, Ryan Breidenbach.** *Spring in Action.* Second Edition. United States of America : Manning Publications Co., 2008. 1-933988-13-4.

24. **Gallardo,David, Burnette, Ed and McGovern , Robert.** *Eclipse in Action: A Guide for Web Developers.* s.l. : Manning Publications, 2003.

25. **BAUER ,CHRISTIAN and KING ,GAVIN.** *Hibernate in Action.* United States of America : Manning Publications Co., 2005. 1932394-15-X.

26. **González, Carlos Sánchez.** *Seguridad no intrusiva con Acegi Security System for Spring.* España : s.n.

27. **Baptiste, Juan Luis.** *Introducción a QtJambi.* Colombia : Campus Party Colombia, 2008.

28. **Murphy, David J.** *Managing Software Development with Trac and Subversion .* s.l. : Packt Publishing Ltd, 2007. 978-1-847191-66-3.

29. **Berlin ,Daniel and Rooney ,Garrett.** *Practical Subversion.* United States of America : s.n., 2006. 1-59059-753-2.

30. **JENS-CHRISTOPH BRENDEL, MARC SCHÖCHLIN.** *Mejores Copias de Seguridad con el Sistema de Backup Bacula. [documento publicado en http://www.linux-magazine.es/issue/10/Bacula.pdf] 2005.*

31. **Escobar, Miguel A. Paco.** *La calidad de software y su importancia en el mercado .* [PDF] s.l. : Kernel Microsystems, 2003.

32. **Rosa Virginia Icedo Ojeda, Jorge Moisés Trejo Vargas.** *Software Architecture Assesment.* [PDF] s.l. : CIMAT, 2003.

33. **Estructure Media Systems**. *Estructure. [En línea] 2001. [Citado el: 25 de marzo de 2009.]* http://www.estructuretv.com/.

34. **García, Antonio Albacete**. *GESTIÓN COMPLETA DE VÍDEO. [PDF publicado en http://www.unideco.net/docs/VIDEOMA.pdf] Madrid : s.n., 2002.*

Anexos

Anexo 1:

En la imagen se recrean los flujos de trabajo de la metodología RUP así como el esfuerzo realizado encada flujo en cada iteración. (30)

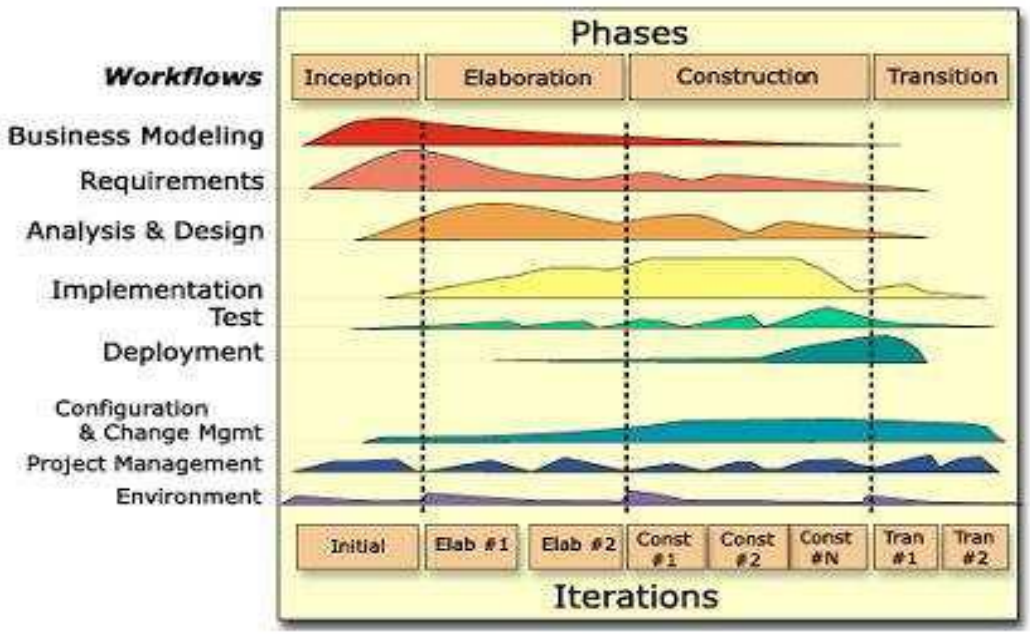

Figura 35: Flujos de Trabajo, Fases y esfuerzo por Iteración

Anexo 2:

Fase	Objetivos	Puntos de Control
Inicio	• Definir el alcance del proyecto • Entender que se va a construir	Objetivo del proyecto
Elaboración	• Construir una versión ejecutable de la arquitectura de la aplicación • Entender cómo se va a construir	Arquitectura de la Aplicación
Construcción	• Completar el esqueleto de la aplicación con la funcionalidad	Versión operativa inicial de la aplicación

	• Construir una versión beta	
Transición	• Disponibilizar la aplicación para los usuarios finales • Construir la versión final	Liberación de la versión de la aplicación

Tabla 8: Fases de la Metodología RUP

Anexo 3:

La figura muestra cada una de las fases del desarrollo siguiendo esta metodología, Planificación, Diseño, Codificación y Pruebas. (27)

Figura 36: Flujos de Trabajo de la Metodología XP

Anexo 4:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean  id="programaGrabadoDaoBean" class="vprogramas.ProgramaGrabadoDaoImpl"/>
<bean id="programaGrabadoServiceBean" class="vprogramas.ProgramaGrabadoServiceImpl">
<property name="programaGrabadoDao" ref="programaGrabadoDaoBean"/>
</bean>
<bean id="programaGrabadoFacadeBean" class="vprogramas.ProgramaGrabadoFacadeImpl">
<property name="programaGrabadoService" ref="programaGrabadoServiceBean"></property>
</bean>
</beans>
```

Figura 37: Representación del framework Spring

Anexo 5:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
                            "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">

<hibernate-mapping>
    <class name="net.sf.hibernate.examples.quickstart.Cat" table="CAT">

        <!-- A 32 hex character is our surrogate key. It's automatically
            generated by Hibernate with the UUID pattern. -->
        <id name="id" type="string" unsaved-value="null" >
            <column name="CAT_ID" sql-type="char(32)" not-null="true"/>
            <generator class="uuid.hex"/>
        </id>
        <!-- A cat has to have a name, but it shouldn' be too long. -->
        <property name="name">
            <column name="NAME" sql-type="varchar(16)" not-null="true"/>
        </property>
        <property name="sex"/>
        <property name="weight"/>

    </class>

</hibernate-mapping>
```

Figura 38: Representación del framework Hibernate