UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 8

Análisis, Diseño e Implementación del Sistema Estudio de Contrarios para el Voleibol.

Trabajo de Diploma para optar por el título de Ingeniero Informático.

Autores: Marcia Leidis Rodríguez Guerra.
Alcides Aguilera Acosta.

Tutor: Ing. Aliuska Sánchez Ibarria.
Cotutor: Lic. Jeanne Cadet Ochoa.

Ciudad de La Habana, Julio del 2009

¨Año 50 del Triunfo de la Revolución ¨

DECLARACIÓN DE AUTORÍA.

Declaramos ser los autores de la presente tesis y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio.

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

_____	_____
Firma del autor.	Firma del autor.
Marcia Leidis Rodríguez Guerra.	Alcides Aguilera Acosta.
_____	_____
Firma del tutor.	Firma del cotutor.
Ing. Aliuska Sánchez Ibarria.	Lic. Jeanne Cadet Ochoa.

"Las ciencias tienen las raíces amargas, pero muy dulces los frutos."

Aristóteles.

AGRADECIMIENTOS.

A mi familia por toda su confianza.

A mi tutora y cotutor porque se que hicieron su mejor esfuerzo para mostrarnos el camino.

A todos los profesores que de una forma u otra contribuyeron a mi educación e instrucción.

A la Revolución por todas las oportunidades.

Marcia Leidis Rodríguez Guerra.

Por ser quien soy y por hacer lo que he hecho quiero agradecer:

A mis padres, por todo su amor, cariño y comprensión, porque nunca me han faltado y se que nunca lo harán, gracias mami, gracias papi.

A todos mis amigos por el apoyo y los consejos que siempre estuvieron cuando los necesité, a mi hermano Yoisell, a mi hermano Kiki, a mi primo Migue y a mi compañera de tesis por tanta paciencia, gracias a todos.

A esta revolución que ha hecho realidad el sueño de muchos, al Comandante Fidel que ha sido el guía de esta obra que es la Universidad de las Ciencias Informáticas.

A todas mis tías, a todos mis tíos, a mi abuelo Acosta y a toda mi familia por todo su apoyo y cariño.

Quiero agradecer a todos, a los que estuvieron y ya no están, a quien fue y ya no es, gracias por todo.

Alcides Aguilera Acosta

DEDICATORIA.

A todos mis compañeros de grupo porque cada uno de ellos fue un ingrediente imprescindible de esta mezcla que yo considero perfecta.

A mis hermanos del alma por toda su ayuda y su cariño y por ser los mejores hermanos del mundo.

Especialmente a mis padres porque todo lo que soy se los debo a ellos, a mi mami por ser la impulsora y a papi por ser la inspiración.

Marcia Leidis Rodríguez Guerra.

Esta tesis es de ustedes, mis padres, se que este es su sueño también y hoy ustedes se gradúan conmigo.

Quiero dedicar este trabajo también a todos los profesores y maestros que he tenido estos casi 24 años y a todas las personas que he conocido a lo largo del camino y han tenido que ver de una forma u otra con mi formación.

Alcides Aguilera Acosta.

RESUMEN.

A partir de la solicitud realizada por la Federación Nacional de Voleibol se lleva a cabo un análisis a los procesos de gestión de la información de estudio de contrarios en el voleibol y se determina que la elaboración de una aplicación que facilite dichos procesos agilizará la toma de decisiones estratégicas a la hora de enfrentarse a las diferentes selecciones por parte de los técnicos y entrenadores, mantendrá centralizada la información generada en los juegos, ayudará a la planificación de los entrenamientos de los equipos nacionales y permitirá además la evaluación de las acciones defensivas y ofensivas por parte de los entrenadores para valorar posteriormente el desempeño de los jugadores.

En el presente trabajo investigativo se lleva a cabo un análisis comparativo de las herramientas y tecnologías alternativas para el desarrollo de la aplicación planteada y se seleccionan las más apropiadas con el objetivo de implementar una solución óptima; se deja constancia de todo el proceso de desarrollo y de ingeniería de software asociado; se realiza un estudio para presentar de la forma más apropiada y eficaz las funcionalidades de dicha aplicación; y por último se realiza la implementación de la misma.

Palabras clave: Estudio de contrarios, voleibol, automatización, aplicación.

TABLA DE CONTENIDOS.

ÍNDICE DE TABLAS.

ÍNDICE DE FIGURAS.

INTRODUCCIÓN.

“Si conoces a los demás y te conoces a ti mismo, ni en cien batallas correrás peligro; si no conoces a los demás, pero te conoces a ti mismo, perderás una batalla y ganarás otra; si no conoces a los demás ni te conoces a ti mismo, correrás peligro en cada batalla.”(1). Esta frase fue pronunciada por un guerrero chino y data del año 500 a.C., como se puede apreciar desde tiempos tan remotos ya se reconocía la necesidad de conocer tanto a los propios como a los contrarios en un enfrentamiento.

En el deporte el estudio realizado para conocer el equipo contrario se conoce con el término de estudio de contrarios y consiste en aprender de un partido o situaciones de juego, de entender el juego y saber resolver los problemas que se presentan en un partido determinado, incluye el conocimiento anticipado logrado a través del procesamiento de las informaciones para contribuir a la toma de decisiones estratégicas y así poder alcanzar objetivos de seguridad, bienestar o predominio.

En los últimos tiempos el uso de las tecnologías en el deporte para realizar el estudio de contrarios ha devenido en un aumento del nivel competitivo en las diferentes disciplinas. Se ha desatado una revolución a nivel mundial en este campo, liderada por los países desarrollados cuyas selecciones y entrenadores cuentan ya con sofisticados sistemas para evaluar patrones de rendimiento y conducta y predecir el futuro comportamiento tanto de jugadores como de equipos.

Cada vez se hace más necesario el uso de la computación para el procesamiento de la cantidad de información que se genera en cada partido. Realizar este procesamiento de manera manual, significa una pérdida de tiempo y recursos así como mano de obra.

Una herramienta que permita a los entrenadores y técnicos hacer un estudio mediante videos tomados a los equipos propios y contrarios permitiría marcar tendencias y costumbres de los jugadores mientras se desarrolla la competencia y así proponer una mejor planificación del entrenamiento que abarcaría todos los componentes del mismo, desde el punto de vista técnico, táctico, físico, teórico y psicológico. Actualmente la selección que no cuente con estas facilidades está en marcada desventaja frente a su rival.

En el deporte cubano los entrenadores ven las claras ventajas que estos sistemas ofrecen y expresan la necesidad de integrarlos al sistema deportivo cubano: “Es la ciencia aplicada al deporte. Necesitábamos esa contribución, nosotros hacíamos un escauteo manual desde hace muchos años; aunque ya esta innovación da más resultado y ahorra tiempo y esfuerzo. En la medida en que sepamos utilizar las herramientas, se logrará un mejor provecho.”(2)

Los equipos de la selección nacional de voleibol masculino y femenino no cuentan en este momento con un sistema informático que les permita a los entrenadores y técnicos realizar un estudio tanto del equipo rival como del propio, además, el software que utilizan para el procesamiento de las estadísticas generadas en el partido es rentado y obsoleto, lo que significa una pérdida de tiempo y de dinero para el país y para nuestros técnicos.

Por lo que el **problema a resolver** es ¿cómo facilitar de forma segura, rápida y eficiente el proceso de gestión de información para el estudio de contrarios en voleibol?

Por tanto el **objeto de estudio** de este trabajo son los procesos de gestión de información de estudio de contrarios en voleibol.

De esto se deriva que el **campo de acción** son los sistemas de gestión de información para el estudio de contrarios en voleibol.

Planteando la **idea a defender** que si se desarrolla un sistema que facilite de forma rápida y eficiente el proceso de gestión de estudio de contrarios, permitirá optimizar el entrenamiento de los atletas y la creación de efectivas tácticas de juego.

El **objetivo general** de este trabajo será desarrollar una aplicación de escritorio que les permita a los entrenadores del equipo nacional de voleibol realizar de forma eficiente el proceso de gestión de información para el estudio de contrarios.

De esto se derivan los siguientes **objetivos específicos:**

- Realizar una caracterización de los lenguajes candidatos a ser utilizados en la implementación.
- Determinar las herramientas y metodologías óptimas para desarrollar una aplicación que satisfaga las necesidades del cliente.

- Elaborar y validar la propuesta de solución.

Para llevar a cabo estos objetivos se le debe dar cumplimiento a las siguientes **tareas**:

- Realizar un estudio de los lenguajes de programación a utilizar en la implementación de la aplicación propuesta.
- Caracterizar las diferentes metodologías y herramientas que existen para el desarrollo de aplicaciones de escritorio.
- Implementar todas las funcionalidades necesarias para lograr una aplicación de escritorio que de solución al problema planteado.

La solución propuesta es de gran importancia para los equipos de voleibol y para los técnicos de este deporte en el país, ya que no existe un sistema producido en Cuba de este tipo. Tecnológicamente automatiza el proceso de estudio de contrario mediante el procesamiento de imágenes de video tomadas en cada partido. Económicamente, la implementación de la solución propuesta evitaría al país pagar los altos costos de los sistemas de estudio de contrarios y procesamiento de estadísticas en el extranjero, pudiendo destinar este presupuesto a otros fines.

El presente trabajo de investigación está conformado por cuatro capítulos con los siguientes temas a tratar:

Capítulo1. Fundamentación teórica: Incluye un estudio y análisis de los lenguajes, metodologías, herramientas y tecnologías posibles a usar para dar una solución óptima a la problemática planteada, incluye además una descripción del estado del arte.

Capítulo 2. Características del sistema: Se hace una presentación de las características y funcionalidades con que el sistema debe contar, así como de los casos de uso que guiarán todo el desarrollo de este y una propuesta de sistema acorde con el estudio realizado.

Capítulo 3. Análisis y diseño del sistema: Contiene toda la documentación generada durante el diseño de la aplicación que engloba una descripción de forma general de las clases utilizadas en los diagramas de clases del diseño, los diagramas de secuencia por cada caso de uso a cumplir, así como el modelo de la base de datos.

Capítulo 4. Implementación y prueba: En este capítulo se describe la arquitectura empleada en el sistema, se define la organización de este en subsistemas, se presentan el modelo de implementación y el modelo de despliegue. Se incluyen además los casos de prueba necesarios para asegurar que el sistema cumpla con las expectativas del cliente.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

1.1 Introducción.

En este capítulo se ofrece una vista panorámica de los temas relacionados a los sistemas informáticos usados en el mundo y en Cuba en el voleibol para el estudio de contrarios. También se abordarán las diferentes teorías y fundamentos aplicados para el diseño del sistema, así como la descripción de los principales conceptos asociados al dominio del problema. En este capítulo se realiza un estudio y análisis de las tecnologías apropiadas para el desarrollo de aplicaciones *desktop* eficientes, escogiendo la mejor opción para la construcción del sistema propuesto.

1.2 Estado del arte.

A nivel mundial son muchas las aplicaciones que se utilizan en la actualidad para el estudio de contarios, a continuación se realiza una breve caracterización de las más usadas.

1.2.1 Utilius VS.

Es el perfecto sistema de edición de video para todas las aplicaciones en el deporte. Fue creado por la empresa *Campus-Computer-Center GmbH* de *Leipzig* y asistido por el Instituto de Formación Profesional de Ciencias Aplicadas (*IAT Leipzig*). El programa es usado en todo el mundo por centros de entrenamiento olímpico, selecciones nacionales, universidades y equipos de alto nivel para archivar y editar tomas de vídeo análogo o digital.

Se usa *utilius VS* en hockey, balonmano, voleibol, fútbol, en tenis y tenis de mesa, en gimnasia en aparatos y gimnasia rítmica, en golf, atletismo y equitación.

Utilius VS existe en alemán, inglés, español, japonés y en chino. Actualmente están en preparación las versiones en holandés, turco. (3)

1.2.2 StatTrak.

StatTrak de voleibol es un programa completo de estadísticas con muchas opciones, muy fácil de usar. Este software brinda y calcula más de 45 estadísticas sobre acciones en un juego, incluidos ataques y servicios.

Procesa informes estadísticos en cuestión de minutos. Después de haber añadido el nombre del equipo y la lista de jugadores, agrega un juego y los resultados de un jugador desde su libro de resultados. Las estadísticas se calculan automáticamente sobre la base de los resultados introducidos. Permite dar seguimiento a un equipo, varios equipos, o un conjunto de la liga. (4)

1.2.3 DataVolley.

DataVolley es un paquete de *software* estadístico proporcionado por *DataProject*. Este *software* se comercializa contando con el apoyo de un gran número de ligas de voleibol influyentes incluyendo las ligas italianas y americanas.

DataVolley sólo requiere que una persona tome las estadísticas para un juego, aunque recomiendan a una segunda persona durante el entrenamiento. El estadístico puede ser localizado en cualquier parte, y registrará sólo los ataques y otras acciones usando un conjunto de códigos compuestos. La decisión de sólo registrar los tiros de ataque se diferencia de *VIS* (otro *software* estadístico) que registra cada tiro. Los códigos compuestos son el modo de acortar la cantidad de información estadística que se genera en cada encuentro.

La versión comercial de *DataVolley* contiene posibilidades casi ilimitadas para almacenar cualquier aspecto de un equipo de voleibol. Las opciones que brinda este *software* son numerosas, pero se debe resaltar que el precio es elevado. (5)

1.2.4 VIS.

El Sistema de Información de Voleibol por sus siglas en inglés *VIS*, es el *software* estadístico producido por la Federación Internacional de Voleibol (FIVB). El *software* se utiliza para generar estadísticas en todos los eventos de la FIVB incluyendo campeonatos del mundo.

VIS ofrece la posibilidad de manipular estadísticas para todos los tiros en el juego y filas de cada disparo como:

- Excelente (Éxito).
- Normal.
- Error (Fallo).

Las estadísticas son tomadas por dos grupos de tres personas, ubicado a cada extremo de la cancha de voleibol. Una persona es responsable de la inserción de las estadísticas en un ordenador, otro anuncia los disparos y la última persona es responsable de la escritura de las estadísticas sobre el papel, en caso de fallo del *software*. Hay una persona que coordina las estadísticas de ambos extremos y se sienta detrás del anotador, en el centro de la corte. Esta persona es responsable de corregir los errores cometidos por ambos extremos y garantizar que coinciden con los puntos correctamente.

Las desventajas principales de *VIS* son, que se emplea mucha mano de obra, requiere 7 personas por partido y el *software* es bastante obsoleto. El sistema consiste en una ventana de MS-DOS por la que el usuario introduce los datos a través del teclado. Además, requiere de alguien que tenga un conocimiento a fondo para que solucione los problemas que a menudo ocurrirán al azar. (6)

Como se ha descrito anteriormente cada una de estas aplicaciones tiene sus potencialidades y debilidades, unas son destinadas al manejo de estadísticas y otras al de información visual referente a los eventos de la disciplina que se efectúan en todo el mundo. Dadas las prestaciones que brindan estos sistemas, su adquisición se dificulta para los países subdesarrollados debido a los altos precios de las licencias. Por este motivo la Federación Cubana de Voleibol en conjunto a la Universidad de las Ciencias Informáticas se dio a la tarea de crear un producto de factura nacional que facilitara a los técnicos y entrenadores de los equipos nacionales el proceso de estudio de contrarios, ahorrando el presupuesto que puede ser dedicado a otros fines para el desarrollo del país.

1.3 Metodologías de desarrollo.

Metodología: Conjunto de métodos empleados para el desarrollo de sistemas automatizados. Las metodologías proponen un proceso disciplinado sobre el desarrollo de *software* con el fin de hacerlo más predecible y eficiente. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar, inspirado por otras disciplinas de la ingeniería.

1.3.1 RUP.

RUP o Proceso Unificado de *Rational* por su traducción al español no es simplemente un proceso, sino un marco de trabajo extensible que puede ser adaptado a organizaciones o proyectos específicos.

Flujos de trabajo:

- Modelamiento del negocio.
- Requerimientos.
- Análisis y diseño.
- Implementación.
- Prueba (Testeo).
- Instalación.
- Administración del proyecto.
- Administración de configuración y cambios.
- Ambiente.

Fases:

- Conceptualización (Concepción o Inicio).
- Elaboración.
- Construcción.
- Transición.

Características:

- Metodología pesada.
- Dirigido por casos de uso.
- Centrado en la arquitectura.
- Iterativo e Incremental.
- Dividido en fases.
- Las fases se dividen en iteraciones.
- El discurrir del proyecto se define en flujos de trabajo.
- Los artefactos son el objetivo de cada actividad.

- Se basa en roles.
- Muy organizativo.
- Genera mucha documentación.
- Basado en *UML*.(7)

Ventajas:

- Evaluación en cada fase que permite cambios de objetivos.
- Funciona bien en proyectos de innovación.
- Sigue los pasos intuitivos necesarios a la hora de desarrollar el *software*.
- Seguimiento detallado en cada una de las fases.
- Mitigación temprana de posibles altos riesgos.
- Progreso visible en las primeras etapas.
- Temprana retroalimentación que se ajuste a las necesidades reales.
- Gestión de la complejidad.
- El conocimiento adquirido en una iteración puede aplicarse en las próximas.

Desventajas:

- La evaluación de riesgos es compleja.
- Excesiva flexibilidad para algunos proyectos
- El cliente queda en una situación que puede ser muy incómoda para él.
- El cliente deberá ser capaz de describir y entender a un gran nivel de detalle para poder acordar un alcance del proyecto con él.

1.3.2 XP.

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo del *software*, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. *XP* se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. *XP* se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. (8)

XP según Beck, su creador es: “Un proceso ligero, de bajo riesgo, flexible, predecible, científico y divertido de desarrollar *software*” (9)

Características:

- Metodología ligera.
- Cercano al desarrollo.
- Se basa en *UserStories*.
- Fuerte comunicación con el cliente.
- El código fuente pertenece a todos.
- Programación por parejas.
- *Tests* como base de la funcionalidad.
- Sólo el mínimo de organización.
- Pobre en cuanto a documentación.(7)

Prácticas básicas de la programación extrema:

- **Equipo completo:** Forman parte del equipo todas las personas que tienen algo que ver con el proyecto, incluido el cliente y el responsable del proyecto.
- **Planificación:** Se hacen las historias de usuario y se planifica en qué orden se van a hacer las mini-versiones. La planificación se revisa continuamente.
- ***Test* del cliente:** El cliente, con la ayuda de los desarrolladores, propone sus propias pruebas para validar las mini-versiones.
- **Versiones pequeñas:** Las mini-versiones deben ser lo suficientemente pequeñas como para poder hacer una cada pocas semanas. Deben ser versiones que ofrezcan algo útil al usuario final y no código que no pueda ver funcionando.
- **Diseño simple:** Hacer siempre lo mínimo imprescindible de la forma más sencilla posible. Mantener siempre sencillo el código.
- **Pareja de programadores:** Los programadores trabajan por parejas (dos delante del mismo ordenador) y se intercambian las parejas con frecuencia (un cambio diario).
- **Desarrollo guiado por las pruebas automáticas:** Se deben realizar programas de prueba automática y deben ejecutarse con mucha frecuencia. Cuantas más pruebas se hagan, mejor.

- **Mejora del diseño:** Mientras se codifica, debe mejorarse el código ya hecho que sea susceptible de ser mejorado. Extraer funcionalidades comunes, eliminar líneas de código innecesarias.
- **Integración continua:** Debe tenerse siempre un ejecutable del proyecto que funcione y en cuanto se tenga una nueva pequeña funcionalidad, debe recompilarse y probarse. Es un error mantener una versión congelada dos meses mientras se hacen mejoras y luego integrarlas todas de golpe.
- **El código es de todos:** Cualquiera puede y debe tocar y conocer cualquier parte del código. Para eso se hacen las pruebas automáticas.
- **Normas de codificación:** Debe haber un estilo común de codificación (no importa cuál), de forma que parezca que ha sido realizado por una única persona.
- **Metáforas:** Hay que buscar unas frases o nombres que definan cómo funcionan las distintas partes del programa, de forma que los nombres den una idea de qué es lo que hace cada parte del programa.
- **Ritmo sostenible**: Se debe trabajar a un ritmo que se pueda mantener indefinidamente. Esto quiere decir que no debe haber días muertos en que no se sabe qué hacer y que no se deben hacer un exceso de horas otros días. Al tener claro semana a semana lo que debe hacerse, hay que trabajar duro en ello para conseguir el objetivo cercano de terminar una historia de usuario o mini-versión. (10)

Ventajas:

- Trabajar de dos en dos es beneficioso cuando se emparientan programadores donde alguno no tiene experiencia o posee malas prácticas y el estilo de programación tiende a unificarse.
- Apropiado para entornos volátiles.
- Estar preparados para el cambio, significa reducir su coste.
- Planificación más transparente para nuestros clientes, conocen las fechas de entrega de funcionalidades vitales para su negocio.
- Permite definir en cada iteración cuáles son los objetivos de la siguiente.
- Permite tener realimentación de los usuarios muy útil.
- La presión está a lo largo de todo el proyecto y no en una entrega final.

Desventajas:

- Delimitar el alcance del proyecto con nuestro cliente: el alcance y la funcionalidad exacta del *software* nunca se define formal y contractualmente.
- Evita cualquier tipo de documentación fuera del código fuente: *UML* juega un papel casi nulo, por lo que no se puede representar todo lo que se debería como las dependencias entre componentes lo que dificulta utilizar la experiencia ganada en otros desarrollos ya que no se ha archivado o anotado nada.
- La implementación se desarrolla por parejas: supone un desperdicio de mano de obra ya que mientras uno programa el otro está sentado mirando.

1.3.3 ¿Por qué RUP?

Después de analizar *XP* en representación de las metodologías ligeras y a *RUP* en representación de las metodologías tradicionales o pesadas ya que estas son las más usadas actualmente a nivel mundial, se decidió utilizar en el proceso de desarrollo de software que se plantea a la metodología *RUP* por las razones siguientes: *XP* plantea en una de sus prácticas básicas que el cliente forma parte del equipo de desarrollo, aspecto imposible de cumplir en este caso particular debido a que el cliente no dispone del tiempo suficiente para estar inmerso en el desarrollo del sistema, sin embargo en *RUP* no es necesario que el cliente esté tan comprometido en el proceso de desarrollo pero esto no implica la enajenación ya que cualquier decisión importante que se tome se le comunicará mediante una entrevista; otra práctica básica propone que la programación se realice por parejas, esta práctica quizás en otras condiciones aporte muchas ventajas al desarrollo de un proyecto pero cuando se dispone de poco tiempo implica un atraso. Por otra parte en la Programación Extrema los requisitos no funcionales tienen un papel secundario y en la práctica estos son tan importantes como los funcionales, igualmente la gestión de configuración y cambios son aspectos que se manejan muy ligeramente. Analizando a *RUP* la principal ventaja que tiene es que se basa en las mejores prácticas que se han probado en el campo de desarrollo de *software*, en comparación con XP que se basa en prácticas inestables que aplicadas al unísono pueden dar un buen resultado; otro aspecto positivo del Proceso Unificado de *Rational* es que propone todos los artefactos que tienen utilidad para la ingeniería de *software*, sin embargo se puede utilizar de ellos sólo los que se necesitan para crear un proceso de desarrollo de software propio .

1.4 Lenguaje unificado de modelado (UML).

Desarrollado por Grady Booch, Jim Rumbaugh e Ivar Jacobson, sirve para especificar, visualizar y documentar esquemas de sistemas de *software* orientado a objetos. El *UML* está compuesto por diversos elementos gráficos que se combinan para conformar diagramas que representan alguna parte o punto de vista del sistema. Debido a que el *UML* es un lenguaje, cuenta con reglas para combinar tales elementos.

UML soporta los siguientes tipos de diagramas:

- Diagrama de casos de uso*:* que muestra a los actores, los casos de uso y sus relaciones.
- Diagrama de objetos: conjunto de objetos y sus relaciones.
- Diagrama de clases*:* que muestra las clases y la relaciones entre ellas.
- Diagrama de secuencia*:* muestra los objetos y sus múltiples relaciones entre ellos.
- Diagrama de colaboración*:* que muestra objetos y sus relaciones, destacando los objetos que participan en el intercambio de mensajes.
- Diagrama de estado: muestra estados, cambios de estado y eventos en un objeto o en parte del sistema.
- Diagrama de actividad*:* que muestra actividades, así como los cambios de una a otra actividad junto con los eventos que ocurren en ciertas partes del sistema.
- Diagrama de componentes*:* que muestra los componentes de mayor nivel de la programación.
- Diagrama de implementación*:* que muestra las instancias de los componentes y sus relaciones.

1.5 Herramientas CASE.

Herramientas *CASE* (*Computer Aided Software Engineering*): conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de *software* y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un *software*. Este puede ser generalmente aplicado a cualquier sistema o colección de herramientas que ayudan a automatizar el proceso de diseño y desarrollo de *software*.

1.5.1 Visual Paradigm.

Visual Paradigm es una de las herramientas *UML CASE* del mercado, considerada como muy completa y fácil de usar, con soporte multiplataforma y que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones. Fue creada para el ciclo vital completo del desarrollo de *software* que lo automatiza y acelera, permitiendo la captura de requisitos, análisis, diseño e implementación. Tiene la capacidad de crear el esquema de clases a partir de una base de datos y crear la definición de base de datos a partir del esquema de clases. Permite invertir código fuente de programas, archivos ejecutables y binarios en modelos *UML* al instante, creando de manera simple toda la documentación. Está diseñada para usuarios interesados en sistemas de *software* de gran escala con el uso del acercamiento orientado a objeto, además apoya los estándares más recientes de las notaciones de *Java* y de *UML*. Incorpora el soporte para trabajo en equipo, que permite que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios hechos por sus compañeros.

Características:

- Producto de calidad.
- Soporta aplicaciones *Web*.
- Varios idiomas.
- Generación de código para *Java* y exportación como *HTML*.
- Fácil de instalar y actualizar.
- Compatibilidad entre ediciones.(11)

Se integra con las siguientes herramientas *Java*:

- *Eclipse/IBM WebSphere*.
- *JBuilder.*
- *NetBeans IDE.*
- *Oracle JDeveloper.*
- *BEA Weblogic.*

Ventajas:

- Apoya todo lo básico en cuanto a artefactos generados en las etapas de definición de requerimientos y de especificación de componentes.
- Tiene apoyo adicional en cuanto a generación de artefactos automáticamente.
- Genera modelos *VP-UML* instantáneamente a partir de código binario *.NET*.
- Generación de documentación en formatos *HTML* y *PDF*.
- Disponibilidad en múltiples plataformas: *Microsoft Windows* (98, 2000, *XP*, o Vista), *Linux*, *Mac OS X*, *Solaris* o *Java.*
- Brinda la posibilidad de intercambiar información mediante la importación y exportación de ficheros con aplicaciones como por ejemplo *Visio* y *Rational Rose*.
- Generación de código e ingeniería inversa: brinda la posibilidad de generar código a partir de los diagramas, para plataformas como *.Net*, *Java* y *PHP*, así como obtener diagramas a partir del código.
- Generación de documentación: brinda la posibilidad de documentar todo el trabajo sin necesidad de utilizar herramientas externas.

Desventajas:

- Las imágenes y reportes generados, no son de muy buena calidad. (11)

1.5.2 POSEIDON para UML.

Es una herramienta para modelar cualquier clase de sistema que esté o no relacionado con programación. *Poseidon* para *UML* puede simplificar la compleja tarea de desarrollo de software ayudando a estructurar pensamientos, a clarificar la comunicación, y a encontrar la correcta abstracción. La incorrecta implantación de la herramienta *UML*, le sumergirá en detalles y diagramas llenos de funciones extrañas y excesivamente complicadas, lo que le evitará el ahorro de tiempo y esfuerzo.

La intuitiva interfaz hace de *Poseidon* una de las herramientas más rápidas de *UML* para dominar el análisis orientado a objetos, liberando al diseñador para centrarse solamente en su modelo.

Características:

- Soporta diagramas *UML*.

- Opciones avanzadas de impresión.
- Soporta gráficos en la mayoría de formatos.
- Varios idiomas.
- Generación de código para *Java* y exportación como *HTML*.
- Fácil de instalar y actualizar.
- Compatibilidad entre ediciones.
- Capacidades ampliables a través de *plug-ins*, es posible cargarlos en tiempo de ejecución.
- Generación de documentación en *HTML* y formato *Word 2003*.
- Soporta los formatos gráficos *gif, ps, eps, wmf, jpg y png.*

Ventajas:

- Herramienta hecha completamente en *Java*, por lo que es independiente de la plataforma.
- Interfaz de usuario muy bien diseñada, fácil de aprender a usar e intuitiva.

Desventajas:

- En la versión *Trial* la grabación de proyectos está limitada a ocho diagramas.

1.5.3 ArgoUML.

Es una aplicación de diagramado de *UML* escrita en *Java* y publicada bajo la Licencia *BSD* (*Berkeley Software Distribution*) *open source*. Dado que es una aplicación *Java*, está disponible en cualquier plataforma soportada por *Java*.

Sin embargo, desde la versión 0.20, *ArgoUML* está incompleto. No es conforme completamente a los estándares *UML* y carece de soporte completo para algunos tipos de diagramas incluyendo los diagramas de secuencia y los de colaboración.

Características:

Nuevas características en V0.20:

- Características de extensibilidad mejoradas de *UML* 1.4.

- Diagramas de secuencia.
- Compatibilidad *AndroMDA*.
- Cientos de *bugs* han sido arreglados.
- La mayoría de las funciones ahora soportan la selección múltiple de los elementos del modelo.
- Se puede arrastrar y soltar desde el árbol de exploración al diagrama y dentro del árbol de exploración.

Otras características:

- Construido en diseños críticos, suministra una revisión no obstructiva del diseño y sugerencias para mejoras.
- Interfaz de módulos extensible.
- Soporte de internacionalización para inglés, alemán, francés, español y ruso.
- Restricciones *OCL* para clases.
- Soporte para el lenguaje de generación de código: *Java, PHP, Python, C++ y Csharp*.
- Ingeniería inversa.
- Disposición (*layout*) automática del diagrama de clases.
- Generación de ficheros *png, gif, jpg, svg, eps* desde diagramas.
- Soporte para comentarios para múltiples elementos.
- Todos los diagramas 1.4 están soportados.

Desventajas:

- Instalación costosa.
- Poco amigable.
- Difícil de empezar.(12)
- No tiene botón "deshacer".
- Los modelos a veces no pueden ser re-abiertos.
- No hay llamadas reflexivas en los diagramas de secuencia.
- Se debe seleccionar una clase para crear un diagrama de secuencia.

Ventajas:

- Genera código automáticamente.
- Propone soluciones a algunos errores.
- Panel de propiedades y de tareas pendientes bastante útil.

1.5.4 ¿Por qué Visual Paradigm?

ArgoUML tiene muchos aspectos que mejorar, para comenzar soporta diagramas especificados en *UML* 1.4 quedándose un poco retrasado en este aspecto ya que *UML* está en la versión 2.1 soportada ampliamente por *Visual Paradigm,* requiere de mucha *RAM*, su usabilidad no es la mejor ya que no tiene íconos para reconocer los estereotipos y la interfaz no es muy agradable. *Visual Paradigm* por su parte soporta la generación de código e ingeniería inversa en *Java, C++, CORBA IDL, PHP, XML Schema, Ada y Python.* Además ayuda la generación de código en *C#, VB .NET, Object Definition Language (ODL), Flash ActionScript, Delphi, Perl, Objective-C, y Ruby,* soporta la ingeniería inversa en clases de *Java, dll.NET y exe, JDBC*, y el mapeo de archivos en *Hibernate*. Otro aspecto muy importante es la excelente documentación con que cuenta, en su sitio oficial se explican detalladamente todas las funcionalidades. Además la herramienta es colaborativa pues soporta múltiples usuarios trabajando sobre el mismo proyecto, permite el control de versiones y genera la documentación en varios formatos. En definitivas *Visual Paradigm* es una herramienta *CASE* muy robusta, destacando además su usablilidad y portabilidad.

1.6 Lenguajes de programación.

Un lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Es utilizado para controlar el comportamiento físico y lógico de una máquina.

Un lenguaje de programación permite a uno o más programadores especificar de manera precisa sobre qué datos debe operar una computadora, cómo estos datos deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Una característica relevante de los lenguajes de programación es precisamente que más de un programador pueda tener un conjunto común de instrucciones que puedan ser comprendidas entre ellos para realizar la construcción del programa de forma colaborativa.

1.6.1 Java.

Java es un lenguaje de programación orientado a objetos, desarrollado por *Sun Microsystems* a principios de los años 90. El lenguaje en sí toma mucha de su sintaxis de *C* y *C++,* pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Características:

- Es un lenguaje de programación que ofrece la potencia del diseño orientado a objetos con una sintaxis fácilmente accesible y un entorno robusto y agradable.
- Proporciona un conjunto de clases potente y flexible.
- Facilita la utilización de aplicaciones que se pueden incluir directamente en páginas *Web* (*applets*).

En *Java* se puede crear los siguientes tipos de aplicaciones:

- Aplicaciones: se ejecutan sin necesidad de un navegador.
- *Applets*: se pueden descargar de *Internet* y se observan en un navegador.
- *JavaBeans*: componentes *software Java*, que se puedan incorporar gráficamente a otros componentes.
- *JavaScript:* conjunto del lenguaje *Java* que puede codificarse directamente sobre cualquier documento *HTML.*
- *Servlets:* módulos que permiten sustituir o utilizar el lenguaje *Java* en lugar de programas *CGI* (*Common Gateway Interface*) a la hora de dotar de interactividad a las páginas *Web*.

El *API Java* es una interfaz de programación de aplicaciones provista por los creadores del lenguaje *Java*, y que da a los programadores los medios para desarrollar aplicaciones *Java.*

Como el lenguaje *Java* es un lenguaje orientado a objetos, la *API* de *Java* provee de un conjunto de clases utilitarias para efectuar todo tipo de tareas necesarias dentro de un programa.

Como ejemplo de *API*s se puede citar:

- *Java 2D*: gráficos 2D y manipulación de imágenes.
- *Java Media Framework*: elementos críticos en el tiempo: audio, video.
- *Java Animation*: animación de objetos en 2D.
- *Java Telephony*: integración con telefonía.
- *Java Share:* interacción entre aplicaciones multiusuario.
- *Java 3D*: gráficos 3D y su manipulación.

JAVA presenta los siguientes niveles de seguridad:

- Fuertes restricciones del 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.

Ventajas:

- Simplicidad: el lenguaje elimina ciertas complejidades como pueden ser el manejo de memoria y el uso de apuntadores que existen en otros lenguajes como *C*.
- Familiaridad: la sintaxis de *Java* es muy similar al de otros lenguajes de programación como *C* y *C*++, por lo que no implica un gran esfuerzo familiarizarse con este lenguaje.
- Seguridad: este lenguaje tiene ciertas políticas que restringen lo que se puede hacer con los recursos críticos del sistema para evitar la codificación de virus.
- Portabilidad: no depende de una arquitectura computacional ya que el compilador de *Java* genera un código conocido como *byte code* que puede ser interpretado por cualquier computadora una vez que se haya implementado un intérprete par cada plataforma.
- Multihebra: *Java* soporta sincronización de múltiples hilos de ejecución (*multithreading*) a nivel de lenguaje, mientras un hilo se encarga de la comunicación, otro puede interactuar con el usuario mientras otro presenta una animación en pantalla y otro realiza cálculos.
- *Java* es gratis, universal y de fácil distribución.

- *Java* es algo más que un lenguaje, es toda una plataforma por lo que es apoyada por muchas empresas, lo que le otorga un grado de calidad del que carece *C++.*

Desventajas:

- La velocidad: en ocasiones puede ocurrir un desperdicio de memoria, lo que acarrea que la ejecución del código sea lenta.
- Clases de librerías: al poseer clases de librerías el manejo del lenguaje se hace más difícil y consume mucho tiempo.

1.6.2 C#.

C# es el lenguaje que *Microsoft* desarrolló principalmente para la plataforma *.Net*. Para su creación se usaron conceptos de *C*, *C++, Smalltalk, Modula 2 y Java.*

Características:

- Moderno: mejora la productividad en el desarrollo de *software*, incorpora características del estado del arte de los lenguajes actuales.
- Simple: permite una sintaxis sencilla y elegante, evita la utilización de punteros, la gestión de memoria, la validación de límites de *arrays*.
- Poderoso: permite el desarrollo de código “seguro” y “no seguro”.
- De propósito general: puede ser utilizado para la construcción de aplicaciones *Web*, aplicaciones de escritorio, *servicios Web*, aplicaciones para celulares y componentes.
- Totalmente orientado a objetos.

Ventajas:

- Incorpora las características de un lenguaje de última generación y está en continuo desarrollo.
- Concepto formalizado de los métodos *get* y *set*, con lo que se consigue código mucho más legible.
- Gestión de eventos mucho más limpia.

Desventajas:

- No es multiplataforma.
- Permite el desarrollo de código no seguro.

1.6.3 C++.

En la actualidad, el *C*++ es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar uno de los primeros puestos como herramienta de desarrollo de aplicaciones. El *C*++ mantiene las ventajas del *C* en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del *C* original. La evolución de *C*++ ha continuado con la aparición de *Java*, un lenguaje creado simplificando algunas cosas de *C*++ y añadiendo otras, que se utilizan para realizar aplicaciones en *Internet*. (13)

Características:

Abstracción

- Variables de instancia.
- Métodos de instancia.
- Variables de clase.
- Métodos de clase.

Encapsulación

- De variables: privada, pública y protegida.
- De métodos: privada, pública y protegida.

Herencia

- Sencilla y múltiple.
- Unidades genéricas.
- Polimorfismo.
- No dispone de meta clases.

Además de los siguientes puntos, algunos heredados del lenguaje *C*:

- Es un lenguaje fuertemente tipado.
- Soporta multitarea mediante clases.
- Permite modularidad (vía ficheros: *includes*).

Ventajas:

- Permite un control de la memoria y una capacidad de programación de bajo nivel.
- Es una extensión de *C*. Por eso, muchos programadores encontrarán muy sencilla la transición.

Desventajas:

- No es multiplataforma. Para lograr aplicaciones que se ejecuten en varios SO, se requiere de mucho esfuerzo.
- Al ser una extensión de *C* hereda de este ciertos dogmas que entorpecen el uso del lenguaje.
- Es más complicado de aprender que *Java.*
- Puede resultar en elevados costos y problemas de mantenimiento y fiabilidad.
- La evolución del lenguaje ha sido estable mientras que en *Java* se ha manifestado de forma acelerada.

1.6.4 PHP.

PHP es un acrónimo recursivo que significa ***P****HP* ***H****ypertext* ***P****re-processor* (inicialmente *PHP Tools*, o, *Personal Home Page Tools*). Es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas *web* dinámicas. Es usado principalmente en interpretación del lado del servidor (*server-side scripting*) pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas incluyendo aplicaciones con interfaz gráfica usando las bibliotecas *Qt* o *GTK+.*

Ventajas:

- Lenguaje multiplataforma.
- Capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad, destaca su conectividad con *MySQL*.

- Capacidad de expandir su potencial utilizando la enorme cantidad de módulos (llamados extensiones).
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite las técnicas de programación orientada a objetos.
- Biblioteca nativa de funciones sumamente amplia e incluida.
- No requiere definición de tipos de variables.
- Tiene manejo de excepciones (desde *PHP*5).

Desventajas:

- Promueve creación de código desordenado y con un mantenimiento complejo.
- No posee adecuado manejo de unicode.
- Es muy difícil de optimizar.
- Diseñado especialmente hacia un modo de realizar aplicaciones *web* que es problemático y obsoleto. (14)

1.6.5 ¿Por qué Java?

Java es el lenguaje apropiado para la implementación del sistema por las razones que se exponen a continuación. Respecto a *C++* es válido mencionar que no es multiplataforma, aspecto que hace al lenguaje poco útil, por otra parte y no menos importante *Java* está basado en *C++* pero elimina lo aspectos que entorpecían el uso del lenguaje y que le agregaban cierta complejidad como pueden ser el manejo de memoria y el uso de apuntadores, por lo que se puede afirmar que *Java* es una mejora de *C++.* Analizando los aspectos positivos y negativos de *C#* este tiene mucha similitud con *Java* aunque es dependiente de la plataforma a menos que se utilice *Mono* para poder hacer de C# un lenguaje multiplataforma. Desde otra perspectiva se puede analizar *PHP* que es un lenguaje muy completo, pero está especializado en aplicaciones *Web*, esto no quiere decir que no permita el desarrollo de aplicaciones *desktop* pero en este sentido tiene mucho que mejorar. Otros aspectos de interés que llevan a *Java* a la vanguardia es que restringe el uso de aspectos críticos del sistema para evitar la codificación de virus y además es gratis por lo que no se tiene que pagar licencias para poder usarlo.

1.7 Entorno de Desarrollo Integrado.

Un entorno de desarrollo integrado o, en inglés, *Integrated Development Environment* (*IDE*), es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios. Lo constituye un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica *GUI*. Los *IDE*s pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

1.7.1 Eclipse.

Eclipse es un Entorno de Desarrollo Integrado (*IDE*) que en un principio fue diseñado y desarrollado por *IBM* y que luego fue lanzada a la comunidad de *software* libre.

Eclipse es una plataforma universal que integra herramientas de desarrollo, con una arquitectura abierta y basada en *plug-ins.* Además, da soporte a todo tipo de proyectos que abarcan desde el ciclo de vida del desarrollo de aplicaciones, incluyendo soporte para modelado. Esta arquitectura de *plug-ins* permite integrar diversos lenguajes sobre un mismo *IDE* e introducir otras aplicaciones accesorias. Conservan el registro de las versiones, generan y mantienen la documentación de cada etapa del proyecto.

Características:

- Editor visual con sintaxis coloreada.
- Compilación incremental de código.
- Modifica e inspecciona valores de variables.
- Avisa de los errores cometidos mediante una ventana secundaria.
- Depura código que resida en una máquina remota.

Por ser un *IDE* hereda las **desventajas** propias de estos:

- Consumen más recursos.
- Son más lentos.
- Para manejarlos es necesario cierto aprendizaje.

- Ausencia de un *IDE* universal.(15)

1.7.2 NetBeans.

NetBeans es un entorno de desarrollo escrito en *Java*, pero puede servir para cualquier otro lenguaje de programación. *NetBeans* es una aplicación de código abierto ("*open source*") diseñada para el desarrollo de aplicaciones fácilmente portables entre las distintas plataformas, haciendo uso de la tecnología *Java.*

Dispone de soporte para crear interfaces gráficas de forma visual, desarrollo de aplicaciones *Web*, control de versiones, colaboración entre varias personas, creación de aplicaciones compatibles con teléfonos móviles, resaltado de sintaxis y por si fuera poco sus funcionalidades son ampliables mediante la instalación de *packs.* (16)

Ventajas:

- Incorpora la línea de tecnología de *Sun* dentro del entorno.
- No hay que buscar *plug-ins* por todas partes, por lo regular todo se encuentra en la misma caja.
- La estructura de los proyectos está basada en *ant*, por lo tanto es muy personalizable, por si después de empaquetar un proyecto se quiere enviar en *ftp*, se puede hacer fácilmente con *ant*.
- *Developer Collaboration*: es un *plug-in* que como su nombre lo indica, permite hacer desarrollos en equipo. Se puede modificar el mismo archivo a la vez y ver los cambios en tiempo real, se puede estar chateando o enviar pedazos de códigos.

NetBeans Platform es una base modular y extensible usada como estructura de integración para crear grandes aplicaciones de escritorio. Empresas independientes asociadas, especializadas en desarrollo de *software*, proporcionan extensiones adicionales que se integran fácilmente en la plataforma y que pueden también utilizarse para desarrollar sus propias herramientas y soluciones. (17)

Entre las **características** de la plataforma están:

- Administración de las interfaces de usuario (ej. menús y barras de herramientas).

- Administración de las configuraciones del usuario.
- Administración del almacenamiento (guardando y cargando cualquier tipo de dato).
- Administración de ventanas.
- *Framework* basado en asistentes (diálogo paso a paso).

Ambos productos, *NetBeans IDE y NetBeans Platform* son de código abierto y gratuito para uso tanto comercial como no comercial.

1.7.3 JBuilder.

JBuilder es un *IDE Java* de *Borland*. En la versión 2007 ya es multiplataforma, está disponible para *Windows, Linux y MacOS X.* Soporta plenamente la última versión del *kit* de desarrollo de *Java* (*JDK*) proporcionado por *Sun,* la versión 1.1.x, aunque es posible compilar programas y *applets* basados en versiones anteriores.

Otra característica muy importante de *JBuilder* es su soporte sin compromisos de componentes escritos sola y exclusivamente en *Java*, lo que maximiza la portabilidad real del código, frente a otros entornos que por apoyarse en código no *Java*, ven muy mermada su portabilidad. Sus programas se pueden ejecutar en *PC´s*, *Macintosh*, *Unix*, *Network Computers* o en cualquier plataforma que soporte *Java*. *JBuilder* ha sido desarrollado íntegramente para desarrollos *Java*, de manera que no se encuentra en él ninguna reminiscencia de cualquier otro lenguaje. El entorno visual ofrece un entorno seguro para crear aplicaciones multiplataforma, y también se puede alternar entre las herramientas visuales y el código, manteniendo siempre la coherencia entre las mismas.

Ventajas:

- Potente, rápido y fiable.

Desventajas:

- Consume muchos recursos.
- Es muy caro.

1.7.4 ¿Por qué NetBeans?

NetBeans IDE es multiplataforma, se ejecuta en *Windows, Linux, Mac OS X y Solaris*, es de código abierto, libre y gratis. Tiene la ventaja además de que es patrocinado por los creadores de *Java*, *Sun Microsystems*. Es el *IDE* ideal para crear grandes aplicaciones de escritorio. En el caso de *JBuilder* es muy bueno sobre todo para el desarrollo en *swing y j2ee*, facilita el desarrollo rápido pero esto va contrastado con lo que cuesta la licencia y lo pesada de la aplicación, aunque también es válido mencionar que ha recibido algunas críticas de poca usabilidad porque tiene muchos menús que pierden al usuario. *Eclipse* es también muy bueno pero tiene una arquitectura basada en plug-ins, aspecto que lo hace muy dependiente de estos y tiende a retrasar el trabajo. Por los motivos expuestos anteriormente se ha decidido que el *IDE* más completo para el desarrollo de una aplicación en *Java* es el *NetBeans*.

1.8 FRAMEWORKS.

Es una estructura de soporte definida en la cual un proyecto de *software* puede ser organizado y desarrollado. Típicamente, un *framework* puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros *software* para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Además representa una arquitectura de *software* que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

1.8.1 Java Media Framework (JMF).

Java Media Framework (*JMF*) proporciona herramientas para la captura, procesamiento y almacenamiento de datos multimedia permitiendo su transmisión y recepción a través de *Internet.* Más concretamente permite:

- Reproducir ficheros multimedia en *applets* y aplicaciones.
- Reproducir flujos multimedia recibidos en tiempo real a través de la red.
- Capturar audio y vídeo de un micrófono y una cámara de video.

Arquitectura de *JMF*.

Pasos principales para realizar el tratamiento a datos multimedia:

- La adquisición de datos: captura desde un dispositivo físico, lectura de un fichero o recepción desde la red.
- Procesado: aplicación de efectos como filtrado o realces, compresión y/o descompresión, conversión entre formatos.
- La salida de datos: presentación, almacenamiento en fichero o transmisión a través de la red.

Para entender más la arquitectura de *JMF* a continuación se muestra una representación de su funcionamiento.

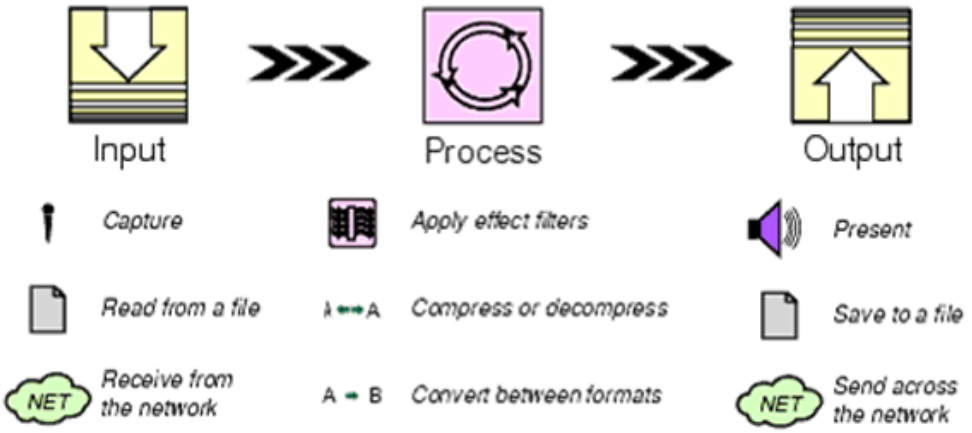

Figura 1: Representación gráfica de la arquitectura *JMF*.

Formatos soportados por *JMF*.

Existe una gran variedad de formatos de audio, imagen y video que soporta el *JMF*. Entre los cuales se pueden detectar destacar*: AIFF, AU, AVI, GSM, MIDI, MPEG, QuickTime, RMF, and WAV*. (18)

1.9 Plataforma software.

Una plataforma *software* es un conjunto de artefactos (componentes o subsistemas) que forman una estructura común a partir de la cual se pueden derivar (desarrollar, construir) sistemas de una forma eficiente.

Los sistemas derivados de una plataforma no sólo comparten código, sino requisitos y arquitectura. Se da por tanto un proceso de reutilización natural de los artefactos de la plataforma en los diferentes sistemas.

Ventajas de una plataforma *software*:

- Reducción del coste de operación. Uno de los parámetros fundamentales en la ingeniería y en la estrategia empresarial es la justificación económica. Una plataforma *software* permitirá reducir el coste de operar en el campo del desarrollo del *software*, ya que hará posible reducir costes tanto a nivel de desarrollo, como de mantenimiento. El elemento esencial aquí es el de reutilización de los artefactos que componen la plataforma.
- Mejora de la calidad. Al ser usados los artefactos de la plataforma en muchos sistemas, estos son revisados y probados de forma intensiva, por lo que es más fácil hacer que dichos artefactos tengan un número muy reducido de errores, buen rendimiento, etc.
- Reducción del tiempo de desarrollo. Tener la capacidad de poder desarrollar proyectos en menos tiempo no tiene sólo un impacto económico directo en el coste individual de los proyectos, sino que tiene varios indirectos: al ser más competitivos se puede optar a realizar más proyectos y a ser por tanto más productivos. Para esto es fundamental una plataforma estable y adaptada a la problemática a la que se quiere aplicar.
- Gestión centralizada de la evolución. La introducción de un nuevo artefacto en la plataforma da la posibilidad de que todos los sistemas derivados de la plataforma usen ese nuevo artefacto. Esta gestión centralizada permite a los sistemas evolucionar de forma ordenada.
- Reducción de la complejidad global. El hecho de usar una plataforma bien definida con una serie de artefactos reconocibles y conocidos reduce la complejidad significativamente tanto a nivel de desarrollo, como de gestión del desarrollo y mantenimiento.
- Mejoras en las estimaciones de costes. Calcular costes para sistemas derivados de la plataforma es más sencillo y tiene menos riesgo porque se tienen ya resultados de otros sistemas derivados de la plataforma y que por tanto pueden ser comparables.
- Interfaz común. Todos los sistemas derivados de la plataforma tendrán en común bastantes elementos en cuanto a su funcionamiento e interfaz, lo que hace que los

usuarios aprendan rápido y se sientan cómodos ante nuevos sistemas derivados de la misma plataforma.

- Estandarización del conocimiento técnico. El conocimiento de los técnicos de la organización se estandariza al estar todos los sistemas (o casi todos) basados en la plataforma y sus artefactos. Esto hace fácil que los técnicos puedan comprender y por lo tanto desarrollar y mantener el conjunto de sistemas generados por la organización, facilitando a su vez la movilidad del personal entre los diferentes proyectos atendiendo a las necesidades corporativas.(19)

Desventaja:

- Inversión en el desarrollo y mantenimiento de la plataforma. Se supone que una vez construida una plataforma la inversión realizada se recupera al utilizarla en 3-5 proyectos.

1.9.1 Máquina virtual de Java (JVM).

La máquina virtual de *Java* (en inglés *Java Virtual Machine*, *JVM*) es un programa gratuito, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el *Java bytecode*), el cual es generado por el compilador del lenguaje *Java*.

Actualmente la mayoría de las plataformas de ejecución de *Java* son máquinas virtuales. La máquina virtual es un programa normal solamente ejecutable en un sistema operativo particular. Esta *JVM* es la que le permite a *Java* ser multiplataforma debido a que cuando se compila el código de *Java*, este queda compilado en *bytecode* que es lo que reconoce la *JVM* instalada en cualquier plataforma *Windows, Linux o MacOS*.(20)

1.10 Patrones de arquitectura.

Definen la estructura de un sistema *software*, los cuales a su vez se componen de subsistemas con sus responsabilidades, también tienen una serie de directivas para organizar los componentes del mismo sistema, con el objetivo de facilitar la tarea del diseño de tal sistema.

1.10.1 Modelo-Vista-Controlador.

MVC (por sus siglas en inglés) es un patrón de diseño de arquitectura de *software* usado principalmente en aplicaciones que manejan gran cantidad de datos y transacciones complejas donde se requiere una mejor separación de conceptos para que el desarrollo esté estructurado de una mejor manera, facilitando la programación en diferentes capas de manera paralela e independiente. *MVC* sugiere la separación del *software* en 3 estratos:

Modelo: Es la representación de la información que maneja la aplicación. El modelo en sí son los datos puros que puestos en contexto del sistema proveen la información al usuario o a la aplicación misma.

Vista: Es la representación del modelo en forma gráfica disponible para la interacción con el usuario.

Controlador: Es la capa encargada de manejar y responder las solicitudes del usuario, procesando la información necesaria y modificando el modelo en caso de ser necesario. (21)

Ventajas:

- Menor acoplamiento.
- Mayor cohesión.
- Las vistas proveen mayor flexibilidad y agilidad.
- Más claridad de diseño.
- Facilita el mantenimiento.
- Mayor escalabilidad. (22)

Desventajas:

- La separación de conceptos en capas agrega complejidad al sistema.
- La cantidad de archivos a mantener y desarrollar se incrementa considerablemente.
- La curva de aprendizaje del patrón de diseño es más alta usando otros modelos más sencillos.
- *MVC* es un patrón de diseño orientado a objetos por lo que su implementación es sumamente costosa y difícil en lenguajes que no siguen este paradigma.

1.10.2 Patrón n capas.

Este patrón permite crear aplicaciones mediante un proceso iterativo de división del problema en piezas manejables denominadas componentes. Estos componentes, o componentes de negocio son modelos *software* basados típicamente en la vista de un objeto real, evento o proceso de negocio. Los componentes *software* individuales pueden formar parte y adaptarse tanto a estructuras independientes como a sistemas colaborativos. El diseño de aplicaciones en n-capas es ideal para la creación de sistemas adaptables, donde cada componente puede ser utilizado y reutilizado en nuevas combinaciones para satisfacer requisitos de negocio dinámicos. Esto permite a los desarrolladores y a las nuevas aplicaciones reutilizar componentes existentes que modelan lógica de negocio sobradamente probada.

Ventajas:

- Reutilización de capas.
- Facilita la estandarización.
- Dependencias se limitan a intra-capa.
- Contención de cambios a una o pocas capas.
- Clara separación de las funciones de control de la interfaz y presentación de datos con la lógica de la aplicación.
- Reusabilidad de componentes.
- Independencia de la interfaz del cliente y la arquitectura de datos.
- No está atado a un lenguaje de programación específico.

Desventajas:

- Pérdida de eficiencia.
- Trabajo innecesario por parte de capas más internas o redundante entre varias capas.
- Dificultad de diseñar correctamente la granularidad de las capas. (23)

1.10.3 ¿Por qué n capas?

Analizar las características, ventajas y desventajas de los patrones de diseño de arquitectura más usados a nivel mundial ofrece las bases para una correcta selección del patrón según los objetivos que se quieren lograr con la arquitectura. El patrón n capas es el que se ha escogido

para utilizar en el diseño de la aplicación ya que sus ventajas se adaptan perfectamente a los objetivos que se quieren lograr. Dentro de los beneficios más significativos que se obtienen con su implementación se pueden mencionar los siguientes. El código está más ordenado y es más flexible a la hora de querer modificarlo o agregar nuevas funciones. Al estar distribuidas las funciones y responsabilidades, es posible reemplazar y actualizar sin que los demás elementos se vean afectados por ese cambio o se afectarán mínimamente. Se logra desarrollar aplicaciones de manera clara, nítida y transparente, con la ventaja de poder desarrollar varias capas al mismo tiempo. El no estar atado a un lenguaje de programación, garantiza una fácil adaptación, su principal fortaleza es que puede hacer convivir aplicaciones creadas en distintos lenguajes.

1.11 Gestor 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. Por tanto 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.

1.11.1 MySQL.

MySQL es un sistema de gestión de bases de datos relacional, licenciado bajo la *GPL* de la *GNU*. Su diseño multihilo le permite soportar una gran carga de forma muy eficiente. *MySQL* fue creada por la empresa sueca *MySQL AB*, que mantiene el *copyright* del código fuente del servidor *SQL*, así como también de la marca. Aunque *MySQL* es *software* libre, *MySQL AB* distribuye una versión comercial de *MySQL*, que no se diferencia de la versión libre más que en el soporte técnico que se ofrece, y la posibilidad de integrar este gestor en un *software* propietario, ya que de no ser así, se vulneraría la licencia *GPL*. Este gestor de bases de datos es, probablemente, el gestor más usado en el mundo del *software* libre, debido a su gran rapidez y facilidad de uso. Esta gran aceptación es debida, en parte, a que existen infinidad de

librerías y otras herramientas que permiten su uso a través de gran cantidad de lenguajes de programación, además de su fácil instalación y configuración.(24)

Características:

- Aprovecha la potencia de sistemas multiprocesador, gracias a su implementación multihilo.
- Soporta gran cantidad de tipos de datos para las columnas.
- Dispone de *API'*s en gran cantidad de lenguajes (*C, C++, Java, PHP*).
- Gran portabilidad entre sistemas.
- Soporta hasta 32 índices por tabla.
- Gestión de usuarios y *passwords*, manteniendo un muy buen nivel de seguridad en los datos.

Ventajas:

- Alta velocidad a la hora de realizar las operaciones.
- El bajo consumo lo hace apto para ser ejecutado en una máquina con escasos recursos sin ningún problema.
- Las utilidades de administración de este gestor son envidiables para muchos de los gestores comerciales existentes, debido a su gran facilidad de configuración e instalación.
- Tiene una probabilidad muy reducida de corromper los datos, incluso en los casos en que los errores no se produzcan en el propio gestor, sino en el sistema en el que está.
- El conjunto de aplicaciones *Apache-PHP-MySQL* es uno de los más utilizados en *Internet* en servicios de foro. (24)

Desventajas:

- Carece de soporte para transacciones, *rollback's* y subconsultas.
- El hecho de que no maneje la integridad referencial, hace de este gestor una solución pobre para muchos campos de aplicación, sobre todo para aquellos programadores que provienen de otros gestores que sí poseen esta característica.

- No es viable para su uso con grandes bases de datos, a las que se acceda continuamente, ya que no implementa una buena escalabilidad. (24)

1.11.2 PostgreSQL.

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional (*ORDBMS*) basado en el proyecto *POSTGRES*, de la universidad de *Berkeley*, es una derivación libre (*OpenSource*) de este proyecto, y utiliza el lenguaje *SQL92/SQL99*.

Fue el pionero en muchos de los conceptos existentes en el sistema objeto-relacional actual, incluido, más tarde en otros sistemas de gestión comerciales. *PostgreSQL* es un sistema objeto-relacional, ya que incluye características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. A pesar de esto, *PostgreSQL* no es un sistema de gestión de bases de datos puramente orientado a objetos. (24)

Características:

- Implementación del estándar *SQL92/SQL99*.
- Soporta distintos tipos de datos: además del soporte para los tipos base, también soporta datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes (*MAC, IP*) y cadenas de *bits*. También permite la creación de tipos propios.
- Incorpora una estructura de datos *array*.
- Incorpora funciones de diversa índole: manejo de fechas, geométricas y orientadas a operaciones con redes.
- Permite la declaración de funciones propias, así como la definición de disparadores.
- Soporta el uso de índices, reglas y vistas.
- Incluye herencia entre tablas (aunque no entre objetos, ya que no existen), por lo que a este gestor de bases de datos se le incluye entre los gestores objeto-relacionales.
- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos. (24)

Ventajas:

- Posee una gran escalabilidad. Es capaz de ajustarse al número de *CPU*s y a la cantidad de memoria que posee el sistema de forma óptima, haciéndole capaz de soportar una mayor cantidad de peticiones simultáneas de manera correcta, en algunos *benchmarks* se dice que ha llegado a soportar el triple de carga de lo que soporta *MySQL*.
- Implementa el uso de *rollback's*, subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz, y ofreciendo soluciones en campos en los que *MySQL* no podría.
- Tiene la capacidad de comprobar la integridad referencial, así como también la de almacenar procedimientos en la propia base de datos, equiparándolo con los gestores de bases de datos de alto nivel, como puede ser *Oracle*. (24)
- Es altamente extensible: soporta operadores y tipos de datos definidos por el usuario.
- Cuenta con una *API* flexible lo cual ha permitido dar soporte para el desarrollo con *PostgreSQL* en diversos lenguajes de programación entre los que se incluyen: *Object Pascal, Python, Perl, PHP, ODBC, Java/JDBC, Ruby, TCL, C/C++, y Pike.*
- Tiene soporte para lenguajes procedurales internos, incluido un lenguaje nativo denominado *PL/pgSQL*, el cual es comparable con el lenguaje procedural de *Oracle PL/SQL.*
- Es totalmente libre.

Desventajas:

- Consume gran cantidad de recursos.
- Tiene un límite de 8K por fila, aunque se puede aumentar a 32K, con una disminución considerable del rendimiento.
- Es de 2 a 3 veces más lento que *MySQL*. (24)

1.11.3 Oracle

Oracle es un sistema de gestión de base de datos relacional (o *RDBMS* por el acrónimo en inglés de *Relational Data Base Management System*), desarrollado por *Oracle Corporation*.

Se considera a *Oracle* como uno de los sistemas de bases de datos más completos, destacando su:

- Soporte de transacciones.

- Estabilidad.
- Escalabilidad.
- Soporte multiplataforma.

Ha sido criticada por algunos especialistas la seguridad de la plataforma, y las políticas de suministro de parches de seguridad, modificadas a comienzos de 2005 y que incrementan el nivel de exposición de los usuarios. En los parches de actualización provistos durante el primer semestre de 2005 fueron corregidas 22 vulnerabilidades públicamente conocidas, algunas de ellas con una antigüedad de más de 2 años.

Ventajas:

- *Oracle* es el motor de base de datos relacional más usado a nivel mundial. Puede ejecutarse en todas las plataformas, desde una *PC* hasta un supercomputador.
- *Oracle* soporta todas las funciones que se esperan de un buen servidor: un lenguaje de diseño de bases de datos muy completo (*PL/SQL*) que permite implementar diseños activos, con *triggers* y procedimientos almacenados, con una integridad referencial declarativa bastante potente.
- Permite el uso de particiones para la mejora de la eficiencia, de replicación e incluso ciertas versiones admiten la administración de bases de datos distribuidas.
- El *software* del servidor puede ejecutarse en multitud de sistemas operativos.
- *Oracle* es la base de datos con más orientación hacia *Internet*.

Desventajas:

- Este sistema ha comenzado a evolucionar hacia la orientación a objetos, añadiendo tipos de clases, referencias, tablas anidadas, matrices y otras estructuras de datos complejas. Desafortunadamente, la implementación actual del sistema no ofrece una ventaja clara en eficiencia, como sería de esperar, y sí provocan la incompatibilidad de los diseños que aprovechan las nuevas características con otras bases de datos.
- La remodelación del sistema de almacenamiento por causa de la introducción de extensiones orientadas a objetos ha sido motivo de muchos fallos.
- El mayor inconveniente de *Oracle* es quizás su precio. Incluso las licencias de *Personal Oracle* son excesivamente caras.

- Otro problema es la necesidad de ajustes. Un error frecuente consiste en pensar que basta instalar el *Oracle* en un servidor y enchufar directamente las aplicaciones clientes. Un *Oracle* mal configurado puede ser desesperantemente lento.
- También es elevado el coste de la formación, y sólo últimamente han comenzado a aparecer buenos libros sobre asuntos técnicos distintos de la simple instalación y administración.

1.11.4 ¿Por qué PostgreSQL?

Una vez analizados los sistemas gestores de base de datos se decidió utilizar en la elaboración del sistema planteado al *PostgreSQL* por todas las ventajas que este representa respecto a otros sistemas. A continuación se plantean algunas razones de las que se tuvieron en cuenta para la selección. *PostgreSQL* es un gestor de base de datos de tipo objeto relacional mientras que *Oracle* es sólo relacional, por ende *PostgreSQL* maneja herencia que es una ventaja. En *Oracle* no se pueden crear tipos de datos personalizados, mientras que en *PostgreSQL* sí. El *Oracle* tiene un coste muy elevado, en tanto el hecho de que *PostgreSQL* es un producto *Open Source*, sin costos de licencia lo hacen extremadamente atractivo. Girando la comparación ahora hacia *MySQL* se puede decir que este no considera las claves ajenas e ignora la integridad referencial, aspecto que se deja en manos del programador de la aplicación sin embargo *PostgreSQL* soporta transacciones y desde la versión 7.0, considera las claves ajenas, con comprobaciones de integridad referencial. *MySQL* es considerado más rápido, pero también es cierto que por lograr la rapidez se sacrificaron aspectos muy importantes que lo hacen menos útil, también es importante destacar que *PostgreSQL* ha soportado el triple de peticiones simultáneas de lo que ha soportado *MySQL*. Mencionando otros puntos a favor del *PostgreSQL* lo primero, sin duda, es la economía tangible de decenas de miles de dólares en licencias de *software*. Otro punto importante es la simplificación del proceso de administración de licencias de *software*, que no es necesario cuando se usa *software* libre. Por otra parte, también ofrece beneficios como el aumento de la estabilidad y *performance* del sistema.

1.12 ¿Por qué una aplicación de escritorio?

Una vez realizado el estudio correspondiente a algunos de los sistemas que fueron construidos con fines similares y después de valorar las disponibilidades de los usuarios se ha determinado realizar una aplicación de escritorio por las razones que se exponen a continuación:

Con una aplicación de escritorio no es necesaria una conexión a internet ya que funciona localmente, este aspecto a tener en cuenta es muy importante ya que la aplicación va a ser usada en una computadora sin una ubicación física estable, por lo que se dificultaría su uso en caso de que el entrenamiento del equipo o el evento competitivo se lleve a cabo en un lugar donde no haya conexión.

Una aplicación de escritorio es una mejor opción cuando hay que acceder a recursos locales como pueden ser memoria de video, sonido, teclado, cámara digital.

La navegación e interfaz de usuario es más rápida que en una aplicación web. Cuando se navega hace falta más tiempo para mostrar o enviar los datos, cosa que en una aplicación de escritorio normalmente es instantáneo.

1.13 Conclusiones.

En este capítulo se describió de manera general el estado del arte, donde se analizaron los *software* que existen a nivel internacional para dar solución a la problemática de la gestión de la información generada en un juego de voleibol, se determinó que no es factible la utilización de ninguno de ellos debido a que todos contienen licencias propietarias. También se han caracterizado las herramientas, tecnologías y lenguajes a emplear para el diseño e implementación de un sistema de estudio de contrarios para el voleibol, teniendo en cuenta los requerimientos planteados por los técnicos de este deporte en el país.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA.

2.1 Introducción.

En este capítulo se dejan plasmados cuáles son los objetivos estratégicos de la organización, se presenta el modelo de dominio para lograr un entendimiento de la estructura y la dinámica de la organización y poder definir los problemas actuales existentes e identificar las mejoras potenciales.

Es objetivo además especificar las funcionalidades y cualidades con que debe contar el sistema que estarán incluidas en el plan de requerimientos, de igual forma se presentará el prototipo de interfaz de usuario con el objetivo de lograr un acercamiento a satisfacer las expectativas del usuario final. Se incluirán los casos de uso que guiarán la implementación del sistema y una descripción textual de estos para un mejor entendimiento.

2.2 Modelo de dominio.

El Modelo de Dominio o Modelo Conceptual es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa clases conceptuales del dominio del problema, conceptos del mundo real, no de los componentes de *software*.

Una clase conceptual puede ser una idea o un objeto físico (símbolo, definición y extensión).

El modelo de dominio se representa en *UML* con un diagrama de clases en los que se muestra:

- Conceptos u objetos del dominio del problema: clases conceptuales.
- Asociaciones entre las clases conceptuales.
- Atributos de las clases conceptuales.

En el Modelo de Dominio no se muestra comportamiento. Las clases conceptuales pueden tener atributos pero no métodos. (25)

2.2.1 Glosario de términos.

Este glosario está concebido para ayudar a comprender algunos de los términos utilizados en la modelación del dominio del problema y se presentan con fines informativos para aquellas personas que no estén relacionados con estos.

Cámara de video: Es un dispositivo electrónico usado para capturar y almacenar videos electrónicamente, se utiliza para captar los videos de los juegos de voleibol.

Jefe técnico: Representa al grupo de entrenadores, los guía en su trabajo y dirige las preparaciones metodológicas de estos, es uno de los entrenadores más capacitados y con más experiencia.

Entrenador: Es la persona encargada de la instrucción y entrenamiento del colectivo de deportistas.

Director de equipo: Maneja los aspectos tácticos, técnicos, sicológicos, metodológicos y preparación física de un equipo. Analiza los criterios valorativos emitidos por los demás entrenadores para tomar decisiones sobre la estrategia de juego y el entrenamiento del equipo.

Preparador físico: Es el encargado de planificar y mejorar la preparación física de los atletas para optimizar su rendimiento.

Preparador técnico: Tiene la responsabilidad de planificar y llevar a cabo la preparación técnica del equipo.

Entrenador asistente: Guía conjuntamente con los preparadores físicos y técnicos el entrenamiento.

Video del juego: Hace referencia a la captación, procesamiento, transmisión y reconstrucción de una secuencia de imágenes y sonidos que representan escenas en movimiento, contiene la captación íntegra del juego de voleibol.

Estudio de contrarios: El estudio de contrarios es un análisis que se hace para obtener información técnico-táctica de los equipos contrarios. Tiene como objetivo lograr una mejor preparación física y psicológica de los jugadores y determinar estrategias de juego efectivas.

Documentación: Hace referencia a todas las anotaciones que hacen los entrenadores de aspectos que ellos creen importantes resaltar cuando están observando el video del juego.

Equipo de la TVC: Está conformado por un pequeño grupo de camarógrafos y técnicos de televisión que acompañan al equipo de deportistas en sus viajes para realizar la filmación de todos sus eventos competitivos.

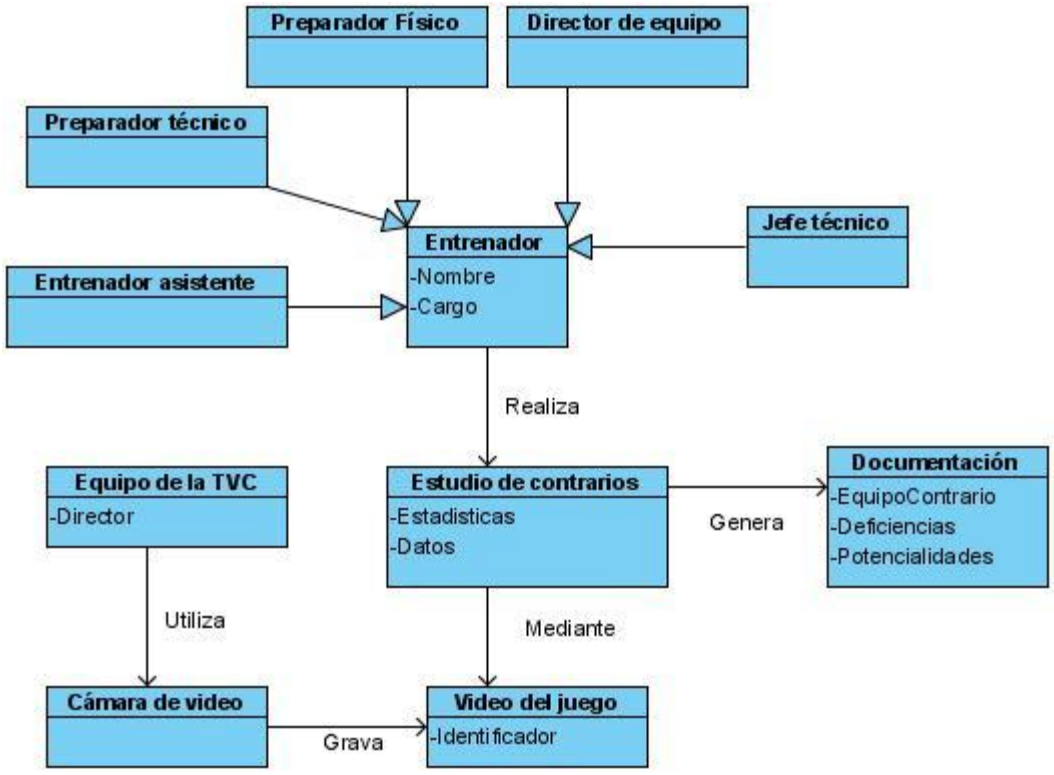

Figura 2: Modelo de domino.

2.3 Modelo del sistema.

El modelo del sistema es un modelo de lo que el sistema debe hacer para satisfacer los requerimientos del usuario, diciendo lo mínimo posible o nada acerca de cómo se implantará el sistema. Esto significa que este modelo supone que se tiene disponible una tecnología perfecta y que se puede obtener fácilmente y sin costo.

El modelo debe describir la función que el sistema debe realizar sin importar su implantación final; debe describir el contenido de los flujos o almacenes de datos, sin describir el medio u organización física de los datos.

Específicamente el modelo de sistema describe:

- La política o lógica esencial de las funciones que se requiere realizar.

- El contenido esencial de los datos que almacena el sistema y que se mueven a través de él.
- El comportamiento esencial dependiente del tiempo que el sistema debe exhibir para manejar señales e interrupciones del ambiente exterior.(26)

2.3.1 Requerimientos funcionales.

De las muchas definiciones que existen para requerimiento, ha continuación se presenta una de las más completas.

1. Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo.
2. Una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal.
3. Una representación documentada de una condición o capacidad como en 1 ó 2. (27)

Los requerimientos funcionales definen las funciones que el sistema será capaz de realizar. Describen las transformaciones que el sistema realiza sobre las entradas para producir salidas. (28)

RF1: Gestionar video.

RF1.1: Guardar el video en tiempo real.

RF1.2: Eliminar video.

RF1.3: Mostrar listado de videos guardados.

RF2: Gestionar escena.

RF2.1: Delimitar escena utilizando teclas de acceso rápido.

RF2.2: Guardar escena delimitada.

RF2.3: Mostrar listado de escenas guardadas por tipo: K1, K2, ataque, recibo, pase, bloqueo, defensa, saque.

RF2.4: Eliminar escena.

RF3: Reproducir hasta 4 escenas al mismo tiempo.

RF3.1: Pausar cada una de las escenas en reproducción.

RF3.2: Detener cada una de las escenas en reproducción.

RF4: Abrir archivo de video desde directorio físico.

RF1.4: Pausar video en reproducción.

RF1.5: Detener Video en reproducción.

RF5: Reproducir video en tiempo real desde una cámara digital.

2.3.2 Requerimientos no funcionales.

Los requerimientos no funcionales tienen que ver con características que de una u otra forma puedan limitar el sistema, como por ejemplo, el rendimiento en tiempo y espacio, interfaces de usuario, fiabilidad (robustez del sistema, disponibilidad de equipo), mantenimiento, seguridad, portabilidad, estándares, etc. (28)

Portabilidad:

- El sistema será multiplataforma (*Windows, Linux, Solaris*).

Soporte:

- Garantía de instalación y prueba del sistema.

Apariencia o interfaz externa:

- Diseño sencillo, permitiendo la utilización del sistema sin mucho entrenamiento.
- El producto final debe tener una interfaz fácil de usar y amigable.
- Todos los mensajes en pantalla aparecerán en idioma español.

Confiabilidad:

- Garantía de un tratamiento adecuado de las excepciones.

Disponibilidad:

- El sistema deberá estar disponible en todo momento.

Usabilidad:

- El sistema debe contar con menús que agrupen las funcionalidades que están relacionadas.
- Para utilizar el sistema es necesario poseer conocimientos elementales de computación.

- El sistema contará con teclas de acceso rápido que faciliten y aceleren su utilización.

***Software*:**

- Un gestor de base de datos *PostgreeSQL*.
- Máquina virtual de *Java*.
- Se debe instalar el Java Madia Framework correspondiente al sistema operativo que esté instalado en la computadora como se recomienda continuación.

  Para Windows: JMF-2_1_1e-windows-i586.

  Para Solaris: jmf-2_1_1e-solaris-sparc.

  Para Linux: jmf-2_1_1e-linux-i586.

Hardware:

- Tarjeta de video: VGA 128 Mb ATI Radeon 7000/7200/7500/9000 DDR o superior.
- Tarjeta capturadora de video: FireWire IEEE 1394 recomendada.
- Cámara de video.
- Microprocesador P4 2.4 GHz recomendado.
- Disco Duro: 80 GB o superior.
- RAM: 256 MB o superior.

2.3.3 Descripción del sistema propuesto.

Teniendo como punto de partida los requerimientos funcionales que son los que determinan las necesidades que el usuario desea satisfacer, se ha propuesto la realización de un sistema que brinde la posibilidad de reproducir y guardar un video en tiempo real. La propuesta de solución debe satisfacer la necesidad que tiene el usuario de consultar el video cuando lo estime conveniente y fragmentarlo en escenas de interés que pueden ser las escenas de las acciones básicas en el voleibol (saque, ataque, bloqueo, pase, recibo, defensa) o las escenas de los complejos 1 y 2 ó k1 (recepción, pase, ataque y aseguramiento del ataque) y k2 (saque, bloqueo, apoyo y defensa), que son los elementos de importancia para que un entrenador o un técnico de deporte pueda evaluar el desempeño del equipo contrario de voleibol. Otro elemento crítico es que el sistema pueda listar las escenas editadas por su tipo que puede ser por cada una de las acciones básicas o los complejos expuestos anteriormente. Con el sistema para

estudio de contrarios en voleibol se espera darles una herramienta a los técnicos y entrenadores que les ayude a procesar información visual para la toma de decisiones acertadas en los partidos de voleibol.

2.3.4 Actores del sistema.

Un actor representa un rol que asume alguien o algo que interactúa con el sistema pero que se encuentra fuera de su frontera, es decir, que el nombre adecuado para el actor será el del papel que juegue dicha entidad externa, desde la perspectiva del sistema.

Actor	Justificación
Entrenador	Es la persona encargada de realizar todo el procesamiento del video relacionado con cada juego.

Tabla 1: Descripción del actor del sistema.

2.3.5 Diagrama de casos de uso del sistema.

Un diagrama de casos de uso (*Use Case Diagram*) es una representación gráfica de parte o el total de los actores y casos de uso del sistema, incluyendo sus interacciones. Todo sistema tiene como mínimo un diagrama *Main Use Case*, que es una representación gráfica del entorno del sistema (actores) y su funcionalidad principal (casos de uso).

Un diagrama de casos de uso muestra, por tanto, los distintos requisitos funcionales que se esperan de una aplicación o sistema y cómo se relaciona con su entorno (usuarios u otras aplicaciones). (29)

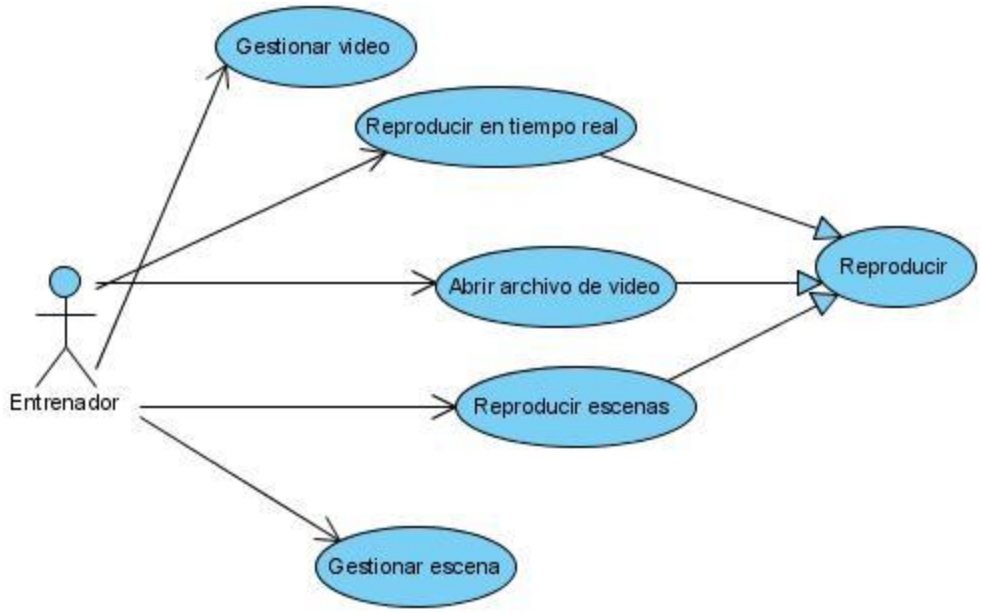

Figura 3: Diagrama de CU del sistema.

2.3.6 Descripción de los casos de uso del sistema.

CU-1.	Reproducir (caso de uso generalizado).
Actor	Entrenador.
Descripción	El caso de uso le permite al entrenador ver la reproducción en tiempo real directamente desde una cámara de video, buscar un archivo de video para reproducirlo y reproducir hasta 4 escenas. Su objetivo principal es permitirle al entrenador que pueda manejar toda la información visual referente a un partido de voleibol.
Referencia	RF3, RF3.1, RF3.2, RF4, RF4.1, RF4.2, RF5.

Tabla 2: Breve descripción del CU Reproducir.

CU-2.	Reproducir en tiempo real (caso de uso especializado).
Actor	Entrenador.
Descripción	El caso de uso le permite al entrenador ver la reproducción en tiempo real directamente desde una cámara de video. Su objetivo principal es permitirle al entrenador que esté actualizado con lo que está pasando instantáneamente en el juego y a la vez poder delimitar las escenas de interés.

Referencia	RF5.

Tabla 3: Breve descripción del CU Reproducir en tiempo real.

CU-3.	Abrir archivo de video (caso de uso especializado).
Actor	Entrenador.
Descripción	El objetivo principal de este caso de uso es que el entrenador pueda analizar videos que no hayan sido grabados y almacenados a través del sistema. Le permite al usuario importar un video desde el directorio físico y reproducirlo.
Referencia	RF4, RF4.1, RF4.2.

Tabla 4: Breve descripción del CU Abrir archivo de video.

CU-4.	Reproducir escenas (caso de uso especializado).
Actor	Entrenador.
Descripción	El caso de uso tiene como objetivo que el entrenador tenga la opción de reproducir hasta cuatro escenas al mismo tiempo para poder hacer comparaciones entre ellas y hacer un estudio más minucioso de los videos de juego. El entrenador puede reproducir las escenas del mismo tipo o de tipos diferentes.
Referencia	RF3, RF3.1, RF3.2.

Tabla 5: Breve descripción del CU Reproducir escena.

CU-5.	Gestionar video.
Actor	Entrenador.
Descripción	El caso de uso tiene como objetivo que el entrenador tenga la opción de guardar un video, mostrar el listado de videos guardados y eliminar un video si lo estima conveniente.
Referencia	RF1, RF1.1, RF1.2, RF1.3.

Tabla 6: Breve descripción del CU Gestionar video.

CU-6.	Gestionar escena.
Actor	Entrenador.

Descripción	El caso de uso tiene como objetivo que el entrenador tenga la opción de delimitar las escenas que le interesen, guardar las escenas una vez delimitadas y mostrar el listado de las escenas guardadas. El entrenador puede reproducir las escenas del mismo tipo o de tipos diferentes.
Referencia	RF2, RF2.1, RF2.2, RF2.3, RF2.4.

Tabla 7: Breve descripción del CU Gestionar escena.

2.3.7 Casos de uso expandidos.

Caso de uso		
CU-2.	Reproducir en tiempo real (caso de uso especializado).	
Propósito	El caso de uso le permite al entrenador ver la reproducción en tiempo real directamente desde una cámara digital.	
Actores: Entrenador.		
Resumen: Su objetivo principal es permitirle al entrenador que esté actualizado con lo que está pasando instantáneamente en el juego y a la vez poder delimitar las escenas de interés.		
Referencias	RF5.	
Acción del actor		**Respuesta del sistema**
1- El actor selecciona la opción “Capturar”.		2- El sistema verifica que haya alguna cámara digital conectada. 3- El sistema reproduce la imagen tomada por la cámara digital.
Flujo alternativo de los eventos		
Acción del actor		**Respuesta del sistema**
		3- Si no existe cámara digital conectada, el sistema muestra un mensaje de error “Se produjo un error al reproducir desde cámara.”, informando al actor.
Prioridad: Crítico.		

Tabla 8: Descripción textual del CU Reproducir en tiempo real.

Caso de uso	
CU-3.	Abrir archivo de video (caso de uso especializado).
Propósito	El objetivo principal de este caso de uso es que el entrenador pueda analizar videos que no hayan sido grabados y almacenados a través del sistema.
Actores: Entrenador.	
Resumen Le permite al usuario importar un video desde un directorio físico y reproducirlo, el entrenador puede hacerle el mismo manejo a este video que a los videos que han sido guardados por el sistema.	
Referencias	RF4, RF4.1, RF4.2.

Acción del actor	**Respuesta del sistema**
1- El actor selecciona la opción “Abrir” disponible en el menú principal. 3- El actor selecciona el video que desea abrir.	2- El sistema muestra un cuadro de diálogo para que el entrenador busque el video que desea abrir. 4- El sistema comprueba si el video seleccionado se encuentra en el espacio de trabajo definido por el actor. Si el video está en el espacio de trabajo el sistema reproduce el video seleccionado.
Flujo alternativo de los eventos	
Acción del actor	**Respuesta del sistema**
	4- Si el video seleccionado no se encuentra en el espacio de trabajo definido por el actor se muestra el mensaje “El archivo (dirección del archivo) no se encuentra en su espacio de trabajo. ¿Desea copiarlo para su espacio de trabajo ahora?”
5- El actor selecciona “Sí”.	6- El sistema copia el archivo para el espacio de trabajo del actor y reproduce el video seleccionado.
5- El actor selecciona “No”.	6- El sistema no copia el archivo y lo reproduce.

Prioridad: Crítico.

Tabla 9: Descripción textual del CU Abrir archivo de video.

Caso de uso		
CU-4.	Reproducir escenas (caso de uso especializado).	
Propósito	El caso de uso tiene como objetivo que el entrenador pueda consultar las escenas que más le interesan para poder realizar un estudio de las mismas.	
Actores: Entrenador.		
Resumen: El caso de uso tiene como objetivo que el entrenador tenga la opción de reproducir hasta cuatro escenas al mismo tiempo para poder hacer comparaciones entre ellas y hacer un estudio más minucioso de los videos de juego. El entrenador puede reproducir las escenas del mismo tipo o de tipos diferentes.		
Referencias	RF3, RF3.1, RF3.2.	
Acción del actor		**Respuesta del sistema**
1- El actor selecciona la escena que desea reproducir y oprime el botón “Reproducir”.		2- El sistema reproduce la escena seleccionada.
3- El actor selecciona la segunda escena que desea reproducir y oprime el botón “Reproducir”.		4- El sistema reproduce la segunda escena seleccionada.
5- El actor selecciona la tercera escena que desea reproducir y oprime el botón “Reproducir”.		6- El sistema reproduce la tercera escena seleccionada.
7- El actor selecciona la cuarta escena que desea reproducir y oprime el botón “Reproducir”.		8- El sistema reproduce la cuarta escena seleccionada.
Flujo alternativo de los eventos		
Acción del actor		**Respuesta del sistema**

9- El actor selecciona la quinta escena que desea reproducir y oprime el botón "Reproducir".	2- El sistema muestra un mensaje de error: "Sólo es posible reproducir 4 escenas".
Prioridad: Crítico.	

Tabla 10: Descripción textual del CU Reproducir escenas.

Caso de uso		
CU-5.	Gestionar video.	
Propósito	El caso de uso tiene como objetivo que el entrenador tenga la opción de realizar la operación que más estime conveniente sobre los videos guardados.	
Actores: Entrenador.		
Resumen: El entrenador puede realizar las siguientes operaciones: guardar un video, mostrar el listado de videos guardados y eliminar un video.		
Referencias	RF1, RF1.1, RF1.2, RF1.3.	
Sección: Guardar video.		
Acción del actor		**Respuesta del sistema**
1- El actor presiona el botón "Grabar". 3- El actor presiona el botón "Detener grabación".		2- El sistema comienza a guardar el video que está captando la cámara. 4- El sistema finaliza la grabación y guarda el archivo de video.
Flujo alternativo de los eventos		
Acción del actor		**Respuesta del sistema**
3- El actor desconecta la cámara mientras se realiza la grabación.		4- El sistema guarda el video hasta el momento en que la cámara fue desconectada y muestra un mensaje informando al actor que se ha desconectado la cámara: "La cámara ha sido desconectada".
Sección: Mostrar videos guardados.		
Acción del actor		**Respuesta del sistema**

1- El actor selecciona la opción "Ver estudios".	2- El sistema muestra una interfaz con la lista de los videos estudiados.
Sección: Eliminar video.	
Acción del actor	**Respuesta del sistema**
1- El actor selecciona el video que desea eliminar y presiona el botón "Eliminar". 3- El actor presiona el botón "Sí".	2- El sistema muestra el mensaje de confirmación: "Si elimina el video se eliminarán también todas las escenas asociadas a este. ¿Confirma que desea eliminar el video?" 4- El sistema elimina el video y todas las escenas asociadas.
Flujo alternativo de los eventos	
Acción del actor	**Respuesta del sistema**
3- El actor presiona el botón "No".	4- El sistema cancela la acción.
Prioridad: Crítico.	

Tabla 11: Descripción textual del CU Gestionar video.

Caso de uso	
CU-6.	Gestionar escenas.
Propósito	El caso de uso tiene como objetivo que el entrenador pueda realizar las acciones necesarias sobre una escena.
Actores: Entrenador.	
Resumen: El entrenador tiene la opción de delimitar la escena que le interesa consultar, guardar dicha escena, consultar todas las escenas en una misma lista o puede definir el criterio de selección y eliminar la escena que ya no le interese.	
Referencias	RF2, RF2.1, RF2.2, RF2.3, RF2.4.
Sección: Delimitar escenas.	
Acción del actor	**Respuesta del sistema**
1- El actor presiona el botón "Tomar escenas". 3- El actor presiona el botón para	2- El sistema habilita los botones correspondientes a cada uno de los tipos de escenas.

delimitar el inicio de un tipo de escena. 5- El actor presiona el mismo botón para delimitar el final de la escena.	4- El sistema marca el inicio del intervalo de tiempo de duración de la escena. 6- El sistema marca el fin del intervalo de tiempo de duración de la escena.
Flujo alternativo de los eventos	
Acción del actor	**Respuesta del sistema**
5- El actor presiona un botón diferente al que había presionado inicialmente para delimitar el fin de la escena.	6- El sistema cancela la escena que había comenzado a marcar y comienza a marcar una nueva escena del tipo perteneciente al botón que presionó último.
Sección: Mostrar listado de escenas.	
Acción del actor	**Respuesta del sistema**
1- El actor selecciona en la lista de videos estudiados el video del que desea ver las escenas.	2- El sistema muestra una lista con todas las escenas relacionadas al video que se seleccionó.
3- El actor selecciona el tipo de escena que desea consultar, que puede ser saque bloqueo, recibo, pase, ataque, defensa, K1 o k2.	4- El sistema muestra las escenas del tipo especificado vinculadas al video seleccionado.
Sección: Eliminar escenas.	
Acción del actor	**Respuesta del sistema**
1- El actor selecciona la escena que desea eliminar y presiona el botón “Eliminar”. 3- El actor presiona el botón “Sí”.	2- El sistema muestra el mensaje de confirmación: “¿Desea eliminar la escena seleccionada?” 4- El sistema elimina la escena seleccionada.
Flujo alternativo de los eventos	
Acción del actor	**Respuesta del sistema**
3- El actor presiona el botón “No”.	4- El sistema cancela la acción.
Prioridad: Crítico.	

Tabla 12: Descripción textual del CU Gestionar escenas.

2.4 Conclusiones.

En este capítulo se concibió la descripción de la propuesta de solución, en la que se incluye la elaboración del modelo de dominio, donde se describen los principales conceptos existentes y se modela la relación entre ellos mediante un diagrama de clases, evidenciando la interacción de los mismos y brindando la posibilidad de mostrar los eventos que tienen lugar en torno al problema que se presenta para lograr así una mejor comprensión de la aplicación a desarrollar. También fue realizada la captura del listado de requerimientos que posteriormente fueron modelados en términos de casos de uso permitiendo la confección del diagrama de casos de uso del sistema, modelo que representa la relación existente entre actor y sistema, incluyendo de forma implícita el cumplimiento de los requerimientos funcionales y no funcionales para obtener los resultados esperados, viabilizando así una vista global del funcionamiento del sistema.

CAPÍTULO 3. DISEÑO DEL SISTEMA.

3.1 Introducción.

En el presente capítulo se exponen los artefactos generados en el flujo de trabajo de diseño, se podrán encontrar los diagramas de clases del diseño y la descripción de cada una de las clases con sus respectivos atributos y funcionalidades. Este capítulo finaliza con el diagrama entidad relación correspondiente al diseño de la base de datos de la aplicación y con la descripción de cada una de sus tablas.

3.2 Diseño.

Consiste en refinar la definición de los elementos de diseño, incluyendo cápsulas y protocolos, trabajando en describir a detalle cómo los elementos de diseño realizan los comportamientos requeridos de ellos. Se actualizan además la realización de los casos de uso basados en los nuevos elementos de diseño identificados. Incluye identificar cuáles clases serán persistentes, diseñar estructuras de datos adecuadas para almacenar estas clases y definir estrategias para el almacenamiento y recuperación de estas clases persistentes.

3.2.1 Diagramas de interacción.

Los diagramas de interacción se utilizan para modelar los aspectos dinámicos de un sistema, lo que conlleva modelar instancias concretas o prototípicas de clases interfaces, componentes y nodos, junto con los mensajes enviados entre ellos, todo en el contexto de un escenario que ilustra un comportamiento.

Cada caso de uso es una telaraña de escenarios primarios y secundarios. Por tanto, para un caso de uso se puede definir diferentes escenarios que ayudan a la identificación de objetos, clases e interacciones entre objetos necesarios para llevar a cabo la parte de funcionalidad que especifica el caso de uso.

Un diagrama de secuencia es un diagrama de interacción que destaca la ordenación temporal de los mensajes; muestra las interacciones entre objetos ordenadas en secuencia; ilustra los objetos que se encuentran en el escenario y la secuencia de mensajes intercambiados entre los objetos para llevar a cabo la funcionalidad descrita por el escenario. (29)

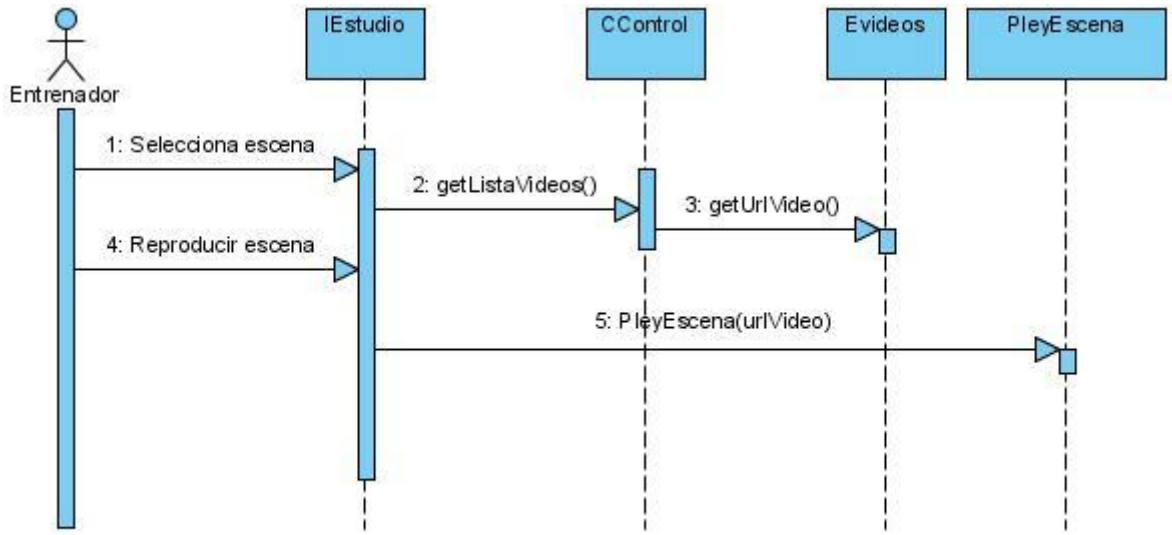

Figura 4: Diagrama de secuencia del CU Reproducir Escena.

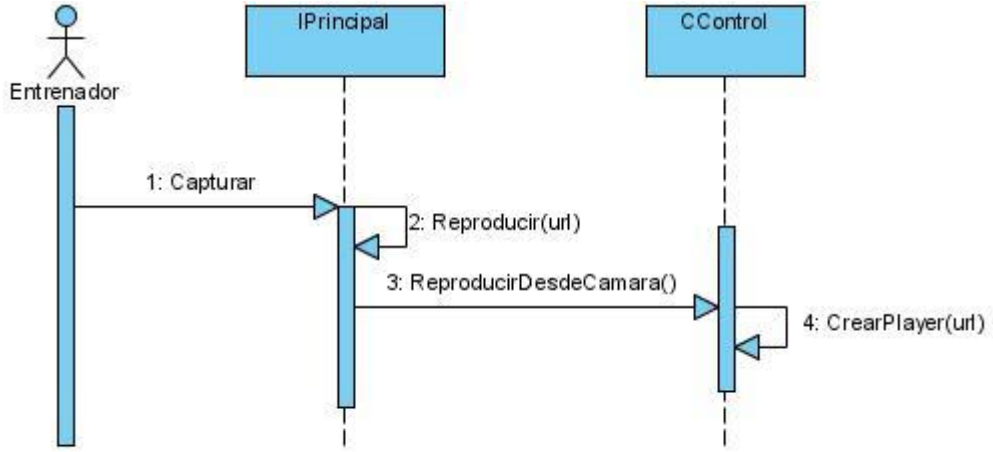

Figura 5: Diagrama de secuencia del CU Reproducir en Tiempo Real.

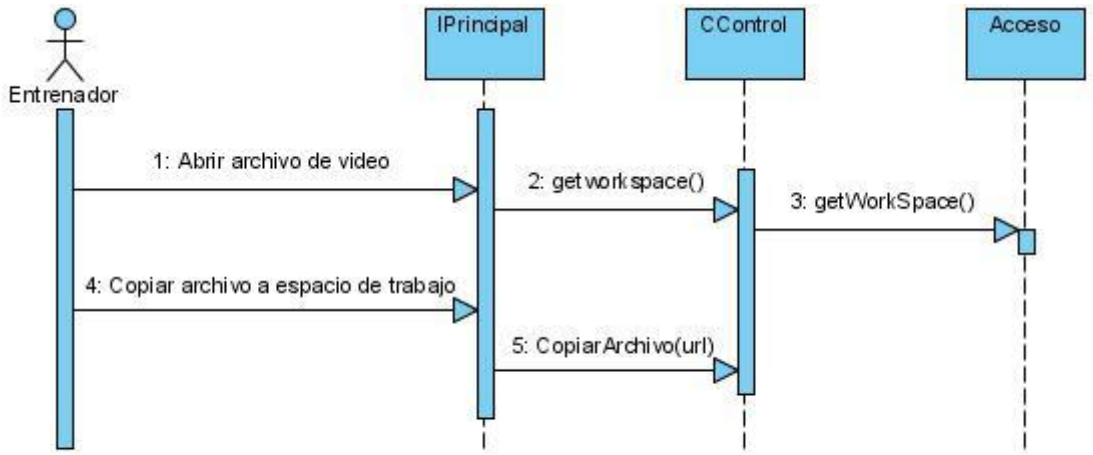

Figura 6: Diagrama de secuencia del CU Abrir Archivo de Video.

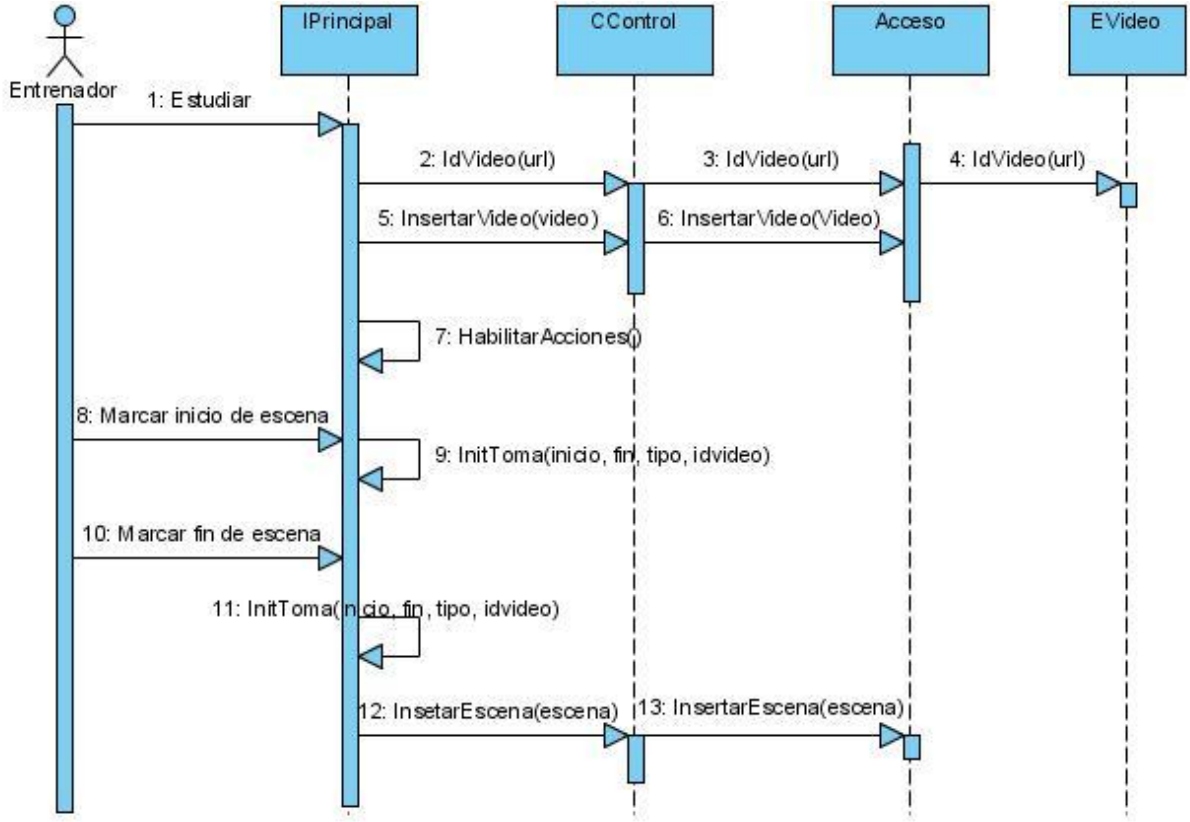

Figura 7: Diagrama de secuencia del CU Gestionar Escenas sección Delimitar Escenas.

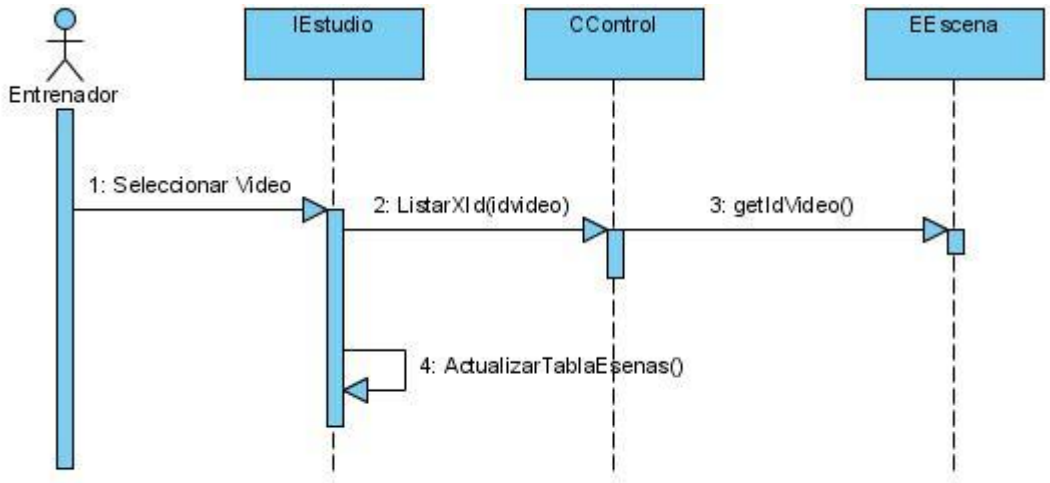

Figura 8: Diagrama de secuencia del CU Gestionar Escenas sección Mostrar Listado de Escenas.

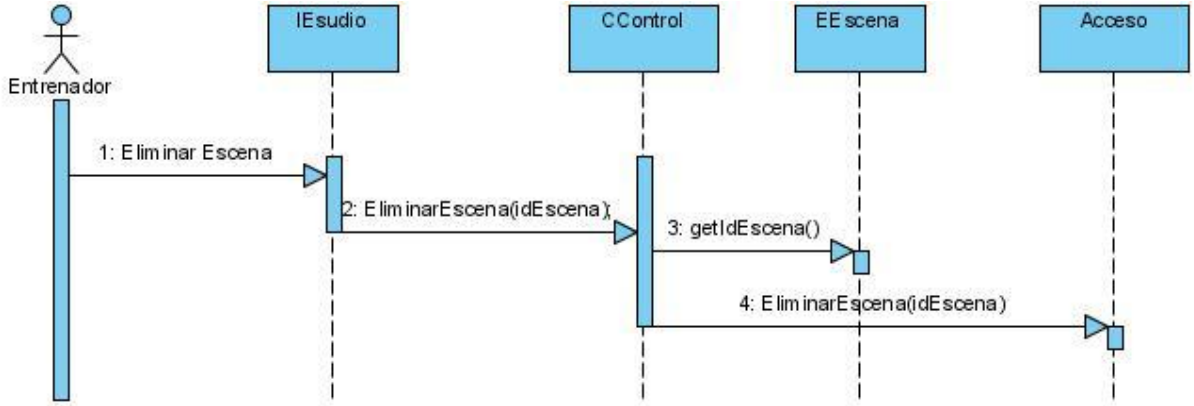

Figura 9: Diagrama de secuencia del CU Gestionar Escenas sección Eliminar Escenas.

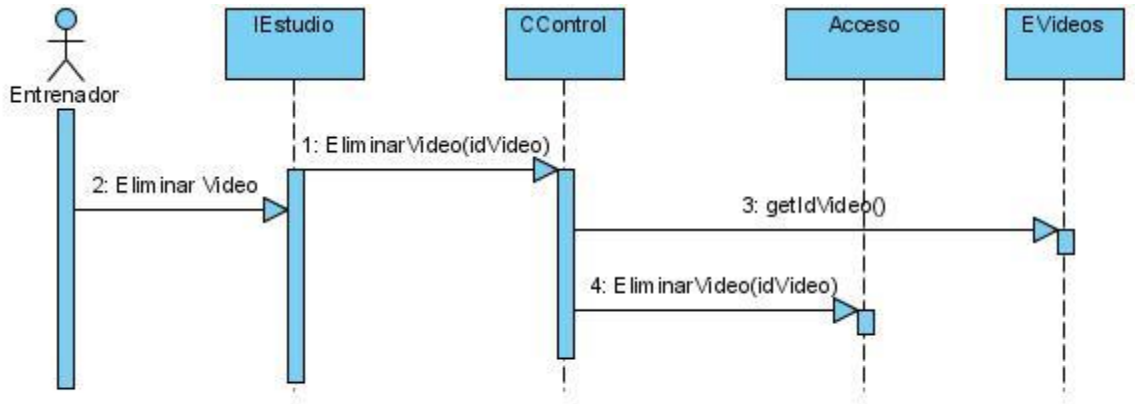

Figura 10: Diagrama de secuencia del CU Gestionar Video sección Eliminar Video.

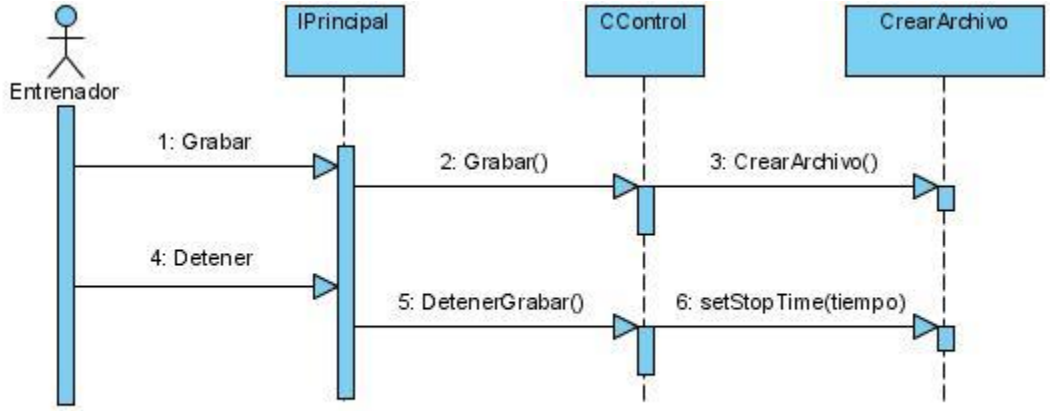

Figura 11: Diagrama de secuencia del CU Gestionar Video sección Guardar Video.

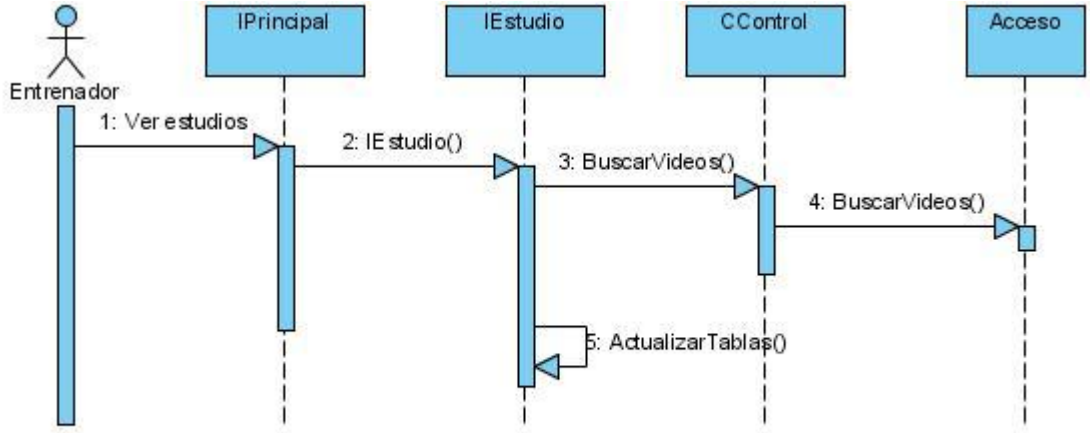

Figura 12: Diagrama de secuencia del CU Gestionar Video sección Mostrar Videos Guardados.

3.2.2 Diagramas de Clases del diseño.

El diagrama de clases del diseño es una representación más concreta que el diagrama de clases del análisis; describe gráficamente las especificaciones de las clases de *software* y de las interfaces en una aplicación; representa la parte estática del sistema. Normalmente contiene la siguiente información:

- Clases, asociaciones y atributos.
- Interfaces, con sus operaciones y constantes.
- Métodos.
- Información sobre los tipos de los atributos.
- Navegabilidad.
- Dependencias.

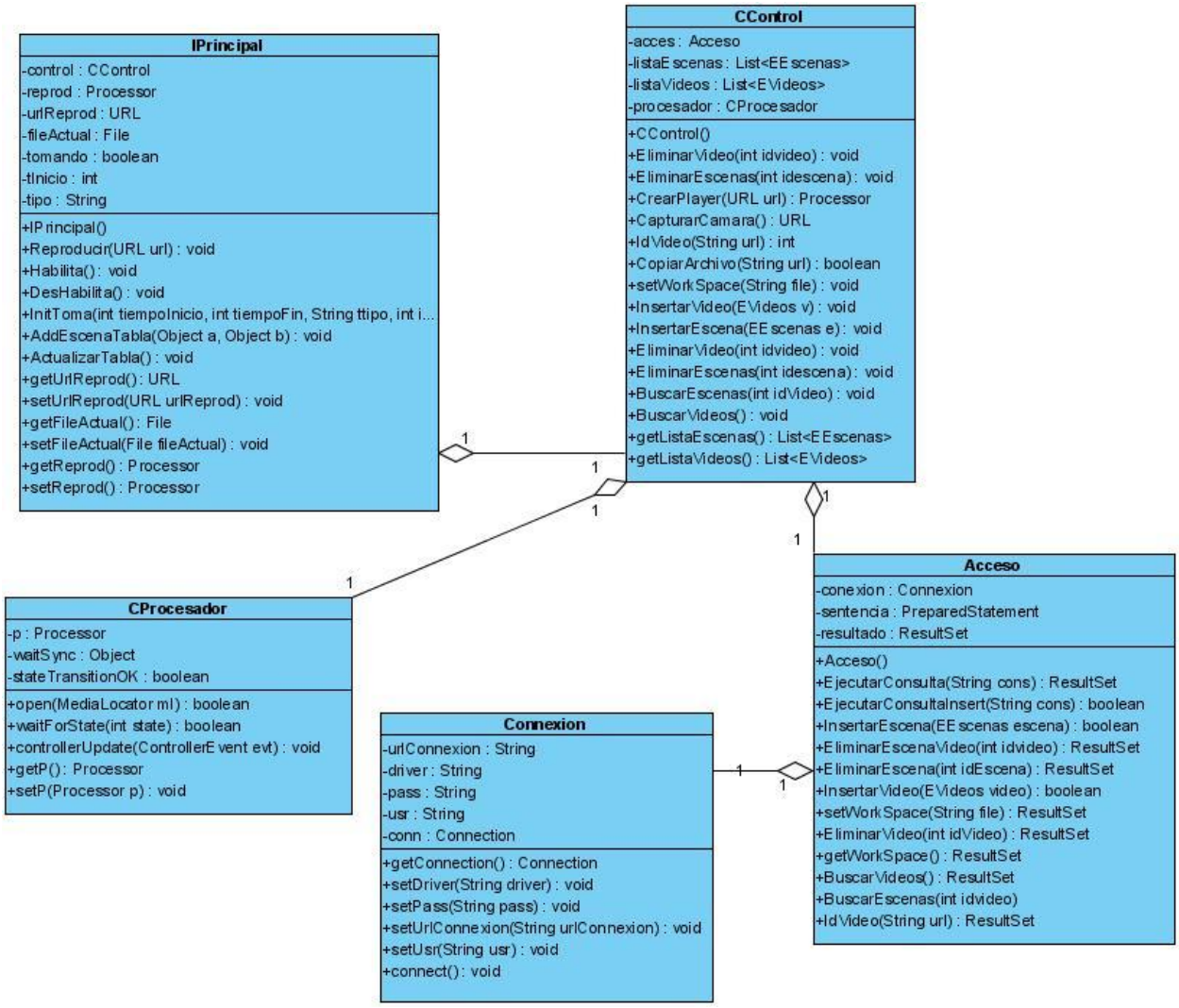

Figura 13: Diagrama de clases del CU Abrir Archivo de Video.

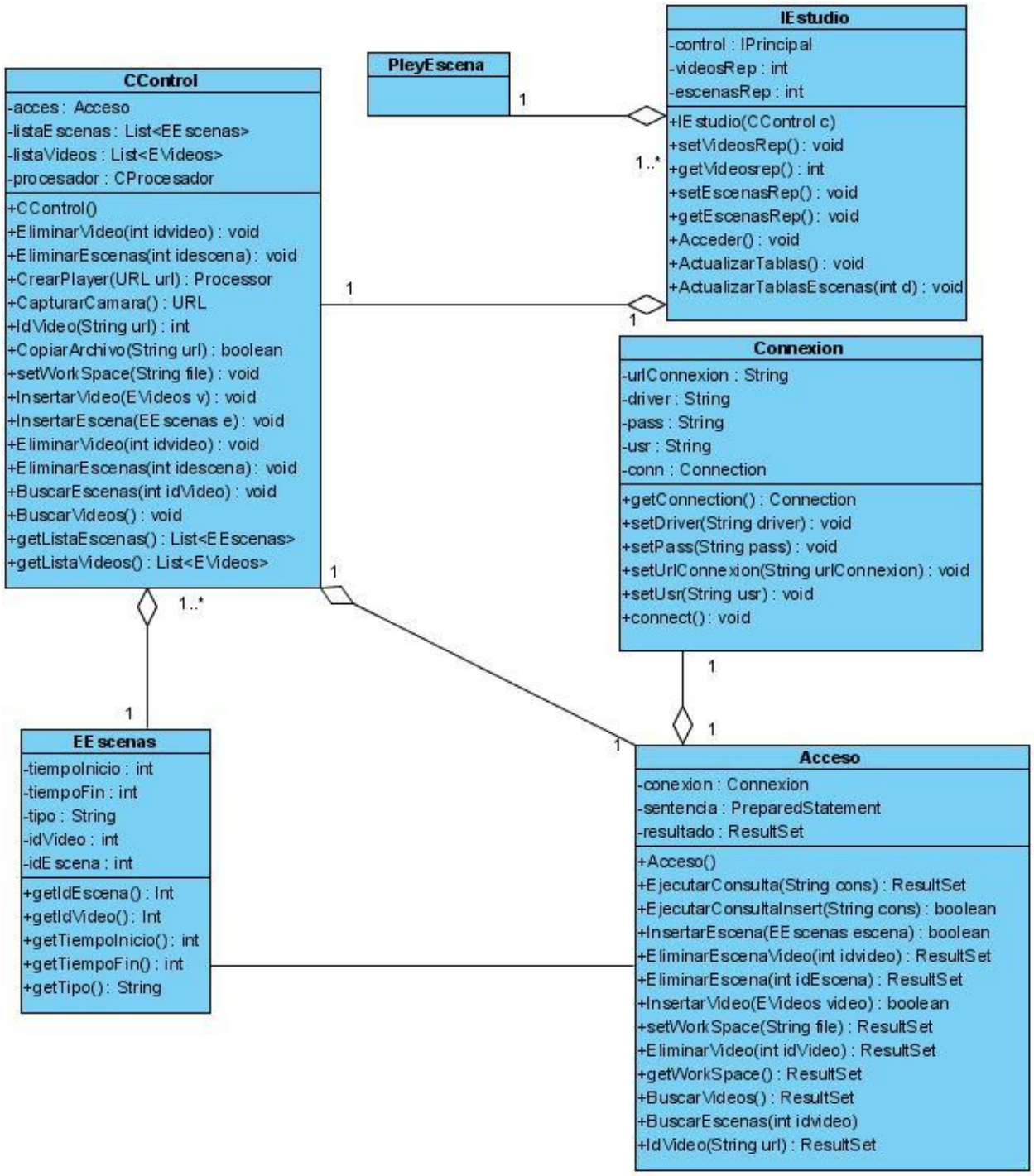

Figura 14: Diagrama de clases del CU Reproducir Escena.

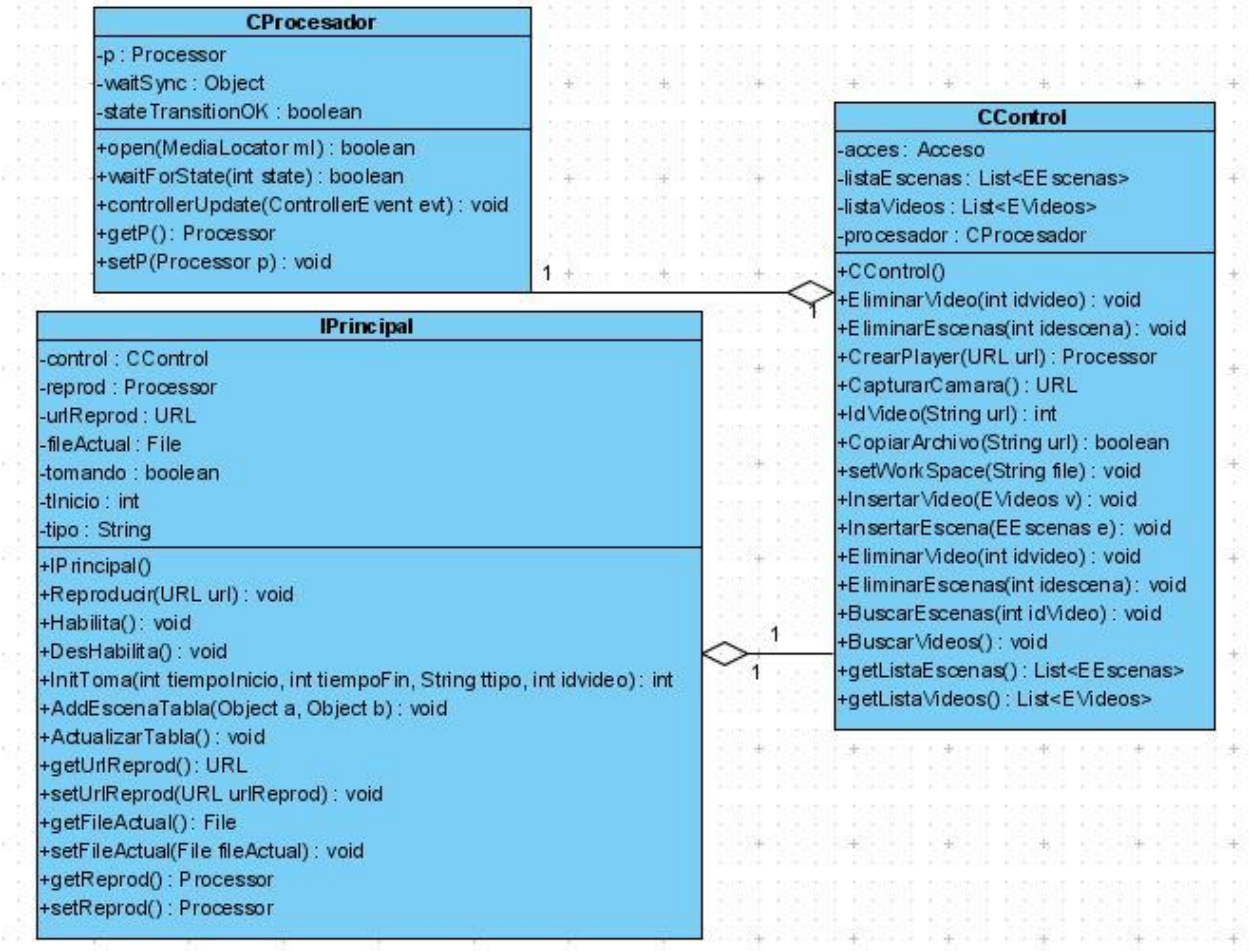

Figura 15: Diagrama de clases del CU Reproducir en Tiempo Real.

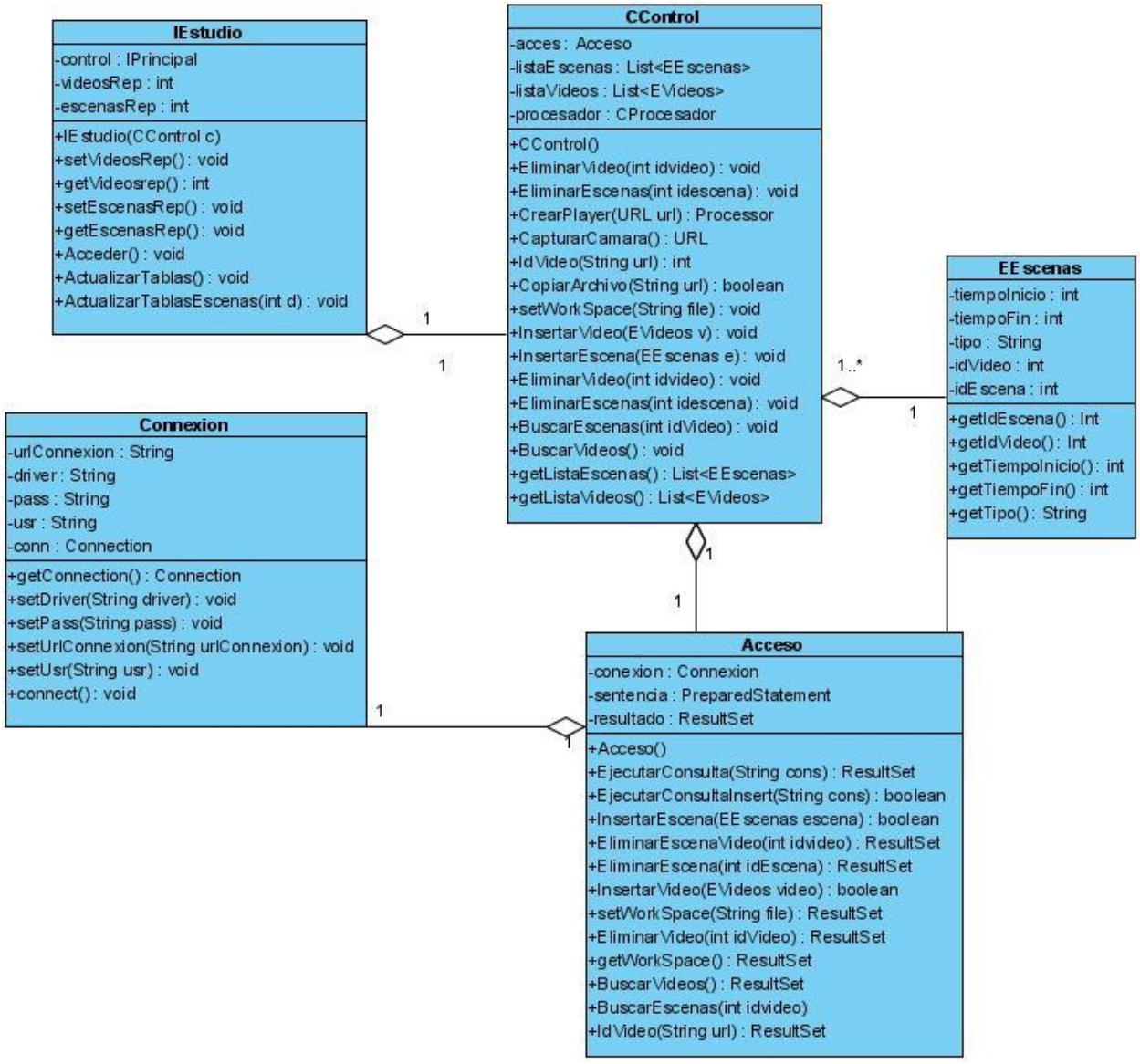

Figura 16: Diagrama de clases del CU Gestionar Escena.

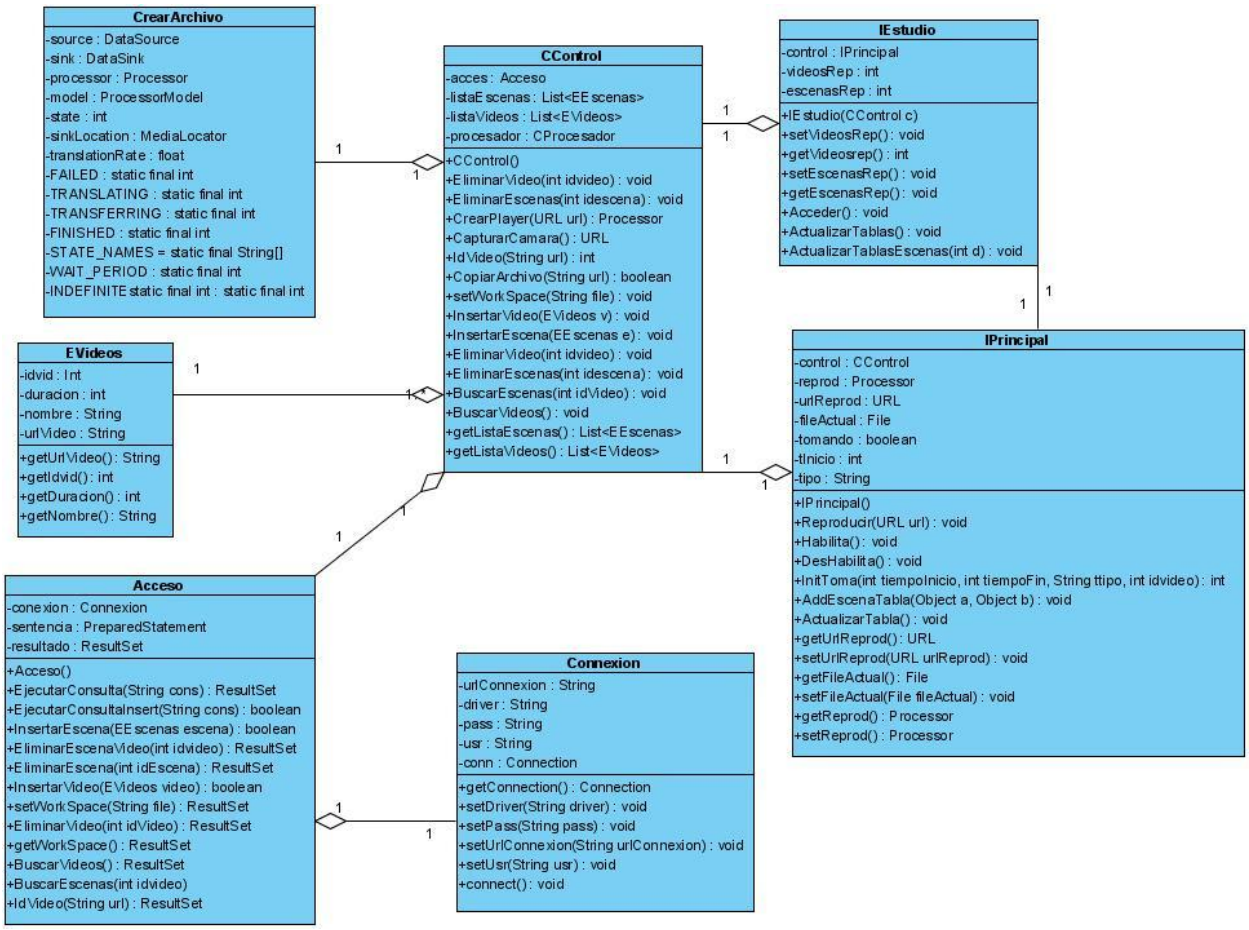

Figura 17: Diagrama de clases del CU Gestionar Video.

3.2.3 Descripción de las Clases.

Las clases se especifican utilizando la sintaxis del lenguaje de programación elegido; se especifica la visibilidad de los atributos y operaciones que estas contienen (*public*, *protected*, *private*)*;* se implementan las relaciones que tienen un significado en la implementación; se especifican los métodos que tienen correspondencia directa con los métodos en la implementación; puede realizar interfaces si tiene sentido en el lenguaje de programación y a menudo se utilizan estereotipos que se corresponden con una construcción de lenguaje de programación.

Nombre: IPrincipal	
Tipo de clase: Interfaz.	
Atributo	**Tipo**
control	CControl
reprod	Processor
urlReprod	URL
fileActual	File
tomando	boolean
tInicio	int
tipo	String
Para cada responsabilidad	
Nombre	**Descripción**
Reproducir(URL url): void	Crea los componentes del reproductor del video cuya URL se pasa por parámetro y lo reproduce.
Habilita():void	Habilita los botones para tomar las escenas.
DesHabilita():void	Deshabilita los botones para tomar las escenas.
InitToma(int tiempoInicio, int tiempoFin,String ttipo, int idvideo): int	Crea una escena y la guarda.
AddEscenaTabla(Object a, Object b): void	Agrega una escena a la tabla de tomas de la interfaz y la actualiza.
ActualizarTabla():void	Inicializa la tabla de tomas de la interfaz con valores predeterminados.
getUrlReprod():URL	Devuelve la URL del video que se está reproduciendo en el momento.
setUrlReprod(URL urlReprod): void	Asigna un nuevo valor a la URL del reproductor.
getFileActual():File	Devuelve en una variable File la dirección física del video que está actualmente en reproducción.
setFileActual(File fileActual): void	Modifica el valor de la variable File que guarda la dirección física del video en reproducción.
getReprod():Processor	Devuelve el objeto Processor.
setReprod():Processor	Modifica o cambia el valor de la variable reprod.

reprod): void	

Tabla 13: Descripción de la clase IPrincipal.

Nombre: IEstudio.	
Tipo de clase: Interfaz.	
Atributo	**Tipo**
control	CControl
videosRep	int
escenasRep	int
Para cada responsabilidad.	
Nombre	**Descripción**
setVideosRep():void	Cambia o modifica el contador del número de videos que están en reproducción.
getVideosrep():int	Devuelve el contador de videos que están en reproducción.
setEscenasRep():void	Cambia o modifica el contador de las escenas que están en reproducción.
getEscenasRep():void	Devuelve el contador de escenas que están en reproducción.
Acceder():void	Actualiza las listas de escenas y videos de la clase CControl.
ActualizarTablas():void	Llena la tabla de videos a partir de la lista de videos de la clase CControl.
ActualizarTablasEscenas(int d): void	Llena la tabla de escenas a partir de la lista de escenas de la clase Ccontrol.

Tabla 14: Descripción de la clase IEstudio.

Nombre: CControl	
Tipo de clase: Controladora.	
Atributo	**Tipo**
acces	Acceso
listaEscenas	List<EEscenas>
listaVideos	List<EVideos>

procesador	CProcesador
Para cada responsabilidad	
Nombre	**Descripción**
EliminarVideo(int idvideo): void	Elimina un video dado su identificador en la base de datos.
EliminarEscenas(int idescena): void	Elimina una escena dado su identificador en la base de datos.
CrearPlayer(URL url): Processor	Crea y devuelve el objeto que maneja el video.
CapturarCamara():URL	Devuelve la dirección URL de la cámara digital conectada a la computadora.
IdVideo(String url): int	Devuelve el identificador de un video a partir de su URL.
CopiarArchivo(String url): boolean	Copia un archivo de video al espacio de trabajo.
setWorkSpace(String file): void	Modifica la dirección del espacio de trabajo en la base de datos.
InsertarVideo(EVideos v): void	Inserta un video nuevo a la base de datos.
InsertarEscena(EEscenas e): void	Inserta una escena nueva a la base de datos.
EliminarVideo(int idvideo): void	Elimina un video de la base de datos a partir de su identificador.
EliminarEscenas(int idescena): void	Elimina una escena de la base de datos a partir de su identificador.
BuscarEscenas(int idVideo): void	Actualiza la lista de escenas con las escenas correspondientes al video cuyo identificador se pasa por parámetro.
BuscarVideos():void	Actualiza la lista de videos con los videos existentes en la base de datos.
getListaEscenas():List<EEscenas>	Devuelve la lista de escenas.
getListaVideos():List<EVideos>	Devuelve la lista de videos.

Tabla 15: Descripción de la clase CControl.

Nombre: EVideos	
Tipo de clase: Entidad	
Atributo	**Tipo**
idvid	int
duracion	int
nombre	String
urlVideo	String
Para cada responsabilidad:	
Nombre	**Descripción**
getUrlVideo():String	Devuelve la dirección física del video.
getIdvid():int	Devuelve el identificador del video.
getDuracion(): int	Devuelve el valor de la duración del video.
getNombre():String	Devuelve el nombre del video.

Tabla 16: Descripción de la clase entidad EVideo.

Nombre: EEscenas	
Tipo de clase: Entidad	
Atributo	**Tipo**
tiempoInicio	int
tiempoFin	int
tipo	String
idVideo	int
idEscena	int
Para cada responsabilidad:	
Nombre	**Descripción**
getIdEscena():int	Devuelve el identificador de la escena.
getIdVideo():int	Devuelve el identificador del video al que pertenece la escena.
getTiempoInicio():int	Devuelve el valor del momento en que comienza la escena.
getTiempoFin():int	Retorna el valor del tiempo en que termina de reproducirse la escena.
getTipo():String	Devuelve el tipo de la escena.

Tabla 17: Descripción de la clase entidad EEscenas.

Nombre: CProcesador	
Tipo de clase: Controladora	
Atributo	**Tipo**
p	Processor
waitSync	Object
stateTransitionOK	boolean
Responsabilidad: Esta clase tiene la responsabilidad de crear el objeto *Processor* que aplica el *plugin* y cambia el formato del video.	

Tabla 18: Descripción de la clase CProcesador.

Nombre: CrearArchivo	
Tipo de clase: Controladora	
Atributo	**Tipo**
source	DataSource
sink	DataSink
processor	Processor
model	ProcessorModel
state	int
sinkLocation	MediaLocator
translationRate	float
FAILED	static final int
TRANSLATING	static final int
TRANSFERRING	static final int
FINISHED	static final int
STATE_NAMES	static final String[]
WAIT_PERIOD	static final int
INDEFINITE	static final int
Responsabilidad: Guardar un archivo de video con las imágenes capturadas por la cámara.	

Tabla 19: Descripción de la clase CrearArchivo.

Nombre: Acceso

Tipo de clase: Auxiliar	
Atributo	**Tipo**
conexion	Connexion
sentencia	PreparedStatement
resultado	ResultSet
Para cada responsabilidad:	
Nombre	**Descripción**
EjecutarConsulta(String cons): ResultSet	Ejecuta la consulta cons en la base de datos.
EjecutarConsultaInsert(String cons): boolean	Ejecuta la consulta cons en la base de datos.
InsertarEscena(EEscenas escena): boolean	Inserta una escena a la base de datos.
EliminarEscenaVideo(int idvideo): ResultSet	Elimina todas las escenas correspondientes al video cuyo identificador es pasado por parámetro.
EliminarEscena(int idEscena): ResultSet	Elimina la escena correspondiente al identificador que es pasado por parámetro.
InsertarVideo(EVideos video): boolean	Inserta un video a la base de datos.
EliminarVideo(int idVideo): ResultSet	Elimina el video correspondiente al identificador pasado por parámetro.
setWorkSpace(String file): ResultSet	Modifica la dirección del espacio de trabajo en la base de datos.
getWorkSpace():ResultSet	Devuelve la dirección del espacio de trabajo.
BuscarVideos():ResultSet	Devuelve todos los videos de la base de datos.
BuscarEscenas(int idvideo): ResultSet	Devuelve las escenas correspondientes al video cuyo identificador es pasado por parámetro.
IdVideo(String url): ResultSet	Devuelve el identificador del video cuya URL es pasada por parámetro.

Tabla 20: Descripción de la clase Acceso.

Nombre: Connexion
Tipo de clase: Auxiliar

Atributo	**Tipo**
urlConnexion	String
driver	String
usr	String
pass	String
conn	Connection
Para cada responsabilidad:	
Nombre	**Descripción**
getConnection():Connection	
setDriver(String driver): void	Cambia o modifica el valor del driver de la conexión.
setPass(String pass): void	Cambia o modifica el valor de la contraseña.
setUrlConnexion(String urlConnexion): void	Cambia o modifica el valor de la URL de la conexión.
setUsr(String usr): void	Cambia o modifica el usuario.
connect():void	Realiza la conexión a la base de datos, con el usuario, contraseña, driver y URL predefinidos.

Tabla 21: Descripción de la clase Connexion.

Nombre: PlayVideo
Tipo de clase: Interfaz
Responsabilidad: Crea una interfaz que reproduce el video cuya URL es pasada por parámetro.

Tabla 22: Descripción de la clase PlayVideo.

Nombre: PleyEscena
Tipo de clase: Interfaz
Responsabilidad: Crea una interfaz que reproduce la escena perteneciente al video cuya URL es pasada por parámetro.

Tabla 23: Descripción de la clase PleyEscena.

3.2.4 Diseño de la Base de Datos.

Una base de datos es una colección de información perteneciente a un mismo contexto y que está organizada de forma que un programa pueda seleccionar rápidamente los fragmentos de datos que necesite. Una finalidad de la base de datos es eliminar la redundancia o al menos minimizarla.

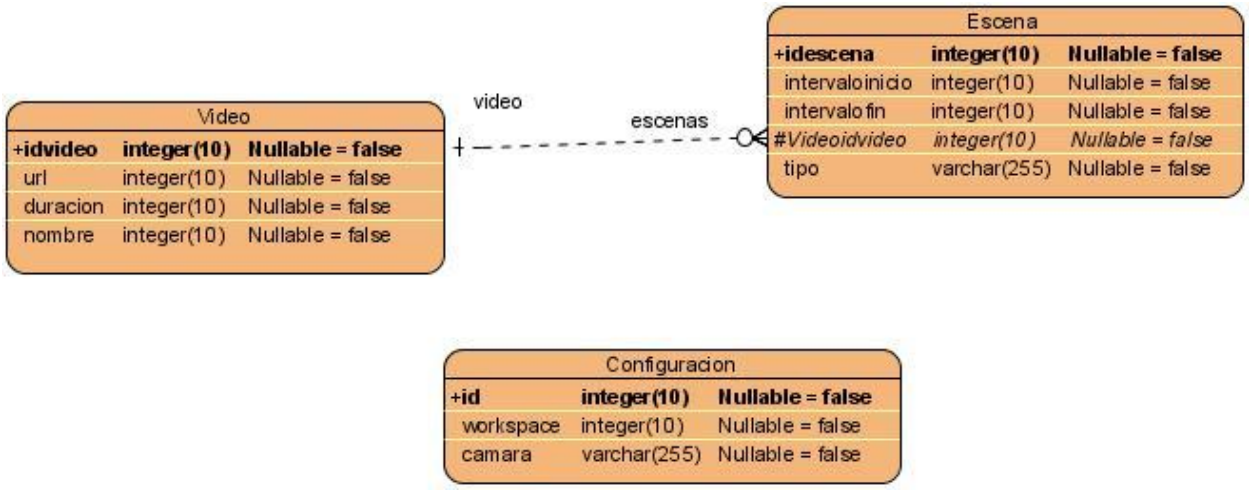

Figura 18: Modelo entidad relación de la base de datos.

3.2.5 Descripción de las Tablas de la Base de Datos.

En las tablas se representan elementos que existen y están bien diferenciados entre sí, que poseen propiedades y entre los cuales se establecen relaciones; son objetos concretos pero también puede ser algo no tangible, como un suceso cualquiera o un concepto abstracto. Cada tabla creada debe tener un nombre único en la Base de Datos.

Nombre: Video		
Descripción: Contiene todos los datos de importancia referentes al video, que son útiles para facilitar el manejo de este.		
Atributo	**Tipo**	**Descripción**
idvideo	Integer	Es el identificador de cada video.
url	Text	Guarda la dirección del video.
nombre	Text	Guarda el nombre del video.
duracion	Integer	Hace referencia a la duración del video.

Tabla 24: Descripción de la tabla Video.

Nombre: Escena		
Descripción: Contiene todos lo datos de la escena de los que son necesarios mantener información.		
Atributo	**Tipo**	**Descripción**
idescena	Integer	Es el identificador de la escena.
idvideo	Integer	Es el identificador del video al que pertenece la escena.
intervaloinicio	Integer	Contiene el tiempo exacto donde comienza la escena.
intervalofin	Integer	Contiene el tiempo exacto donde finaliza la escena.
tipo	Text	Guarda el tipo al que pertenece la escena.

Tabla 25: Descripción de la tabla Escena.

Nombre: Configuración		
Descripción: Tiene el objetivo de guardar los datos de la configuración visual que estuvo abierta últimamente.		
Atributo	**Tipo**	**Descripción**
Id	Integer	Es un identificador de la configuración.
workspace	Varchar	Guarda la dirección del espacio de trabajo.
camara	Varchar	Guarda el nombre de la cámara.

Tabla 26: Descripción de la tabla Configuración.

3.3 Conclusiones.

El desarrollo de este capítulo ha permitido una mejor comprensión del funcionamiento del sistema, del mismo modo las características y restricciones que debe tener el mismo para cumplir con los requerimientos del cliente. El diagrama de clases del análisis contribuyó a un mejor entendimiento de las clases que colaboran para dar respuesta a cada caso de uso de los definidos en el capítulo anterior. Por otra parte el diagrama de clases del diseño sirvió de

antesala a la elaboración del diseño de la base de datos, elemento prioritario para lograr un desarrollo de *software* eficiente. Con los artefactos obtenidos a partir del desarrollo de este flujo de trabajo, se puede pasar al flujo de trabajo de implementación para comenzar la construcción de la solución propuesta.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA.

4.1 Introducción.

En este capítulo se presentarán los artefactos más importantes del flujo de trabajo de implementación, el modelo de despliegue y el modelo de implementación para una mejor descripción de las vistas estáticas del sistema. Se presentan además los tipos de prueba a realizarle al sistema para verificar su correcto funcionamiento con el objetivo de encontrar posibles fallos y corregirlos antes de la liberación del producto.

4.2 Diagramas de Implementación.

Un diagrama de implementación muestra las dependencias entre las partes de código del sistema (diagrama de componentes) o la estructura del sistema en ejecución (diagrama de despliegue): los diagramas de componentes se utilizan para modelar la vista de implementación estática de un sistema, mientras que los diagramas de despliegue se utilizan para modelar la vista de despliegue estática.

4.2.1 Diagrama de Despliegue.

Un diagrama de despliegue muestra las relaciones físicas entre los componentes *hardware* y *software* en el sistema final, es decir, la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes *software* (procesos y objetos que se ejecutan en ellos). Estarán formados por instancias de los componentes *software* que representan manifestaciones del código en tiempo de ejecución (los componentes que sólo sean utilizados en tiempo de compilación deben mostrarse en el diagrama de componentes).

Un diagrama de despliegue es un grafo de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes *software*, objetos, procesos (caso particular de un objeto). (29)

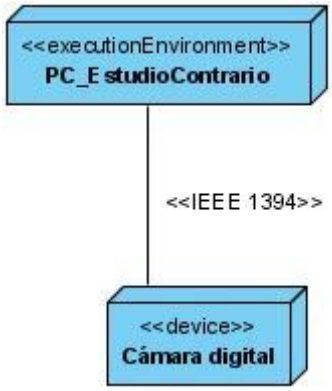

Figura 19: Diagrama de despliegue.

4.2.2 Diagrama de Componentes.

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes *software*, sean estos componentes de código fuente, binarios o ejecutables. Desde el punto de vista del diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del *software*, la reutilización, y las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo. Los elementos de modelado dentro de un diagrama de componentes serán componentes y paquetes. En cuanto a los componentes, sólo aparecen tipos de componentes, ya que las instancias específicas de cada tipo se encuentran en el diagrama de despliegue. (29)

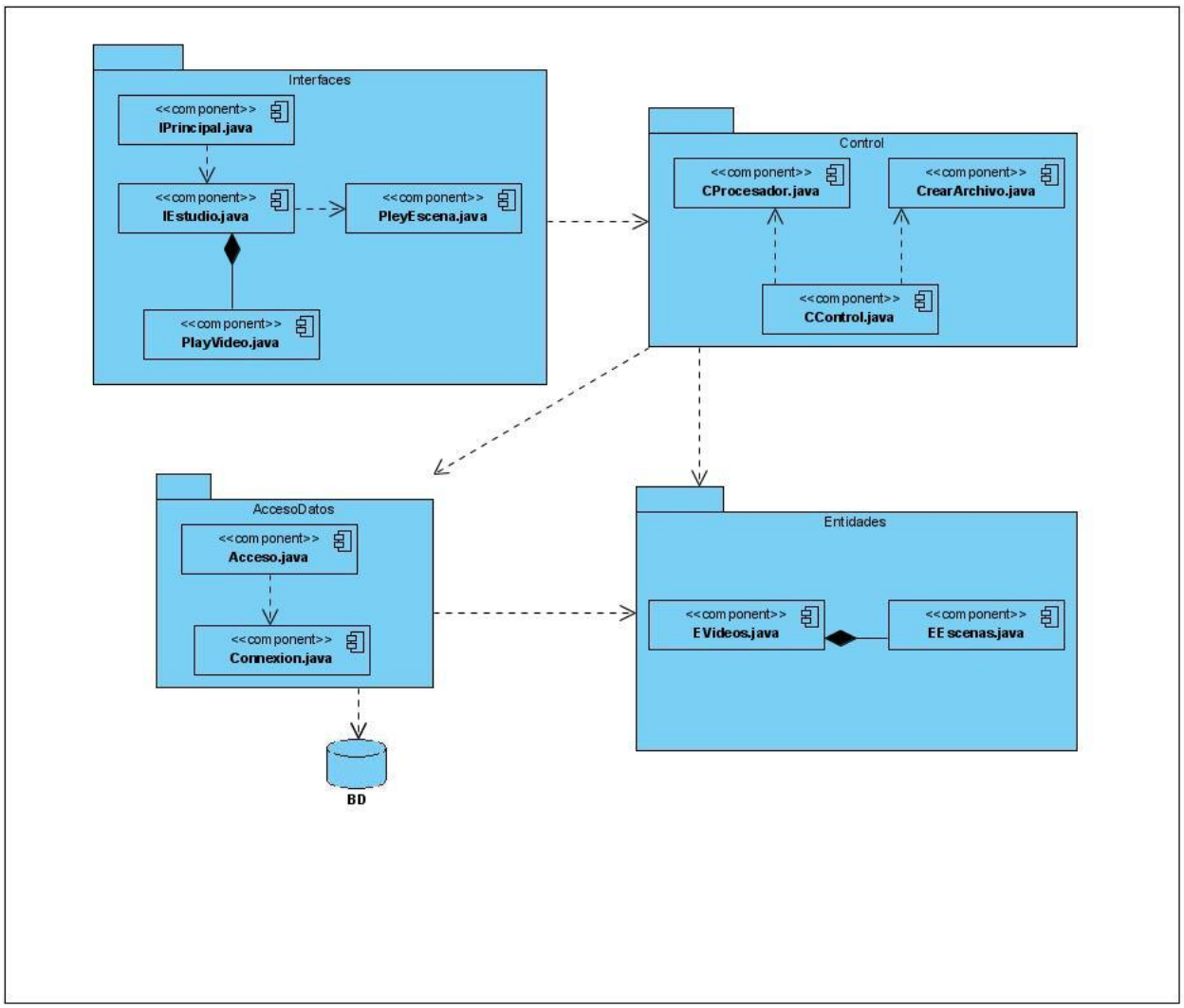

Figura 20: Diagrama de componentes.

4.3 Pruebas de validación del software.

El principal objetivo de realizarle pruebas al *software* es evaluar la calidad del producto que se está desarrollando a través de las diferentes fases por las cuales este pasa, mediante la aplicación de pruebas concretas para validar que las suposiciones hechas en el diseño y los requerimientos se estén cumpliendo satisfactoriamente, se verifica que el producto funcione como se diseñó y que los requerimientos sean satisfechos cabalmente. Esta disciplina brinda soporte para encontrar, documentar y solucionar defectos en la calidad del sistema a las otras disciplinas.

Sus objetivos específicos son:

- Encontrar y documentar defectos en la calidad del *software*.
- Notificar la calidad percibida del *software*.
- Proveer un medio de validación para las suposiciones hechas en el diseño y especificaciones de requerimientos por medio de demostraciones concretas.
- Validar las funciones del producto de *software* según lo diseñado.
- Validar que los requerimientos fueron implementados apropiadamente.

4.3.1 Casos de prueba.

Los casos de prueba son un conjunto de condiciones o variables bajo las cuales el analista determinará si el requisito de una aplicación es parcial o completamente satisfactorio. Definen un conjunto de datos de entradas, condiciones de ejecución y resultados esperados de las pruebas, identificados para hacer una evaluación de los aspectos específicos de un elemento objeto de prueba. Cada caso de prueba está asociado a un escenario de un caso de uso en particular.

Entrada	Resultados	Condiciones
Se selecciona la escena que se desea reproducir y se presiona el botón "Reproducir" y así sucesivamente hasta 4 escenas.	El sistema muestra un reproductor por cada escena seleccionada. Si se seleccionan 5 escenas el sistema muestra el mensaje: "Sólo es posible reproducir 4 escenas".	El video al que pertenece la escena debe estar guardado.

Tabla 27: Caso de prueba reproducir escena.

Entrada	Resultados	Condiciones
Se selecciona la opción "Capturar".	Aparece en pantalla la visualización del video que la cámara está captando.	La cámara digital tiene que estar conectada a la computadora.

	En caso de que la cámara no esté conectada se muestra el mensaje “Se produjo un error al reproducir desde cámara.”	

Tabla 28: Caso de prueba reproducir en tiempo real.

Entrada	Resultados	Condiciones
Se selecciona la opción “Abrir” disponible en el menú principal.	Si el archivo de video tiene un formato compatible con el reproductor del sistema, se reproduce el archivo de video seleccionado. Si el archivo seleccionado tiene un formato incompatible con el reproductor del sistema se muestra un mensaje de error informando al usuario: “No es posible reproducir el formato del archivo especificado”.	Existe un archivo de video en un directorio físico de la computadora.

Tabla 29: Caso de prueba abrir archivo de video.

Entrada	Resultados	Condiciones
Se presiona la tecla correspondiente a un tipo de escena para delimitar el inicio de la escena y se presiona una tecla diferente para delimitar el fin de la escena.	El sistema cancela la escena que había comenzado a marcar y comienza a marcar una nueva escena del tipo perteneciente al botón que presionó último.	El sistema tiene que estar reproduciendo un video.

Tabla 30: Caso de prueba delimitar escenas.

Entrada	Resultados	Condiciones
Se selecciona el video del cual se quieren consultar las escenas asociadas y se selecciona el tipo de escena "Saque".	El sistema muestra todas las escenas de tipo saque asociadas al video que se seleccionó.	Las escenas fueron delimitadas anteriormente para poder consultarlas.

Tabla 31: Caso de prueba mostrar listado de escenas.

Entrada	Resultados	Condiciones
Se selecciona la escena que se desea eliminar y se presiona el botón "Eliminar".	El sistema muestra el mensaje "¿Desea eliminar la escena seleccionada?", se presiona el botón "Sí" y el sistema elimina la escena seleccionada.	Las escenas fueron delimitadas anteriormente para poder eliminarlas.

Tabla 32: Caso de prueba eliminar escenas.

Entrada	Resultados	Condiciones
Se selecciona el video que se desea eliminar y se presiona el botón "Eliminar".	El sistema muestra un mensaje de confirmación: "Si elimina el video se eliminarán también todas las escenas asociadas a este. ¿Confirma que desea eliminar el video?", se presiona el botón "Sí" y el sistema elimina el video seleccionado.	El video seleccionado está guardado en un directorio físico.

Tabla 33: Caso de prueba eliminar video.

Entrada	Resultados	Condiciones
Se presiona el botón "Grabar" para dar inicio a la grabación que finaliza al presionar el botón "Detener grabación"	El sistema guarda un archivo de video correspondiente al video que se grabó. El video es adicionado al listado de videos disponibles para reproducir.	La cámara digital está conectada y el sistema está visualizando la reproducción.

Tabla 34: Caso de prueba guardar video.

Entrada	Resultados	Condiciones
Se selecciona la opción "Ver videos" disponible en el menú principal.	El sistema muestra una nueva interfaz donde aparecen listados todos los videos que hay guardados en el sistema.	Los videos se han grabado y guardado con anterioridad.

Tabla 35: Caso de prueba mostrar videos guardados.

4.4 Conclusiones.

Concluido este capítulo se puede afirmar que el diagrama de componentes es una herramienta muy importante y versátil cuando se refiere a modelar la vista estática de un sistema, y ayuda a determinar qué componentes pueden compartirse entre diferentes partes funcionales. El modelo de despliegue genera una organización de los dispositivos que se necesitan y su distribución física. Los casos de prueba por su parte colaboran con el objetivo de lograr un sistema completamente funcional y libre de errores.

CONCLUSIONES.

A partir de un análisis exhaustivo realizado a los procesos de gestión de información de estudio de contrarios en el voleibol que se llevan a cabo en la Federación Nacional de Voleibol, se determinó que la principal causa de los problemas relacionados con el entrenamiento y la toma de decisiones acertadas en las competencias es la falta de una aplicación que automatice el procesamiento de la información vinculada a estos procesos.

Para la elaboración de la aplicación se realizaron las siguientes actividades:

- Se realizó un estudio a los sistemas que se utilizaban para fines similares a través del cual se detectaron deficiencias que fueron el punto de partida para la creación del nuevo sistema.
- Se hizo un estudio y caracterización de las metodologías, herramientas y lenguajes de programación factibles para el desarrollo de aplicaciones.
- Se llevó a cabo el proceso de desarrollo de *software* referente a la aplicación, del cual se deja constancia en el presente documento.

Con la elaboración de una aplicación que facilite el proceso de gestión de información a los entrenadores del equipo nacional de voleibol se ha dado solución al problema planteado y por consiguiente se ha cumplido el objetivo general.

La aplicación fue diseñada para los equipos nacionales, femenino y masculino de voleibol, pero su utilización puede ser aplicada a cualquier equipo.

RECOMENDACIONES.

Se recomienda:

- Optimizar el *software* con la implementación de otras funcionalidades útiles al usuario como pueden ser:
    1. Gestión de jugadores.
    2. Gestión de equipos.
    3. Gestión de eventos.
    4. Gestión de juegos.
    5. Gestión de anotaciones por juego.
    6. Mostrar estadísticas por jugador, equipo, juego y evento.
- Crear el manual de usuario para un mejor entendimiento de las funcionalidades que posee la aplicación.
- Utilizar la aplicación no sólo en los equipos nacionales de voleibol sino en todos aquellos en los que sea factible su uso.

GLOSARIO DE TÉRMINOS.

Federación Internacional de Voleibol: La Federación Internacional de Voleibol (FIVB) es el organismo mundial que se dedica a regular las normas del voleibol a nivel competitivo, así como de celebrar periódicamente competiciones y eventos en sus dos disciplinas.

Voleibol: es un deporte donde dos equipos se enfrentan sobre un terreno de juego liso separados por una red central, tratando de pasar el balón por encima de la red hacia el suelo del campo contrario.

Contrario: de forma general, contrario es todo aquello que se opone de cualquier forma o manera posible a algo.

Estadística: es una ciencia con base matemática referente a la recolección, análisis e interpretación de datos.

K1: es una abreviatura de la palabra técnica complejo uno utilizada en el voleibol para hacer referencia a la combinación de un grupo de acciones que engloba recepción, pase, ataque y aseguramiento del ataque.

K2: es una abreviatura de la palabra técnica complejo dos utilizada en el voleibol para hacer referencia a la combinación de un grupo de acciones que engloba saque, bloqueo, apoyo y defensa.

Mono: es el nombre de un proyecto de código abierto iniciado por *Ximian* y actualmente impulsado por *Novell* para crear un grupo de herramientas libres, basadas en *GNU/Linux* y compatibles con *.NET*.

Multiplataforma: es un término usado para referirse a los programas, sistemas operativos, lenguajes de programación, u otra clase de *software*, que puedan funcionar en diversas plataformas. Por ejemplo, una aplicación multiplataforma podría ejecutarse en *Windows* en un procesador x86, en *GNU/Linux* en un procesador x86 o en un *PowerPC*.

CASE: *Computer Aided Software Engineering*, en español Ingeniería de *Software* Asistida por Ordenador, son diversas aplicaciones informáticas destinadas a facilitar todos los aspectos del ciclo de vida de desarrollo del *software* en tareas como el proceso de realizar un diseño del

proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

UML: Lenguaje Unificado de Modelado (por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de *software* más conocido y utilizado en la actualidad; está respaldado por el *OMG* (*Object Management Group*). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

API: interfaz de programación de aplicaciones (del inglés *Application Programming Interface*) es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizados por otro *software* como una capa de abstracción. Uno de los principales propósitos de una *API* consiste en proporcionar un conjunto de funciones de uso general. Los programadores se benefician de las ventajas de la *API* haciendo uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio.

GUI: *Graphical User Interface* (en español interfaz gráfica de usuario) es un tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

IDE: son las siglas de *Integrated Development Environment, por* su traducción al español entorno de desarrollo integrado. Es un entorno de programación y consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica.

SO: es el acrónimo de sistema operativo y se refiere a un conjunto de programas de computación destinados a realizar muchas tareas entre las que destaca la administración eficaz de sus recursos.

HTML: siglas de *HyperText Markup Language* (Lenguaje de Marcas de Hipertexto), es el lenguaje de marcado predominante para la construcción de páginas *Web*. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

Internet: es un conjunto descentralizado de redes de comunicación interconectadas, que utilizan la familia de protocolos *TCP/IP*, garantizando que las redes físicas heterogéneas que la componen funcionen como una red lógica única, de alcance mundial.

PDF: acrónimo del inglés *Portable Document Format*, formato de documento portátil por su traducción al español es un formato de almacenamiento de documentos, desarrollado por la empresa *Adobe Systems*. Está especialmente ideado para documentos susceptibles de ser impresos.

.NET: es un proyecto de *Microsoft* para crear una nueva plataforma de desarrollo de *software* con énfasis en transparencia de redes, con independencia de plataforma de *hardware* y que permita un rápido desarrollo de aplicaciones.

AndroMDA: (pronunciado "Andromeda") es un programa informático de tipo *framework*, de generación extensible de código, que se adhiere al paradigma de la arquitectura dirigida por modelos.

GTK+: también llamado *The GIMP Toolkit* es un conjunto de bibliotecas multiplataforma para desarrollar interfaces gráficas de usuario (GUI), principalmente para los entornos gráficos *GNOME*, *XFCE* y *ROX* aunque también se puede usar en el escritorio de *Windows*, *MacOS* y otros. *GTK+* se ha diseñado para permitir programar con lenguajes como *C, C++, C#, Java, Ruby, Perl, PHP o Python.*

Qt: es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario.

OCL: (*Object Constraint Language*) es un lenguaje para la descripción formal de expresiones en los modelos *UML*. Sus expresiones pueden representar invariantes, precondiciones, postcondiciones, inicializaciones, guardias, reglas de derivación, así como consultas a objetos para determinar sus condiciones de estado.

VGA: *Video Graphics Adapter,* es un tipo de tarjeta de video. Tiene multitud de modos de video posibles, aunque el más común es el de 640x480 puntos con 256 colores, conocido generalmente como "*VGA* estándar" o "resolución *VGA*".

RAM: La memoria principal o *RAM* (*Random Access Memory*, Memoria de Acceso Aleatorio) es donde el computador guarda los datos que está utilizando en el momento presente. El almacenamiento es considerado temporal por que los datos y programas permanecen en ella mientras que la computadora esté encendida o no sea reiniciada.

PCI: Un *Peripheral Component Interconnect* (Interconexión de Componentes Periféricos) consiste en un bus de ordenador estándar para conectar dispositivos periféricos directamente a su placa base. Estos dispositivos pueden ser circuitos integrados ajustados en ésta (los llamados dispositivos planares en la especificación *PCI*) o tarjetas de expansión que se ajustan en conectores.

CPU: La unidad central de procesamiento**,** o *CPU* (por el acrónimo en inglés *Central Processing Unit*), o, simplemente, el procesador, es el componente en una computadora digital que interpreta las instrucciones y procesa los datos contenidos en los programas de la computadora.

IEEE 1394: conocido como *FireWire* por *Apple Inc.*, como *i.Link* por *Sony* y como *Lynx por Texas Instruments*, es un estándar multiplataforma para entrada/salida de datos en serie a gran velocidad. Suele utilizarse para la interconexión de dispositivos digitales como cámaras digitales y videocámaras a computadoras.

REFERENCIAS BIBLIOGRÁFICAS.

1. TZU, S. *El arte de la guerra* [Consultado el: 10 de enero de 2009]. Disponible en: http://www.gorinkai.com/textos/suntzu.htm.

2. HERNÁNDEZ, H. El Software se Desliza 2008, nº [Consultado el: 20 de enero de 2009]. Disponible en: http://nelson-panoramaganador.blogspot.com/2008_12_01_archive.html.

3. VIDEO, A. *Utilius VS* [Consultado el: 4 de febrero de 2009]. Disponible en: http://www.ansa-video.de/esp/soft01.htm

4. SOFTWARE, V. *StarTrack* [Consultado el: 4 de febrero de 2009]. Disponible en: http://www.allprosoftware.com/vb/

5. SCORES, V. *DataVolley* [Consultado el: 4 de febrero de 2009]. Disponible en: http://www.volleyballscores.co.uk/about-this-site/datavolley.html

6. ---. *VIS* [Consultado el: 7 de febrero de 2009]. Disponible en: http://www.volleyballscores.co.uk/about-this-site/vis.html.

7. MOLPECERES, A. *Procesos de desarrollo* [Consultado el: 18 de febrero de 2009]. Disponible en: http://www.willydev.net/descargas/Articulos/General/cualxpfddrup.PDF.

8. JOSÉ H. CANÓS, P. L., Mª CARMEN PENADÉS. *Metodologías Ágiles en el Desarrollo de Software* [Consultado el: 8 de febrero de 2009]. Disponible en: http://www.willydev.net/descargas/prev/TodoAgil.pdf.

9. BECK, K. *Extreme Programming Explained. Embrace Change.* 1999. Traducido de: Una explicación de la programación extrema. Aceptar el cambio.

10. COMMONS, C. *Programación extrema* [Consultado el: 18 de febrero de 2009]. Disponible en: http://www.chuidiang.com/ood/metodologia/extrema.php.

11. YULIANA ZAPATA, L. G. *Herramientas de desarrollo de ingeniería de SW para Linux* [Consultado el: 18 de febrero de 2009]. Disponible en: http://hugolopez.phi.com.co/docs/download/file=Giraldo-Zapata-Herramientas%20de%20ISW.pdf,_id=17

12. MARTÍNEZ, M. I. S. *Guía de prácticas diagramas de UML en ARGO-UML* [Consultado el: 19 de abril de 2009]. Disponible en: http://users.dsic.upv.es/asignaturas/facultad/lsi/trabajos/272002.ppt.

13. JAVIER GARCÍA DE JALÓN, J. I. R., JOSÉ MARÍA SARRIEGUI, ALFONSO BRAZALES. *Aprenda C++ como si estuviera en primero* [Consultado el: 17 de abril de 2009]. Disponible en: http://mat21.etsii.upm.es/ayudainf/aprendainf/Cpp/manualcpp.pdf.

14. WEBSITE, H. *PHP-Hipertexto-Pre-Procesado* [Consultado el: 17 de abril de 2009]. Disponible en: http://hardz.wordpress.com/2008/02/07/php-hipertexto-pre-procesado/.

15. LAURA BERMEJO SANZ, E. G. M. *Eclipse como IDE* [Consultado el: 9 de febrero de 2009]. Disponible en: http://kybele.escet.urjc.es/documentos/HC/Exposiciones/EclipseIDE.pdf.

16. RAMÍREZ, I. *NetBeans IDE* [Consultado el: 7 de febrero de 2009]. Disponible en: http://netbeans-ide.softonic.com/.

17. NETBEANS. *NetBeans* [Consultado el: 7 de febrero de 2009]. Disponible en: http://www.netbeans.org/index_es.html.

18. RODRIGO LOYOLA A, G. V. A. *Introducción a Java Media Framework* [Consultado el: 8 de febrero de 2009]. Disponible en: http://profesores.elo.utfsm.cl/~agv/elo330/2s04/projects/JMF/presentacion.ppt.

19. DÍAZ, M. D. *Plataforma de software* [Consultado el: 15 de febrero de 2009]. Disponible en: http://www.moisesdaniel.com/bloc/archives/2007/12/entry_17.html.

20. ALCALÁ, U. D. *Manual de Instalación de la Máquina Virtual de Java (JVM)* [Consultado el: 8 de febrero de 2009]. Disponible en: http://www2.uah.es/ice/aulaVirtual/Documentos/Manuales_doc/JVM.pdf.

21. LÓPEZ, A. R. *Sistema asistente para la generación de horarios de cursos* [Consultado el: 19 de febrero de 2009]. Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/rivera_l_a/capitulo2.pdf.

22. RICARDO, F. Á. C. *Utilización del patrón Modelo-Vista-Controlador (MVC) en el diseño de software educativo* [Consultado el: 19 de febrero de 2009]. Disponible en: http://www.monografias.com/trabajos43/patron-modelo-vista/patron-modelo-vista2.shtml.

23. TERUEL, A. *Arquitectura de capas* [Consultado el: 19 de febrero de 2009]. Disponible en: http://www.ldc.usb.ve/~teruel/ci3715/clases/arqCapas.html

24. PECOS, D. *PostGreSQL vs. MySQL* [Consultado el: 15 de marzo de 2009]. Disponible en: http://www.netpecos.org/docs/mysql_postgres/index.html.

25. URUGUAY, U. D. L. R. D. *Expendedora: Modelo de Dominio* [Consultado el: 4 de marzo de 2009]. Disponible en: http://iie.fing.edu.uy/ense/asign/desasoft/practico/hoja8/ejemplos_clase2.pdf.

26. I. JACOBSON, G. B., AND J. RUMBAUGH. *El Proceso Unificado de Desarrollo de Software* [Consultado el: 6 de marzo de 2009]. Disponible en: http://bibliodoc.uci.cu/pdf/reg00060.pdf.

27. IEEE. *Requerimiento* [Consultado el: 4 de marzo de 2009]. Disponible en: http://iteso.mx/~juanjo/IEEE_Std1233_1998_esp_desarrollo_de_especificacion_de_requ e.pdf.

28. HERRERA, L. J. *Ingeniería De Requerimientos* [Consultado el: 4 de marzo de 2009]. Disponible en: http://www.monografias.com/trabajos6/resof/resof.shtml#inge.

29. VILAS, A. F. *Introducción a UML* [Consultado el: 2 de abril de 2009]. Disponible en: http://tvdi.det.uvigo.es/~avilas/UML.

BIBLIOGRAFÍA.

ALFARO, F. M. Herramientas case. 1999. 53 p.

DAN PILONE, R. M. Head First Software Development. 2008. 447 p.

DIEV, S. RUP vs XP [Consultado el: 19 de abril de 2009]. Disponible en: http://www.diev.com/RUPandXPFrame1Source1.htm.

GRACIA, J. Análisis y Diseño. [Consultado el: 4 de febrero de 2009]. Disponible en: http://www.ingenierosoftware.com/analisisydiseno/uml.php.

JAMES NEWKIRK, R. C. M. La Programación Extrema en la Práctica. 2002. 216 p.

LARMAN, C. UML y Patrones. Introducción al análisis y diseño orientado a objetos. La Habana: Editorial Félix Varela, 2004.

MOLPECERES, A. Procesos de desarrollo: RUP, XP y FDD [Consultado el: 20 de febrero de 2009]. Disponible en: http://www.javahispano.org/contenidos/es/procesos_de_desarrollo/;jsessionid=D18902321F30D7A6EAC2295C4A63D31F.

MORALES, E. A. B. Herramienta case “ArgoUML”, [Consultado el: 19 de abril de 2009]. Disponible en: http://erickbenitez.iespana.es/Herramientas%20case.pdf.

ROBERTO HÉRNANDEZ SAMPIERI, C. F. C., PILAR BAPTISTA LUCIO. Metodología de la Investigación. 2006. 850 p.

VICENTE AGUILAR, P. S. MySQL vs. PostgreSQL [Consultado el: 11 de abril de 2009]. Disponible en: http://www.fedora-es.com/node/189.