Universidad de las Ciencias Informáticas

Propuesta de arquitectura de software para laboratorios virtuales.

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

Autora

Nailiuvis Rojas Pileta

Tutora

Ing. Sailyn Salas Hechavarria

La Habana, Junio 2012

Año 54 de la Revolución

Frase

Para alcanzar su sueño un guerrero de la luz necesita de una voluntad firme y de una inmensa capacidad de entrega. Aunque tenga un objetivo, el camino para lograrlo no siempre es aquel que se imagina.

Paulo Coelho.

Declaración de Autoría

Declaro ser autor del presente trabajo de diploma y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo y a realizar uso en su beneficio del mismo. Para que así conste firmo el presente a los ____ días del mes de _____ del año _____.

Autor: Nailiuvis Rojas Pileta. _____

Tutor: Ing. Sailyn Salas Hechavarria. _____

Datos de Contacto

Tutor: Ing. Sailyn Salas Hechavarria

Edad: .26

Ciudadanía: cubana

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Ciencias Informáticas

Categoría Docente: Profesor Instructor

E-mail: ssalas@uci.cu

Agradecimientos

A la Universidad de las Ciencias Informáticas por haber hecho de mí la profesional que hoy soy.

A mi novio Alfredo que durante estos años de carrera estuvo a mi lado apoyándome en todo, gracias amor por ser parte de mi vida.

A mi tutora Sailyn que me aguantó y ayudó hasta último momento, say eres una tutora excelente.

A mi familia por apoyarme en todo y darme fuerzas para seguir especialmente a mis padres y abuelos.

A mi otra familia Elena y Vilató gracias por traer al mundo a ese hombre tan maravilloso que tengo por novio y por aceptarme como una más de la familia.

Al profesor Prevot por su ayuda para terminar este trabajo.

Dedicatoria

Dedico este trabajo a mi familia especialmente a mi mamá Carmen y mi papá Chichi que me apoyaron incondicionalmente y me supieron guiar durante todos estos años de vida, porque hicieron de mí una persona honrada.

A mi novio que ha sido mi mano derecha y que en todo este tiempo se ha mantenido a mi lado en las buenas y en las malas. Amor le agradezco a dios que te haya puesto en mi camino.

A mis abuelos que nunca han faltado sus sabios consejos y que siempre me han guiado por el buen camino.

A mi hermana que me ha apoyado siempre y a mi sobrino Yudel Ángel por darle tanta alegría a mi vida.

Resumen

El presente trabajo es una propuesta de arquitectura de software para laboratorios virtuales del Proyecto de Laboratorios Virtuales (PROLAVI) de la Universidad de las Ciencias Informáticas en la Facultad 5 debido a que no cuenta con una arquitectura documentada para el desarrollo de dichos software.

Por la importancia que tiene hoy en día el uso de un laboratorio virtual que cumpla las mismas expectativas de un laboratorio tradicional en la educación debido al déficit de estos últimos, fue necesario el diseño de una arquitectura que permita mejorar la calidad, modularidad y funcionalidad de los software resultantes. La selección de estilos, patrones de arquitectura y de diseño, herramientas, metodología para el desarrollo de software y métodos de evaluación de arquitecturas van a permitir la obtención de una arquitectura de software que cumpla con los requisitos deseados facilitando además la reutilización de la misma lo que va a ayudar en el perfeccionamiento continuo de las practicas de laboratorios.

Palabras Clave: *Arquitectura de software, estilo de arquitectura, laboratorios virtuales, patrón arquitectónico, patrón de diseño.*

Tabla de Contenidos

Índice de figuras

Índice de tablas

Introducción

El desarrollo de las tecnologías de la información y la comunicación (TICS) ha impactado tanto en la sociedad que su uso es inevitable. El avance continuo de las mismas ha permitido una nueva forma de comunicación: la realidad virtual, con el propósito de producir una apariencia de realidad que permita al usuario tener la sensación de estar presente en ella.

En el ámbito educativo, la incorporación de la realidad virtual ha llegado para quedarse porque mediante la misma se ha logrado fortalecer aún más los conocimientos adquiridos en las diferentes materias, por lo que es clave su utilización en el proceso de enseñanza.

Como parte de la realidad virtual en la educación surgen los entornos virtuales de enseñanza-aprendizaje *(*EVEA*),* aportando resultados positivos en esta área. Su concepto viene asociado a un nuevo modelo en el que se centran las tendencias actuales de la educación, donde las teorías y estilos de aprendizaje centran sus procesos en el estudiante, que le permiten construir su conocimiento basado en sus propias expectativas y necesidades de acuerdo al contexto en que se desarrolla, aplicando métodos investigativos que le permitan tomar acciones para alcanzar resultados positivos, lo cual deviene en un revolucionario modelo pedagógico-tecnológico que asegura una educación pertinente, cuyo mayor reto es mantener y elevar la calidad del proceso docente-educativo [1].

Un EVEA sirve para distribuir materiales educativos en formato digital (textos, imágenes, audio, simulaciones, juegos) y acceder a ellos, para realizar debates y discusiones en línea sobre aspectos del programa de la asignatura, para integrar contenidos relevantes de la red o para posibilitar la participación de expertos o profesionales externos en los debates o charlas.

El uso de los entornos virtuales de enseñanza-aprendizaje ha permitido incorporar varios espacios virtuales: las bibliotecas, tablón de anuncios, cuadernos digitales, ejercicios interactivos, videos conferencias y los laboratorios virtuales que son herramientas de aprendizaje que pretenden aproximar el ambiente de un laboratorio tradicional, teniendo como mayor característica la relación hombre-máquina [2].

En un sentido más amplio, un laboratorio virtual es un tipo de apoyo centrado en el logro de determinados objetivos creativos o de ayuda a la toma de decisiones, por lo tanto pueden utilizarse en la mayoría de las esferas de la actividad intelectual humana, por lo que aumentado su manejo en las ramas de la salud, la ingeniería y la investigación además de la educación. Con el aumento cada vez mayor del uso de las nuevas tecnologías los laboratorios virtuales han logrado expandirse a la mayoría de ellas incluyendo la computación móvil, que consiste en el uso de dispositivos capaces de operar en red y que los estudiantes llevan consigo todo el tiempo.

Un número cada vez más grande de profesores y personal de Tecnología Educativa experimentan con las posibilidades que la computación móvil ofrece a la colaboración y la comunicación. Dispositivos como los teléfonos inteligentes o los ordenadores ultra-portátiles son herramientas transportables útiles para la productividad, el aprendizaje así como la comunicación y ofrecen una variedad cada vez mayor de actividades que pueden realizarse plenamente con aplicaciones diseñadas especialmente para móviles, entre las que se incluyen los laboratorios virtuales [3].

En Cuba se hace uso también de los laboratorios virtuales, debido a que se carece de laboratorios tradicionales casi en su totalidad a causa del bloqueo impuesto por Estados Unidos. Varios centros educativos principalmente las universidades hacen uso de esta tecnología de aprendizaje para permitirles a sus estudiantes un mejor conocimiento. En este caso se encuentra el Instituto Superior Politécnico José Antonio Echeverría (ISPJAE) que desde 1991 ha venido incrementado los laboratorios virtuales como una forma más de estudio práctico. Se han visto avances en el desarrollo de laboratorios virtuales en la Universidad de La Habana, con laboratorios de Electrónica así como el desarrollo de laboratorios de Bioquímica en la Universidad de Oriente.

En las Ciencias Médicas, el desarrollo de las TICS ha permitido la creación de laboratorios virtuales que apoyan el aprendizaje y la investigación de alumnos, técnicos y profesionales como parte del proceso docente educativo que se desarrolla en estas instituciones. La Universidad Virtual de Ciencia Tecnología y Medio Ambiente (CITMA) desde al año 2000 hace uso de los laboratorios virtuales de Física, Álgebra y Matemática para fomentar el aprendizaje a distancia. La Universidad Central de las Villas Marta Abreu de la provincia Villa Clara cuenta con un sitio web donde los estudiantes pueden acceder a realizar prácticas de laboratorios virtuales de todos los

temas de la Física que se imparten en las carreras de ingeniería. Cuba avanza en el uso de estas herramientas educativas y cada día se obtienen productos con mayor calidad lo que hace cada vez más real el trabajo en los laboratorios virtuales por parte de los docentes.

Por la importancia que tienen los laboratorios virtuales en la Educación, surge en la facultad 5 de la Universidad de las Ciencias Informáticas el proyecto de laboratorios virtuales (PROLAVI), que hasta el momento ha desarrollado tres laboratorios virtuales: "Ensamblaje de un Computador", "Diseño e instalación de una red de área local (LAN)" y "Administración y Configuración de una red de área local (LAN)", los cuales permiten a los estudiantes una mejor preparación y formación relacionado con estos temas. En el proyecto se trabaja en base a que los productos que allí se desarrollen tengan la mejor calidad posible y para ello es muy importante la utilización de una adecuada y conveniente arquitectura de software.

La arquitectura de software es: "la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución" [4]. Los errores u omisiones en el diseño arquitectónico pueden destruir todo el trabajo logrado hasta el momento.

El proyecto PROLAVI no tiene definida una arquitectura de software con la documentación necesaria que asegure que la construcción de los laboratorios virtuales que se desarrollen cumplan con los servicios y las funcionalidades que espera el usuario. Los módulos desarrollados hasta el momento solo están basados en algunos patrones propios de la herramienta de programación utilizada para su implementación, provocando que una vez terminados los laboratorios virtuales se realizaran cambios en los mismos por no cumplir con algunos requisitos de software, por lo que fueron entregados fuera de la fecha prevista.

A partir de la situación problémica anteriormente expuesta se formula el siguiente **problema científico**: ¿Cómo obtener una arquitectura para guiar el proceso de desarrollo de software de los laboratorios virtuales?

Por lo que el **objeto de estudio es:** La arquitectura de software.

Para resolver el problema referenciado anteriormente se plantea como **objetivo general** de esta investigación: Diseñar una propuesta de arquitectura de software para el desarrollo de laboratorios virtuales de PROLAVI.
La investigación estará centrada en el siguiente **campo de acción:** Arquitectura de software para laboratorios virtuales.

Para sustentar la realización de este trabajo se toma como **idea a defende**r: El diseño de la arquitectura de software a proponer para los laboratorios virtuales de PROLAVI ayudará en el proceso de desarrollo de los mismos, favoreciendo su calidad y reutilización.

Para dar cumplimiento al objetivo planteado de este trabajo se definieron las siguientes **tareas de investigación:**

- Revisión bibliográfica de materiales relacionados con el objeto de estudio para identificar los fundamentos teóricos del problema.
- Estudio de estilos y patrones arquitectónicos para identificar los posibles a utilizar.
- Identificación de la estructura de diseño (o sus relaciones) que debe tener la arquitectura de software propuesta.
- Selección de las herramientas y metodología de software a utilizar.
- Diseño de la arquitectura de software a utilizar.
- Evaluación de la arquitectura de software propuesta.
- Elaboración del documento de tesis.

Los **métodos** a utilizar en la investigación son los siguientes:

Empíricos:

- **Consultas bibliográficas**. Consultar bibliografías sobre estilos y patrones de arquitectura que permitan realizar un buen diseño de la arquitectura de software.

Teóricos:

- **Analítico – sintético**. Este método permitirá analizar las teorías y documentos existentes sobre el tema a desarrollar, permitiendo la extracción de los elementos más importantes que se relacionan con el objeto de estudio y el campo de acción.

- **Histórico – lógico.** Mediante este método se podrá constatar teóricamente como ha ido evolucionando la arquitectura de software, el desarrollo de los laboratorios virtuales así como su conexión a lo largo de la historia.
- **Modelación.** Se utilizará para representar parte del conocimiento acumulado durante la investigación, como los diferentes diagramas que apoyarán el proceso de desarrollo de la arquitectura de software.

El presente trabajo de diploma se conforma de tres capítulos descritos a continuación.

Capítulo I. Fundamentación teórica. En este capítulo se realiza un estudio de las arquitecturas de software más usadas actualmente en los laboratorios virtuales. Se estudian las tendencias y antecedentes de la arquitectura de software. Además se analizan las herramientas, tecnologías, estilos arquitectónicos y patrones de diseño a utilizar.

Capítulo II. Descripción de la propuesta de solución. En este capítulo se presenta la propuesta de arquitectura de software para el desarrollo de laboratorios virtuales de PROLAVI en la cual se describen cada uno de los elementos que componen el modelo, especificándose también las distintas vistas que permiten diseñar la arquitectura de software basada en el estudio realizado en el capítulo anterior.

Capítulo III. Validación de la propuesta de solución. En este capítulo se realiza la validación de la arquitectura de software desarrollada mediante el método de evaluación de arquitectura seleccionada.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

Introducción

El principal objetivo de la arquitectura de software es aportar elementos que ayuden a la toma de decisiones y al mismo tiempo, proporcionar conceptos y un lenguaje común que permitan la comunicación entre los equipos que participen en un proyecto. La necesidad de una adecuada arquitectura de software es una preocupación cada vez mayor en el mundo de la informática, cuyos resultados se aprecian en la realización de un software y luego en la aceptación del mismo por el cliente. En el presente capítulo se abordan los fundamentos generales con las bases conceptuales que permitan el entendimiento del problema.

1.1. Arquitectura de software

Hoy en día se requiere cada vez más de productos con buena calidad, los que necesitan a su vez diferentes tecnologías y plataformas de hardware y software para alcanzar el funcionamiento deseado. La calidad de estos productos es indispensable, por lo que los sistemas deben ser modificables, interoperables, seguros, funcionales y disponibles, asegurando el mantenimiento, flexibilidad y reusabilidad. Para lograr estos requisitos es necesario el diseño de una arquitectura de software robusta y confiable que se adapte a las necesidades de cada software.

Una buena arquitectura de software debería estar bien documentada, con al menos una vista dinámica y una vista estática, utilizando una notación para que terceras personas puedan entenderla fácilmente. Debe ser evaluada y analizada con técnicas cuantitativas y cualitativas, presentarse como una implementación incremental para poder diseñar un esqueleto del sistema donde primero se muestre la mínima funcionalidad y después como puede ir creciendo, además tiene que ser diseñada por arquitectos que cuentan con los requisitos funcionales y no funcionales del sistema.

1.1.1. Definiciones fundamentales

Existen varias definiciones sobre la arquitectura de software ya que cada autor aporta ideas que van enriqueciendo su definición, por ejemplo:

Paul Clements, 1996: “La arquitectura de software es a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones”. [5]

A causa de la abundancia de definiciones del campo de la arquitectura de software, existe en general acuerdo de que ella se refiere a la estructura a grandes rasgos del sistema, consistente en componentes y las relaciones entre ellos. [5]

Es muy habitual encontrar varias definiciones referentes a este tema, sin embargo, ya desde el año 2000 el “Instituto de Ingenieros en Electricidad y Electrónica”, conocida como la IEEE publicó en su documento IEEE Std 1471-2000 la definición oficial para ella: “La arquitectura de software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orienten su diseño y evolución” [5].

Esta definición es importante porque destaca el hecho de que un sistema puede contemplarse desde distintas perspectivas que hacen énfasis en los distintos aspectos del sistema, por lo que se tendrá en cuenta esta definición para el diseño de la arquitectura de software a proponer.

1.1.2. Antecedentes históricos.

Los antecedentes de la arquitectura de software se remontan a la década de 1960 aunque su estudio e historia no ha sido tan profunda e incesante como la Ingeniería de Software que es el campo que la incluye. A continuación se muestran los antecedentes de la misma que hasta nuestros días ha sido de mucha ayuda en la construcción de un software.

Entre los primeros científicos en hacer planteamientos que se acercaran a lo que se conoce hoy como arquitectura de software se destaca Edsger Dijkstra de la Universidad Tecnológica de Holanda en 1968, quien propuso que se hiciera una estructuración correcta de los sistemas de software antes de lanzarse a programar. Más tarde Fred Brooks Jr y Ken Iverson, en 1969, llamaban arquitectura de software a la estructura conceptual de un sistema en la perspectiva del programador. En 1975

Brooks utilizaba el concepto de arquitectura del sistema para designar “la especificación completa y detallada de la interfaz de usuario” y consideraba que el arquitecto es un agente del usuario, igual que lo es quien diseña su casa, empleando una nomenclatura que ya nadie aplica de ese modo [5].

En la década de 1980 fueron perfeccionadas las técnicas descriptivas, las notaciones formales y para la caracterización de lo que sucedería en la siguiente década, ellos formulan esta otra frase que ha quedado inscrita en la historia de la especialidad: “La década de 1990, creemos, será la década de la arquitectura de software” [5].

La arquitectura de software quedó en estado de vida latente durante unos cuantos años hasta comenzar su expansión explosiva con los manifiestos de Dewayne Perry y TBell de laboratorios de New Jersey y Alexander Wolf de la Universidad de Colorado.

Puede decirse que Perry y Wolf fundaron la disciplina, puesto que el primer estudio en que aparece la expresión arquitectura de software como se conoce hoy fue realizado por ellos en 1992. La arquitectura de software se encuentra en una etapa de formación constante y están surgiendo nuevos aportes que desarrollan y amplían la disciplina por la importancia que tiene para el desarrollo de un software.

1.1.3. ¿Por qué es importante la arquitectura de software?

Se han identificado varias razones claves por las cuales la arquitectura de software es importante:

- Las representaciones de la arquitectura de software permiten la comunicación entre todas las partes interesadas en el desarrollo de un sistema de cómputo.
- Destaca las decisiones iniciales relacionadas con el diseño que tendrán un impacto profundo en todo el trabajo de la Ingeniería de software que le sigue y lo que también resulta importante en el éxito final del sistema como entidad operacional.
- Constituye un modelo relativamente pequeño e intelectualmente comprensible de cómo está estructurado el sistema y como trabajan juntos sus componentes.
- El modelo de diseño arquitectónico y los patrones arquitectónicos que contiene son transferibles.
- Los estilos y patrones arquitectónicos se aplican al diseño de otros sistemas y representan un conjunto de abstracciones que permiten a los ingenieros de software describir la arquitectura de manera predecible.

1.2. Laboratorios virtuales

Un laboratorio virtual es un espacio electrónico de trabajo concebido para la colaboración y la experimentación a distancia con objeto de investigar o realizar otras actividades creativas, elaborar y difundir resultados mediante tecnologías difundidas de información y comunicación [6].

El uso de laboratorios virtuales representa una oportunidad para el profesor de estimular al alumno en la responsabilidad de aprender por sí mismo y transferir ese aprendizaje al mundo real, con Tecnología Educativa. También implica el reto de desarrollar materiales semejantes a los juegos con intereses educativos. Los laboratorios virtuales se pueden dividir en tres grupos como se muestra a continuación:

- **Laboratorios virtuales web**. Este tipo de laboratorios se basa en un software que depende de los recursos de un servidor determinado. Esos recursos pueden ser determinadas bases de datos, software que requiere ejecutarse en su servidor, es decir la exigencia de determinado hardware para ejecutarse [2]. Estos programas el usuario no los puede descargar en su equipo para ejecutarlo localmente de forma independiente, se abren mediante un navegador web que utiliza la información guardada en un servidor distinto de la máquina donde se está ejecutando el laboratorio virtual.

- **Laboratorios remotos**. Se trata de laboratorios que permiten operar remotamente cierto equipamiento, bien sea didáctico como maquetas específicas o industriales. En general estos laboratorios requieren de servidores específicos que les den acceso a las máquinas para operar de forma remota. [2].

- **Laboratorios virtuales software**. Son laboratorios desarrollados como un programa de software independiente destinado a ejecutarse en la máquina del usuario y cuyo servicio no requiere de un servidor web. Es el caso de programas con instalación propia que pueden estar destinados a plataformas Unix, Linux, Windows e incluso necesitar que otros componentes de software estén instalados previamente pero que no necesitan los recursos de un servidor determinado para funcionar [2]. Los laboratorios que se desarrollan en el proyecto PROLAVI son de

tipo software o desktop como también son conocidos ya que no necesitan de un servidor de base de datos para ejecutarse.

1.2.1. Ventajas y desventajas

Los laboratorios virtuales poseen varias ventajas y desventajas como se muestra a continuación.

Ventajas [2]:

- Reducen el costo del montaje y mantenimiento de los laboratorios tradicionales, siendo una alternativa barata y eficiente, donde el estudiante simula los fenómenos a estudiar como si los observase en el laboratorio tradicional.
- Fomenta la formación académica de los alumnos.
- Ayudan a proteger el medio ambiente, el estudiante y al profesor en casos de prácticas de laboratorios que ponen en peligro la vida de los mismos.
- Permite que el profesor analice los resultados desde su ordenador en cualquier horario.

Los laboratorios virtuales del proyecto PROLAVI permiten a los estudiantes ver los errores que han ido cometiendo al realizar la práctica de laboratorio así como guardar dicha práctica para analizarla o resolverla en otro momento sin perder lo que ha hecho hasta ese instante, fomentándose de esta forma el aprendizaje práctico en los estudiantes.

Entre las desventajas que presentan los laboratorios virtuales se pueden encontrar [2]:

- Es necesario disponer de ordenadores para realizar la práctica de laboratorio.

1.3. Arquitecturas de software más usadas en los laboratorios virtuales

Hoy en día se hace uso de los tres tipos de laboratorios virtuales pero son más utilizados los laboratorios virtuales web así como remoto y en menor medida los laboratorios virtuales software. Los laboratorios virtuales web y remoto mayormente usan la arquitectura cliente/servidor. Este modelo describe la estructura tradicional de

los sistemas distribuidos. Los principales componentes de la arquitectura de software son:

- Un conjunto de servidores locales que ofrecen servicios a otros subsistemas.
- Un conjunto de clientes que invocan los servicios ofrecidos por los servidores.
- Una red que permite que los clientes accedan a los servicios.

A continuación se muestran ejemplos de laboratorios virtuales que usan dicha arquitectura de software:

- Laboratorio virtual para optimizar el uso de un laboratorio de robótica real, el mismo surge como un proyecto colaborativo entre el laboratorio de robótica denominado Laboratorio de Manufactura Flexible Asistido por Computadora (LMFACCCS) ubicado en el Centro de Ciencias de Sinaloa y el Centro de Investigación Aplicada en Tecnologías de Información y Comunicaciones (CIATIC-UAS) de la facultad de Informática Culiacán de la Universidad Autónoma de Sinaloa. Para el diseño del laboratorio virtual se consideró un esquema cliente/ servidor con el fin de realizar la interconexión entre el mundo virtual y el laboratorio real.

- Laboratorio virtual de Física Cuántica, en el departamento de Física de La Universidad de Villa Clara. Esta práctica está fundamentada en los conceptos de Física Cuántica con ayuda de un programa de simulación nombrado Visual de Mecánica Cuántica y está basado en la simulación del espectro de luz que emiten diferentes sustancias.

Para el caso de los laboratorios virtuales software se usa la arquitectura en capas, arquitectura orientada a objetos y la arquitectura basada en componentes. Ejemplo de laboratorios virtuales que usan estas arquitecturas son:

- Laboratorio Virtual de Química (VLabQ) en la Universidad Miguel de Cervantes en Lucena, Córdoba. Entre las principales características de este laboratorio virtual se destacan: la posibilidad de guardar en cualquier momento todo el contenido del laboratorio, tanto el equipo como su contenido y condiciones para así poder continuar con la práctica posteriormente. Una vez cargada una práctica, el simulador muestra diferentes textos que sirven como guía para realizarla. El

laboratorio consta de tres apartados que muestran el marco teórico, el procedimiento y las conclusiones que contiene cada simulación.

1.4. Estilos y patrones arquitectónicos

Los estilos y patrones arquitectónicos ayudan al arquitecto a definir la composición y el comportamiento del sistema de software, y una combinación adecuada de ellos permite alcanzar los requisitos de calidad. Se estima que los estilos se pueden comprender como clases de patrones, o tal vez más adecuadamente como lenguajes de patrones y que definen los patrones posibles de las aplicaciones.

Estilos [7]

Los estilos arquitectónicos describen una clase de arquitecturas, o piezas significantes de una arquitectura. Permiten evaluar arquitecturas de software alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requisitos no funcionales. El estilo arquitectónico es también un patrón de construcción. Asociado a cada estilo hay propiedades que lo caracterizan:

- Determinan sus ventajas e inconvenientes.
- Condicionan la elección de uno u otro estilo.
- Cada estilo describe una categoría de sistemas que abarca.
- Un conjunto de componentes, por ejemplo, una Base de Datos (BD), módulos computacionales que realizan una función requerida por el sistema.
- Un conjunto de conectores que permiten la "comunicación, coordinación y cooperación entre los componentes".
- Restricciones que definen cómo se integran los componentes para formar el sistema.
- Modelos semánticos que permiten a un diseñador mediante el análisis de las propiedades conocidas de las partes que lo integran, comprender las propiedades generales de un sistema.

Patrones [7]

Según Buschmann, 1996, los patrones de arquitectura de software se pueden ver como la descripción de un problema en particular y recurrente de diseño que aparece

en contextos de diseño arquitectónicos específicos y representa un esquema genérico demostrado con éxito para su solución.

El esquema de solución se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades y desarrollos, así como también la forma en que estos colaboran entre sí. Los patrones arquitectónicos se definen sobre aspectos fundamentales de la estructura del sistema software, donde se especifican un conjunto de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes.

Diferencias entre patrón y estilo [7]

- El alcance de un patrón es menor ya que se concentra en un aspecto en lugar de hacerlo en toda la arquitectura.
- Un patrón impone una regla sobre la arquitectura, pues describe la manera en que el software manejará algún aspecto de su funcionalidad al nivel de la infraestructura.
- Los patrones arquitectónicos tienden a abarcar aspectos específicos del comportamiento dentro del contexto de la arquitectura.

Los patrones se usan junto con un estilo arquitectónico para determinar la forma de la estructura general de un sistema.

1.4.1. Clasificación de los estilos arquitectónicos

Los principales estilos arquitectónicos que se usan en la actualidad están divididos por clases de estilos que engloban una serie de estilos arquitectónicos específicos:

Estilos de flujo de datos.

- tubería y filtros.

Estilos centrados en datos.

- Arquitectura de pizarra o repositorio.

Estilos de llamada y retorno.

- Modelo-vista-controlador.
- Arquitecturas en capas.
- Arquitecturas orientadas a objetos.
- Arquitecturas basadas en componentes.

Estilos de código móvil.

- Arquitectura de máquinas virtuales.

Estilos heterogéneos.

- Sistemas control de procesos.
- Arquitecturas basadas en atributos.

Estilos peer-to-peer.

- Arquitecturas basadas en eventos.
- Arquitecturas orientadas a servicios.
- Arquitecturas basadas en recursos.

1.4.1.1. Estilos de llamada y retorno

Estos estilos reflejan la estructura del lenguaje de programación, además de que enfatiza la modificabilidad y la escalabilidad porque son más generalizadores en sistemas de gran escala. Dentro de ellos se encuentran las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objetos y los sistemas jerárquicos en capas [7].

Arquitectura en capas

Este estilo se define como una organización jerárquica, tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Al dividir un sistema en capas, cada capa puede tratarse de forma independiente, sin tener que conocer los detalles de las demás. La división de un sistema en capas facilita el diseño modular, en la que cada capa encapsula un aspecto concreto del sistema y permite además la construcción de sistemas débilmente acoplados lo que significa que si se minimiza las dependencias entre capas, resulta más fácil sustituir la implementación de una capa sin afectar al resto del sistema [7].

Arquitectura orientada a objetos

Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. Los objetos representan una clase de componentes denominada manager (administradores), debido a que son responsables de preservar la integridad

de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos. [7]

Arquitectura basada en componentes

Los sistemas de software basados en componentes se basan en principios definidos por una Ingeniería de Software específica, los componentes son las unidades de modelado, diseño e implementación. Las interfaces están separadas de las implementaciones y conjuntamente sus interacciones son el centro de incumbencias en el diseño arquitectónico. Las funcionalidades y propiedades de los componentes pueden ser descubiertas y utilizadas en tiempo de ejecución [7].

1.4.1.2. Estilos de código móvil

Esta familia de estilos enfatiza la portabilidad. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando. Fuera de las máquinas virtuales y los intérpretes, los otros miembros del conjunto han sido rara vez estudiados desde el punto de vista estilístico de la arquitectura [7].

- **Arquitectura de máquinas virtuales**

Esta arquitectura se conoce como intérpretes basados en tablas o sistemas basados en reglas. Estos sistemas se representan mediante un seudo-programa a interpretar y una máquina de interpretación. Estas variedades incluyen un extenso espectro que está comprendido desde los llamados lenguajes de alto nivel hasta los paradigmas declarativos no secuenciales de programación. Las aplicaciones inscritas en este estilo simulan funcionalidades no nativas al hardware y software en que se implementan o capacidades que exceden a (o que no coinciden con) las capacidades del paradigma de programación que se está implementando [7].

1.4.1.3. Estilos peer to peer

Esta agrupación de estilos, también llamada de componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras

entidades, pero no controlarlas directamente. Los mensajes pueden ser enviados a componentes nominados o propalados mediante emisiones.

Los ejemplos de sistemas que utilizan esta arquitectura son numerosos. El estilo se utiliza en ambientes de integración de herramientas, en sistemas de gestión de base de datos para asegurar las restricciones de consistencia, en interfaces de usuario para separar la presentación de los datos de los procedimientos que gestionan datos entre otros. [7]

- **Arquitecturas basadas en eventos**

Las arquitecturas de software basadas en eventos se han llamado también de invocación implícita. La idea dominante en la invocación implícita es que en lugar de invocar un procedimiento en forma directa (como se haría en un estilo orientado a objetos) un componente puede anunciar mediante difusión uno o más eventos. Un componente de un sistema puede anunciar su interés en un evento determinado asociando un procedimiento con la manifestación de dicho evento [7].

Otros nombres propuestos para el mismo estilo han sido integración reactiva o difusión selectiva. Por supuesto que existen estrategias de programación basadas en eventos, sobre todo referidas a interfaces de usuario y hay además eventos en los modelos de objetos y componentes, pero no es a eso a lo que se refiere primariamente el estilo, aunque esa variedad no está del todo excluida.

1.5. Patrones de diseño

Los patrones de diseño tratan los problemas de diseño que se repiten y que se presentan en situaciones particulares del diseño, con el fin de proponer soluciones a ellas. Estos expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas de software. Nos brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares, para esto debemos tener presente los siguientes elementos de un patrón: su nombre, el problema, la solución y las consecuencias. Los patrones de diseño permiten identificar las clases, instancias, roles, colaboraciones y la distribución de responsabilidades. A continuación se muestran los patrones de diseño más conocidos así como sus clasificaciones.

1.5.1. Patrones GoF

En el libro Patrones de Diseño publicado en los años 90, escrito por los que comúnmente se conoce como GoF (Gang of Four, "pandilla de los cuatro") se recopilaron y documentaron 23 patrones de diseño clasificados en tres grandes categorías.[8]

- **De creación**: abstraen el proceso de creación de instancias.
- **Estructurales**: se ocupan de cómo clases y objetos son utilizados para componer estructuras de mayor tamaño.
- **De comportamiento**: atañen a los algoritmos y a la asignación de responsabilidades entre objetos.

A continuación algunos ejemplos de estos patrones de diseño.

Patrones creacionales

- ***Abstract factory*** (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.
- ***Builder*** (Constructor virtual): Abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.
- ***Prototype*** (Prototipo): Crea nuevos objetos clonándolos de una instancia ya existente.
- ***Singleton*** (Instancia única): El Singleton es quizás el más sencillo de los patrones que se presentan en el catálogo del GoF .Es también uno de los patrones más conocidos y utilizados. Su propósito es asegurar que sólo exista una instancia de una clase.

Patrones de comportamiento

- ***Interpreter*** (Intérprete): Dado un lenguaje define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.
- ***Mediator*** (Mediador): Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
- ***State*** (Estado): Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.

1.5.2. Patrones de Asignación de Responsabilidades

Conocidos como patrones GRAPS que describen los principios fundamentales de la asignación de responsabilidades a objetos y más que patrones propiamente dicho son una serie de buenas prácticas de aplicación recomendable en el diseño de software. El nombre se eligió para indicar la importancia de captar estos principios, si se quiere diseñar eficazmente el software orientación a objeto. Entre los más conocidos se encuentran los siguientes patrones: experto, creador, bajo acoplamiento, alta cohesión y controlador [9].

Experto:

El GRASP de experto en información es el principio básico de asignación de responsabilidades. Nos indica, por ejemplo, que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo obtendremos un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento) [9].

Problema: ¿Cuál es el principio general para asignar responsabilidades a los objetos?

Solución: Asignar una responsabilidad al experto en información.

Creador:

El patrón creador nos ayuda a identificar quién debe ser el responsable de la creación de nuevos objetos o clases. Si se asignan bien el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización.

La nueva instancia deberá ser creada por la clase que:

- Tiene la información necesaria para realizar la creación del objeto.
- Contiene o agrega la clase.

Bajo acoplamiento

- Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima

repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases.

Alta cohesión

- Nos dice que la información que almacena una clase debe de ser coherente y está en la mayor medida de lo posible relacionada con la clase.
- La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase.

1.6. Metodologías del desarrollo de software

Se entiende por metodología de desarrollo una colección de documentación formal referente a los procesos, las políticas y los procedimientos que intervienen en el desarrollo del software. La finalidad de una metodología de desarrollo es garantizar la eficacia y la eficiencia en el proceso de generación de software [10].

En términos informáticos, una metodología de desarrollo de software es un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevos software [10].

Las metodologías se clasifican en robustas y ágiles.

Las metodologías robustas son aquellas en las que se hace una fuerte planificación del proyecto durante todo el proceso de desarrollo, encaminadas para proyectos de gran tamaño. Están divididas por etapas en las cuales se genera gran cantidad de documentación. En este tipo de metodologías se realiza además un profundo desarrollo en el análisis y diseño de los sistemas antes de su construcción. Ejemplo de ellas:

- **Proceso Unificado de Desarrollo (RUP)**. Provee un acercamiento disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad que satisfaga los requerimientos de los usuarios finales.
- **Marco de Soluciones Microsoft (MSF)**. Es un resumen de las mejores prácticas en cuanto a administración de proyectos se refiere. Más que una metodología rígida de administración de proyectos, es una serie de modelos que puede adaptarse a cualquier proyecto de tecnología de informa.

Las metodologías ágiles por otra parte están encaminadas para proyectos de menos volumen y en los que la documentación no juega el papel más importante. Son orientadas a las personas y no a los procesos. Ejemplos:

- **Programación Extrema (XP).** La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad.
- **Proceso Unificado Ágil.** También conocido como AUP es un acercamiento aerodinámico a desarrollo del software basado en RUP, en disciplinas y entregables incrementales con el tiempo. El ciclo de vida en proyectos grandes es serial mientras que en los pequeños es iterativo.
- **Scrunm**. Es un proceso ágil y liviano que sirve para administrar y controlar el desarrollo de software. Se focaliza en priorizar el trabajo en función del valor que tenga para el negocio, maximizando la utilidad de lo que se construye y el retorno de inversión.

1.6.1. Programación Extrema

La Programación Extrema es una metodología ligera de desarrollo de software que se basa en la simplicidad, la comunicación y la realimentación o reutilización del código desarrollado. Consiste en una programación rápida o extrema y es utilizada para proyectos de corto plazo. Esta metodología es basada en pruebas unitarias, la re-fabricación y la programación en pares [10].

1.6.2. Proceso Unificado de Desarrollo

El Proceso Unificado de Desarrollo conocido como RUP es una metodología para el desarrollo de software orientado objeto. Es un proceso de desarrollo de software, definido como un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. No obstante, el proceso unificado es más que un proceso de trabajo, es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones y diferentes niveles de aptitud [11].

Está constituido por 9 flujos de trabajo: modelamiento del negocio, requerimientos, análisis y diseño, implementación, prueba, instalación, administración de configuración

y cambios, administración de proyectos, ambiente, los cuales tienen lugar sobre 4 etapas o fases: inicio, elaboración, construcción y transición. Es adaptable para proyectos a largo plazo y establece refinamientos sucesivos de una arquitectura ejecutable. RUP es un proceso de desarrollo de software que junto al Lenguaje de Modelado (UML) constituyen la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos [11].

Características específicas de RUP [11]

- **Dirigido por casos de uso**. Significa que el proceso de desarrollo sigue una trayectoria que avanza a través de los flujos de trabajo generados por los casos de uso. Los casos de uso se especifican y diseñan al principio de cada iteración y son la fuente a partir de la cual los ingenieros de prueba construyen sus casos de prueba, describiendo así la funcionalidad total del sistema.

- **Centrado en la arquitectura**. Los casos de uso guían a la arquitectura del sistema que influye en la selección de los casos de uso. La arquitectura involucra los elementos más significativos del sistema y está influenciada, entre otros por las plataformas de software, sistemas operativos, sistemas de gestión de bases de datos, además de otros como sistemas heredados y requisitos no funcionales. Se presenta mediante varias vistas que se centran en aspectos concretos.

- **Iterativo e incremental.** RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y se definen según el nivel de madurez que alcanzan los productos que se van obteniendo con cada actividad ejecutada. La terminación de cada fase ocurre en el hito correspondiente a cada una, donde se evalúa que se hayan cumplido los objetivos de la fase en cuestión.

1.6.3. Selección de la metodología de desarrollo

Como metodología para el desarrollo de software se propone RUP debido a que es muy popular y utilizada para la documentación y realización de software orientados a objetos. El equipo de desarrollo cuenta con experiencia en el uso de esta metodología lo que aumenta la rapidez del desarrollo del producto.

Propone varias iteraciones en el proceso de desarrollo de software posibilitando la corrección de errores y mejoras del software a medida que avanza el proyecto, lo que permite a los desarrolladores poder realizar un producto con gran calidad. Brinda una fuerte y abundante documentación, permitiendo un futuro entendimiento de cada componente del producto.

La realización de los laboratorios de PROLAVI como producto hace necesaria una buena documentación favoreciendo el soporte, mantenimiento y producción de nuevas versiones y nuevos laboratorios virtuales los cuales se realizarían con mucha más calidad si existe la documentación necesaria. La experiencia del equipo de desarrollo en esta metodología constituye un factor fundamental en la selección de la misma.

1.7. El Lenguaje Unificado de Modelado

UML ofrece soporte para clases, clases abstractas, relaciones, comportamiento por iteración, empaquetamiento, entre otros. Estos elementos se pueden representar mediante nueve tipos de diagrama, que son: de clases, de objetos, de casos de uso, de secuencia, de colaboración, de estados, de actividades, de componentes y de desarrollo. [11]

Presenta características generales y razones por las que resulta interesante su aplicación para efectos de la representación de una arquitectura de software. Permite el soporte para algunos de los conceptos asociados a las arquitecturas de software, como los componentes, los paquetes, las librerías y la colaboración. Mediante este lenguaje se pueden describir los componentes que integran la arquitectura de software y permitiendo además especificar su nombre, clases o interfaces que los implementan.

Está compuesto por tres elementos fundamentales, los elementos, los diagramas y las relaciones. Los elementos son abstracciones que constituyen los bloques básicos de construcción, los diagramas representan un conjunto de elementos lo que permite visualizan un sistema desde diferentes perspectivas y las relaciones permiten ligar los elementos.

Las herramientas que se han elaborado para representar la arquitectura de software son los Lenguajes de Descripción de Arquitecturas (ADL). Sin embargo, son muy pocos quienes los utilizan como instrumento en el diseño arquitectónico de sus proyectos debido a que presentan diferentes problemas para su utilización, como:

- Requieren una extensa capacitación.
- No son amigables para presentar la arquitectura a personas ajenas a la construcción del software.
- No tienen herramientas ni metodologías de apoyo.
- Algunos se encuentran especializados solo en un tipo particular de sistemas.
- Sólo tienen en cuenta una sola estructura del sistema.

Esta es una de las razones por las que UML ha sido seleccionado como lenguaje de modelado para los laboratorios virtuales. Además es un lenguaje sencillo, visual, fácil de utilizar y comprender, que omite detalles específicos y permite la comprensión necesitada para el desarrollo de un sistema en concreto. En otras palabras simplifica la complejidad real, permite la reutilización de soluciones o de modelos, permite descubrir fallas y ahorra tiempo de desarrollo.

1.8. Herramientas de Ingeniería de Software Asistidas por Computadora.

Conocidas como herramientas CASE representan un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases. Brindan ayuda a los analistas, ingenieros de software y desarrolladores, permitiendo el modelado de los sistemas mediante diferentes diagramas y generación de código a partir de estos y viceversa. El uso de estas herramientas aumenta la calidad del software desarrollado, debido a que una funcionalidad se basa en la comprobación automática de errores. Además permiten la reutilización de componentes de software como librerías, acelera el proceso de desarrollo del software y permite un desarrollo gradual e iterativo.

Entre las herramientas CASE más utilizadas se encuentran **Visual Paradigm**. Su mayor éxito consiste en ser multiplataforma. Utiliza UML como lenguaje de modelado ofreciendo soluciones de software que permiten a las organizaciones desarrollar las aplicaciones de calidad más rápido, bien y más barato.

Otra de estas herramientas es **Rational Rose** una de las más poderosas herramientas de modelado visual para el análisis y diseño de sistemas basados en objetos. Se utiliza para modelar un sistema antes de proceder a construirlo. Además es la herramienta CASE que comercializan los desarrolladores de UML y que soporta de forma completa sus especificaciones. [11]

1.8.1. Selección de la Herramienta de Ingeniería de Software Asistida por Computadora.

Como herramienta CASE se va a hacer uso de Visual Paradigm para UML en su versión 6.4 ya que soporta el ciclo de vida completo del desarrollo de software permitiendo un software más robusto, con mayor calidad, menor costo y mejor entendimiento por parte delo equipo de desarrollo. Posee una interfaz amigable, profesional, permite modelar varios idiomas, realizar ingeniería directa e inversa y posee una ventaja muy importante que es útil para el desarrollo de los laboratorios virtuales y es su capacidad de ejecutarse en varios sistemas operativos.

Consideraciones del capítulo

En el capítulo se abordaron las principales temáticas que permiten comprender en qué se basa la descripción de una arquitectura de software dentro del desarrollo de software a nivel mundial así como el avance del uso de laboratorios virtuales. Se realizó un estudio de los elementos fundamentales que van a permitir estructurar la arquitectura de software a proponer y se seleccionó el lenguaje de programación, herramientas y metodología del desarrollo de software a partir de un estudio previo.

CAPITULO II: DESCRIPCIÓN DE LA ARQUITECTURA

Introducción

En el presente capítulo se describe la propuesta de arquitectura de software para los laboratorios virtuales de PROLAVI representada a través de las vistas de RUP. Se realiza además la selección de los elementos que la van a permitir estructurar.

2.1. Ambiente de desarrollo

A continuación se define el conjunto de herramientas que conforman el ambiente de desarrollo para la construcción de los laboratorios virtuales.

2.1.1. Entorno de desarrollo integrado

Un entorno de desarrollo integrado conocido como IDE es un programa informático compuesto por un conjunto de herramientas de programación. Pueden ser aplicaciones por si solos o pueden ser parte de aplicaciones existentes [12].

En el proyecto PROLAVI se hace uso de QT Creator como IDE de desarrollo por las características presentadas a continuación:

- Posee un avanzado editor de código C++.
- Es multiplataforma.
- Posee también una interfaz de usuario integrada y diseñador de formularios.
- Resaltado y auto-completado de código
- Soporte para refactorización de código.
- Se emplea para el desarrollo de aplicaciones de escritorio.

2.1.2. Motor gráfico

En el proyecto se hace uso del Motor gráfico orientado a objeto (OGRE) para visualizar las escenas y los objetos que integran los laboratorios. Su principal ventaja radica en que su uso es gratuito y solo existen unas pocas exigencias para el uso del mismo. Está diseñado desde el principio con la idea de la orientación a objetos, por lo que su interfaz es clara, intuitiva y fácil de usar.

2.1.3. Lenguaje de programación

C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos. Posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel como la sobrecarga de operadores y la identificación de tipos en tiempo de ejecución [11].

Permite trabajar tanto a alto como a bajo nivel. Posibilita programar desde sistemas operativos, compiladores, aplicaciones de bases de datos, procesadores de texto o juegos. Genera código eficiente, altamente transportable, flexible y con él se pueden realizar muchas funciones escribiendo pocas líneas de código.

Por todo lo anterior expuesto se selecciona este lenguaje para implementar los componentes de los laboratorios virtuales además de que es lenguaje que soporta la programación orientada a objetos lo que permite una mayor reutilización de dichos componentes.

2.1.4. Framework

En el desarrollo de software, un framework o componente es una estructura conceptual y tecnológica de soporte definida normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto. 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 [13].

Como framework de desarrollo en el proyecto se hace uso de QT ya que utiliza el lenguaje de programación C++ de forma nativa, es un framework para el desarrollo de aplicaciones multiplataforma, tiene disponibilidad de código fuente y una excelente documentación [13].

2.2. Selección de estilos y patrones arquitectónicos

En la mayor parte de la bibliografía consultada el término de estilo o patrón arquitectónico se utiliza de forma indiferente y quizás en alguna de las diferentes tendencias o clases que agrupan los tipos de arquitectura sea más significativo este

hecho de marcar las diferencias que solo radican principalmente en el nivel de abstracción.

En la práctica dentro la metodología RUP y la corriente de la arquitectura de software como etapa de la ingeniería y el diseño orientado a objeto la utilización del término estilo arquitectónico o patrón arquitectónico indistintamente no sobrepone relevancias, de ahí que los estilos y patrones arquitectónicos adecuados a utilizar para el diseño de la arquitectura de los laboratorios virtuales de PROLAVI sean los que se muestran a continuación.

2.2.1. Estilo de Llamada y Retorno

Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. Con este estilo se obtiene una estructura de programa que resulta relativamente fácil de modificar y cambiar de tamaño.

Entre sus características podemos encontrar [14]:

- Hilo de control simple soportado por los lenguajes de programación.
- Usa una estructura implícita de subsistemas.
- Reflejan la estructura del lenguaje de programación. Permite al diseñador del software construir una estructura de programa relativamente fácil de modificar y ajustar a escala.
- Se basan en la bien conocida abstracción de procedimientos/funciones/métodos.
- Utilizados en grandes sistemas de software.
- Persiguen escalabilidad y modificabilidad.

2.2.2. Arquitectura orientada a objeto

La arquitectura de los laboratorios virtuales estará basada en los principios de la programación orientada a objeto (POO): abstracción, herencia y polimorfismo. Los principales elementos de ella estarán centrados hacia los objetos y serán estos las unidades de modelado a partir de las clases que los definen interactuando a través de funciones y métodos.

Características [14]:

- Los componentes del estilo se basan en principios orientado a objeto: encapsulamiento, herencia y polimorfismo.

- Las interfaces están separadas de las implementaciones. En general la distribución de objetos es transparente, y en el estado de arte de la tecnología apenas importa si los objetos son locales o remotos.
- En cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases. Hay muchas variantes del estilo, algunos sistemas por ejemplo, admiten que los objetos sean tareas concurrentes, otros permiten que los objetos posean múltiples interfaces.

Ventajas [14]:

- Se puede modificar la implementación de un objeto sin afectar a sus clientes.
- Un objeto es una entidad reutilizable en el entorno de desarrollo.
- Encapsulamiento.

Desventajas [14]:

- Para invocar métodos de un objeto se debe conocer su identidad.

2.2.3. Arquitectura basada en componentes

Actualmente en el desarrollo de software hay una gran necesidad de hacer uso de la reutilización de partes o módulos de software existente, que podrían ser utilizadas para la generación de nuevas extensiones de las aplicaciones o las aplicaciones completas. Cuando se habla de reutilización en los procesos de ingeniería está muy implícito el concepto de componente, que representa una parte del software caracterizado por una interfaz.

Una de las características más importantes de los componentes es que son reutilizables. Para ello los componentes deben satisfacer como mínimo el siguiente conjunto de características [14]:

- **Identificable**. Un componente debe tener una identificación clara y consistente que facilite su catalogación y búsqueda en repositorios de componentes.
- **Accesible sólo a través de su interfaz**. El componente debe exponer al público únicamente el conjunto de operaciones que lo caracteriza y ocultar sus detalles de implementación. Esta característica permite que un componente sea reemplazado por otro que implemente la misma interfaz.

- **Servicios son invariantes**. Las operaciones que ofrece un componente, a través de su interfaz, no deben variar. La implementación de estos servicios puede ser modificada, pero no deben afectar la interfaz.
- **Documentado**. Un componente debe tener una documentación adecuada que facilite su búsqueda en repositorios de componentes, evaluación, adaptación a nuevos entornos, integración con otros componentes y acceso a información de soporte.

Ventajas [15]:

- **Reutilización del software**. Lleva a alcanzar un mayor nivel de reutilización de software.
- **Simplifica las pruebas**. Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- **Simplifica el mantenimiento del sistema.** Cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
- **Mayor calidad.** Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

Desventajas [15]:

- Si no existen los componentes hay que desarrollarlos por lo que se puede perder mucho tiempo.
- Las actualizaciones de los componentes adquiridos no están en manos de los desarrolladores del sistema.

2.2.4. Arquitectura en capas

Conocida también como arquitecturas estratificadas donde se crean diferentes capas y cada una realiza operaciones que progresivamente se aproximan más al cuadro de instrucciones de la máquina. En la capa externa, los componentes sirven a las operaciones de interfaz de usuario. En la capa interna, los componentes realizan operaciones de interfaz del sistema. Las capas intermedias proporcionan servicios de

utilidad y funciones de software de aplicaciones [15]. Esta arquitectura permite que un sistema se pueda estructurar jerárquicamente de manera que si se realizan cambios en una de las capas no se afecten las demás capas permitiendo la calidad del software así como una mejor organización del diseño del software. A continuación se muestran sus ventajas y desventajas.

Ventajas:

- Reutilización de capas.
- Facilita la estandarización.
- Contención de cambios a una o pocas capas.

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.

2.3. Selección de los patrones de diseño

En el diseño de la arquitectura propuesta para los laboratorios virtuales se hará uso de los siguientes patrones para asignación de responsabilidades: creador, experto, bajo acoplamiento además del patrón GoF singleton debido a las ventajas presentadas a continuación.

Patrón creador:

- Se selecciona este patrón porque permite el bajo acoplamiento, lo cual supone facilidad de mantenimiento y reutilización en los laboratorios que se desarrollen en el proyecto. La creación de instancias es una de las actividades más comunes en un sistema orientado a objeto. En consecuencia es útil contar con un principio general para la asignación de las responsabilidades de creación.

En la capa de interfaz de usuario la clase **Main Window** es la encargada de crear las instancias de las clases **Dialogo de Inicio** y **MainWidget.** En la capa de lógica del negocio la clase **FManage** es la encargada de crear la instancia de la clase **Error** y **ReportPDF** ya que contiene los datos necesarios de los objetos de dichas clases para crearla. En la capa de acceso a datos la clase creadora es **ComponentManager.**

Patrón experto:

- Se conserva el encapsulamiento de la orientación a objeto porque los objetos se valen de su propia información para hacer lo que se les pide.
- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases sencillas y más cohesivas, las cuales son fáciles de comprender y mantener.

En el diseño a proponer se posibilita la reutilización ya que el mismo se basa en componentes y las responsabilidades son asignadas a las clases que poseen la información de los objetos.

Patrón bajo acoplamiento:

- No afectan los cambios en otros componentes.
- Fácil de entender de manera aislada.
- Conveniente para reutilizar.

La arquitectura que se quiere proponer posee un bajo acoplamiento entre los componentes ya que la dependencia entre clases es mínima.

Patrón singleton:

El patrón singleton como se muestra en la **figura 1** asegura que exista una única instancia de una clase. A primera vista, uno puede pensar que pueden utilizarse clases con miembros estáticos para el mismo fin; mas los resultados no son los mismos, ya que en este caso la responsabilidad de tener una única instancia recae en el cliente de la clase. El patrón singleton hace que la clase sea responsable de su única instancia, quitando así este problema a los clientes.

El funcionamiento de este patrón es muy sencillo y podría reducirse a los siguientes conceptos [16]:

- Ocultar el constructor de la clase singleton, para que los clientes no puedan crear instancias.

- Declarar en la clase singleton una variable miembro privada que contenga la referencia a la instancia única que queremos gestionar.

- Proveer en la clase singleton una función o propiedad que brinde acceso a la única instancia gestionada por el singleton. Los clientes acceden a la instancia a través de esta función o propiedad.

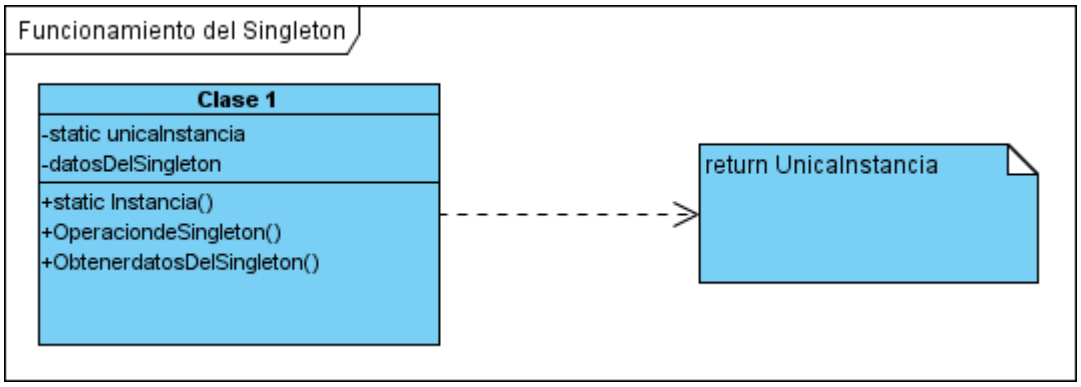

Figura 1: Estructura del patrón singleton.

En el caso de la arquitectura que se va a proponer este patrón se ve de manifiesto en las clases pertenecientes a OGRE que utiliza la máscara singleton para asegurarse que solo haya una instancia de cada. Significa que cuando se llame a los métodos por primera vez haciendo referencia a las clases que los definieron se verifica que si el estado es falso se crea el objeto y se cambia el estado a verdadero evitando que durante la sesión se pueda instanciar otro objeto de la misma clase.

2.4. Representación arquitectónica

La arquitectura de software es muy importante en el desarrollo de un sistema porque permite representar de forma concreta la estructura y funcionamiento interno del mismo. Dicha arquitectura se va a representar desde el punto de vista de la metodología RUP basada en diferentes vistas que juntan los elementos más significativos del desarrollo de un sistema. Las vistas mencionadas están organizadas en el documento en el siguiente orden:

- **Vista de casos de uso**. Representa el modelo de casos de uso, el diagrama de casos de uso arquitectónicamente significativo para la arquitectura además de una descripción de cada uno de los casos de usos.

- **Vista lógica**. Permite descomponer el modelo de diseño en subsistemas, sus interfaces y las dependencias entre ellos, ya que constituyen la estructura fundamental del sistema. Además se incluye una descripción de las clases más importantes, su organización en paquetes y subsistemas.

- **Vista de despliegue.** Suministra una base para la comprensión de la distribución física de un sistema a través de nodos mediante el diagrama de despliegue.

- **Vista de implementación**. Describe la estructura general del modelo de implementación, en correspondencia con la vista lógica, se representa la descomposición del software en paquetes importantes para la arquitectura.

2.5. Objetivos y restricciones arquitectónicas

La IEEE define un requerimiento como condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo. Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. Normalmente están vinculados a requisitos funcionales que son capacidades o condiciones que el sistema debe cumplir. Una vez se conozca lo que el sistema debe hacer podemos determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser [17].

El desarrollo de los laboratorios virtuales tiene como objetivo principal proporcionar una herramienta de apoyo a la docencia que cumpla con los requisitos no funcionales y restricciones siguientes, los cuales tienen un impacto significativo para la arquitectura.

Requerimientos de software

RNF1. Las PC deben tener instalado, Windows 2000 o superior o alguna distribución de GNU/Linux.

Requisitos de hardware.

RNF2. PC con 256 MB de memoria RAM como mínimo.
RNF3. Microprocesador Intel Pentium 4 a 3.00GHz.

Requisitos de usabilidad

RNF4: Los Laboratorios Virtuales deben tener una interfaz organizada y de fácil entendimiento para el usuario.

Requisitos de eficiencia

RNF5: El tiempo promedio de respuesta del sistema no deberá ser mayor de 4 segundos.

Requisitos de soporte

RNF6: El sistema debe ser multiplataforma**.**

Restricciones de diseño o implementación

RNF7. El sistema será una aplicación de escritorio.

RNF8. El sistema debe ser implementado utilizando el lenguaje C++.

RNF9. El sistema deberá utilizar como IDE de desarrollo QT Creator.

RNF10. El sistema usará como herramienta de modelación el Visual Paradigm for UML 6.4 Enterprise Edition.

RNF11. Como motor gráfico se debe utilizar OGRE.

RNF12. Como framework se debe hacer uso de QT.

2.6. Vistas de la arquitectura del sistema

2.6.1. Vista de casos de usos

A partir de la vista de casos de usos, se puede definir los escenarios o los casos de usos que serán de interés para cada iteración del ciclo de desarrollo así como los actores que interactuarán con el sistema.

Un caso de uso ayuda a idear y a desarrollar una arquitectura de software permitiendo así llevar a cabo el proceso iterativo de la misma. Mediante la selección de los casos de usos arquitectónicamente significativos para llevarlos a cabo durante las primeras iteraciones se puede implementar un sistema con una arquitectura estable que pueda utilizarse en muchos ciclos de desarrollos siguientes [17].

Para el desarrollo de los laboratorios virtuales se definieron 13 requisitos funcionales básicos que fueron agrupados en 9 casos de uso como muestra la **figura 2** del sistema de acuerdo a su impacto en la arquitectura.

Requisitos funcionales de los laboratorios virtuales

RF 1. Autenticar usuario.

RF 2. Agregar objetos a la escena.

RF 3. Eliminar objetos de la escena.

RF 4. Modificar posición de objetos de la escena.

RF 5. Integrar objetos en la escena.

RF 6. Navegar en la escena.

RF 7. Exportar práctica de laboratorio

RF 8 . Cargar práctica de laboratorio.

RF 10. Salvar reporte de la práctica.

RF 11. Mostrar errores cometidos.

RF 12. Mostrar el tiempo de duración de la práctica.

RF 13. Cargar repositorio de componentes.

Casos de Usos del sistema		
Críticos	Secundarios	Auxiliares
Autenticar	Exportar práctica de laboratorio.	Mostrar el tiempo de duración de la práctica
Gestionar objetos de la escena.	Cargar práctica de laboratorio	
Cargar repositorio de componentes		
Salvar reporte de la práctica		
Navegar en la escena		
Mostrar errores cometidos		

Tabla 1: Casos de usos del sistema

- Los casos de uso críticos contienen las principales funcionalidades que el sistema debe cumplir y que son de vital importancia para los usuarios, por lo que definen la arquitectura básica.
- Los casos de uso secundarios sirven de apoyo a los casos de uso críticos, contienen aquellas funcionalidades de menor impacto sobre la arquitectura, aunque también deben implementarse tempranamente pues responden a requisitos de interés para los usuarios.
- Los casos de uso auxiliares no son claves para la arquitectura y son usados sobre todo para completar casos de usos críticos o secundarios.

Actores del sistema

- **Usuario.** El actor usuario es genérico ya que la práctica la puede realizar tanto un estudiante como un profesor.

Diagrama de casos de usos del sistema

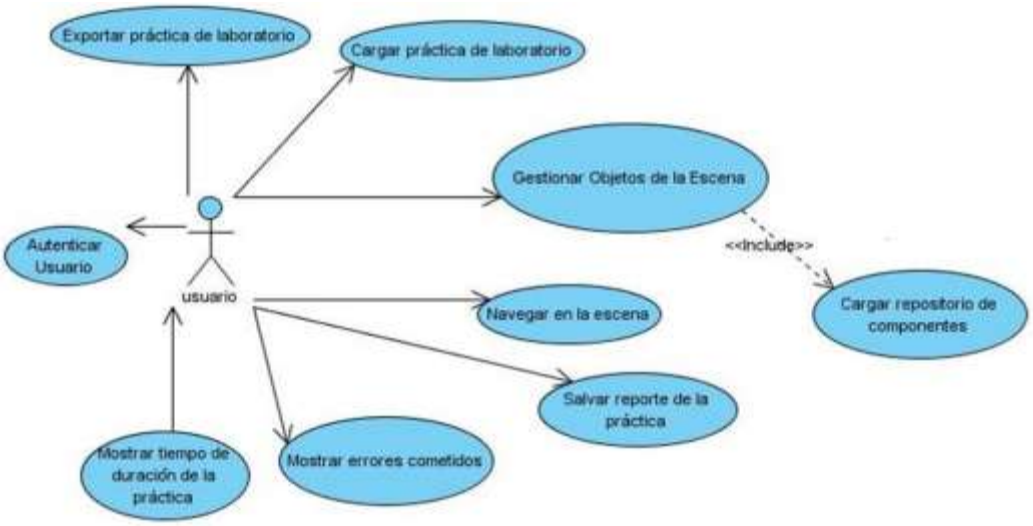

Figura 2: Diagrama de casos de uso del sistema.

Especificación de casos de usos (Ver Anexo 1)

Diagrama de casos de usos arquitectónicamente significativos

Los casos de usos arquitectónicamente significativos (CUAS) expuestos en el diagrama de la **figura 3** describen las principales funcionalidades para el software y permiten verificar que la arquitectura propuesta esté correcta.

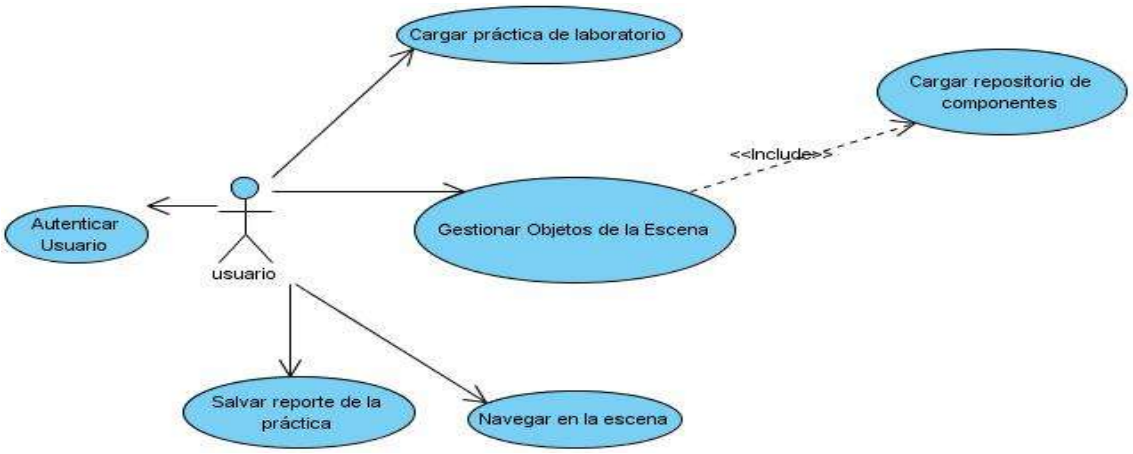

Figura 3: Diagrama de casos de uso arquitectónicamente significativos.

A continuación se muestran los diagramas de secuencia de los casos de usos que tienen un mayor significado para la arquitectura.

Diagrama de secuencia del CU: Autenticar

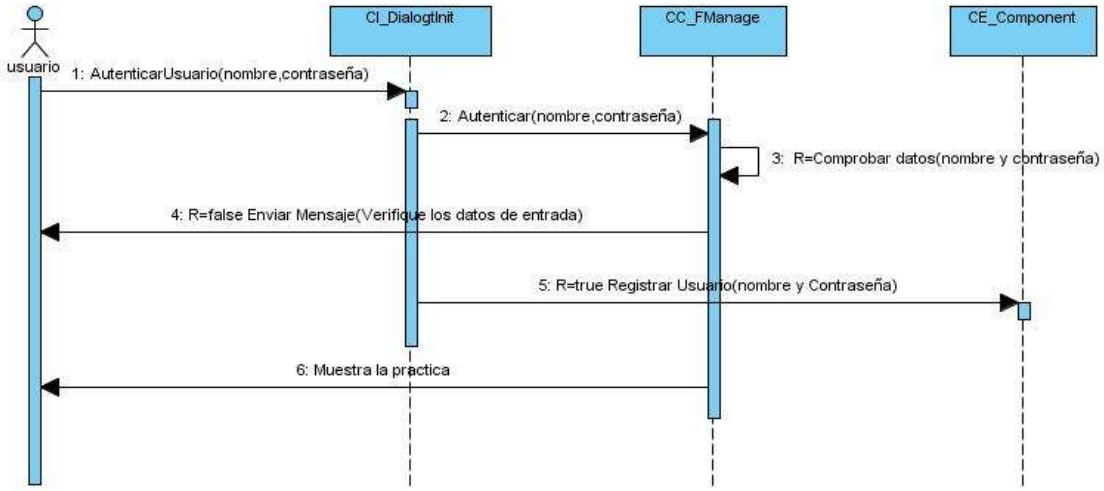

Figura 4: Diagrama de secuencia del CU Autenticar.

Diagrama de secuencia del CU: Salvar reporte de la práctica

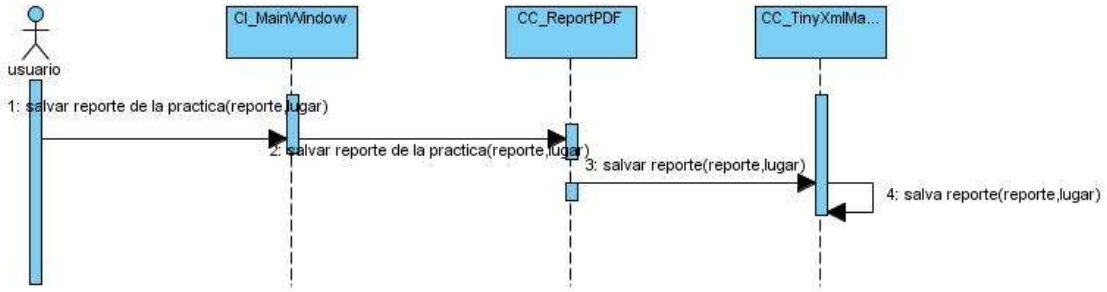

Figura 5: Diagrama de secuencia del CU Salvar reporte de la práctica.

Diagrama de secuencia del CU: Mostrar errores cometidos

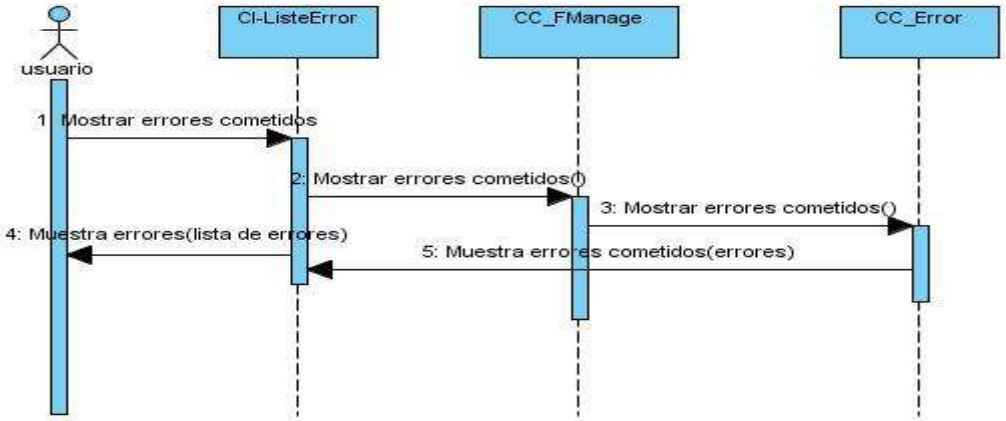

Figura 6: Diagrama de secuencia del CU Mostrar errores cometidos.

Diagrama de secuencia del CU: Gestionar objetos de la escena

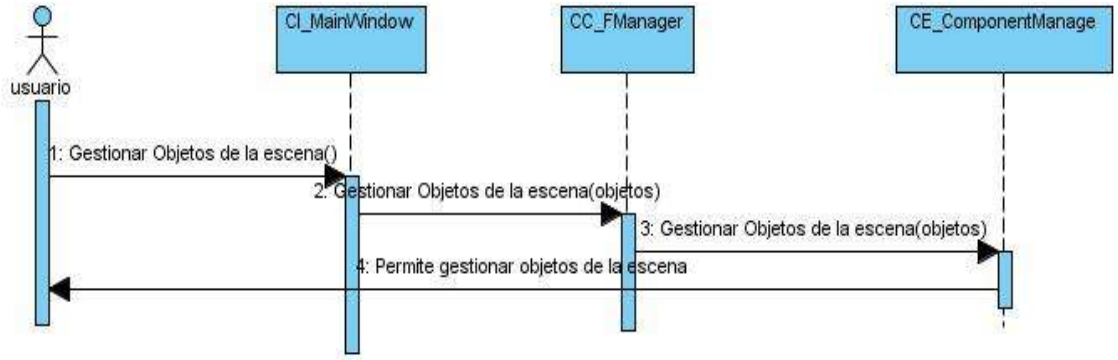

Figura 7: Diagrama de secuencia del CU gestionar objetos de la escena.

Diagrama de secuencia del CU: Exportar práctica de laboratorio

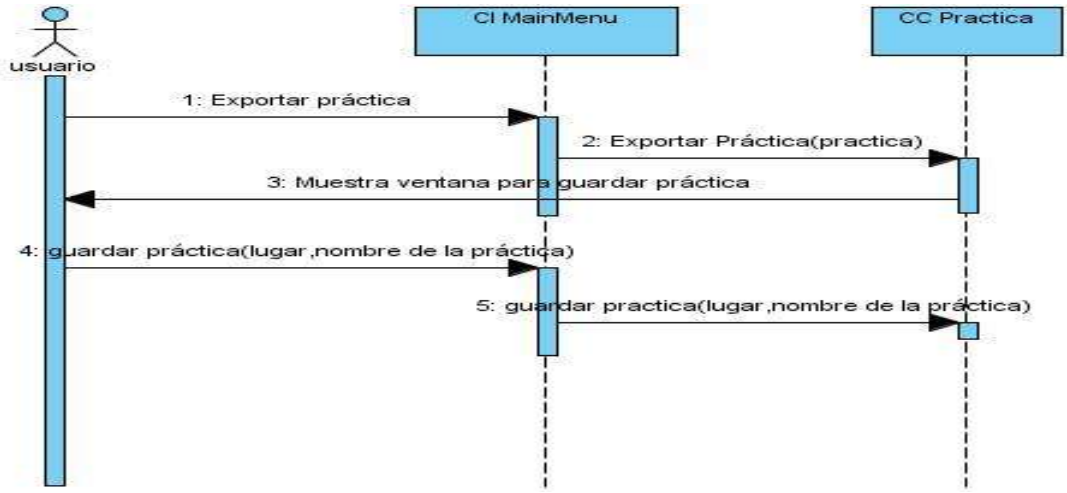

Figura 8: Diagrama de secuencia del CU Exportar práctica de laboratorio.

2.6.2. Vista lógica

La vista lógica como se muestra en la **figura 9** comprende los elementos del diseño significativos para la arquitectura. Esta vista representa un subconjunto del artefacto modelo de diseño, representando los elementos de diseño más importantes para la arquitectura del sistema, aquí se describen las clases más importantes, su organización en paquetes y subsistemas. Esta descripción se realiza a través de diagramas de clases para ilustrar la relación entre las clases arquitectónicamente significativas, subsistemas y paquetes. El diagrama de la **figura 9** muestra cómo va queda estructurado la arquitectura del sistema en capas donde:

- La capa **Interfaz de Usuario** contiene las interfaces con las que el usuario va a interactuar, captura la información entrada por él y hace las peticiones a la capa inferior mostrando al usuario la respuesta proveniente de ella. Únicamente se comunica con la capa de lógica del sistema. La misma está integrada por el paquete **Interfaces** que va a contener las clases que implementan las interfaces con las que el usuario interactúa mediante los componentes del paquete framework QT que usa las librerías de OGRE para visualizar los modelos 3D.

- En la capa de **Lógica de Sistema** se asocia aquellos componentes arquitectónicamente significativos para el sistema. El paquete **Clases de Control**

contiene las clases controladoras que integran los componentes de la capa. Solo tendrá acceso a la capa de Acceso a Datos.

- La Capa **Acceso a Datos** contiene los paquetes **Clase de Datos** y **Repositorio de imágenes y modelos 3D** que contienen las clases persistentes del sistema.

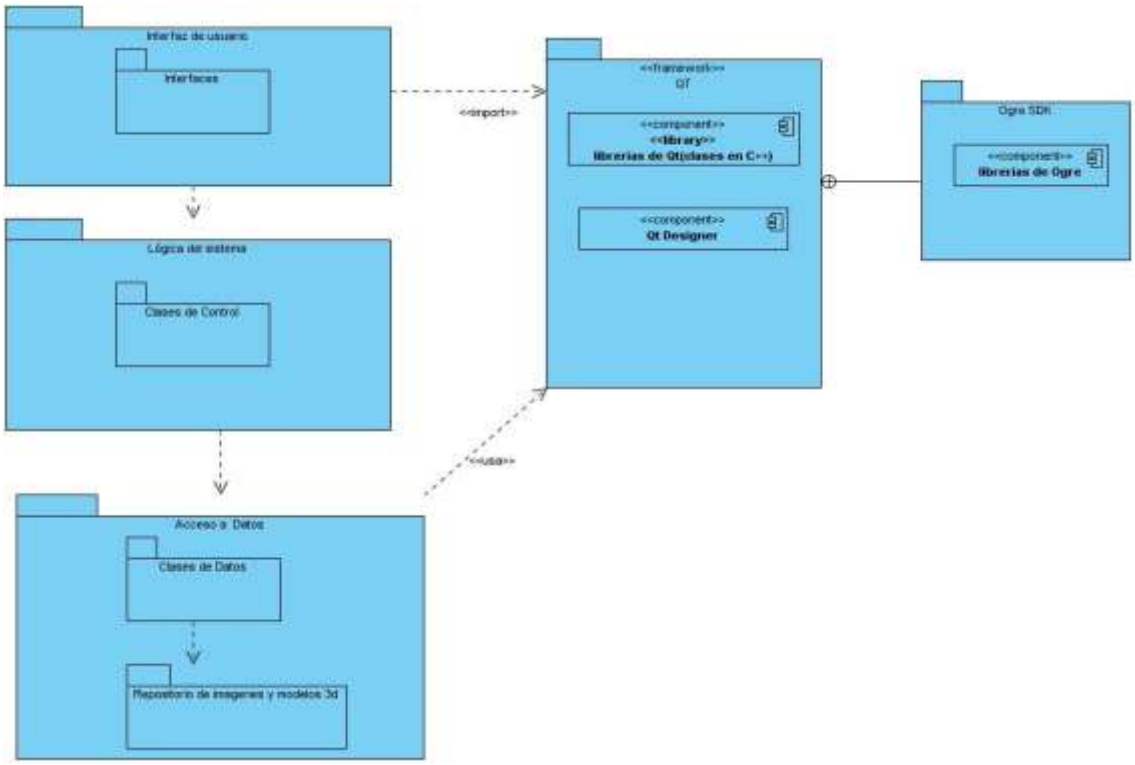

Figura 9: Diagrama de paquetes.

A continuación se muestra una vista general de las clases principales que debe contener cada capa:

- **Capa Interfaz de Usuario**

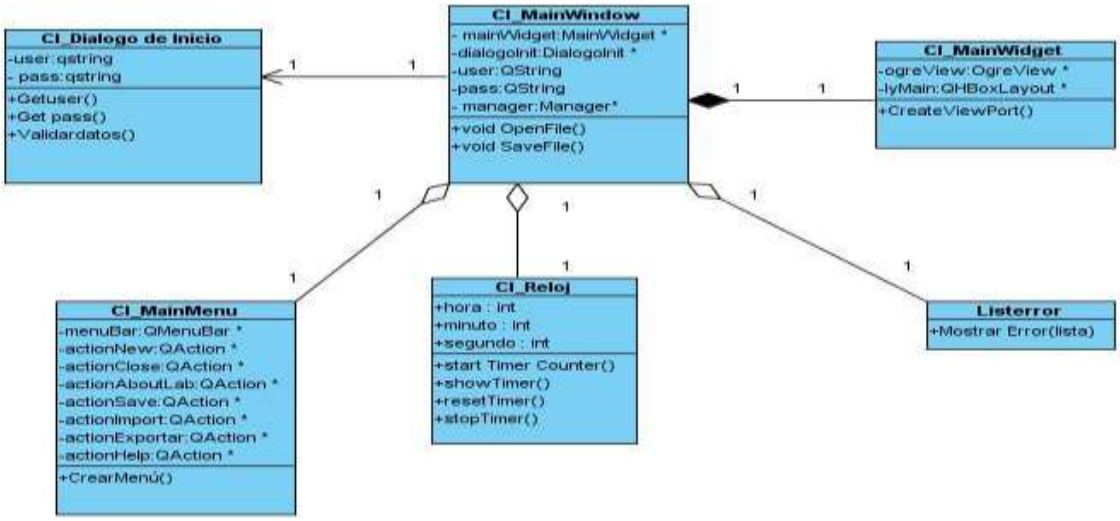

Figura 10: Diagrama de clases del paquete Interfaz de Usuario.

Cl_Dialogo de Inicio. Esta clase es la encargada de pedirle los datos de entrada al usuario para poder registrarlo en la aplicación.

Cl_Reloj. Es la clase encargada de mostrar el tiempo de realización de la práctica.

Cl_Main Menu. Es la clase principal encargada de mostrar el menú de la aplicación.

Listeerror. Es la clase encargada de mostrar en una lista los errores que el usuario cometió durante la realización total o parcial de la práctica.

Cl_MainWindow. Es la clase principal encargada de permitirle al usuario abrir un fichero para cargarlo en la práctica o guardar si desea la práctica de laboratorio una vez realizada.

Cl_MainWidget. Es la clase encargada de que el usuario pueda visualizar todo lo que la escena y los objetos que la integran.

Capa Lógica del Sistema

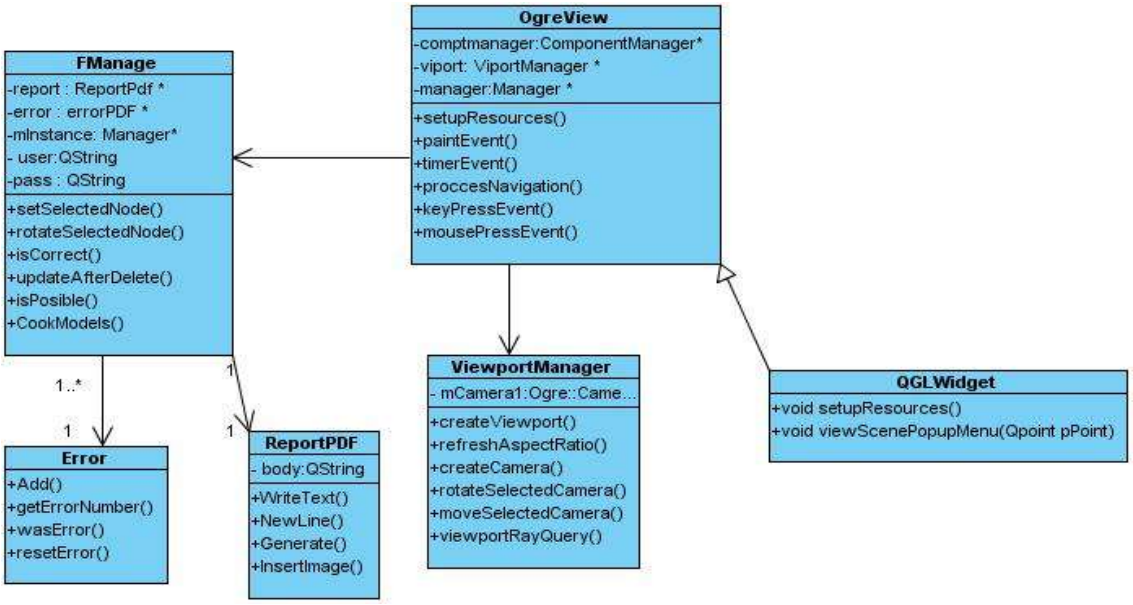

Figura 11: Diagrama de clases del paquete Lógica del Sistema.

FManage. Es la clase encargada de cambiar la posición de los objetos de la escena, de actualizando cada posición de movimiento de los mismos así como de validar los datos de entrada al sistema del usuario.

Error. Esta clase permite saber cuáles son los errores que se han cometido en la práctica.

ReportPDF. Clase que contiene los datos para crear el PDF de la practica realizada.

OgreView. Esta clase va a ser la encargada de que se pueda visualizar la escena de la práctica de laboratorio y va a permitir que se pueda dibujar en la misma.

ViewportMange. La funcionalidad de esta clase es que la cámara se pueda mover por toda la escena.

QGLWidge. Esta clase es la encargada de renderizar la escena.

Capa Acceso a Datos

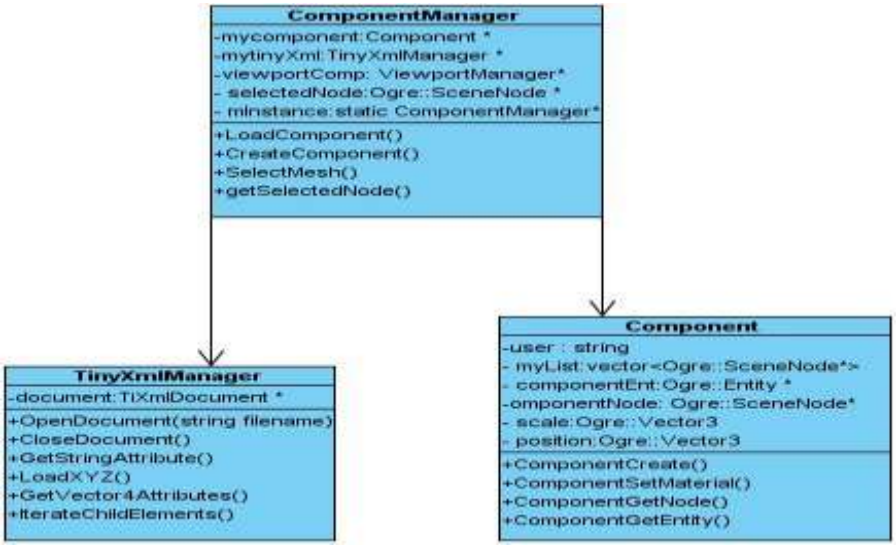

Figura 12: Diagrama de clases del paquete Acceso a Datos.

ComponentManager. Esta clase es la que permite el trabajo con los componentes de la escena para seleccionar con la que se desea trabajar así como los nodos que se van a utilizar.

Component. Esta clase se encarga de crear las entidades y los componentes con los cuales se van a trabajar en la escena y de guardar los datos del usuario que está interactuando con el sistema.

TinyXmlManager. Clase que va a permitir la lectura de documentos desde los archivos XML.

2.6.3. Vista de despliegue

La vista de despliegue permite ver el sistema en términos de nodos de procesamiento, servidores o dispositivos mediante el diagrama de despliegue como se muestra en la **figura 13**. Muestra la comunicación entre los diferentes nodos que componen los escenarios de distribución física del sistema. Los diagramas de despliegue muestran la configuración en funcionamiento del sistema, incluyendo hardware y software.

Para realizar el despliegue de un laboratorio virtual se requiere de una o varias PC clientes, donde se ejecutará el .**exe** del laboratorio virtual. Las computadoras deben cumplir con los requisitos de hardware y software anteriormente expuestos.

Figura 13: Diagrama de despliegue del sistema.

Nodo PC Cliente. Contiene el ejecutable de la aplicación.

2.6.4. Vista de implementación

La vista de implementación describe la descomposición del software en componentes y subsistemas de implementación. Un componente de software constituye una parte física de un sistema, ya sea un módulo, una base de datos entre otros. Algunos de los elementos de esta vista son:

- Subsistemas de implementación.
- Diagramas de componentes.

A continuación se muestra el diagrama de componentes por cada capa así como la explicación de los mismos.

Diagrama de componentes de la capa Interfaz de Usuario

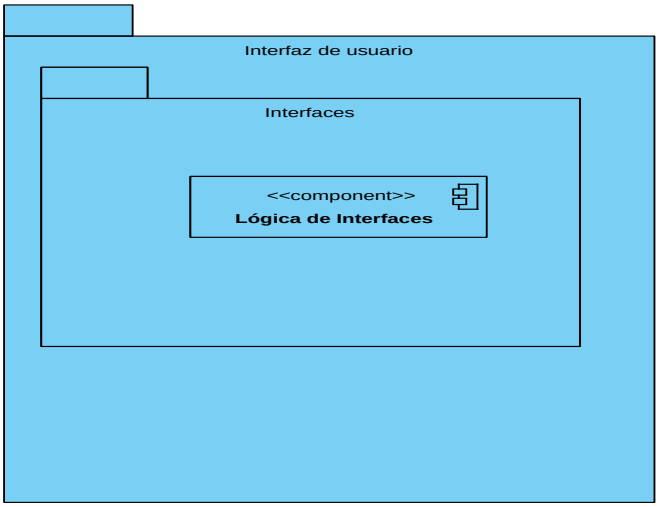

Figura 14: Diagrama de compontes de la capa Presentación.

El componente **Lógicas de Interfaz** contiene las clases que permiten al usuario interactuar con el sistema.

Diagrama de componentes de la capa Lógica del Sistema

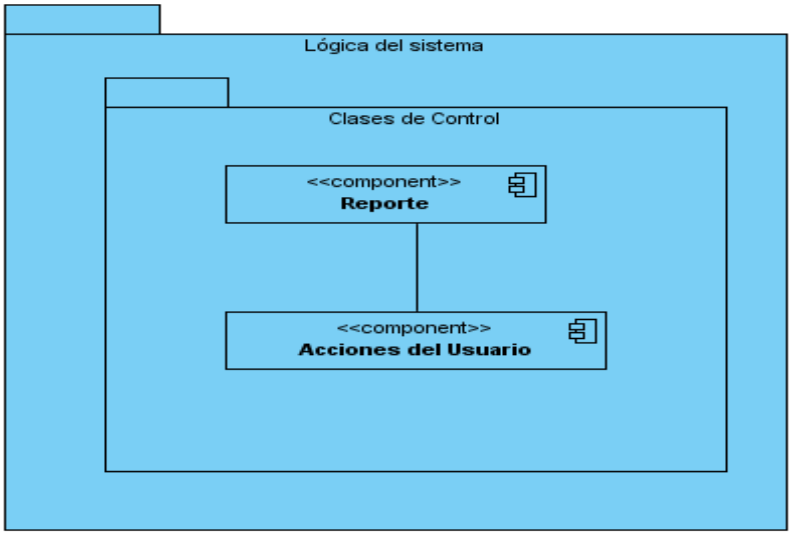

Figura 15: Diagrama de compontes del paquete Lógica del sistema.

El componente **Reporte** contiene las clases que controlan la creación del reporte en PDF para un mejor estudio del usuario. El componente **Acciones del Usuario** será el encargado de contener las clases controladoras de las funciones principales que se deben realizar en un laboratorio virtual.

Diagrama de componentes del paquete Acceso a Datos

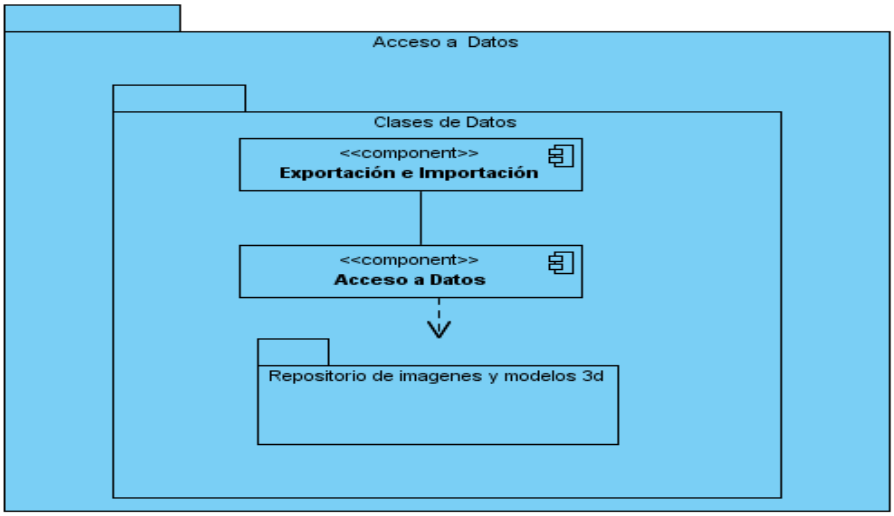

Figura 16: Diagrama de compontes del paquete Acceso a Datos.

El paquete de Acceso a Datos está formado por el componente **Exportar e Importar** que contiene las clases que permitan cargar o exportar un fichero xml con la ayuda de QT. El componente **Acceso a Datos** contiene las clases persistentes encargadas de acceder a las imágenes y modelos 3D ubicadas en el paquete Repositorio de componentes.

Consideraciones del capítulo

En este capítulo se describió la arquitectura de software a proponer para el desarrollo de un laboratorio virtual definiendo para ello los estilos, patrones de diseño y de arquitectura así como IDE, framework y lenguaje de programación a emplear en el desarrollo de un laboratorio virtual, además se describió y modeló cada una de las vistas que conforman la propuesta.

CAPITULO III: VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN

Introducción

La evaluación de una arquitectura de software permite identificar dónde están los riesgos, fortalezas y debilidades identificadas de dicha arquitectura. Si un sistema no cumple con los atributos de calidad que se que especifican en los requisitos no funcionales entonces el producto final no va a cumplir con las expectativas de los clientes es por ello que diseñar una correcta arquitectura va a determinar el éxito o fracaso de un sistema de software, en la medida que esta cumpla o no con sus objetivos.

Debido a esto “Para reducir tales riesgos, y como buena práctica de ingeniería, es recomendable realizar evaluaciones a la arquitectura” [18]. En el presente capítulo se realiza un estudio de los métodos y técnicas de validación existentes para realizar una buena selección de los mismos y aplicarlo a la arquitectura que se quiere proponer para ver si la misma está correctamente diseñada .

3.1. Objetivos de evaluar una arquitectura

El objetivo de evaluar una arquitectura es saber si puede habilitar los requisitos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los involucrados [19].

La evaluación de una arquitectura de software es una tarea no superficial, puesto que se pretende medir propiedades del sistema en base a especificaciones abstractas, como por ejemplo los diseños arquitectónicos. Por ello, la intención es más bien la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos.

Las mediciones que se realizan sobre una arquitectura de software pueden tener distintos objetivos, dependiendo de la situación en la que se encuentre el arquitecto y la aplicabilidad de las técnicas que emplea. Algunos de estos objetivos son: cualitativos, cuantitativos y máximos y mínimos teóricos [19].

• La medición cualitativa se aplica para la comparación entre arquitecturas candidatas y tiene relación con la intención de saber la opción que se adapta mejor a cierto

atributo de calidad. Este tipo de medición brinda respuestas afirmativas o negativas, sin mayor nivel de detalle.

• La medición cuantitativa busca la obtención de valores que permitan tomar decisiones en cuanto a los atributos de calidad de una Arquitectura de Software.

3.2. ¿Cuándo una arquitectura puede ser evaluada?

Es posible realizarla en cualquier momento según Kazman, pero propone dos variantes que agrupan dos etapas distintas: temprano y tarde [15].

• **Temprana**. No es necesario que la arquitectura esté completamente especificada para efectuar la evaluación, y esto abarca desde las fases tempranas de diseño y a lo largo del desarrollo.

• **Tarde**. Cuando la arquitectura de software se encuentra establecida y la implementación se ha completado. Este es el caso general que se presenta al momento de la adquisición de un sistema ya desarrollado.

3.3. Atributos por los cuales puede ser evaluada una arquitectura

Atributo de Calidad	Descripción
Disponibilidad	Es la medida de disponibilidad del sistema para el uso.
Confidencialidad	Es la ausencia de acceso no autorizado a la información.
Funcionalidad	Habilidad del sistema para realizar el trabajo para el cual fue concebido.
Desempeño	Es el grado en el cual un sistema o componente cumple con sus funciones designadas dentro de ciertas restricciones dadas como: velocidad, exactitud o uso de memoria.
Confiabilidad	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo.
Seguridad externa	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información.
Seguridad interna	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos.

Configurabilidad	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema.
Integrabilidad	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.
Integridad	Es la ausencia de alteraciones inapropiadas de la información.

Tabla 2: Atributos para evaluar la calidad de una arquitectura de software.

3.4. Métodos y técnicas de evaluación de la arquitectura de software

En la actualidad existen varios planteamientos sobre como evaluar la arquitectura de software pero los más usados se refieren a los planteados por Bosch y Kazman los cuales exponen que la evaluación de la arquitectura puede ser realizada mediante el uso de diversas técnicas y métodos, mostrados en un estudio a continuación.

3.4. 1. Técnicas de evaluación

Las técnicas de evaluación de arquitectura de software se utilizan para la evaluación de atributos de calidad y requieren grandes esfuerzos por parte de los ingenieros de software para crear especificaciones y predicciones respecto a si la propuesta arquitectónica satisface las cualidades del Software. Las mismas se clasifican en cualitativas y cuantitativas como se muestra a continuación.

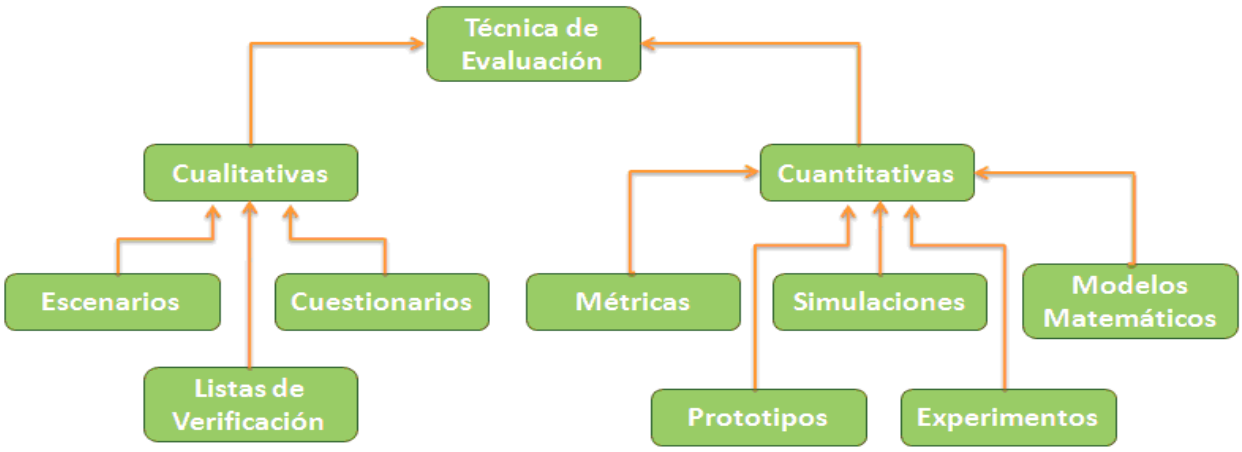

Figura 17: Clasificación de las técnicas de evaluación de la arquitectura de software.

3.4.2. Métodos para evaluar la arquitectura

SAAM: Conocido como Método de Análisis de Arquitecturas de Software es el primero que fue ampliamente promulgado y documentado. Originalmente creado para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad.

Este método se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro. Con la aplicación de este método, si el objetivo de la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requisitos de modificabilidad [15].

ATAM: El Método de Análisis de Acuerdos de Arquitectura como también se le conoce está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM. El nombre del método ATAM surge del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros.

El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. Estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas, entre otros [15].

ADR: Conocido como Revisiones Activas de Diseño es utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos. Las preguntas giran en torno a la calidad y completitud de la documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto [15].

ARID: Conocido como Revisiones Activas de Diseño Inmediato es un método de bajo costo y gran beneficio, el mismo es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. ARID es un híbrido entre ADR y ATAM. Se basa en ensamblar el diseño para articular los escenarios de usos importantes o significativos y probar el diseño para ver si satisface los escenarios.

Como resultado de la aplicación de dicho procedimiento se obtiene un diseño de alta fidelidad acompañado de una alta familiarización con el diseño de los terceros. Este método consta de 9 pasos agrupados en dos fases [15].

Losavio (2003): Es un método contemplado por siete actividades que se usa para evaluar y comparar arquitecturas de software candidatas, que hace uso del modelo de especificación de atributos e calidad adaptado del modelo ISO/IEC 9126. La especificación de los atributos de calidad haciendo uso de un modelo basado en estándares internacionales ofrece una vista amplia y global de los atributos de calidad, tanto a usuarios como arquitectos del sistema para efectos de la evaluación [15].

A continuación se muestra una tabla de comparación de los métodos explicados anteriormente.

	ATAM	SAAM	ARID	Bosch(2000)	Losavio(2003)
Atributos de Calidad Contemplados	Modificabilidad Seguridad Confiabilidad Desempeño	Modificabilidad Funcionabili-dad	Conveniencia del diseño evaluado	Seleccionados por el arquitecto, de acuerdo a la importancia sobre el sistema	Funcionabilidad Confiabilidad Usabilidad Eficiencia Mantenimiento Portabilidad
Objetos Analizados	Estilos Arquitectónicos, Documentación, Flujo de Datos y Vistas Arquitectónicas.	Documentación y Vistas Arquitectónicas	Especificación de los componentes	Estilos Arquitectónicos, Vistas Arquitectónicas, Patrones Arquitectónicos, Patrones de Diseño y Patrones de Idioma.	Especificación de Atributos de Calidad
Etapas del Proyecto en las que se aplica	Luego que el diseño de la arquitectura ha sido establecido	Luego que la arquitectura cuenta con funcionalidad ubicada en módulos	A lo largo del diseño de la arquitectura	Luego que el diseño de la arquitectura ha sido establecido.	Luego que el diseño de la arquitectura ha sido establecido.
Enfoques Utilizados	Árbol de utilidad y lluvia de ideas para articular los requisitos de calidad. Análisis arquitectónico que detecta	Lluvia de ideas para escenarios y articular los requisitos de calidad Análisis de los escenarios para verificar	Revisiones de diseño, lluvia de ideas para obtener escenarios	Análisis de perfiles	Análisis y comparación con de los resultados para las arquitecturas candidatas.

	puntos sensibles, puntos de balance y riesgos.	funcionalidad o estimar el costo de los cambios.			

Tabla 3: Comparación entre métodos de evaluación [15].

3.5. Selección de la técnica y método de evaluación a utilizar

Para evaluar la arquitectura que se quiere proponer se hará uso del método ARID ya que al ser una propuesta se hace de manera temprana en relación con el desarrollo del software como tal. Por esto se necesita la evaluación en las fases tempranas del diseño, se propone la utilización de las características que proveen tanto ADR como ATAM por separado. De ADR, resulta conveniente la fidelidad de las respuestas que se obtiene de los involucrados en el desarrollo.

Así mismo, la idea del uso de escenarios generados por los involucrados con el sistema es tomada del ATAM. De la combinación de ambas filosofías surge ARID para efecto de la evaluación temprana de los diseños de una arquitectura de software que es el método utilizado en la evaluación de este trabajo. Se decidió pre-validar la arquitectura evaluando cómo las vistas seleccionadas responden a cada uno de los principales requisitos y restricciones del sistema enfocándose en el uso de la técnica cualitativa basada en escenario ya que los mismos establecen un vehículo que permite concretar y entender los atributos de calidad.

Algunos autores consideran que los escenarios son una herramienta importante para relacionar vistas arquitectónicas, porque recorriendo un escenario puede mostrar las formas en que fragmentos o escenas de esas vistas se corresponden entre sí [15].

3.5.1. Pasos del método seleccionado

Una evaluación ARID progresa a través de dos fases que abarquen nueve pasos.

Fase 1: Actividades Previas	
1. Identificación de los encargados de la revisión	Los encargados de la revisión son los ingenieros de software que se espera que usen el diseño, y todos los involucrados en el diseño. En este punto, converge el concepto de *encargado de revisión* de ADR e *involucrado* del ATAM.
2. Preparar el informe De diseño	El diseñador prepara un informe que explica el diseño. Se incluyen ejemplos del uso del mismo para la resolución de problemas reales. Esto permite al facilitador anticipar el tipo de preguntas posibles, así como identificar áreas en las que la presentación puede ser mejorada.
3. Preparar los escenarios base	El diseñador y el facilitador preparan un conjunto de escenarios base. De forma similar a los escenarios del ATAM y el SAAM, se diseñan para ilustrar el concepto de escenario, que pueden o no ser utilizados para efectos de la evaluación.
4. Preparar los materiales	Se reproducen los materiales preparados para ser presentados en la segunda fase. Se establece la reunión, y los involucrados son invitados.
Fase 2: Revisión	
5. Presentación del ARID	Se explica los pasos del ARID a los participantes.
6. Presentación del diseño	El líder del equipo de diseño realiza una presentación, con ejemplos incluidos. Se propone evitar preguntas que conciernen a la implementación o argumentación, así como alternativas de diseño. El objetivo es verificar que el diseño es conveniente.
7. Lluvia de ideas y establecimiento de prioridad de escenarios	Se establece una sesión para la lluvia de ideas sobre los escenarios y el establecimiento de prioridad de escenarios. Los involucrados proponen escenarios a ser usados en el diseño para resolver problemas que esperan encontrar. Luego, los escenarios son sometidos a votación, y se utilizan los que resultan ganadores para hacer pruebas sobre el diseño.
8. Aplicación de los escenarios	Comenzando con el escenario que contó con más votos, el facilitador solicita pseudo-código que utiliza el diseño para proveer el servicio, y el diseñador no debe ayudar en esta tarea. Este paso continúa hasta que ocurra alguno de los siguientes eventos: ✓ Se agota el tiempo destinado a la revisión ✓ Se han estudiado los escenarios de más alta prioridad ✓ El grupo se siente satisfecho con la conclusión alcanzada. Puede suceder que el diseño presentado sea conveniente, con la exitosa aplicación de los escenarios, o por el contrario, no conveniente, cuando el grupo encuentra problemas o deficiencias.
9. Resumen	Al final, el facilitador recuenta la lista de puntos tratados, pide opiniones de los participantes sobre la eficiencia del ejercicio de revisión, y agradece por su participación.

Figura 18: Fases del método ARID.

Atributos de calidad contemplados. Los que el equipo seleccione.

Objetos analizados. Detalles de los componentes.

Etapa del proyecto en la que se aplica. A lo largo del diseño de la arquitectura.

Enfoques utilizados. Revisiones de diseños, lluvias de ideas para obtener escenarios.

3.5.2. Árbol de utilidades

Para la especificación de la calidad se hace uso de un árbol de utilidad, el cual tiene como nodo raíz la “bondad” o “utilidad” del sistema. En el segundo nivel del árbol se encuentran los atributos de calidad. Las hojas del árbol de utilidad son escenarios, los que representan mecanismos mediante los cuales extensas (y ambiguas) declaraciones de cualidades son hechas específicas y posibles de evaluar.

La salida de la generación del árbol de utilidad es una lista priorizada de los requisitos de los atributos de calidad comprendidos como escenarios. Le dice al equipo de evaluación donde gastar su tiempo y en particular donde buscar propuestas arquitectónicas y riesgos. El árbol de utilidad provee un mecanismo que en forma directa y eficiente traduce las pautas del negocio del sistema en escenarios concretos de los atributos de calidad. Los participantes priorizan al árbol de utilidad en dos dimensiones [15]:

- Por la importancia que cada escenario tiene para el éxito del sistema(X).
- Por el grado de dificultad que posee el escenario para ser realizado según la estimación del arquitecto(Y).

Habitualmente la escala utilizada para ambas dimensiones es alto(A), medio(M) y bajo(B).

3.6. Evaluando la arquitectura de software propuesta

La selección de los escenarios concretos para validar que la arquitectura cumpliese con requisitos específicos se realizó en base a las cualidades o capacidades que la arquitectura debe cumplir hasta el estado en que se encuentra desarrollada .Se consideró que el arquitecto de software realizará una breve evaluación de la arquitectura con vista a evaluar los atributos de calidad considerados de mayor importan según las características del tipo de sistema a desarrollar. Para la evaluación de la arquitectura se seleccionaron los siguientes atributos de calidad para evaluar: seguridad (A, A), integrabilidad(A, A), modularidad(A, A), modificabilidad (A, A) y desempeño (M, M).Además se muestra un ejemplo en el **anexo 2** de la prueba del funcionamiento de un módulo de una práctica de un laboratorio virtual que utiliza una parte de la arquitectura propuesta así la integración del mismo.

3.6.1. Vista del árbol de utilidad

A continuación se muestra como quedó el árbol de utilidad una vez seleccionado los atributos y escenarios para evaluar la arquitectura.

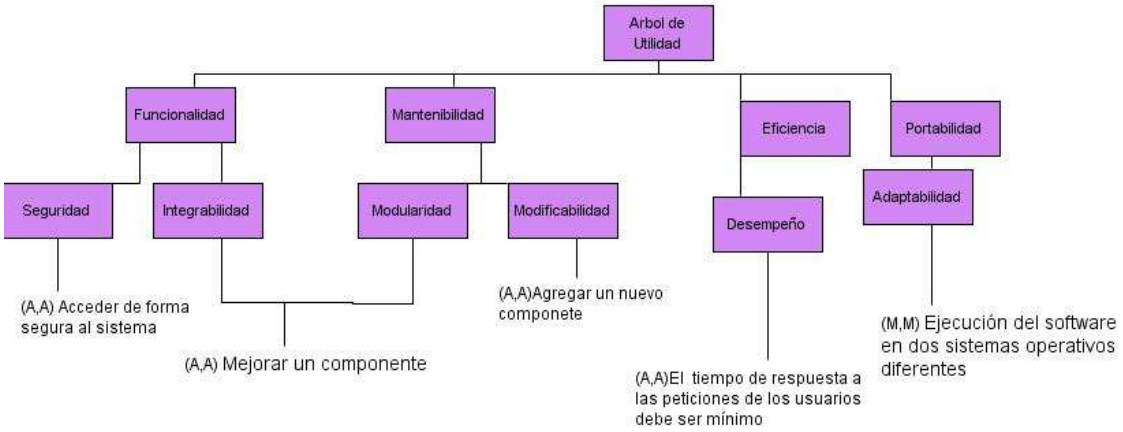

Figura 19: Árbol de utilidad.

3.6.2. Vista de escenarios

A continuación se explican cada uno de los escenarios que fueron seleccionados.

Escenario#1

Escenario	Acceder de forma segura al sistema.
Atributo	Funcionabilidad (Seguridad).
Ambiente	Operación normal.
Estimulo	Usuario no autenticado intenta acceder a las funcionalidades del sistema.
Contexto	No permite el acceso.
Respuesta	Se muestra la página de autenticación para que el usuario introduzca sus credenciales.
Decisiones de la arquitectura	La autenticación será la primera acción del usuario en el sistema y consistirá en suministrar un nombre de usuario único o cédula y una contraseña que debe ser de conocimiento exclusivo de la persona que

	se autentica. Una vez que el usuario ha entrado sus datos estos son verificados en la clase FManage de la capa lógica del sistema y guardados temporalmente en la clase Component de la capa de acceso a datos.

Escenario# 2

Escenario	El tiempo de respuesta a las peticiones de los usuarios debe ser mínimo.
Atributo	Eficiencia (desempeño).
Ambiente	Operación normal.
Estimulo	El usuario agrega un objeto a la escena.
Contexto	Se agrega el objeto que el usuario quiere en la escena en menos de dos segundos.
Respuesta	El sistema responde a la petición del usuario rápidamente.
Decisiones de la arquitectura	La clase **ComponentManager** de la capa accesos a datos permite que el objeto que se seleccione por el usuario sea creado en la escena en la posición que el usuario seleccionó. La rapidez con que el objeto es visualizado en la escena por el usuario se debe a la rápida comunicación de las clases controladoras en cada capa todo ello incluido en la vista lógica del sistema.

Escenario#3

Escenario	Mejorar un componente.
Atributo	Mantenibilidad (modularidad)/Funcionalidad (integrabilidad).
Ambiente	Operación normal.
Estimulo	Modificación de un componte del software.
Contexto	El sistema se mantiene intacto ya que el componente es modificado y probado por separado antes de integrarlo nuevamente al sistema.
Respuesta	El sistema se debe mantener funcional y una vez que se integre el componte al mismo debe funcionar correctamente.
Decisiones de la arquitectura	La arquitectura está basada en componentes y en capas lo que va a permitir que si un componente de una capa se quiere mejorar se puede hacer sin afectar los demás componentes de una misma capa o de

	capas inferiores, y como existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.

Escenario#4

Escenario	Agregar un nuevo módulo.
Atributo	Mantenibilidad (modificabilidad).
Ambiente	Expansión del sistema.
Estimulo	Agregar un componte al software.
Contexto	El sistema funciona correctamente.
Respuesta	El sistema debe aceptar el nuevo módulo y funcionar correctamente.
Decisiones de la arquitectura	La arquitectura está basada en capas lo que va a permitir que si se desea agregar un nuevo módulo se seleccione la capa donde se quiere agregar y se realice la integración con los demás componentes de dicha capa sin que se afecten los componentes de las otras capas .

Escenario#5

Escenario	Ejecución del software en dos sistemas operativos diferentes.
Atributo	Portabilidad (adaptabilidad).
Ambiente	El software es ejecutado en el Sistema Operativo Windows y Linux.
Estimulo	Se ejecuta el sistema en los sistemas operativos de Windows y Linux.
Contexto	El sistema responde rápidamente sin mostrar errores.
Respuesta	El sistema debe ejecutarse en los sistemas operativos ya que fue diseñado para que fuese multiplataforma.
Decisiones de la arquitectura	En la capa lógica del sistema se maneja que el software sea multiplataforma. El sistema tiene la facilidad de adaptarse a los sistemas operativos Linux y Windows ya que contiene clases que van a permitir que el sistema se ejecute en ambientes diferentes.

3.6.3. Resultados de la evaluación

Con la utilización de la técnica de evaluación basada en escenarios y el método de evaluación ARID se realizó la evaluación de la arquitectura, lo que ayudó a demostrar que la arquitectura que se quiere proponer es correcta y que la misma posee un alto grado de mantenibilidad, reusabilidad, modificabilidad, integrabilidad, portabilidad y eficiencia atributos que fueron seleccionados como mas importantes para esta evaluación debido a la fase de desarrollo en que se encuentra la arquitectura.

Consideraciones del capítulo

En este capítulo se realizó un estudio de las técnicas y métodos de evaluación de la arquitectura de software y se seleccionó la más adecuada para evaluar la misma. Se analizaron brevemente los atributos de calidad de la arquitectura propuesta, arrojando como resultado el árbol de utilidad y los escenarios identificados.

Conclusiones generales

El desarrollo de esta investigación permitió elaborar una propuesta de arquitectura de software para el desarrollo de laboratorios virtuales del proyecto PROLAVI. Para ello:

- Se realizó un estudio de los estilos arquitectónicos, patrones de arquitectura, de diseño y se escogieron los adecuados para la elaboración de la arquitectura de software.
- Se realizó la descripción de la arquitectura con todos los elementos necesarios para la misma entre los que se encuentran el ambiente de desarrollo y la selección de los objetivos y restricciones del sistema.
- Se definió la metodología y herramientas de software a utilizar.
- La arquitectura fue diseñada teniendo en cuenta las 4 + 1 vistas propuestas por RUP.
- Se evaluó la arquitectura propuesta aplicando el método de evaluación ARID y la técnica basada en escenario usando como instrumento de evaluación el árbol de utilidad, lo cual contribuyó al diseño de una arquitectura flexible y robusta lo que permitió darle cumplimiento al objetivo general y las tareas propuestas.

Recomendaciones

Al término del presente trabajo de diploma se recomienda:

- Perfeccionar la arquitectura continuamente a través del ciclo de desarrollo de RUP.
- Validar la arquitectura en otra etapa de diseño y con otro método de evaluación para mejorar los escenarios propuestos que ayuden en la mejora de la calidad de la arquitectura propuesta.

Bibliografía

1. **Ledo, María Vidal; Ruiz, Susana; Diego, Francisca; Vialart, Niurka**. *Entornos virtuales de enseñanza-aprendizaje*. *2012.*
2. **Salas, Carlos Vásquez.** *Laboratorios virtuales.*2009.
3. **Johnson, L; Smith, R; Levine, A; Stone, S**. *Informe Horizon Report. 2010.*
4. **Brooks, Frederick.** *The mythical man-month: Addison-Wesley.* 2005.
5. **Reynoso, Carlos Billy.** *Introducción a la Arquitectura de Software.* 2004.
6. *Informe de la reunión de expertos sobre laboratorios virtuales. Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura. 2000.*
7. **Kiccillof, Nicolás; Reynoso, Carlos Billy.** *Estilos y Patrones en la estrategia de arquitectura de Microsoft.* 2004.
8. **Canal, Carlos.** *Arquitectura,Marco de trabajo y Patrones.* 2005.
9. **Lescano, German.** *Patrones Fundamentales en el diseño Orientado a Objeto.*
10. **Sánchez, María A. Mendoza.** *Metodologías de Desarrollo de Software.* 2004.
11. **Zarzuela, Jorge Ferrer.** *Metodologías Ágiles.* [Online]. 2003. http://libresoft.dat.escet.urjc.es/html/downloads/ferrer-20030312.pdf.
12. **Jacobson, Ivar; Booch, Grady; Rumbaugh, James.** *El proceso Unificado de Desarrollo de Software.*1999.
13. *Wikipedia, la enciclopedia libre.* [Online]. 2012. http://www.Wikipedia.com/Entornos de desarrollo integrado
14. **Meyer**, **Pérez**. *Desarrollo multiplataforma con QT 4.* [Online]. 2012. http://perezmeyer.com.ar/files/introduccionAQT/
15. **Buschmann, Frank; Meunier, Regine; Rohnert, Hans.** *Pattern-Oriented of Software Architecture. A system of patterns.*
16. **Camacho, Erika; Cardeso, Fabio; Núñez, Gabriel.** *Arquitectura de Software.* 2004.
17. **Grados, Luis Ismael Rondón.** *Patrón Singleton - Implementación.* [Online]. 2009. http://patron-singleton-implementacion.html.
18. *Introducción a la Disciplina de Requisitos de RUP. Ingeniería de Software I .2011-1012.*
19. **Gómez, Omar Salvador.** *Evaluando Arquitecturas de Software.Panorama General. Parte 1.* 2007.

18. **Gómez, Omar Salvador.** *Evaluando Arquitecturas de Software.Métodos de Evaluación. Parte 2.* 2007.

Anexos

Anexo 1: Especificación de los Casos de Usos

1.1 CU: Autenticar

Nombre del CU	Autenticar.
Actores	Usuario genérico.
Propósito	Permitir a un usuario autenticarse en el sistema.
Resumen	El caso de uso se inicia cuando el usuario decide autenticarse en el sistema y finaliza cuando éste ingresa al sistema.
Referencias	R 1.1
Precondiciones	-
Poscondiciones	El usuario es autenticado.

Curso Normal de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
1. El usuario decide autenticarse y llena los siguientes campos: nombre de usuario y contraseña.	1.1. El sistema verifica que los datos estén correctos y le da acceso al sistema con los permisos correspondientes al rol del usuario terminando así el caso de uso.
Flujo Alternativo	
Acciones del Actor	**Respuesta del Sistema**
	1.1 El sistema muestra un cuadro de dialogo con un mensaje de error. ”Verifique los datos de entrada” terminando así el caso de uso.

1.2 CU: Gestionar Objetos de la Escena

Nombre del CU	Gestionar Objetos de la Escena.
Actores	Usuario genérico.
Propósito	El propósito del caso de uso es que el usuario pueda **RF1.2.1 Agregar objetos en la escena.** **RF1.2.2 Eliminar objetos de la escena.** **RF1.2.3 Modificar objetos de la escena.** **RF1.2.4 Integrar objetos en la escena.**
Resumen	El caso de uso inicia una vez el usuario se haya autenticado y entrado a la escena de trabajo.

Referencias	R 1.2
Precondiciones	El usuario debe estar autenticado.
Poscondiciones	El usuario puede agregar, eliminar o modificar los objetos de la escena.

Curso Normal de los Eventos	
Sección: " Agregar Objeto a la escena"	
Acciones del Actor	**Respuesta del Sistema**
1. Selecciona el objeto a arrastrar al área de trabajo.	1.1. Verifica que el usuario se encuentre en el área de trabajo adecuada. 1.2 Permite que el usuario agregue el objeto a la escena terminando así el caso de uso.
Flujo Alternativo	
Acciones del Actor	**Respuesta del Sistema**
	1.1 Muestra un cuadro de dialogo al usuario con un mensaje de error terminando así el caso de uso.
Sección: " Eliminar Objeto a la escena"	
Acciones del Actor	**Respuesta del Sistema**
1. Selecciona el objeto a eliminar.	1.1 Permite eliminar el objeto seleccionado por el actor terminando así el caso de uso.
Flujo Alternativo	
Acciones del Actor	**Respuesta del Sistema**
Sección: " Modificar posición de objetos en la escena"	
Acciones del Actor	**Respuesta del Sistema**
1. Selecciona el objeto a modificar.	1.2 Permite que el usuario elimine el objeto de la escena terminando así en caso de uso.
Flujo Alternativo	
Acciones del Actor	**Respuesta del Sistema**
Sección: " Integrar objetos en la escena"	
Acciones del Actor	**Respuesta del Sistema**

1. Selecciona los objetos que se van a integrar para confeccionar la escena.	1.2 Permite que el usuario integre los objetos de la escena.
Flujo Alternativo	
Acciones del Actor	**Respuesta del Sistema**

1.3 CU: Navegar en la escena

Nombre del CU	Navegar en la escena
Actores	Usuario genérico.
Propósito	Permitir al usuario que navegue en la escena.
Resumen	El caso de uso se inicia una vez el usuario se autentica permitiéndole al mismo que navegue en la escena.
Referencias	R 1.3
Precondiciones	El usuario debe estar autenticado.
Poscondiciones	-
Curso Normal de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
1. Selecciona la escena donde va a trabajar.	1.1. Permite que el usuario navegue en la escena terminando así el caso de uso.
Flujo Alternativo	
Acciones del Actor	**Respuesta del Sistema**

1.4 CU: Exportar Práctica de Laboratorio

Nombre del CU	Exportar práctica de laboratorio.
Actores	Usuario genérico.
Propósito	Permitirle al usuario exportar la práctica de laboratorio para usarla en otro momento.
Resumen	Este caso de uso va a permitirle al usuario guardar en un documento xml la práctica de laboratorio realizada para cargarla en

	otro momento.
Referencias	R 1.4
Precondiciones	El usuario debe estar autenticado.
Poscondiciones	

Curso Normal de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
1. Selecciona la opción de "exportar práctica". 2. Selecciona el lugar para exportar el fichero, escribe el nombre del mismo terminando así el caso de uso.	1.1. El sistema le muestra una ventana para que el usuario seleccione el lugar y el nombre con que desea guardar el fichero a exportar.
Flujo Alternativo	
Acciones del Actor	**Respuesta del Sistema**

1.5 CU: Importar práctica de laboratorio

Nombre del CU	Importar práctica de laboratorio.
Actores	Usuario genérico.
Propósito	Permitir a un usuario importe la práctica de laboratorio deseada.
Resumen	El caso de uso se inicia cuando el usuario decide importar la práctica de laboratorio en la que desea trabajar.
Referencias	R 1.5
Precondiciones	El usuario debe estar autenticado y debe haber exportado en algún momento el fichero que desea importar.
Poscondiciones	El usuario puede trabajar en la práctica importada.

Curso Normal de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
1. El Selecciona la opción "importar práctica". 2. Selecciona el fichero a importar y da clic en el botón aceptar.	1.1. Muestra una ventana para que seleccione el documento a importar. 1.2. Verifica que el documento que se quiere importar está en formato xlm terminando así el caso de uso.

Flujo Alternativo	
Acciones del Actor	**Respuesta del Sistema**
	1.2 Muestra un mensaje de error diciendo que el documento seleccionado no se puede importar terminando así el caso de uso.

1.6 CU: Salvar Reporte de la Práctica

Nombre del CU	Salvar reporte de la práctica.	
Actores	Usuario genérico.	
Propósito	Que el usuario salve en formato .pdf la práctica realizada.	
Resumen	Este caso de uso es el encargado de que el estudiante una vez terminada la práctica de laboratorio pueda salvar el reporte de la misma.	
Referencias	R 1.6	
Precondiciones	El usuario debe estar autenticado.	
Poscondiciones		
Curso Normal de los Eventos		
Acciones del Actor		**Respuesta del Sistema**
1. Selecciona la opción salvar práctica de laboratorio. 2. Escribe el nombre y el lugar donde desea guardar el documento y da clic en el botón aceptar terminando así el caso de uso.		1.1. Muestra una ventana para que el usuario seleccione donde desea guardar la práctica.
Flujo Alternativo		
Acciones del Actor		**Respuesta del Sistema**

1.7 CU: Mostrar errores cometidos

Nombre del CU	Mostrar errores cometidos
Actores	Usuario genérico.

Propósito	Que el usuario pueda ver los errores cometidos en la escena.
Resumen	Este caso de uso es el encargado de que el usuario vea los errores que cometió durante la realización de la práctica de laboratorio.
Referencias	R 1.7
Precondiciones	El usuario debe estar autenticado.
Poscondiciones	Se le muestran los errores cometidos.

Curso Normal de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
1. Selecciona la opción mostrar errores cometidos.	1.1. Muestra los errores cometidos por el usuario terminando así el caso de uso.
Flujo Alternativo	
Acciones del Actor	**Respuesta del Sistema**

Anexo 2 Prueba de un módulo del sistema de Redes II por casos de uso

Validación del Caso de Uso Autenticar

Entrada

El usuario debe entrar el nombre, el número de pasaporte o número de cedula.

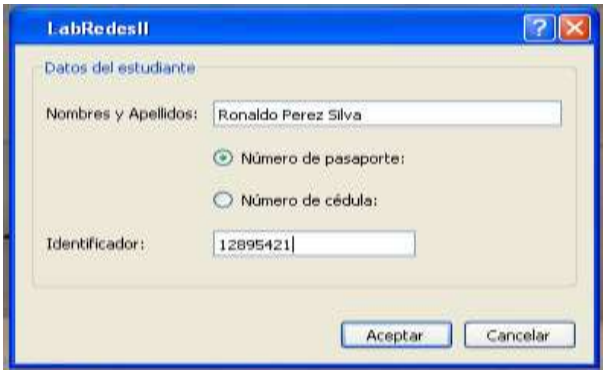

Figura 20: Interfaz de Autenticación.

Resultado

Si la autenticación esta correcta el usuario ingresa a la práctica si no se le muestran las ventanas emergentes del error cometido.

- Error de campos vacíos.

Figura 21: Interfaz de Alerta de Campo Vacío.

- Error en los datos de autenticación.

Figura 22: Interfaz de Alerta Datos Incorrectos.

Descripción

Se verificó la correcta autenticación del usuario mediante las validaciones requeridas.

Validación caso de uso Comprobar Configuración

Entrada

Analizar las configuraciones bajo diferentes condiciones, chequeando que se realicen todas las tareas el proceso de compilación.

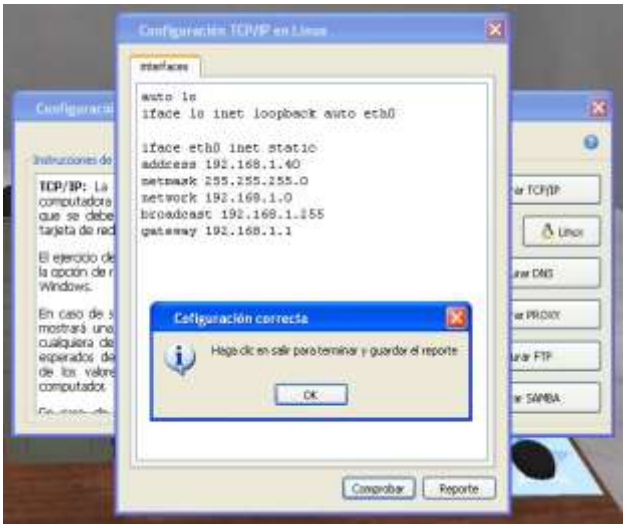

Figura 23: Interfaz de Configuración.

Resultado

Si la configuración esta correcta todas las tareas comprendidas en el proceso de compilación se desarrollan y se muestra la ventana de guardar el reporte, sino se le muestran los errores de la configuración realizada.

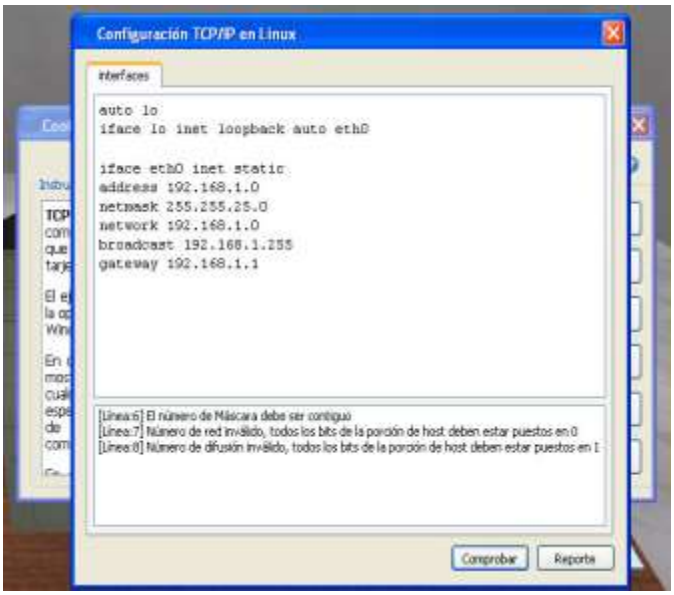

Figura 24: Interfaz de Errores de Configuración.

Descripción

Se verifica el correcto funcionamiento de los fases que conforman el proceso de compilación entre ellas llenar lista de errores.

Validación caso de uso Reporte de la Configuración

Entrada

Guardar las configuraciones realizadas por los usuarios en la práctica.

Figura 25: Interfaz de Guardar el Reporte.

Resultado

El fichero generado contiene cada operación contiene las configuraciones realizadas en la práctica.