Universidad de las Ciencias Informáticas

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

Título:

Arquitectura de software para el Ambiente Virtual de Aprendizaje ProteSIM del proyecto UCI-CIMEQ

Autor:

Rafael Corpas Crehuet

Tutor:

Lic. Luis Gabriel Viciedo Caraballoso

Asesores:

Ing. Roberto Elías Pérez Ozete

Ing. Yaima Fiallo Valle

La Habana, 2014

"A las estrellas no se sube por caminos llanos..."

José Martí

Declaración de autoría

Declaro ser el autor del trabajo de diploma titulado: “Arquitectura de software para el Ambiente Virtual de Aprendizaje ProteSIM del proyecto UCI-CIMEQ” y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Autores:

_____

Rafael Corpas Crehuet

Tutores:

_____

Lic. Luis Gabriel Viciedo Caraballoso

Datos de contacto

Tutor:

Nombre y Apellidos: Luis Gabriel Viciedo Caraballoso.

Institución: Universidad de las Ciencias Informáticas.

Título: Licenciado en Educación, especialidad Física.

Categoría docente: Profesor Auxiliar.

E-mail: viciedo@uci.cu

Se desempeña como Profesor Auxiliar desde el año 2000. Ha realizado tutorías en más de 15 trabajos de diploma en la carrera de Ingeniería Informática y se ha desempeñado en tribunales de eventos científicos y de cambios de categoría docente en la UCI. Tiene más de 20 publicaciones sobre el tema de Laboratorios Virtuales y Ambientes de Aprendizaje Interactivos desde el año 2008.

Asesores:

Nombre y apellidos: Yaima Fiallo Valle

Institución: Universidad de Ciencias Informáticas

Categoría docente: Instructor

E-mail: yfiallo@uci.cu

Graduado de la Universidad de Cienfuegos Carlos Rafael Rodríguez (UCF) y ISPJAE, con siete años de experiencia en el área de ingeniería de software.

Nombre y apellidos: Roberto Elías Pérez Ozete

Institución: Universidad de Ciencias Informáticas

E-mail: reperez@uci.cu

Graduado de la Universidad de Ciencias Informáticas, actualmente en adiestramiento con experiencia en el trabajo con el motor de juegos Unity 3D.

Dedicatoria

A mis padres y mi hermana:

por ser las personas que más quiero en la vida y que son todo para mí.

A mi abuela Amelia:

por todo el amor y la dedicación que me ha dado.

Agradecimientos

Agradecimientos a todas aquellas personas que de una forma u otra influyeron para que me graduara en la universidad y que contribuyeron a mi formación tanto personal como profesional.

De forma especial agradecer:

A mi tutor Luis Gabriel Viciedo por todo su apoyo.

A los profesores Yidier Romero, Roberto Elías y la profesora Yaima Fiallo por la ayuda brindada.

A toda mi familia sin excepciones, primeramente por ser los profesionales que han sido mi ejemplo a seguir y mi fuente de inspiración, y porque todos contribuyeron en la realización de mis estudios.

A mis padres y mi hermana por ser todo para mí.

A mi cuñado Juan Ignacio por toda la ayuda que me brindó en momentos de apuro.

A mi familia de la Habana: mi abuela Amelia, mi tía Belkys y mis primos Joan y Joanne, por hacer de su casa la mía y por toda la ayuda que me dieron desde que pasé el servicio militar aquí en la capital hasta hoy que termino mis estudios en la UCI.

A mi tía Suraya por ser mi otra madre y a mi prima Adriana por ser mi otra hermana.

A todos mis compañeros de grupo durante estos cinco años, por todo lo vivido, por las noches de desvelo de estudio y por todo lo que aprendí con ellos, especialmente a Judit, Anisley, Elizabeth, Orlenis, Joaquín, Pedro y Manuel Chang por ser las personas que siempre me ayudaron cuando lo necesitaba y con las cuales tuve el placer de compartir una buena amistad.

A mi amiga Madeline Rodríguez por todos los momentos vividos dentro y fuera de la universidad, y aunque ya no se encuentra en el país me apoyó en todo momento en la realización de este trabajo.

A Pelly por toda su ayuda y por ser más que un amigo un hermano durante estos cinco años.

Al piquete de freakies del café por todos los buenos momentos compartidos y todo lo que aprendí junto a ellos, especialmente a Jorgito, Samuel, William, Daniela, La Flaca, Marisbelis, Rojas y Yadini.

A todos muchas gracias.

Resumen

La arquitectura de software para guiar el desarrollo de aplicaciones informáticas tiene un peso fundamental en el éxito del producto final. El presente trabajo tiene como objetivo definir una arquitectura para el ambiente virtual de aprendizaje ProteSIM del proyecto UCI-CIMEQ, que facilite a los desarrolladores la adecuada comprensión del sistema y contribuya a la implementación del producto con una alta aceptación. A partir del análisis de diferentes estilos arquitectónicos se propone al proyecto, una arquitectura basada en capas y componentes para estructurar la aplicación. Para ello se toman en cuenta los conceptos asociados a la arquitectura de software y su importancia para el proceso de desarrollo. Además se consideran los patrones arquitectónicos y de diseño a tener en cuenta en la solución.

La representación detallada de la arquitectura se realizó basándose en los requerimientos funcionales del sistema y representada mediante el modelo 4+1 vistas propuesto por la metodología de desarrollo RUP. De igual forma, fueron detallados los requerimientos no funcionales y las restricciones de diseño e implementación que debe satisfacer la arquitectura. La evaluación de la misma fue realizada de dos formas: con la aplicación del método ARID y la técnica basada en prototipos, donde se implementó algunas de las funcionalidades significativas.

Palabras claves: Arquitectura de software, ProteSIM, ambiente virtual de aprendizaje, estilos arquitectónicos, patrones de diseño.

Índice de contenidos

Índice de Figuras

Índice de Tablas

Introducción

El desarrollo continuo de las Tecnologías de la Información y las Comunicaciones (TIC) en su esfuerzo por generar mayor riqueza y aumentar la calidad de vida de las personas, ha logrado intervenir en gran cantidad de procesos, por lo que es difícil mencionar un sector de la sociedad donde no estén presentes. Uno de los sectores que ha sido beneficiado por las TIC es el sector de la Educación, donde se han puesto en práctica medios informáticos de enseñanza para apoyar el proceso docente-educativo. Uno de estos medios de enseñanza son los ambientes virtuales de aprendizaje, donde se simulan los conocimientos adquiridos en las clases por los estudiantes, contribuyendo a una mayor calidad en su formación.

A través de un ambiente virtual de aprendizaje, los estudiantes pueden acceder y desarrollar acciones que son propias de un proceso de enseñanza presencial tales como conversar, leer documentos, realizar ejercicios, formular preguntas al docente, trabajar en equipo, realizar prácticas de laboratorio, etcétera, todo ello de forma simulada. De esta manera, no se necesita hacer uso de herramientas y materiales reales para la realización de prácticas, por los que constituyen una opción viable ante la carencia de recursos u otros impedimentos para llevar a cabo esta actividad.

En la actualidad existen disímiles ambientes virtuales entre los que se puede mencionar: laboratorios virtuales de Física, ejemplo de ellos son: **Phet**, **Physion** e **Interactive Physic** que son un conjunto de simuladores didácticos e interactivos diseñados para el aprendizaje de los conceptos básicos de la Física. También existen estas simulaciones en la asignatura de Química como es el caso de **LiveChem, ChemLAB** y **Chemmol**, este último con el objetivo del cálculo de masa molecular y análisis químico de componentes. En Cuba no se está ajeno al uso de estas nuevas tecnologías, debido a que también se han desarrollado ambientes virtuales para apoyar las actividades docentes en instituciones de diferentes niveles de enseñanza. En el caso de la Universidad de Ciencias Informáticas (UCI), se han creado laboratorios virtuales 3D para el aprendizaje de Metalografía, el ensamblaje de computadoras y para la asignatura de Sistemas Operativos, desarrollados por el centro VERTEX de la Facultad 5.

En el proyecto de investigación desarrollo UCI-CIMEQ perteneciente al centro VERTEX, se desarrolla la aplicación **ProteSIM**, en colaboración con el Centro de Rehabilitación de Cara y Prótesis Bucomaxilofacial del Hospital CIMEQ. La aplicación mencionada se define como un Ambiente Virtual de Aprendizaje Interactivo en 3D (AVAI), que tiene como objetivo desarrollar y entrenar habilidades en los estudiantes de las carreras médicas de Estomatología y Tecnologías de la Salud, específicamente en las tecnologías asociadas a la construcción de prótesis bucomaxilofaciales. Con esta herramienta informática, los estudiantes pueden insistir sobre tareas simuladas complejas dentro de la tecnología médica sujeta a aprendizaje y adquirir un nivel de familiarización, sin repercutir en pérdidas de materiales por errores humanos y gastos de insumos. Luego de ser evaluada la preparación previa de los estudiantes por parte del profesor e interactuar de forma

colaborativa en el ambiente virtual diseñado, los estudiantes pueden enfrentarse a los procederes médicos reales con una mejorada certeza en objetivos a cumplir en su interacción con los pacientes.

Para abordar el desarrollo de estas aplicaciones informáticas con fines educativos, la calidad de software cobra un papel esencial, pues permite competir con productos similares con mayores posibilidades de éxito. En el proyecto implicado en este trabajo, sus miembros deben ejecutar sus responsabilidades en base a que la aplicación en desarrollo tenga la mayor calidad posible, para ello uno de los aspectos que debe ser bien considerado resulta la utilización de una adecuada y conveniente Arquitectura de Software (AS) documentada y bien estructurada, que sirva como guía del proceso de desarrollo.

Actualmente el proyecto UCI-CIMEQ no cuenta con una AS documentada que guíe el proceso de desarrollo de la aplicación **ProteSIM** para que esta cumpla con los servicios y funcionalidades que requiere el usuario al interactuar con los escenarios virtuales diseñados y que además le permita a los desarrolladores del software compartir una misma línea de trabajo, cubriendo todos los objetivos y restricciones de la aplicación. La arquitectura empleada hasta el momento tiene entre sus principales problemas que no satisface las nuevas exigencias para el proyecto, como es la necesidad de que los estudiantes trabajen de forma colaborativa en cada uno de los roles previstos en las escenas de la aplicación dentro de una red académica o subred de un laboratorio docente.

Igualmente asociado al trabajo colaborativo en equipo por medio de una subred, será necesario incluir funcionalidades asociadas a la comunicación e intercambio de archivos entre los integrantes del equipo, exigencia determinada por el cliente para la evaluación integral de los estudiantes. Finalmente, al concluir las tareas simuladas en el ambiente de aprendizaje, el sistema debe ser capaz de generar las trazas de cada miembro del equipo de estudiantes de manera individualizada, en un archivo de formato determinado como parte de la evaluación de sus integrantes que posteriormente realizará el profesor o tutor.

En un futuro próximo se pretende combinar las interacciones tradicionales por teclado y ratón, con otras más novedosas, como es la captura de movimiento de manos y dedos, para la interactividad de los estudiantes con los objetos y escenarios simulados del Ambiente Virtual de Aprendizaje Interactivo (AVAI), por lo que la arquitectura debe ser lo suficientemente flexible como para poder integrar estas funcionalidades al sistema.

Otros problemas constatados son el fuerte acople y la alta de dependencia entre sí de los elementos de la arquitectura empleada hasta el momento, además de la débil reusabilidad que se puede lograr con los mismos, lo cual dificulta la realización de los nuevos requisitos exigidos, así como la falta de una estructura lógica de sus componentes, lo cual dificulta el aprendizaje a los nuevos integrantes del proyecto cada año.

Lo planteado anteriormente deriva como **problema científico** la siguiente interrogante: ¿Cómo obtener una Arquitectura de Software que guíe el proceso de desarrollo de la aplicación **ProteSIM** del proyecto UCI-CIMEQ?

Teniendo como **objeto de estudio**: La Arquitectura de Software y **como campo de acción**: Arquitectura de Software para ambientes virtuales de aprendizajes en 3D.

El **objetivo general** del trabajo propone desarrollar y validar una Arquitectura de Software para encauzar el proceso de desarrollo de la aplicación **ProteSIM** del proyecto UCI-CIMEQ.

Para darle cumplimiento al objetivo propuesto se plantean las siguientes **tareas investigativas**:

1. Análisis de fuentes bibliográficas para la elaboración del diseño teórico de la investigación y el estado del arte.
2. Análisis de las tecnologías empleadas a nivel mundial para el desarrollo de ambientes virtuales de aprendizaje en 3D, teniendo en cuenta principalmente la arquitectura empleada para su construcción.
3. Descripción de los componentes de la arquitectura y determinación de aquellos que serán empleados en el desarrollo de la aplicación.
4. Diseño de la Arquitectura de Software para la aplicación ProteSIM.
5. Implementación de la arquitectura propuesta.
6. Evaluación de la arquitectura propuesta.

Entre los **métodos científicos de investigación** que serán utilizados para llevar a cabo el proceso de investigación y elaboración del trabajo son:

Métodos teóricos:

- **Histórico-Lógico**: Método que será utilizado para analizar la trayectoria, evolución y tendencias actuales de la Arquitectura de Software.
- **Inductivo-Deductivo**: Este método será utilizado para solucionar problemas particulares partiendo de principios y conocimientos generales sobre el problema de investigación.
- **Analítico-Sintético**: Método que será utilizado para analizar los elementos más representativos que se definen en la Arquitectura de Software para luego establecer las relaciones entre ellos y arribar a conclusiones del problema.
- **Modelación**: Método utilizado que será para representar mediante diagramas las diferentes vistas arquitectónicas que le dan solución al problema.

Métodos empíricos:

- **Observación**: Método que será utilizado para analizar resultados obtenidos en la evaluación de la arquitectura.
- **Pruebas**: Método que será utilizado para la realización de pruebas a la arquitectura definida para determinar si la misma constituye una propuesta factible.
- **Consultas a fuentes de información**: Método que será utilizado para consultar información actualizada referente a la Arquitectura de Software.

La estructura de la tesis se conforma de la siguiente manera:

Capítulo # 1: Fundamentación teórica.

Se exponen los diferentes conceptos asociados a la Arquitectura de Software, así como la descripción de los estilos y patrones arquitectónicos. Se analizan los diseños arquitectónicos de proyectos similares y se hace un estudio de las diferentes metodologías y herramientas de desarrollo de software adecuadas para la propuesta de arquitectura.

Capítulo # 2: Propuesta de arquitectura.

Se definen los elementos que describen el sistema a desarrollar. Se describen los requerimientos funcionales y no funcionales que debe satisfacer la arquitectura propuesta y se presenta la descripción detallada de la arquitectura mediante varias vistas.

Capítulo # 3: Validación de la arquitectura propuesta.

Se analizan los principales métodos y técnicas de evaluación de las arquitecturas de software y se seleccionan los adecuados para aplicarlo a la arquitectura propuesta.

Capítulo 1: Fundamentación Teórica

En el presente capítulo se abordan los elementos teóricos que dan solución al problema científico planteado. Se analizan las arquitecturas de ambientes virtuales de aprendizaje con funciones similares a los de ProteSIM. Además se analizan los principales conceptos que se emplearán en el trabajo asociados a la Arquitectura de Software, así como la descripción de estilos, patrones arquitectónicos y de diseño. También se profundiza en el estudio de metodologías y herramientas de desarrollo.

1.1 Ambientes virtuales de aprendizaje (AVA).

Los ambientes virtuales de aprendizaje (AVA) son herramientas informáticas que han surgido en los últimos tiempos con el avance de las TIC impactando de forma positiva en el proceso de enseñanza aprendizaje. Se entiende por AVA al espacio físico donde las nuevas tecnologías, han rebasado el entorno escolar tradicional, favoreciendo al conocimiento y a la apropiación de contenidos, experiencias y procesos pedagógicos-comunicacionales. Están conformadas por el espacio, el estudiante, el asesor, los contenidos educativos, la evaluación y los medios de información y comunicación. Los AVA siguen el mismo objetivo principal que sigue la pedagogía, el cual es aprender. Siguiendo sus mismas funciones pedagógicas: actividades de aprendizaje, situaciones de enseñanza, materiales de aprendizaje, apoyo y autorización, evaluación, entre otros. Además, incluyen características más específicas y la diferencia clave es el uso de herramientas de telecomunicación en el proceso enseñanza-aprendizaje. (1)

En la actualidad pueden ser consideradas AVA diversas aplicaciones informáticas con fines educativos, tal es el caso de los Entornos Virtuales de Aprendizaje (EVA). Estos pueden ser desde un campus virtual sin interacción presencial hasta una clase convencional que utiliza herramientas telemáticas en el proceso de enseñanza-aprendizaje, siempre que los recursos sean también accesibles fuera del horario regular y la clase asignada. Esta característica es la que hace de los EVA un instrumento de innovación dentro de las instituciones convencionales de enseñanza. (2)

Un Ambiente Virtual de Aprendizaje Interactivo (AVAI) es un espacio electrónico, donde se simulan en 2D o 3D situaciones problemáticas provenientes de un diseño de aprendizaje, donde deben cumplirse objetivos instructivos y educativos de un programa de estudio, en que estudiantes y tutores colaboran en escenarios simulados y son cumplidas tareas experimentales o de entrenamiento, asumiendo un rol propio de su profesión y donde son evaluados por ello. (3)

La diferencia entre un EVA y un AVAI es que el primero responde al formato tradicional de las instituciones, esta vez en un entorno electrónico, donde profesores y estudiantes intercambian información y se comunican de manera sincrónica o asíncrona, enfocados en actividades de aprendizaje, situaciones problemáticas, materiales de aprendizaje, de apoyo, de autorización y de evaluación. Además, pueden incluirse actividades

más personalizadas y la diferencia clave está en el uso de herramientas de telecomunicación en el proceso enseñanza-aprendizaje. Mientras que son consideradas AVAI aquellas aplicaciones informáticas que asumen estrategias y experiencias provenientes de la industria de los videojuegos, se emplean tecnologías de la Realidad Virtual, donde situaciones problemáticas son representadas gráficamente y los estudiantes interactúan de manera síncrona con los escenarios de aprendizaje simulados, ofreciendo soluciones más cercanas a la realidad. Entre los ejemplos de estas aplicaciones encontramos los simuladores de conducción y vuelo, simuladores de operaciones quirúrgicas, los juegos serios, los laboratorios virtuales, entre otros.

1.2 Trabajo colaborativo

El trabajo colaborativo permite a un grupo de individuos realizar actividades en colectivo con el objetivo de lograr un fin común. Los proyectos innovadores que usan técnicas de enseñanza aprendizaje involucran esta modalidad de trabajo en la que el ser que aprende se forma como persona.

En el estudio “Software para el Trabajo Colaborativo y Bibliotecas” (4), se presenta un concepto de trabajo colaborativo elaborado a partir de la fusión de definiciones dadas por diferentes autores. El mismo propone que es una estructura social creada donde dos o más personas interactúan entre sí, bajo determinadas circunstancias, donde aparece como rasgo extensivo de significación, el concepto de interacción en un contexto ya existente. De esta manera, se exige un grado mayor de contribución de las personas implicadas para alcanzar un determinado fin común. La colaboración implica la participación intencionada y coordinada de los miembros de un grupo.

La técnica del aprendizaje colaborativo se refiere a la actividad que efectúan pequeños grupos de alumnos dentro de las aulas de clase; éstos se forman después de las indicaciones explicadas por el docente. Durante el inicio de la actividad y al interior del grupo, los integrantes intercambian información, tanto la que activan (conocimientos previos), como la que investigan. Posteriormente trabajan en la tarea propuesta hasta que han concluido y comprendido a fondo todos los conceptos de la temática abordada, aprendiendo así a través de la cooperación. (5)

1.2.1 Entornos virtuales 3D para el trabajo colaborativo.

Desde el punto de vista pedagógico, las TIC poseen ventajas para el proceso de aprendizaje colaborativo debido a que permiten: estimular la comunicación interpersonal; el acceso a información y contenidos de aprendizaje; el seguimiento del progreso del participante, tanto a nivel individual como grupal; la gestión y administración de los alumnos; la creación de escenarios para la coevaluación y autoevaluación, entre otras. Algunas utilidades específicas de las herramientas tecnológicas para el aprendizaje colaborativo son: comunicación sincrónica y asincrónica, la transferencia de datos, la aplicaciones compartidas, convocatoria

de reuniones, chat, lluvia de ideas, mapas conceptuales, navegación compartida, wikis, notas, pizarra compartida, etcétera. (6)

Desde la Realidad Virtual, se consideran los entornos virtuales 3D como aplicaciones informáticas que simulan mundos virtuales y permiten a uno o más usuarios ver, moverse y reaccionar en él mediante una computadora u ordenador. Existen dos métodos de entornos virtuales 3D: los inmersivos y los no inmersivos. Los inmersivos se asocian a una Realidad Virtual donde el ambiente tridimensional creado por una computadora se manipula a través de cascos, guantes u otros dispositivos de interacción, que capturan la posición y rotación de diferentes partes del cuerpo humano y permiten interactuar con los objetos de los escenarios de manera más natural o intuitiva. Los no inmersivos también utilizan la computadora y se vale de los medios y tecnologías que comúnmente ofrece Internet y redes académicas, donde la interactividad en tiempo real con diferentes personas en espacios y ambientes simulados transcurre sin la necesidad de dispositivos adicionales.

En la actualidad diversas instituciones educativas han incluido dentro de sus prácticas pedagógicas el uso de plataformas virtuales 3D orientadas a la enseñanza y que propician el trabajo colaborativo. Estos contextos son conocidos como *Massively Multilearner Online Learning Environment* (MMOLE, por sus siglas en inglés). Es importante mencionar la existencia de estudios relacionados con los espacios tridimensionales en el proceso de enseñanza aprendizaje, entre los cuales se destacan los trabajos de investigación del MIT (Instituto Tecnológico de Massachusetts), que proponen el uso de simuladores en 3D para el apoyo de procesos formativos. A continuación se presentan algunos ejemplos de estas aplicaciones. (7)

***Second Life* o Segunda Vida (SL):** Es una aplicación multiplataforma desarrollado por la compañía *Linden Lab*, accesible gratuitamente en Internet. En ella los usuarios conocidos como residentes, pueden acceder al sistema, mediante el uso de programas de interfaces llamados *viewers* (visores) permitiéndoles interactuar entre ellos mediante un avatar. Los residentes pueden explorar el mundo virtual, establecer relaciones sociales, participar en actividades individuales o grupales. Se utiliza además para el desarrollo de la educación a distancia (*e-learning*), permitiendo plantear nuevas formas de enseñar y aprender, incluso más allá de las ideas de "simulaciones" y de las teorías más clásicas de aprendizaje.

OpenSim: Es un sistema multiplataforma de código abierto que permite crear mundos virtuales que pueden ser accedidos a través de una gran variedad de clientes y protocolos, desde software de escritorio o desde páginas Web. Cuenta con potencialidades de un mundo virtual aplicado a la Educación y tiene un importante desarrollo en el ámbito universitario, donde se personalizan sus desarrollos, permitiendo integrar usuarios existentes en sus LMS (del inglés, Learning Management System) u otros sistemas con la propia base de datos de OpenSim, diseñando sistemas de administración y de creación de contenidos que adaptan a las necesidades de la enseñanza universitaria.

Sloodle: Es un entorno de aprendizaje dinámico donde se combinan un entorno virtual 3D, como Second Life o OpenSIM, a un LMS de código abierto como Moodle[1]. Se considera un entorno amigable, empleado fundamentalmente para los encuentros sincrónicos, discusiones, simulaciones y aprendizaje donde los usuarios pueden encontrarse virtualmente y resolver tareas de forma conjunta.

1.3 Arquitectura de software

La Arquitectura de Software (AS) es un tema que ha sido abordado por diferentes autores en distintas épocas. Los primeros acercamientos al concepto ocurrieron en la década de los años sesenta y durante los siguientes años también se realizaron estudios referentes al tema, pero no fue hasta la década de los años noventa que cobró mayor popularidad debido a que comenzó a gestarse de manera más clara la idea de que las aplicaciones tienen una morfología, una estructura. El punto de partida para lo que hoy se conoce como Arquitectura de Software aparece en 1992 en el trabajo de Dewayne E. Perry y Alexander Wolf: "*Foundations for the study of software architectures*" (8). Hoy día no existe una definición exacta para el término Arquitectura de Software, sin embargo existen varios aspectos comunes en los distintos conceptos dados que facilitan su entendimiento. A continuación se exponen varias definiciones de AS.

La organización internacional *Institute of Electrical and Electronics* (IEEE) a través de sus normas 1471-2000 propone que: La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución (9).

La metodología *Rational Unified Process* (RUP) en 1999 propuso que: Una arquitectura es el sistema de decisiones significativas sobre la organización de un sistema de software, la selección de los elementos estructurales y de sus interfaces por los cuales el sistema es compuesto, junto con su comportamiento según lo especificado en las colaboraciones entre esos elementos, la composición de estos elementos estructurales y del comportamiento en subsistemas progresivamente más grandes, y el estilo arquitectónico que dirige esta organización, los elementos y sus interfaces, sus colaboraciones y su composición (10).

David Garlan y Mary Shaw definen que la arquitectura es un nivel de diseño que hace foco en aspectos más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema (11).

Luego de analizar estas definiciones, se puede considerar la AS como disciplina de la ingeniería de software que desempeña un papel fundamental en la guía para la construcción de aplicaciones informáticas y define la

[1] Es un Control Management System o Sistema de Control y Administración de Contenidos, especializado en educación a distancia y basado en tecnología web de código abierto.

estructura de un sistema, que comprende la organización en subsistemas o componentes y las relaciones entre ellos.

En la presente investigación se adoptará la definición de AS dada por la IEEE en su norma 1471-2000, agregando que esta debe ser comprensible, para lograr de manera fácil una visión común entre los desarrolladores, flexible para la incorporación de nuevas funcionalidades al sistema y además debe permitir la reutilización de componentes.

1.4 Necesidad de una arquitectura de software para un proyecto.

En un sistema de software complejo del que no se tiene el conocimiento de las dimensiones que alcanzará, se necesita de una arquitectura para que los desarrolladores puedan progresar hasta tener una visión común. De forma general una AS es necesaria para:

- Comprender el sistema.
- Organizar el desarrollo.
- Fomentar la reutilización.
- Hacer evolucionar el sistema. (10)

Comprensión del sistema: Para que una organización desarrolle un sistema informático, dicho sistema debe ser comprendido por todos los que vayan a intervenir en él. Lograr que los sistemas modernos sean comprensibles es un reto importante por muchas razones:

- Abarcan un comportamiento complejo.
- Operan en entornos complejos.
- Son tecnológicamente complejos.
- Deben satisfacer demandas individuales de la organización.
- A menudo, combinan computación distribuida, productos y plataformas comerciales y reutilizan componentes y *frameworks* [2]de trabajo. (10)

Organizar el desarrollo: Cuanto mayor sea la organización del proyecto de software, mayor será la sobrecarga de comunicación entre los desarrolladores para intentar coordinar sus esfuerzos. Dividiendo el sistema en subsistemas, con las interfaces claramente definidas y con un responsable o un grupo de responsables establecido para cada subsistema, el arquitecto puede reducir la carga de comunicación entre los grupos de trabajo de los diferentes subsistemas. (10)

[2] En el desarrollo de software, es una estructura de soporte en la que otro proyecto puede ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas, un lenguaje interpretado, entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Fomentar la reutilización: Mientras mayor cantidad de elementos reutilizables existan en el sistema, menor serán los tiempos de desarrollo y costo de otros sistemas futuros. Un elemento reutilizable puede ser un componente, el cual constituye una unidad modular con interfaces bien definidas que es reemplazable dentro de un contexto, una librería la cual proporciona códigos reutilizables por los programadores de una aplicación, un framework que es un conjunto de clases parcialmente funcional (no es una aplicación) para un dominio de aplicación. (10)

Hacer evolucionar el sistema: El sistema debe ser en sí mismo flexible a los cambios o tolerante a los cambios, debe ser capaz de evolucionar sin problemas puesto que las arquitecturas de los sistemas pobres suelen degradarse con el paso del tiempo y necesitan ser parcheadas hasta que al final no es posible actualizarla con un coste razonable. (10)

1.5 Estilos y patrones arquitectónicos

Todo sistema de software está formado por componentes que no son más que los bloques de construcción que lo conforman, estos a nivel de lenguaje de programación pueden ser representados como módulos, clases, objetos o un conjunto de funciones relacionadas. En el diseño de una AS, la forma en que se organizan y se relacionan los componentes de forma tal que cumplan los requerimientos establecidos, puede ser representada mediante estilos arquitectónicos, patrones arquitectónicos y de diseño.

El concepto de estilo aparece en los inicios de Arquitectura de Software como disciplina, al observar que en la práctica del diseño y la implementación ciertas regularidades de configuración aparecían una y otra vez como respuestas a similares demandas. Un estilo describe una clase de arquitectura o piezas identificables de las arquitecturas empíricamente dadas. Esas piezas se encuentran repetidamente en la práctica, sintetizando la existencia de decisiones estructurales coherentes. Una vez que se han identificado los estilos, es lógico y natural pensar en reutilizarlos en situaciones semejantes que se presenten en el futuro. (12)

Las primeras definiciones de estilos fueron propuestas por Perry y Wolf a principios de la década de los noventa los que plantean: “La década de 1990, creemos, será la década de la arquitectura de software. Usamos el término “arquitectura”, en contraste con “diseño”, para evocar nociones de codificación, de abstracción, de estándares, de entrenamiento formal (de arquitectos de software) y de estilo”. (8)

Luego definen un estilo arquitectónico como una abstracción de tipos de elementos y aspectos formales a partir de diversas arquitecturas específicas. Un estilo arquitectónico encapsula decisiones esenciales sobre los elementos arquitectónicos y enfatiza restricciones importantes de los elementos y sus relaciones posibles. (8)

Por otra parte Mary Shaw y Paul Clements identifican los estilos arquitectónicos como un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer en sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo. (13)

Luego de analizar estas definiciones se puede constatar que los estilos pueden tratarse como entidades que ocurren a un nivel muy alto de abstracción, proponiendo una estructura estandarizada en la organización de los elementos que componen el sistema estableciendo para esto las restricciones y relaciones de los mismos.

Complementando a los estilos arquitectónicos existen los patrones arquitectónicos, estos dan solución a un problema de diseño que aparece con frecuencia en reiteradas ocasiones. Cada patrón describe un problema que ocurre una y otra vez en el ambiente, y describe el núcleo de su solución, de tal forma que puede ser utilizada millones de veces sin hacer dos veces lo mismo. (14)

Los patrones arquitectónicos expresan el esquema de organización estructural fundamental para sistemas de software. Provee un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos. (15)

Entre los estilos y patrones existen diversas similitudes, incluso algunos patrones coinciden con estilos hasta en el nombre que se les da: Tuberías y Filtros, Modelo Vista Controlador y Arquitectura en Capas. Al respecto Carlos Reynoso y Nicolás Kiccillof en su libro “Estilos y Patrones en la Estrategia de Arquitectura de Microsoft” expresan: “En cuanto a los patrones de arquitectura, su relación con los estilos arquitectónicos es perceptible, pero indirecta y variable incluso dentro de la obra de un mismo autor. Hay claras convergencias entre ambos conceptos, aun cuando se reconoce que los patrones se refieren más bien a prácticas de reutilización y los estilos conciernen a teorías sobre la estructuras de los sistemas a veces más formales que concretas. Algunas formulaciones que describen patrones pueden leerse como si se refirieran a estilos, y también viceversa”. (16)

La principal diferencia entre ambos conceptos radica en que los estilos se encuentran en un nivel de abstracción más elevado definiendo la configuración de la arquitectura mientras que los patrones se centran más el diseño y pueden representarse mediante lenguajes de programación.

Diversos autores han propuesto diferentes clasificaciones de los estilos y patrones arquitectónicos, pero solo serán analizados los más representativos y vigentes. La notación se establecerá entonces en términos de lo que Shaw y Clements llaman “boxology” (17).

Estilos de Flujo de Datos

- Tubería y filtros

Estilos Centrados en Datos

- Arquitecturas de Pizarra o repositorio

Estilos de Llamada y Retorno

- Modelo-Vista-Controlador (MVC)
- Arquitectura en Capas
- Arquitecturas Orientadas a Objetos
- Arquitecturas Basadas en Componentes

Estilos de Código Móvil

- Arquitectura de Máquinas Virtuales

Estilos heterogéneos

- Sistemas de Control de Procesos
- Arquitecturas Basadas en Atributos

Estilos Peer-to-Peer

- Arquitecturas Basadas en Eventos
- Arquitecturas Orientadas a Servicios
- Arquitecturas Basadas en Recursos

1.5.1 Arquitectura en capas

Se define como un sistema organizado jerárquicamente, donde cada capa proporciona servicios a la capa inmediatamente superior y a su vez se sirve de las prestaciones que le brinda capa inmediatamente inferior (Figura 1).

Cada capa forma una agrupación lógica horizontal de componentes de software o un servicio a diferentes niveles de abstracción. Este estilo ofrece un diseño que permite diferenciar claramente los roles y funcionalidades de cada capa, las diferentes tareas a ser realizadas por los componentes y una delimitación de donde debe estar cada tipo de componente funcional e incluso cada tecnología.

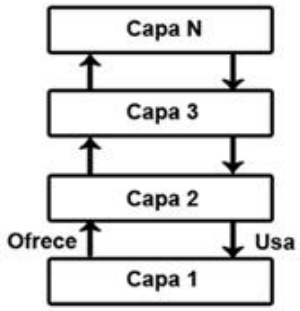

Figura 1. Arquitectura en capas

Algunas de las ventajas que presenta este estilo son (16):

- Soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.
- Admite muy naturalmente optimizaciones y refinamientos.
- Proporciona amplia reutilización.

Algunas desventajas que presenta el este estilo son:

- Muchos problemas no admiten una representación jerárquica en capas.
- En ocasiones se torna difícil encontrar el nivel de abstracción correcto.
- Los cambios en las capas de bajo nivel tienden a filtrarse hacia las de alto nivel.

1.5.2 Arquitectura basada en componentes.

Se basa en principios definidos por una ingeniería de software específica. Un componente de software, es una unidad de composición con interfaces especificadas contractualmente y dependencias del contexto explícitas. (18) En este estilo los componentes son las unidades de modelado, diseño e implementación. Las interfaces están separadas de las implementaciones, y las interfaces y sus interacciones son el centro de incumbencias en el diseño arquitectónico. Los componentes soportan algún régimen de introspección, de modo que su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución. (16)

Algunas ventajas que presenta este estilo son:

- Al ser un componente una unidad de composición, este puede ser comprado o adquirido ya hecho por otra vía, por lo que a veces no es necesario desarrollarlo.
- Admite la reusabilidad.

Algunas desventajas que presenta este estilo son:

- A veces presenta problemas de adaptabilidad.
- Los estados de un componente no son accesibles desde el exterior.

1.6 Patrones de diseño

Un Patrón de Diseño define un esquema de refinamiento de los subsistemas o componentes dentro de un sistema, o las relaciones entre estos. Este describe una estructura común y recurrente de componentes interrelacionados, que resuelve un problema general de diseño dentro de un contexto particular. Los patrones de diseño trabajan a una escala intermedia. Son menores en escala que un patrón de arquitectura, pero logran ser independientes del lenguaje de programación. La aplicación de un patrón de diseño no afecta la

estructura fundamental de un sistema (arquitectura), pero puede tener una fuerte influencia sobre la arquitectura de un subsistema. (19)

En el libro *Design Patterns: Elements of Reusable Object-Oriented Software* (20) se describen los patrones de diseño en tres categorías: patrones creacionales, que se centran en la creación de objetos, patrones estructurales, que tratan lo relacionado con la composición de clases u objetos y los patrones de comportamiento que caracterizan las formas en las que las clases u objetos interactúan y distribuyen responsabilidades. En la siguiente tabla se muestran los patrones distribuidos en las categorías antes mencionadas.

Tabla 1. Patrones de diseño

Creacionales	Estructurales	De Comportamiento
Abstract Factory	Adapter	Chain of responsability
Builder	Bridge	Comand
Factory Method	Composite	Interpreter
Prototype	Decorator	Iterator
Singleton	Facade	Mediator
	Flyweigh	Memento
	Proxy	Observer
		State
		Strategy
		TemplateMethod
		Visitor

- **Abstract Factory:** Proporciona una interfaz para la creación de familias de objetos relacionados o dependientes sin especificar sus clases concretas.
- **Factory Method:** Define una interfaz para crear un objeto pero permite a las subclases decidir cuáles serán sus instancias.
- **Singleton:** Se asegura que una clase solo tenga una instancia y le proporciona un punto de acceso global.
- **Adapter:** Convierte la interfaz de una clase en otra que el cliente puede entender. Permite que clases con interfaces incompatibles trabajen juntas.
- **Facade:** Proporciona una interfaz unificada a un conjunto de interfaces en un subsistema. Define una interfaz de un nivel más alto que hace a un subsistema más fácil de usar.
- **Observer:** Define una relación de uno a muchos entre objetos, por lo que cuando un objeto cambia su estado todas sus dependencias son notificadas y actualizadas automáticamente.
- **State:** Permite a un objeto alterar su comportamiento cuando su estado interno cambia.

1.7 Patrones de software para la asignación de responsabilidades.

Más conocidos como patrones GRASP, acrónimo de General Responsability Assignment Software Patterns (patrones de software para la asignación general de responsabilidades), describen los principios fundamentales de la asignación de responsabilidades. (21) Entre los patrones más conocidos de este tipo se encuentran: experto, creador, bajo acoplamiento, alta cohesión y controlador.

Experto

Este patrón se encarga de asignar una responsabilidad a la clase que posea la información necesaria para cumplir con la misma, o sea el experto.

Algunas ventajas que se logran con el uso de este patrón son:

- Se mantiene el encapsulamiento de la información, puesto que los objetos utilizan su propia información para llevar a cabo las tareas.
- Se distribuye el comportamiento entre las clases que contienen la información requerida.

Creador

Este patrón define quien tiene la responsabilidad de crear instancias de una clase. La clase que se escoja para instanciar a otra debe contener la información necesaria para realizarlo y que agregue o contenga a la clase que será instanciada. Una buena asignación de este patrón propicia que el diseño pueda soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización.

Bajo acoplamiento

La idea fundamental de este patrón es asignar responsabilidades de manera que el acoplamiento entre las clases sea bajo, o sea reducir las dependencias que puedan existir para lograr que las modificaciones que se hagan tengan un impacto bajo e incrementar la reutilización.

Alta cohesión

Este patrón se rige por el principio de asignar responsabilidades de forma tal que permanezca alta la cohesión. La idea fundamental es que las clases tengan responsabilidades altamente relacionadas y que no hagan demasiados trabajos.

Controlador

Este patrón define que clase administra los eventos en un sistema. Un Controlador es un objeto que no pertenece a la interfaz de usuario, responsable de recibir o manejar un evento del sistema. Un Controlador define el método para la operación del sistema.

1.8 Arquitectura de proyectos similares

Con el objetivo de encontrar apoyo para desarrollar la solución del presente trabajo, durante el proceso de investigación fueron analizadas las arquitecturas de varios productos de software que son considerados ambientes virtuales de aprendizaje y que sus objetivos son similares en cierta medida a los ProteSIM. A continuación se detalla el análisis realizado a cada una de las propuestas arquitectónicas seleccionadas:

Adele (Agent for Distance Education - Light Edition): Por sus siglas en inglés: Agente para la Educación a Distancia – Edición Luz, es un agente pedagógico para enseñar a realizar diágnósticos y tratamientos médicos. Fue desarrollado para funcionar en navegadores web en el ISI (*Information Sciences Institute – University of Southern California*). Con esta aplicación los estudiantes pueden realizar diversas acciones sobre el paciente, como realizar preguntas sobre el historial médico, realizar exámenes físicos, ordenar la realización de pruebas y finalmente realizar el diagnóstico, en la Figura 2 se muestra un ejemplo de aplicación Adele en ejecución. (22)

Figura 2. Ejemplo del AVA Adele

Puesto que es una aplicación basada en web, la arquitectura de Adele es, en primera instancia, de tipo cliente/servidor, aunque gran parte de la lógica de la aplicación permanece en el cliente. Esto hace que, a

diferencia de muchos otros sistemas en los que la lógica de la aplicación reside en el servidor, la ejecución de Adele sea más dinámica, al eliminarse los períodos de latencia con el servidor. Básicamente, el sistema se compone de cuatro módulos, como son el agente pedagógico, la simulación, el módulo de comunicación cliente-servidor y el almacén central del servidor. El agente pedagógico, a su vez, se descompone en un módulo de control de Adele y un motor de razonamiento. El almacén del servidor se utiliza para guardar los progresos de los alumnos y, en caso de utilizar entrenamiento multiusuario, para sincronizar las acciones de los distintos estudiantes. En la Figura 3 se muestra la arquitectura de este sistema. (22)

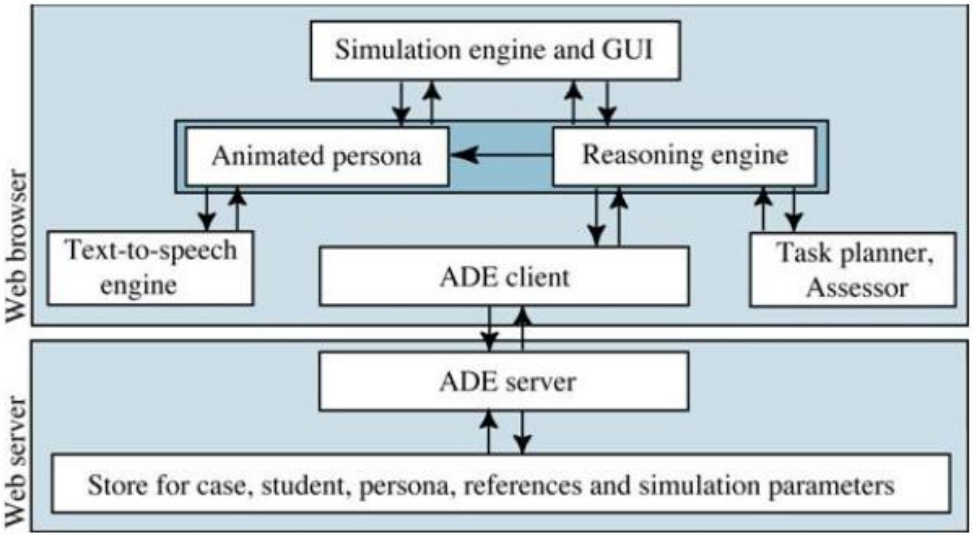

Figura 3. Arquitectura del AVA Adele

Lahystotrain: Es un proyecto desarrollado para entrenar a cirujanos en operaciones de laparoscopia e histeroscopia. Debido a los riesgos que comporta el entrenamiento en el propio quirófano y a las cuestiones éticas que implica la práctica con animales, la utilización de un sistema de entrenamiento basado en realidad virtual ha supuesto una alternativa bastante atractiva para la formación de cirujanos en esta área. El sistema admite dos tipos de usuarios: estudiantes, separados en cuatro niveles de experiencia, e instructores, encargados de supervisar la evolución de los estudiantes. Estos últimos tienen la capacidad de proponer nuevos ejercicios y consultar información respecto a los distintos ejercicios y estudiantes. En la Figura 4 se muestra un ejemplo de Lahystotrain en ejecución. (22)

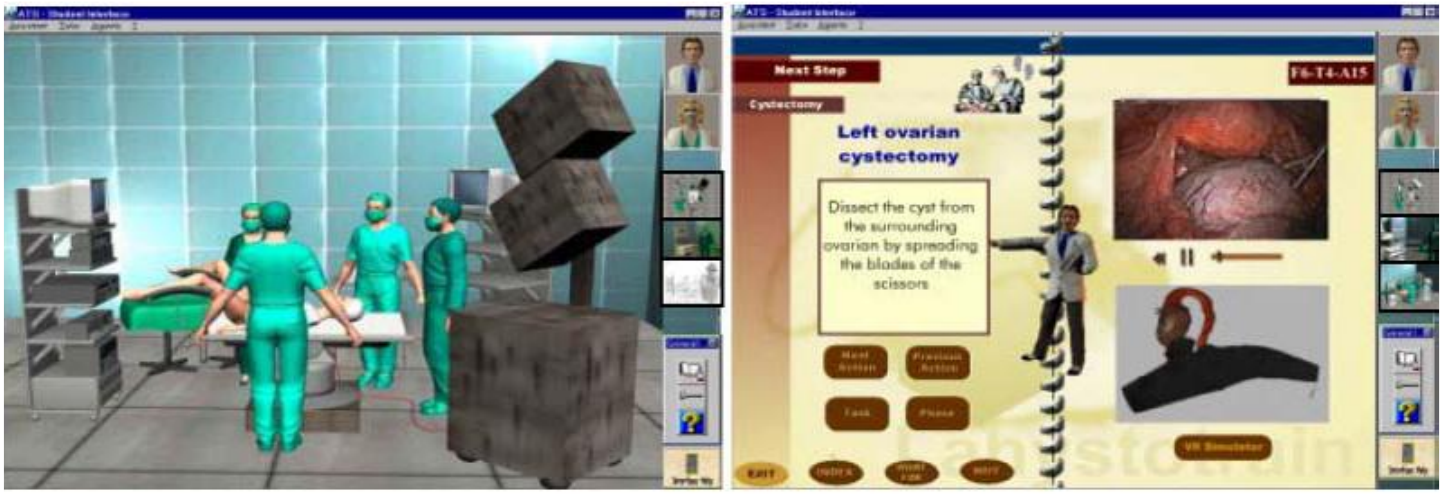

Figura 4. Ejemplo del AVA Lahystotrain

La aplicación está compuesta por una simulación de realidad virtual (RV) y un sistema de entrenamiento, el ATS (*Advanced Training System*). Este, a su vez, se compone de dos agentes pedagógicos, el asistente y el tutor, tres agentes de apoyo, el cirujano auxiliar, la enfermera y el anestesista, y una interfaz de usuario. Esta interfaz se utiliza para que el estudiante solicite explicaciones, así como la intervención del asistente y el tutor. También permite comunicarse con la enfermera, el anestesista y el cirujano auxiliar. Además, existe una base de datos que almacena el modelo del estudiante. Esta contiene una parte estática para guardar los datos del estudiante, y otra dinámica, donde se van almacenando las operaciones realizadas por el estudiante, su resultado, los errores cometidos y su conocimiento de las patologías y el instrumental usado. En la Figura 5 se muestra la arquitectura de este sistema. (22)

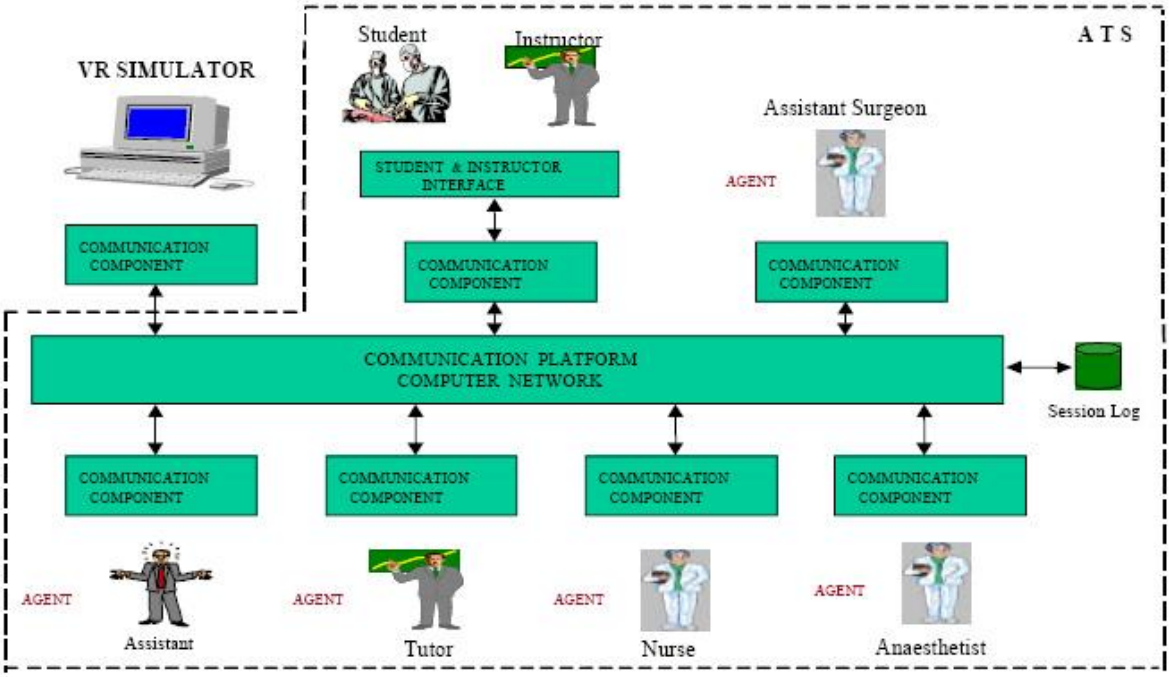

Figura 5. Arquitectura del AVA Lahystotrain

Laboratorios virtuales: Una 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, y elaborar y difundir resultados mediante tecnologías difundidas de información y comunicación. En el centro Vertex de la UCI se encuentra el proyecto Laboratorios Virtuales que tiene como objetivo el desarrollo de este tipo de aplicaciones de apoyo para la educación. Existen varios tipos de laboratorios virtuales, entre ellos se pueden citar:

- **Laboratorios virtuales *desktop*:** Están desarrollados como un programa independiente para ser ejecutados en los ordenadores, no requieren de un servidor web.
- **Laboratorios virtuales web:** Este tipo de laboratorios se basa en un software que depende de los recursos de un servidor determinado.
- **Laboratorios remotos:** Estos laboratorios requieren de equipos servidores específicos que les den acceso a las máquinas a operar de forma remota, y no pueden ofrecer su funcionalidad ejecutándose de forma local. (23)

La arquitectura usada para el desarrollo de laboratorios virtuales combina los estilos arquitectónicos basados en capas y en componentes, analizadas en los epígrafes 1.5.1 y 1.5.2 respectivamente. El uso de estos estilos permite obtener amplios niveles de reusabilidad y facilidad de desarrollo. En la Figura 6 se muestra cómo se encuentra estructurada la arquitectura de los laboratorios virtuales desarrollados por este proyecto.

Figura 6. Arquitectura de Laboratorios Virtuales propuesta por Merzeau Martínez, 2012. (23)

1.9 Metodologías de desarrollo de software

Las metodologías de desarrollo de software imponen un proceso disciplinado sobre el desarrollo de software con el objetivo de hacerlo más eficiente y predecible. 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. (24)

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. (24)

Las metodologías se pueden dividir en dos grupos de acuerdo a sus características y objetivos en robustas y ágiles. Dentro de las robustas se encuentra RUP (acrónimo de *Rational Unified Process* por sus siglas en inglés), cuyas características se analizan el próximo epígrafe.

1.9.2 Metodología de desarrollo RUP

Esta metodología se presenta como un proceso formal que 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. Fue desarrollado por

Rational Software, y está integrado con toda la suite Rational de herramientas. Puede ser adaptado y extendido para satisfacer las necesidades de la organización que lo adopte. Es guiado por casos de uso y centrado en la arquitectura, y utiliza UML como lenguaje de notación. (25)

Principales características de RUP:

Guiado por casos de uso: Los casos de uso son la técnica que utiliza RUP para la especificar los requisitos del sistema y además guían su diseño, implementación y prueba. Constituyen un elemento integrador y guía de trabajo, a través de ellos se genera todo el flujo de trabajo que sigue el proceso de desarrollo.

Centrado en la arquitectura: Para guiar el proceso de desarrollo además de los casos de uso, RUP presta especial atención al establecimiento de una buena arquitectura que no se vea afectada ante cambios durante la construcción y el mantenimiento. La arquitectura brinda una visión completa del sistema en la que los desarrolladores y los usuarios deben estar de acuerdo; esta se representa a través de varias vistas que juntas forman el llamado modelo de 4+1 vistas (26), el cual recibe este nombre porque está compuesto por las vistas lógica, de implementación, de proceso y de despliegue, más la de casos de uso que es la que da cohesión a todas.

Proceso iterativo e incremental: El proceso iterativo e incremental consta de una secuencia de iteraciones. Cada iteración abarca una parte de la funcionalidad total, pasando por todos los flujos de trabajo relevantes y refinando la arquitectura. Cada iteración se analiza cuando termina y se puede determinar si han aparecido nuevos requisitos o han cambiado los existentes afectando a las iteraciones siguientes. Toda la retroalimentación de la iteración pasada permite reajustar los objetivos para las siguientes iteraciones.

RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en dependencia del tipo de proyecto. En la Figura 7 se representa en nivel de esfuerzo que se realiza en cada disciplina según la fase en que se encuentre el proyecto. La fase de concepción o inicio tiene por finalidad definir la visión, los objetivos y el alcance del proyecto, la fase de elaboración tiene como principal finalidad completar el análisis de los casos de uso y definir la arquitectura del sistema, la fase deconstrucción está compuesta por un ciclo de varias iteraciones, en las cuales se van incorporando sucesivamente los casos de uso, de acuerdo a los factores de riesgo del proyecto y la fase de transición se inicia con una versión “beta” del sistema y culmina con el sistema en fase de producción.

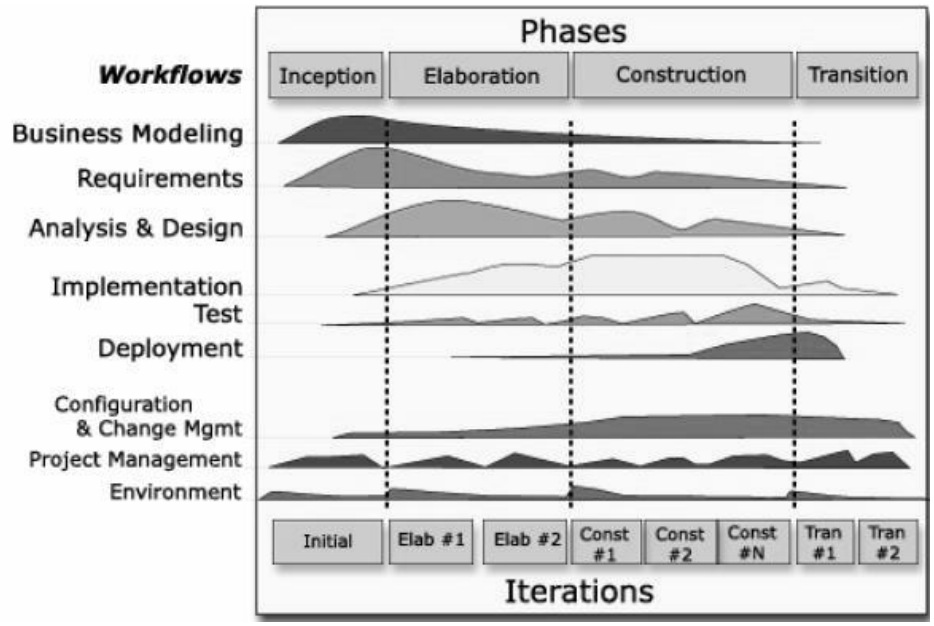

Figura 7. Esfuerzo en actividades según fase de proyecto en metodología RUP

1.10 Lenguaje Unificado de Modelado (UML)

El Lenguaje Unificado de Modelado, UML por sus siglas en inglés es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. (27) Este lenguaje de modelado es utilizado ampliamente por muchas metodologías de desarrollo de software, pero su estrecha relación con RUP se debe a que esta hace uso de todos sus elementos y diagramas.

1.10.1 Herramienta de Ingeniería de Software Asistida por Computadora para UML

Las herramientas conocidas como CASE (acrónimo de *Cumputer Aided Software Engineering,* por sus siglas en inglés), son programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software. Este puede ser generalmente aplicado a cualquier sistema o colección de herramientas que ayudan a automatizar el proceso de diseño y desarrollo de software.

Dentro de las herramientas más usadas en la actualidad se encuentra **Visual Paradigm for UML**, la cual posee soporte multiplataforma, licencia privada en alguna de sus ediciones y libre y gratuita en otras. Esta herramienta está concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas, entre sus principales características se encuentran que

hace uso de un lenguaje estándar y común a todo el equipo de desarrollo que facilita la comunicación y posee capacidades de ingeniería directa e inversa.

1.11 Motores de juego

Un motor de juegos es un sistema creado exclusivamente para el desarrollo de video juegos. La funcionalidad básica de un motor de juegos es proveer al video juego de un motor de renderizado[3] para los gráficos 2D y 3D, motor físico o detector de colisiones, sonidos, *scripting*, animación, inteligencia artificial, redes, *streaming*, administración de memoria y un escenario gráfico. El trabajo de un motor de juegos es el de realizar todas las tareas de bajo nivel tales como comunicarse con el adaptador gráfico, administrar la escena, transformar la geometría del mundo 3D ó 2D y lidiar con diversos procesos matemáticos que afectan su comportamiento. El motor del juego es el elemento más importante de todo proyecto de desarrollo de video juegos ya que está compuesto de todas las instrucciones necesarias para la representación del juego.

A continuación se comparan las principales características de los motores de juego usados para el desarrollo de aplicaciones en el proyecto: Shiva 3D y Unity 3D.

Tabla 2. Comparación de motores de juego Shiva 3D y Unity 3D

Aspectos/Motor de juego	Shiva 3D	Unity 3D
Multiplataforma	Si	Si
Licencias	Posee tres licencias: Personal *Learning* (N/A), sin costo, *Basic* (por maquina), 169 euros, y *Advanced* (por maquina), 1499 euros.	Posee dos tipos de licencias principales: Unity y Unity-Pro. La primera es gratuita (con algunas limitaciones) y está orientada a estudiantes y personas que deseen aprender a trabajar con este motor. La versión Pro no es gratuita, pero tiene características adicionales. Tanto Unity como Unity-Pro incluyen el entorno de desarrollo, tutoriales, ejemplos de proyectos y el contenido, el apoyo a través de la comunidad de foros, wiki, y las futuras de actualizaciones de la versión.
Lenguajes soportados	Soporta los lenguajes Lua y C++.	Unity soporta tres lenguajes de Scripting: JavaScript, C#, y una versión modificada de Python llamada

[3] Proceso que incluye la interpretación y la evaluación de componentes gráficos para un medio en específico.

		Boo. Los tres son igual de rápidos e interoperables. Además pueden utilizar librerías .NET con soporte a base de datos, expresiones regulares, XML, acceso a ficheros y funciones de red.
Red	Servidor embebido, *Video streaming*, *Data streaming.*	Posee funcionalidades multijugador como chat y puntuaciones en línea. Brinda solución a todas las necesidades de red para sus videojuegos.
Editor de escenas	Si	Si
Documentación	Posee poca documentación.	Posee una abundante y buena documentación.

Luego de analizar los principales aspectos de los motores gráficos usados en el proyecto resulta interesante destacar que Unity posee varias ventajas respecto a Shiva, este permite exportar sus aplicaciones para disímiles de formatos incluido para navegadores web mediante un *plugin* especial creado con tal propósito y disponible en su sitio web oficial de forma gratuita, aspecto que se pudiera explotar en futuros proyectos. Además este motor de juegos utiliza lenguajes de programación conocidos y de fácil implementación, tal es el caso de Javascript, lenguaje recomendado para usar en Unity, pero de igual forma pueden ser usados C# o Boo. Otra de sus ventajas es que posee documentación abundante y de fácil acceso en internet lo que facilita el desarrollo de sus aplicaciones.

1.13 Sistema Gestor de Base de Datos SQLite

En un ambiente virtual de aprendizaje es de vital importancia el almacenamiento de forma organizada de información referente a cuestiones propias del proceso de enseñanza-aprendizaje, por lo que, para lograr este objetivo se hace necesario hacer uso de un Sistema Gestor de Base de Datos.

SQLite es una biblioteca libre para cualquier uso y gratuita escrita en leguaje C que implementa un Sistema de Gestión de Base de Datos transaccionales SQL auto-contenido, sin servidor y sin configuración. Entre sus principales características se encuentran:

- **Tamaño:** tiene una pequeña memoria y una única biblioteca que es necesaria para acceder a bases de datos.
- **Rendimiento:** realiza operaciones de manera eficiente y es más rápido que MySQL y PostgreSQL.

- **Portabilidad:** se ejecuta en muchas plataformas y sus bases de datos pueden ser facilmente portadas sin ninguna configuración o administración.
- **Estabilidad:** es compatible con ACID, reunión de los cuatro criterios de Atomicidad, Consistencia, Aislamiento y Durabilidad.

Resulta importante destacar que la integración de una base de datos de las tradicionalmente usadas con Unity 3D resulta una tarea compleja, además que requiere del uso de varias herramientas y servidores, por lo que SQLite constituye una opción más factible para el almacenamiento de forma organizada de información.

1.14 Conclusiones del capítulo

El estudio realizado en este capítulo permitió:

- Analizar los principales conceptos referentes a los ambientes virtuales de aprendizaje, enfocándose en los ambientes virtuales de aprendizaje en 3D y su uso para el trabajo colaborativo, lo que servirá como punto de partida para comprender el tipo de aplicación a la que se le quiere establecer una arquitectura.

- Analizar las arquitecturas de varias aplicaciones consideradas ambientes virtuales de aprendizaje, tomando como principal referencia para la propuesta arquitectónica del presente trabajo, la arquitectura de software para los laboratorios virtuales.

- Analizar los principales conceptos referentes a la Arquitectura de Software y la importancia de esta en la construcción de aplicaciones informáticas. Además se estudiaron las ventajas del uso de estilos y patrones arquitectónicos en el diseño de una arquitectura y se caracterizaron los patrones de diseño y de asignación general de responsabilidades; estos dos últimos sin ser elementos que tienen que estar presentes de manera obligatoria en el diseño, si representan una buena práctica para llegar a soluciones de problemas muy comunes y recurrentes.

- Analizar las características de la metodología de desarrollo de software RUP y el lenguaje de modelado UML, con los cuales se desarrollará la propuesta arquitectónica del presente trabajo, apoyado por la herramienta CASE Visual Paradigm for UML.

- Estudiar las principales características de los motores de juegos usados en el proyecto, prestando especial atención a las ventajas que posee Unity 3D para el desarrollo de la aplicación.

- Analizar las principales características de la biblioteca SQLite para la implementación de una base de datos.

Capítulo 2: Propuesta de arquitectura

En el presente capítulo se describe la arquitectura propuesta mediante el modelo de 4+1 vistas de la Arquitectura de Software propuesto por la metodología RUP. Se especifican los requisitos funcionales y no funcionales que esta debe satisfacer. Además se describen los elementos que permiten estructurarla, tales como los estilos arquitectónicos y los patrones de diseño.

2.1 Descripción general de la aplicación ProteSIM

Teniendo en cuenta el estudio realizado, se define la aplicación ProteSIM como un AVAI en el cual los estudiantes de las carreras de Estomatología y Tecnologías de la Salud podrán entrenarse en la construcción de prótesis bucomaxilofaciales.

Al entrar al sistema se presenta el menú principal de la aplicación, donde se encuentran las opciones de autenticar, videos, imágenes, configurar conexión y salir del programa. A continuación se describen las funciones de cada opción:

Autenticar: Tiene como objetivo permitir a los usuarios autentificarse y acceder a las secciones del sistema en dependencia su categoría (estudiante, profesor o administrador). En cada sección los usuarios realizan distintas funciones, las que se describen a continuación:

- **Estudiante:** En el menú principal de su sección debe escoger un rol de los tres existentes: técnico de laboratorio, asistente o clínico, luego deberá resolver un cuestionario evaluativo referente al rol escogido que indicará si está preparado o no para realizar el entrenamiento en el escenario virtual, en el que trabajará de forma colaborativa con dos estudiantes más, que ocupan el resto de los roles. En caso de existir menos de tres estudiantes y no se pueda completar el equipo de trabajo el sistema debe ser capaz de reemplazarlo de forma automática.
- **Profesor:** Gestiona los equipo de estudiantes, visualiza el reporte del entrenamiento de cada estudiante en escenario virtual y asigna una evaluación a los equipos.
- **Administrador:** Gestiona la información referente a los usuarios del sistema.

Videos: Tiene como objetivo mostrar videos de apoyo referentes a la construcción de prótesis bucimaxiloaciales, sin necesidad de autentificarse en la aplicación.

Imágenes: Tiene como objetivo mostrar imágenes de apoyo referentes a la construcción de prótesis bucimaxiloaciales, sin necesidad de autentificarse en la aplicación.

Configurar conexión: Tiene como objetivo realizar la configuración básica de la conexión en red, para lo cual el usuario no tiene que estar autentificado. Es importante destacar que esta configuración debe ser

supervisada por la persona que se encuentre dirigiendo la práctica en el laboratorio docente, ya que es necesario definir qué estación de trabajo funcionará como servidor, al cual los clientes se conectarán especificando dirección IP y puerto.

Salir: Tiene como objetivo cerrar la aplicación

2.2 Modelo del dominio.

El modelo del dominio es una representación visual mediante diagramas de clases de UML de la forma en que se relacionan los conceptos u objetos asociados a un problema específico. Este permite a desarrolladores y clientes entender el contexto del sistema y su funcionamiento. La metodología RUP propone usarlo cuando los procesos del negocio no están bien definidos. Un modelo de dominio captura los tipos más importantes de objetos que existen o los eventos que suceden en el entorno donde estará el sistema. A continuación se muestra el diagrama de modelo de dominio referente al problema.

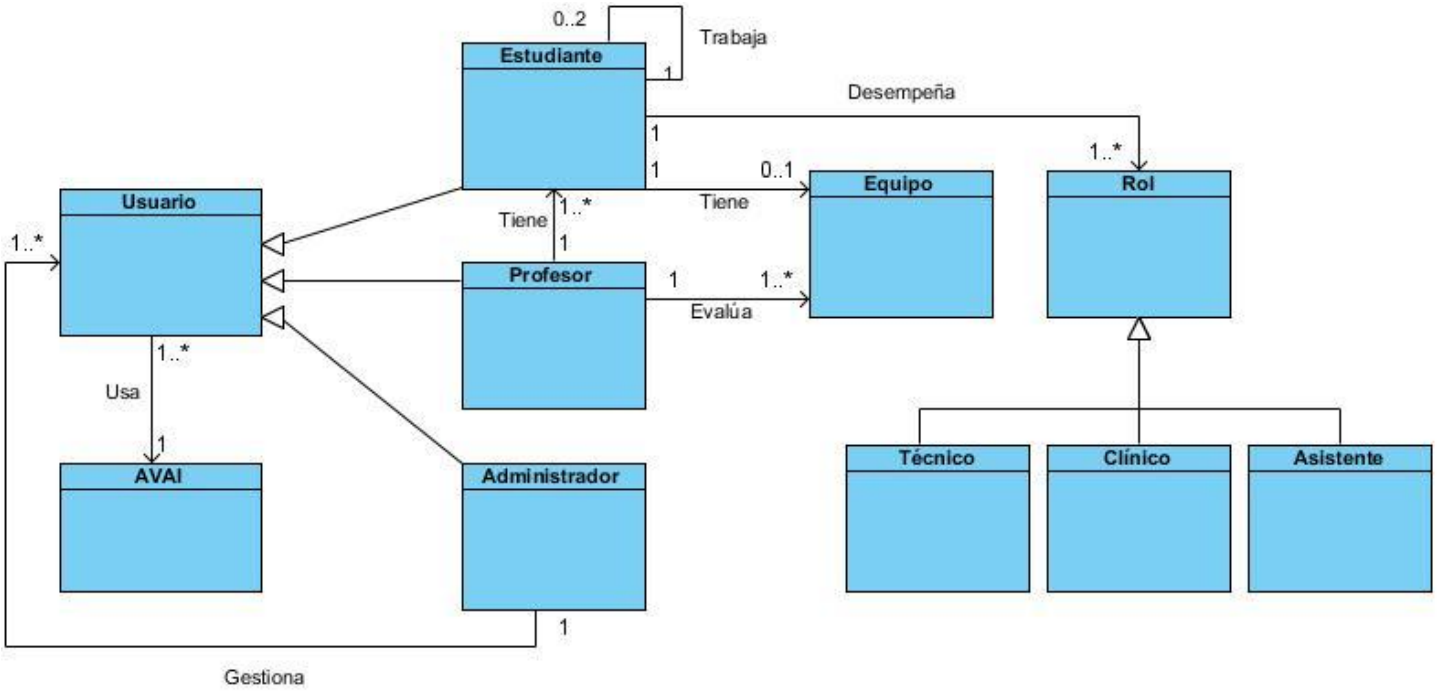

Figura 8. Modelo de dominio

Para mejor comprensión del modelo a continuación se explican los conceptos utilizados:

Estudiante: Es el usuario que usará el AVAI con fines de aprendizaje, haciendo uso de cuestionarios evaluativos y realizando el entrenamiento en el escenario.

Profesor: Es el usuario que desempeñará en al AVAI el rol propio de su profesión, deberá gestionar equipo de estudiantes y evaluarlos según desempeño en el entrenamiento en escenario virtual.

Administrador: Usuario capacitado para gestionar los usuarios que acceden al sistema.

Rol: Rol que desempeñará el usuario estudiante en el entrenamiento en el escenario virtual.

Equipo: Es el equipo que será conformado como máximo con 3 estudiantes para realizar el entrenamiento en el escenario virtual, asumiendo cada uno distintos roles de los tres existentes.

AVAI: Es la aplicación definida como AVAI, cuyo concepto se analizó en el epígrafe 1.1 del Capítulo 1.

2.3 Requerimientos del sistema

Los requerimientos de software son capacidades y condiciones con las cuales debe ser conforme el sistema, estos deben definir lo que realmente se necesita de manera que tenga un significado claro para clientes y desarrolladores. El objetivo fundamental del flujo de trabajo de los de los requisitos es guiar el proceso de desarrollo hacia el sistema correcto. Los requerimientos de un sistema de software se dividen en requisitos funcionales (RF) y requisitos no funcionales (RNF), a continuación se presentan los requerimientos que intervienen en el desarrollo de la AS propuesta.

2.3.1 Requisitos funcionales

Los requisitos funcionales (RF) son las declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares, y de cómo se debe comportar en situaciones particulares. (28) En la Tabla 3 se describen los requisitos funcionales agrupados por los usuarios del sistema que lo realizan.

Tabla 3. Requisitos funcionales

No.	Nombre	Descripción
Requisitos funcionales comunes para todos los usuarios		
RF. 1	Autenticar.	Permitir a los usuarios acceder a las secciones de la aplicación en correspondencia con su categoría.
RF. 2	Mostrar videos.	Permitir a los usuarios visualizar videos de apoyo.
RF. 3	Mostrar imágenes.	Permitir al usuario visualizar imágenes de apoyo.
RF. 4	Conectar como cliente.	Permitir al usuario conectarse como cliente a un servidor de la aplicación previamente conectado.
RF. 5	Conectar como servidor.	Permitir al usuario conectar la aplicación como servidor.
RF. 6	Desconectar	Permitir al usuario desconectar la aplicación.
Requisitos funcionales realizados por el usuario administrador		

RF. 7	Insertar usuario.	Permitir al administrador insertar los datos de un usuario en la aplicación.
RF. 8	Eliminar usuario.	Permitir al administrador eliminar un usuario de la aplicación.
RF. 9	Modificar usuario.	Permitir al administrador modificar uno o varios datos referentes a un usuario de la aplicación.
Requisitos funcionales realizados por el usuario profesor		
RF. 10	Insertar equipo.	Permitir al profesor insertar un equipo de estudiantes en la aplicación.
RF. 11	Eliminar equipo.	Permitir al profesor eliminar un equipo de estudiantes de la aplicación.
RF. 12	Modificar equipo.	Permitir al profesor modificar los integrantes que conforman un equipo de estudiantes.
RF. 13	Evaluar equipo.	Permitir al profesor evaluar el trabajo realizado por un equipo de estudiantes.
RF. 14	Mostrar reportes de práctica de simulación.	Permitir al profesor visualizar el reporte de entrenamiento de los estudiantes en el escenario virtual.
Requisitos funcionales realizados por el usuario estudiante		
RF. 15	Realizar evaluación teórica.	Permitir al estudiante realizar un cuestionario evaluativo.
RF. 16	Mostrar puntuación de evaluación teórica.	Debe permitir al estudiante visualizar la puntuación obtenida en el cuestionario evaluativo.
RF. 17	Realizar entrenamiento en escenario virtual.	Debe permitir al estudiante interactuar en el escenario virtual para entrenamiento
RF. 18	Mostrar resultados de entrenamiento en escenario virtual	Debe permitir al estudiante visualizar el resultado del entrenamiento en el escenario virtual.

2.3.2 Requisitos no funcionales

Los requisitos no funcionales son restricciones de los servicios o funciones ofrecidos por el sistema. Estos no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. (28) A continuación se describen los requisitos no funcionales que intervienen en la AS propuesta:

Requerimientos de software

RNF. 1 Las PC deben tener instalado, Windows 2000 o superior o alguna distribución de Linux.

Requisitos de usabilidad

RNF. 2 El sistema debe tener una interfaz organizada y de fácil entendimiento tanto para usuarios avanzados como para los que no poseen conocimientos profundos de informática.

Requerimientos de hardware:

Requerimientos mínimos:

RNF. 3 Procesador Intel Pentium IV 2Ghz o equivalente.

RNF. 4 512 megabytes (MB) de RAM.

RNF. 5 Tarjeta gráfica con 64MB mínimo de RAM.

Requerimientos recomendados:

RNF. 6 Procesador de doble núcleo o superior.

RNF. 8 1 Gigabyte (GB) de memoria RAM o superior.

RNF. 9 Tarjeta gráfica con 256MB de memoria o superior.

Restricciones de diseño o implementación

RNF. 10 Como motor gráfico se debe utilizar Unity 3D

RNF. 11 El sistema será una aplicación de escritorio.

RNF. 12 El lenguaje de programación recomendado para Unity 3D es JavaScript, pero de igual forma se puede usar C# o Boo.

RNF. 13 El sistema deberá utilizar como plataforma de desarrollo el MonoDevelop el cual viene integrado con Unity, o alternativamente Visual Studio.

RNF. 14 El sistema usará como herramienta de modelación el Visual Paradigm for UML 8.0 Enterprise Edition.

RNF. 15 El sistema utilizará como Sistema Gestor de Bases de Datos la biblioteca SQLite.

2.4 Descripción general de la arquitectura

El primer paso en el diseño de una arquitectura, luego de haber analizado el problema es realizar el diseño arquitectónico a alto nivel de abstracción para luego pasar al diseño más detallado, o sea al diseño de más bajo nivel de abstracción.

Para el diseño de la arquitectura se propone el uso de la combinación de estilos arquitectónicos basados en capas y basados en componentes, teniendo en cuenta las ventajas que estos brindan tal como se expone en el Capítulo 1. En la Figura 9 se muestra las diferentes capas de la arquitectura propuesta, a continuación se describen cada una de ellas:

1. **Capa de presentación:** Es con la que interactuará directamente el usuario, está compuesta básicamente por las interfaces gráficas de usuario y los escenarios virtuales.
2. **Capa lógica:** Contiene los *scripts* que controlan y dan el comportamiento a las escenas de la aplicación. Los *scripts* representan las clases en el motor de juegos Unity 3D.
3. **Capa de soporte**: Es la que le da el soporte y el acceso a los datos a la capa lógica, en ella se encuentran las bibliotecas de la base de datos.

Figura 9. Diseño arquitectónico en capas

2.5 Vistas arquitectónicas

El diseño detallado de una arquitectura se realiza mediante múltiples vistas, para ello se utilizó el Modelo de 4+1 vistas propuesto por Philippe Kruchten (29), en el cual se representan diferentes aspectos y características de la arquitectura en varias vistas. Una vista es una presentación de un modelo, donde se describe de forma completa un sistema desde una particular perspectiva.

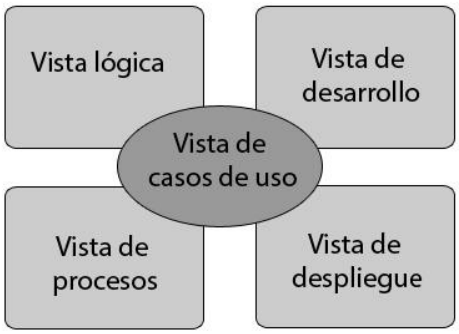

Figura 10. Modelo 4+1 vistas

2.5.1 Vista de casos de uso

La vista de casos de uso (CU) es un subconjunto del artefacto modelo de casos de uso, en esta se representa los casos de uso significativos para el diseño arquitectónico. Cada caso de uso arquitectónicamente significativo (CUAS) se corresponde con funcionalidades críticas o imprescindibles del sistema. En la Figura 11 se representa la vista de CU correspondiente a la arquitectura de software propuesta.

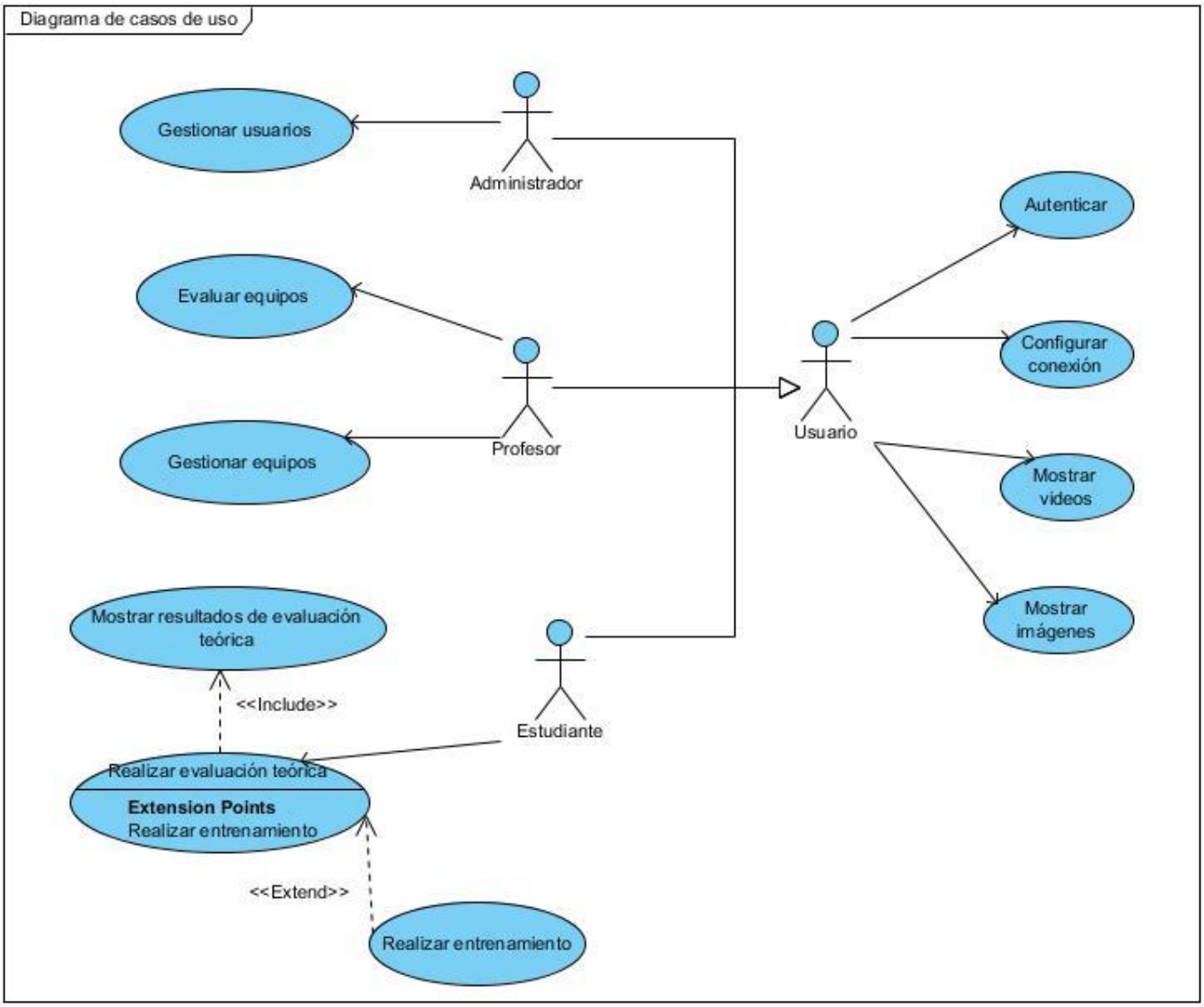

Figura 11. Vista de casos de uso

2.5.1.1 Descripción de los casos de uso.

Tabla 4. Caso de uso Autenticar

Caso de uso	Autenticar
Objetivo	Permitir al usuario autentificarse en el sistema para

	acceder a la sección que pertenece.
Actores	Usuario
Resumen	El caso de uso inicia cuando el usuario (administrador, profesor o estudiante) solicita autentificarse.
Complejidad	Media
Prioridad	Alta
Referencias	RF1
Precondiciones	El usuario tiene que estar registrado.
Postcondiciones	El usuario accede a la sección que pertenece.
Flujo de eventos	
Flujo básico <Autenticar>	
Acción del actor	**Respuesta del sistema**
1. El usuario (administrador, profesor o estudiante) selecciona la opción Autenticar en el menú principal de la aplicación.	2. El sistema muestra los campos de captura de datos: usuario y contraseña.
3. El usuario introduce los datos de autentificación y presiona el botón aceptar.	4. El sistema recoge los datos y los valida 5. El sistema entra a la sección a la que pertenece el usuario.
Flujos Alternos	
	Si los datos no son válidos el sistema muestra el mensaje “Datos no válidos”. Regresa al paso 2.

Tabla 5. Caso de uso Mostrar imágenes

Caso de uso	Mostrar imágenes
Objetivo	Mostrar imágenes de ayuda al usuario.
Actores	Usuario
Resumen	El caso de uso se inicia cuando el usuario (administrador, profesor o estudiante) se encuentra en el menú principal de la aplicación y presiona el botón de “Mostrar Imágenes” y visualiza las imágenes de apoyo.
Complejidad	Baja
Prioridad	Alta
Referencias	RF3
Precondiciones	El usuario debe estar en el menú principal de la aplicación.
Postcondiciones	El usuario visualizará imágenes de ayuda.
Flujo de eventos	

Flujo básico <Mostrar imágenes>	
Acción del actor	**Respuesta del sistema**
1. El usuario selecciona la opción Mostrar imágenes en el menú principal de la aplicación.	2. El sistema muestra imágenes de apoyo.
Flujos alternos	

Tabla 6. Caso de uso Mostrar videos

Caso de uso	Mostrar videos
Objetivo	Mostrar videos de ayuda al usuario.
Actores	Usuario
Resumen	El caso de uso se inicia cuando el usuario (administrador, profesor o estudiante) se encuentra en el menú principal de la aplicación y presiona el botón de “Mostrar videos” y visualiza los videos de apoyo.
Complejidad	Baja
Prioridad	Alta
Referencias	RF2
Precondiciones	El usuario debe estar en el menú principal de la aplicación.
Postcondiciones	El usuario visualizará videos de ayuda.
Flujo de eventos	
Flujo básico <Mostrar videos>	
Acción del actor	**Respuesta del sistema**
1. El usuario selecciona la opción Mostrar videos en el menú principal de la aplicación.	2. El sistema muestra videos de apoyo.
Flujos alternos	

Tabla 7. Caso de uso Configurar conexión

Caso de uso	Configurar conexión
Objetivo	Permitir al usuario trabajar en red con otros usuarios conectados.
Actores	Usuario
Resumen	El caso de uso inicia cuando el usuario (administrador, profesor o estudiante) se encuentra en el menú principal de la aplicación y selecciona la opción Configurar conexión, donde configurará la dirección IP y el puerto, y

	escogerá si trabará como servidor o se conectará como cliente de un servidor previamente conectado.
Complejidad	Alta
Prioridad	Alta
Referencias	RF4, RF5, RF6
Precondiciones	El usuario debe estar en el menú principal de la aplicación.
Postcondiciones	El usuario accede a la sección Configurar conexión.
Flujo de eventos	
Flujo básico <Configurar conexión>	
Acción del actor	**Respuesta del sistema**
1. El usuario selecciona la opción Configurar conexión del menú principal.	2. El sistema muestra la interfaz correspondiente a Configurar la conexión con los campos de captura de datos: dirección IP y puerto, estado de la conexión “Desconectado” y los botones: Conectar como cliente y Conectar como servidor.
3. El usuario (administrador, profesor o estudiante) introduce la dirección IP, el puerto y hace clic en el botón: -Conectar como cliente. -Conectar como servidor.	4. El sistema realiza las siguientes acciones en dependencia de la opción seleccionada por el usuario: -Conectar como cliente (ir a la Sección 1). -Conectar como servidor (ir a la Sección 2).
Flujos alternos	
1. El usuario introduce la dirección IP o el puerto de forma incorrecta.	2. El sistema muestra el mensaje: “Datos de conexión no válidos”. Regresa al paso 2.
Sección 1: Conectar como servidor	
Acción del actor	**Respuesta del sistema**
	1. El sistema muestra “Conectado como servidor”. 2. El sistema muestra la cantidad de clientes conectados a él. 3. El sistema muestra el número de *pings* hechos a los clientes. 4. .El sistema muestra la opción de Desconectar y Volver al menú inicial.
Flujos alternos	
1. El usuario presiona el botón Desconectar.	1.1 Regresa al paso 2 del flujo básico de eventos.

2. El usuario presiona el botón Volver al menú inicial.	2.1 El sistema muestra el menú principal de la aplicación.
Sección Conectar como cliente	
Acción del actor	**Respuesta del sistema**
1. El usuario introduce los datos necesarios para la conexión y presiona el botón Conectar como cliente.	2. El sistema muestra “Conectado como cliente”. 3. El sistema muestra el número de *pings* hechos a al servidor. 4. El sistema muestra la opción de desconectar y volver al menú inicial.
Flujos Alternos	
1. El usuario presiona el botón Desconectar.	1.1 Regresa al paso 2 del flujo básico de eventos.
2. El usuario presiona el botón Volver al menú inicial.	2.1 El sistema muestra el menú principal de la aplicación.

Tabla 8. Caso de uso Gestionar usuarios

Caso de uso	Gestionar usuarios
Objetivo	Permitir al usuario (administrador) adicionar, eliminar o modificar la información referente a los usuarios del sistema.
Actores	Usuario
Resumen	El caso de uso inicia cuando el usuario (administrador) se autentifica en el sistema y accede a la Sección de gestionar usuarios.
Complejidad	Alta
Prioridad	Alta
Referencias	RF7, RF8, RF9
Precondiciones	El usuario (administrador) debe haberse autentificado
Postcondiciones	El sistema queda actualizado en dependencia de la acción que se ejecute. Se adicionan, se actualizan o se eliminan usuarios.
Flujo de eventos	
Flujo básico <Gestionar usuarios>	
Acción del actor	**Respuesta del sistema**
El usuario (administrador) se autentifica y entra a la sección de gestionar usuarios del sistema.	El sistema muestra la interfaz correspondiente a la gestión de usuarios del sistema donde el usuario (administrador) puede realizar las siguientes acciones: -Insertar usuario (Ir a Sección 1). -Eliminar usuario (Ir a Sección 2). -Modificar usuario (Ir a Sección 3).
Sección 1: Adicionar usuarios	

Acción del actor	Respuesta del sistema
	1. El sistema muestra los campos de captura de identificador, datos: nombre, apellidos, usuario, contraseña y categoría para adicionar un usuario.
2. El usuario (administrador) introduce los datos referentes al usuario a registrar en el sistema.	3. El sistema valida los datos. 4. El sistema inserta al usuario en la base de datos.
Flujos Alternos	
	1. Si el nombre contiene caracteres especiales o números el sistema muestra el mensaje “Nombre no válido”. Regresa al paso 1. 2. Si los apellidos contienen caracteres especiales o números el sistema muestra el mensaje “Apellidos no válidos”. Regresa al paso 1. 3. Si los apellidos contienen caracteres especiales o números el sistema muestra el mensaje “Apellidos no válidos”. Regresa al paso 1. 4. Si el nombre de usuario ya está en uso el sistema muestra el mensaje “El usuario se encuentra en uso, seleccione otro”. Regresa al paso 1. 5. Si el identificador ya existe en la base de datos, el sistema muestra el mensaje “El identificador debe ser único”. Regresa al paso 1. 6. Si la categoría del usuario no es estudiante, profesor o administrador, el sistema muestra el mensaje “Categoría no válida”. Regresa al paso 1 7. Si el identificador del usuario ya existe el sistema mostrará el mensaje “El Id ya existe”. Regresa al paso 1.
Sección 2: Eliminar usuarios	
Acción del actor	**Respuesta del sistema**
	1. El sistema muestra el campo para capturar el identificador del usuario a eliminar.
2. El usuario introduce el identificador del usuario a eliminar.	3. El sistema valida los datos. 4. El sistema elimina al usuario de la base de datos.
Flujos Alternos	
	Si no existe un usuario con tal identificador en la base de datos el sistema muestra el mensaje “El usuario no existe”. Regresa al paso 1.

Sección 3: Modificar usuarios	
Acción del actor	**Respuesta del sistema**
	1. El sistema muestra los datos de los usuarios para seleccionar el que se necesite modificar.
2. El usuario (administrador) selecciona el identificador y el dato a modificar de un usuario.	3. El sistema muestra el campo para capturar el nuevo dato.
4. El usuario (administrador) introduce el dato que modificará al anteriormente guardado.	5. El sistema valida el dato.
Flujos Alternos	
	1. Si no existe un usuario con tal identificador en la base de datos el sistema muestra el mensaje “El usuario no existe”. Regresa al paso 1. 2. Si el nombre contiene caracteres especiales o números el sistema muestra el mensaje “Nombre no válido”. Regresa al paso 1. 3. Si los apellidos contienen caracteres especiales o números el sistema muestra el mensaje “Apellidos no válidos”. Regresa al paso 1. 4. Si los apellidos contienen caracteres especiales o números el sistema muestra el mensaje “Apellidos no válidos”. Regresa al paso 1. 5. Si el nombre de usuario ya está en uso el sistema muestra el mensaje “El usuario se encuentra en uso, seleccione otro”. Regresa al paso 1. 6. Si el identificador ya existe en la base de datos, el sistema muestra el mensaje “El identificador debe ser único”. Regresa al paso 1. 7. Si la categoría del usuario no es estudiante, profesor o administrador, el sistema muestra el mensaje “Categoría no válida”. Regresa al paso 1
8. El usuario (administrador) selecciona otro dato a modificar.	9. Regresa al paso 3 del flujo básico de la sección 3.

Tabla 9. Caso de uso Gestionar equipos

Caso de uso	Gestionar equipos
Objetivo	Permitir al usuario (profesor) adicionar, eliminar o modificar equipos de estudiantes.

Actores	Usuario
Resumen	El caso de uso inicia cuando el usuario (profesor) presiona el botón Gestionar equipos de estudiantes.
Complejidad	Alta
Prioridad	Alta
Referencias	RF10, RF11, RF12
Precondiciones	El usuario (profesor) debe estar en el panel del menú profesor.
Postcondiciones	El sistema queda actualizado en dependencia de la acción que se ejecute. Se adiciona, se actualiza o se elimina un equipo de estudiantes.
Flujo de eventos	
Flujo básico <Gestionar equipos>	
Acción del actor	**Respuesta del sistema**
1. El usuario (profesor) selecciona la opción Gestionar equipos del menú del menú profesor.	2. El sistema muestra una interfaz donde el usuario (profesor) puede realizar las acciones: -Adicionar equipo (Ir a Sección 1). -Eliminar equipo (Ir a Sección 2). -Modificar equipo (Ir a Sección 3).
Sección 1: Adicionar equipos	
Acción del actor	**Respuesta del sistema**
	1. El sistema muestra los estudiantes registrados en el sistema para conformar equipos de hasta 3 estudiantes.
2. El usuario (profesor) introduce un identificador para el equipo, selecciona los estudiantes que lo conformarán y lo adiciona.	3. El sistema valida los datos. 4. El sistema inserta el equipo en la base de datos.
Flujos Alternos	
	Si el identificador del equipo ya existe el sistema muestra el mensaje “El equipo ya existe”. Regresa al paso 1.
Sección 2: Eliminar equipos	
Acción del actor	**Respuesta del sistema**
	1. El sistema muestra el campo para capturar el identificador del equipo a eliminar.
2. El usuario (profesor) introduce el identificador del equipo a eliminar.	3. El sistema valida los datos. 4. El sistema elimina el equipo de la base de datos.
Flujos Alternos	
	Si no existe un equipo con tal identificador en la base de

	datos el sistema muestra el mensaje “El equipo no existe”. Regresa al paso 1 del flujo básico de la Sección 2.
Sección 3: Modificar equipos	
Acción del actor	**Respuesta del sistema**
	1. El sistema muestra los equipos registrados.
2. El usuario (profesor) selecciona el identificador y el dato a modificar de un equipo.	3. El sistema muestra el campo para capturar el nuevo dato.
4. El usuario (profesor) introduce el dato que modificará al anteriormente guardado.	5. El sistema valida el dato.
Flujos Alternos	
	1. Si el nombre contiene caracteres especiales o números el sistema muestra el mensaje “Nombre no válido”. Regresa al paso 3. 2. Si los apellidos contienen caracteres especiales o números el sistema muestra el mensaje “Apellidos no válidos”. Regresa al paso 3. 3. Si los apellidos contienen caracteres especiales o números el sistema muestra el mensaje “Apellidos no válidos”. Regresa al paso 3. 4. Si no existe un equipo con tal identificador en la base de datos el sistema muestra el mensaje “El equipo no existe”. Regresa al paso 3.
5. El usuario (administrador) selecciona otro dato a modificar.	6. Regresa al paso 3 del flujo básico de la Sección 3.

Tabla 10. Caso de uso Evaluar equipos

Caso de uso	Evaluar equipos
Objetivo	Permitir al usuario (profesor) evaluar equipos de estudiantes.
Actores	Usuario
Resumen	El caso de uso inicia cuando el usuario (profesor) solicita Evaluar equipos.
Complejidad	Alta
Prioridad	Alta
Referencia	RF13, RF14
Precondiciones	El equipo tiene que estar registrado.
Postcondiciones	El equipo quedará evaluado.
Flujo de eventos	

Flujo básico <Evaluar equipos>	
Acción del actor	**Respuesta del sistema**
1. El usuario (profesor) selecciona la opción Evaluar equipos del menú del profesor.	2. El sistema muestra los reportes del entrenamiento en el escenario virtual hecho por los estudiantes individualmente agrupados en equipos durante la práctica de simulación.
3. El usuario (profesor) selecciona el equipo a evaluar e introduce la evaluación.	4. El sistema valida el datos.
Flujos Alternos	

Tabla 11. Caso de uso Realizar evaluación teórica

Caso de uso	Realizar evaluación teórica
Objetivo	Permitir al usuario (estudiante) realizar la evaluación teórica para pasar al entrenamiento en el escenario virtual.
Actores	Usuario
Resumen	El caso de uso inicia cuando el usuario (estudiante) solicita Realizar evaluación teórica.
Complejidad	Alta
Prioridad	Alta
Referencia	RF15, RF16
Precondiciones	El usuario (estudiante) debe estar en el panel del menú de estudiante y haber escogido el rol que desempeñará.
Postcondiciones	El usuario (estudiante) realizará la evaluación teórica.
Flujo de eventos	
Flujo básico <Realizar evaluación teórica>	
Acción del actor	**Respuesta del sistema**
1. El usuario (estudiante) selecciona la opción Realizar evaluación teórica del menú del estudiante	2. El sistema muestra un cuestionario evaluativo
3. El usuario (estudiante) realiza el cuestionario evaluativo y obtiene el 75 % de la puntuación.	4. El sistema brinda acceso al entrenamiento en el escenario virtual.
Flujos Alternos	
1. El usuario estudiante realiza el cuestionario evaluativo y obtiene menos del 75% de la puntuación.	2. El sistema mostrará los errores cometidos en cada proceder.

Tabla 12 Caso de uso Mostrar resultados de evaluación teórica

Caso de uso	Mostrar resultados de evaluación teórica.
Objetivo	Permitir al usuario (estudiante) ver el resultado y los errores cometidos en el cuestionario evaluativo.
Actores	Usuario
Resumen	El caso de uso inicia cuando el usuario (estudiante) termina la evaluación teórica mediante un cuestionario y el sistema le muestra los resultados de la mismo
Complejidad	Alta
Prioridad	Alta
Referencia	RF17, RF18
Precondiciones	El usuario (estudiante) tiene que haber terminado el cuestionario evaluativo.
Postcondiciones	El sistema mostrará la puntuación obtenida y los errores cometidos.
Flujo de eventos	
Flujo básico <Realizar práctica de simulación>	
Acción del actor	**Respuesta del sistema**
1. El usuario (estudiante) termina el cuestionario evaluativo.	2. El sistema muestra la puntuación obtenida y los errores cometidos
Flujos Alternos	

Tabla 13. Caso de uso Realizar entrenamiento

Caso de uso	Realizar práctica de simulación
Objetivo	Permitir al usuario (estudiante) entrenarse en los procedimientos básicos de la construcción de prótesis bucomaxilofaciales, mediante el trabajo colaborativo con otros usuarios estudiantes conectados en el escenario virtual tridimensional.
Actores	Usuario
Resumen	El caso de uso inicia cuando el usuario (estudiante) termina la evaluación teórica con más del 75 % de la puntuación y el sistema de brinda acceso al entrenamiento en el escenario virtual.
Complejidad	Alta
Prioridad	Alta
Referencia	RF17, RF18

Precondiciones	El usuario (estudiante) tiene que haber terminado la evaluación teórica del rol que desempeñará con más del 75 % de puntuación.
Postcondiciones	El usuario (estudiante) se entrenará en la construcción de prótesis bucomaxilofaciales en el escenario virtual.
Flujo de eventos	
Flujo básico <Realizar práctica de simulación>	
Acción del actor	**Respuesta del sistema**
	1. El sistema muestra el escenario virtual tridimensional donde trabajará el usuario (estudiante).
2. El usuario (estudiante) se mueve en el escenario y manipula los objetos que este contiene.	3. El sistema responde antes los eventos realizados por el usuario (estudiante)
Flujos Alternos	

2.5.2 Vista lógica

La vista lógica representa un subconjunto del artefacto Modelo de Diseño y comprende el conjunto de abstracciones fundamentales tomadas mayormente del dominio del problema. En esta se representan los elementos de diseño más importantes para la arquitectura y se describen las clases más importantes, su organización en paquetes y subsistemas.

Con esta representación se persigue el objetivo de brindar una vista lo más abstracta posible de cómo estará estructurada la aplicación ProteSim, debido a que no se persigue describir los elementos del diseño específicos de la aplicación, sino que se adapte a las necesidades de los desarrolladores y que sirva de base para futuras aplicaciones similares dentro del proyecto. En la Figura 12 se muestran las diferentes capas de la aplicación y la descripción de los principales paquetes que contiene cada una.

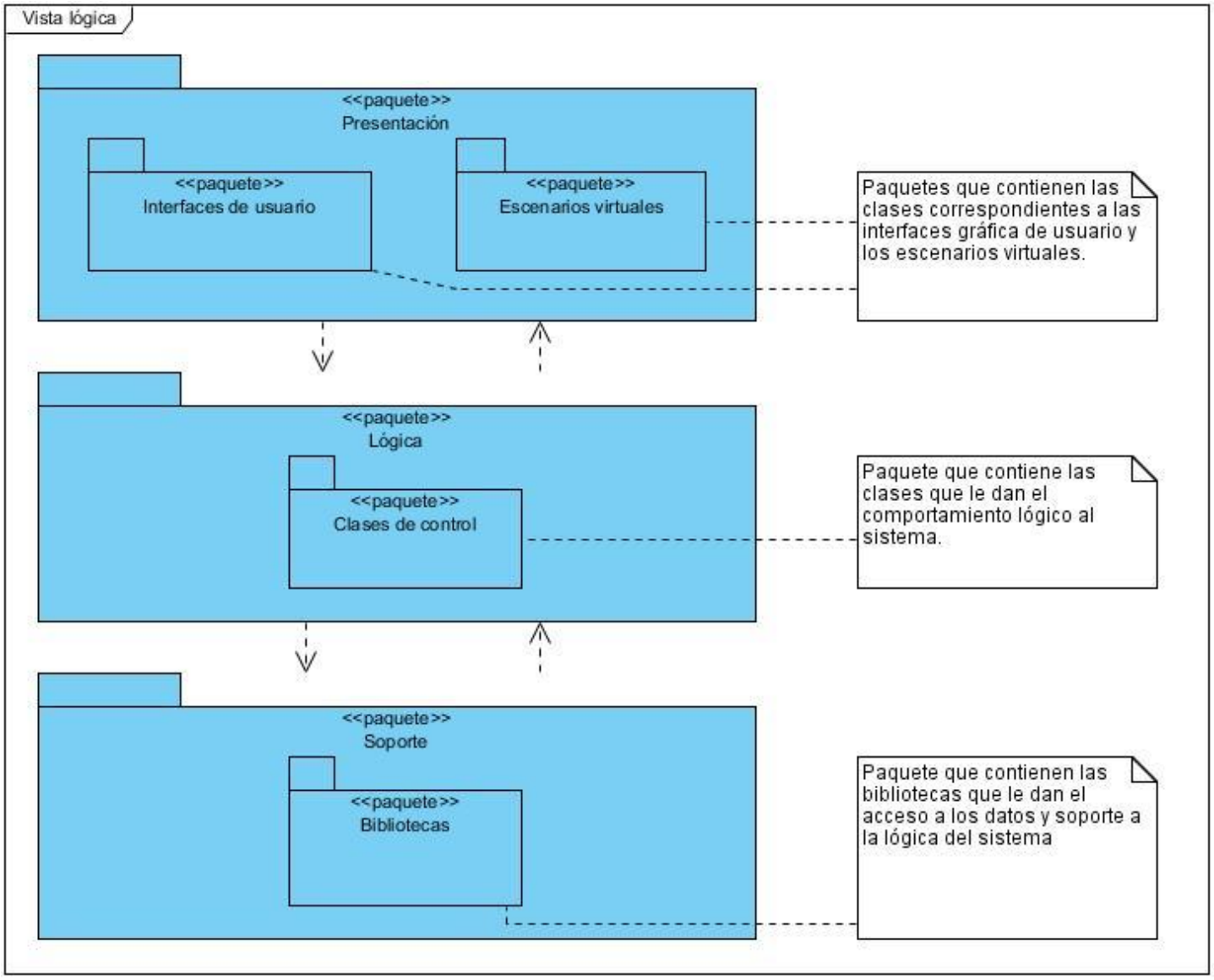

Figura 12. Vista lógica

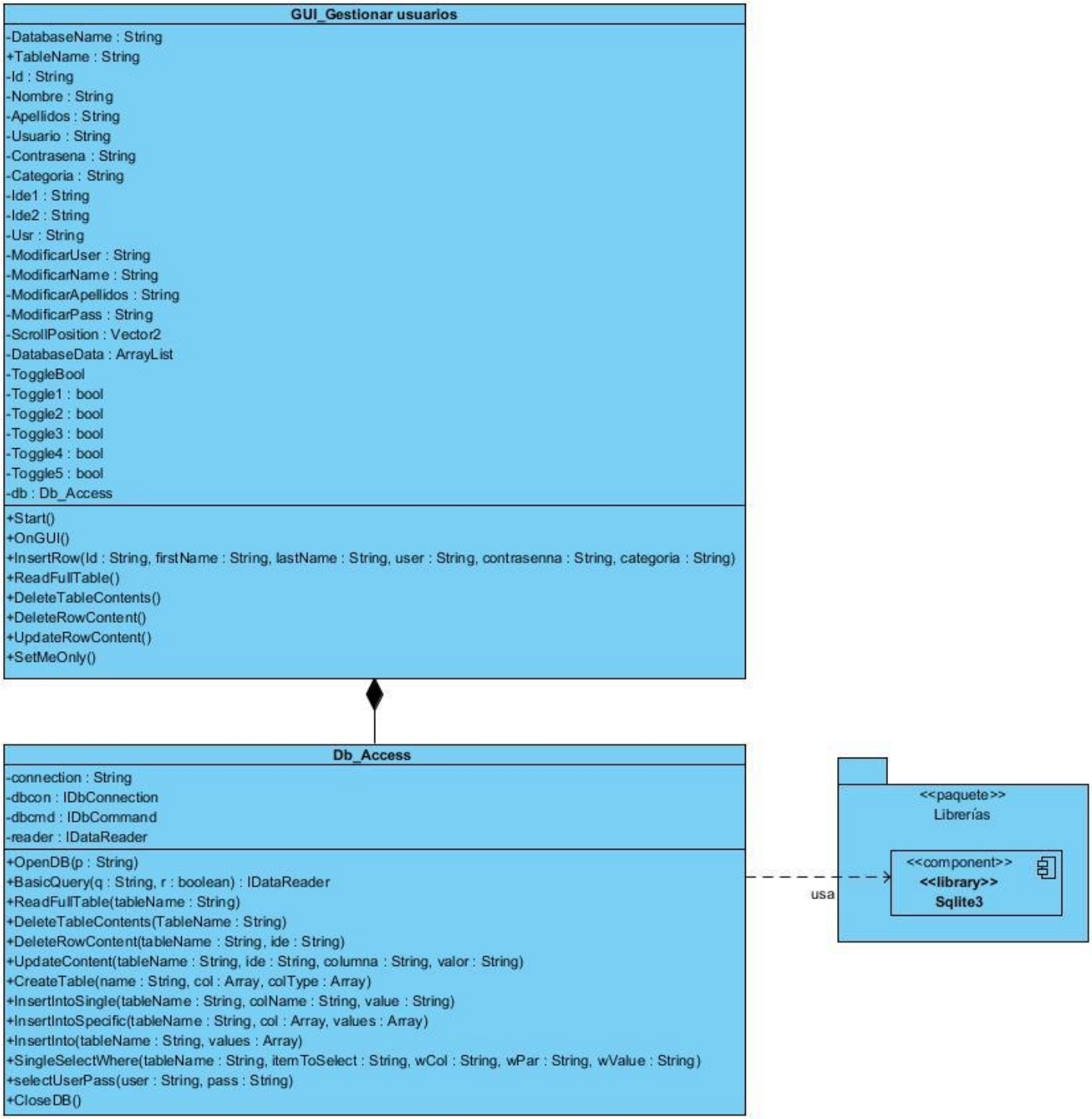

Figura 13. Ejemplo de diagrama de clases del CUAS Gestionar usuarios

2.5.3 Vista de implementación

La vista de implementación o desarrollo describe los elementos físicos del sistema y sus relaciones. Esta se representa a través de diagramas de componentes, los cuales representan todos los tipos de elementos de software que entran en la fabricación de aplicaciones informáticas, pueden ser simples archivos, paquetes, o bibliotecas cargadas dinámicamente. Uno de los usos principales es que puede servir para ver qué

componentes pueden compartirse entre sistemas o entre diferentes partes de un sistema. En la Figura 14 se representa los principales componentes que estructuran la vista de implementación de la arquitectura propuesta.

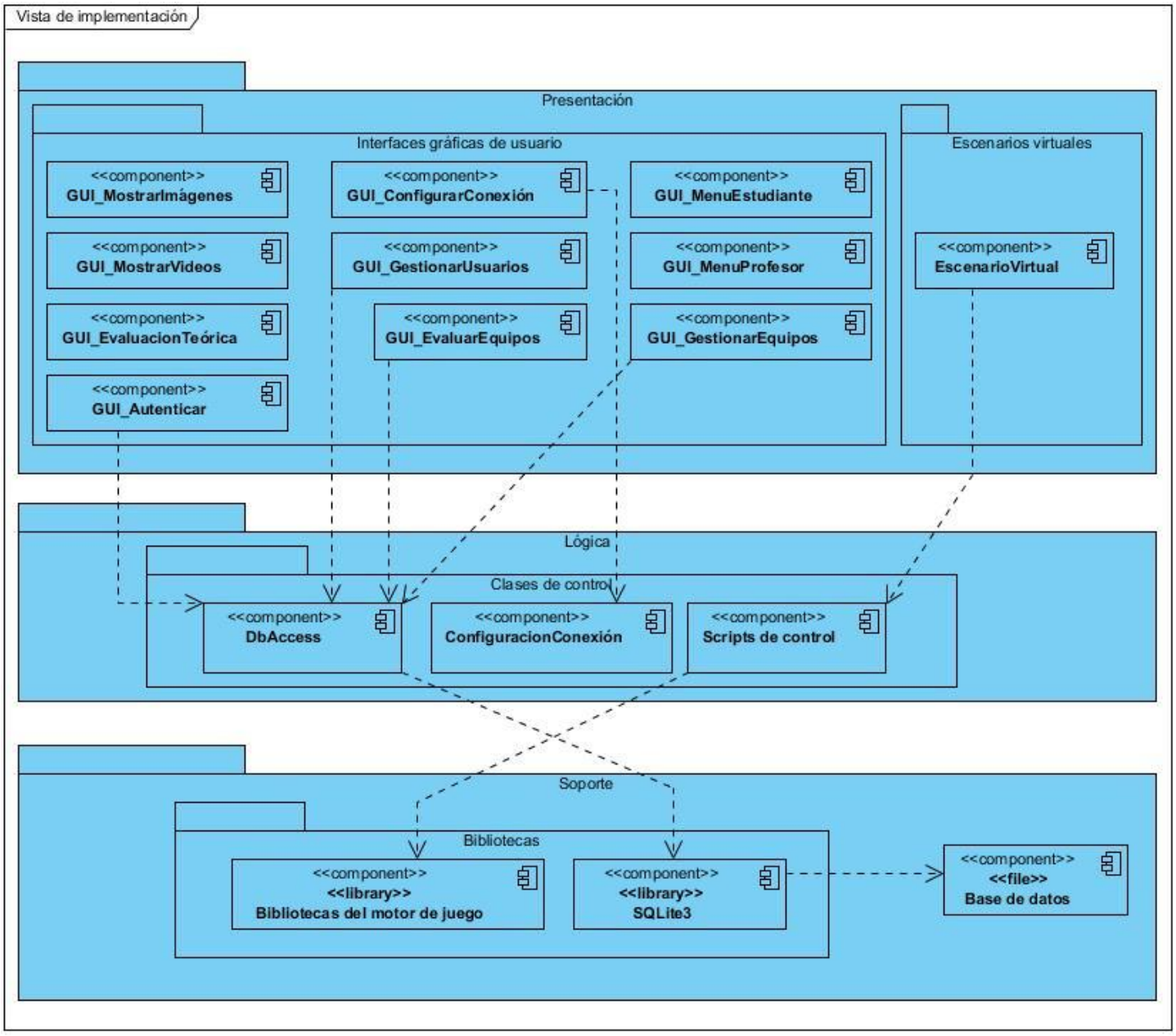

Figura 14. Vista de implementación

Los componentes ubicados en la capa de presentación corresponden a las clases de interfaces de usuarios y de los escenarios virtuales. Los componentes de interfaces de usuario que tienen que acceder al fichero de la base de datos para obtener información harán uso del componente **DbAcces**, ubicado en la capa lógica, el cual a hace uso de las funcionalidades de la biblioteca dinámica **SQLite3**, la cual se encarga de ejecutar las sentencias SQL necesarias para realizar acciones sobre la base de datos. De igual forma el componente

EscenarioVirtual hace uso de los componentes scripts de control ubicados en la capa lógica, los cuales se encargan de darle el control y el comportamiento a al escenario virtual. Es válido destacar que los componentes que se encuentran aislados en el diagrama, implícitamente se integran al núcleo del motor de juegos, en este caso a MonoBehaviour, núcleo básico de Unity 3D.

2.5.4 Vista de despliegue

La vista de despliegue define la arquitectura física del sistema por medio de nodos interconectados. Estos nodos son los elementos de hardware sobre los cuales pueden ejecutarse elementos de software. Los nodos y conexiones de esta vista, y la asignación de los objetos activos a nodos, pueden mostrarse en diagramas de despliegue. Estos diagramas también pueden mostrar cómo se asignan componentes ejecutables a nodos. (10)

La aplicación a desarrollar sobre la base de la arquitectura propuesta deberá ser desplegada en estaciones de trabajo que cumplan con los requerimientos analizados en el epígrafe 2.3.2 de la siguiente forma:

Variante 1: Si se usará de forma distribuida, el fichero ejecutable de la aplicación debe ejecutarse en todas las estaciones de trabajo, una de ellas funcionará como servidor y el resto conectadas como clientes mediante TCP/IP, tal como se muestra en la Figura. 15.

Variante 2: En caso de no usarse de forma distribuida la aplicación se desplegará en una estación de trabajo independiente, tal como se muestra en la Figura. 16.

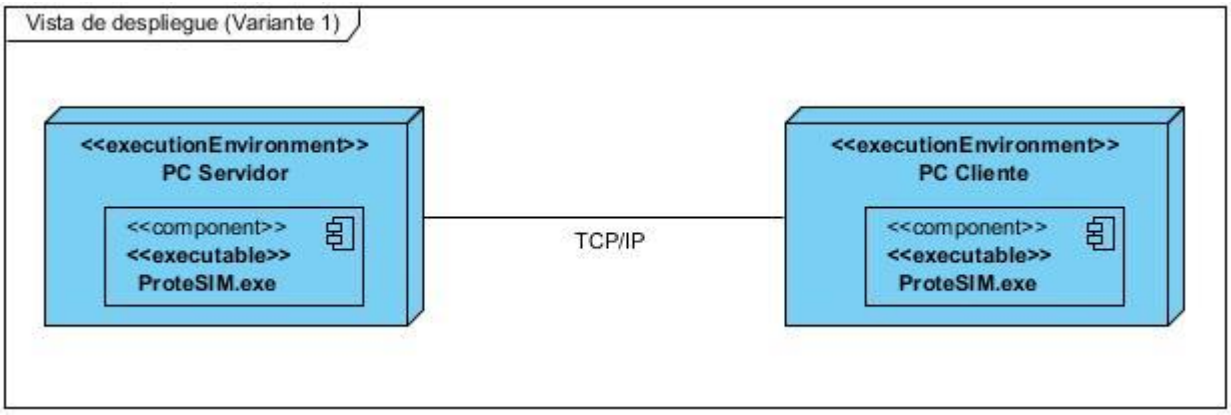

Figura 15. Diagrama de despliegue (Variante 1)

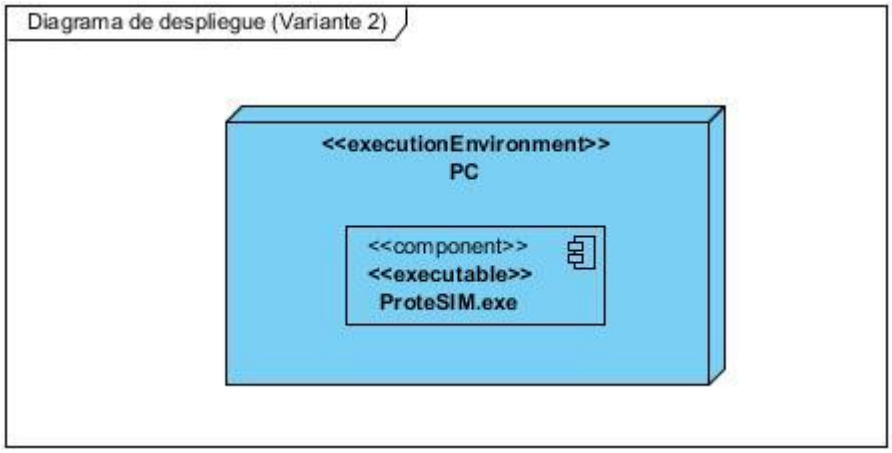

Figura 16. Diagrama de despliegue (Variante 2)

2.5.5 Vista de procesos

La vista de procesos se centra en la concurrencia y distribución de procesos del diseño. Esta muestra los hilos y procesos de ejecución así como la comunicación entre ellos. Esta vista es útil cuando el sistema presenta procesos concurrentes o hilos. El diseño arquitectónico propuesto no presenta procesos concurrentes por lo que esta vista queda omitida.

2.6 Patrones de diseño utilizados

Para el diseño de la solución propuesta se consideró el conveniente uso de patrones de diseño analizados en los epígrafes 1.6 y 1.7 del Capítulo 1. Teniendo en cuenta las características de la arquitectura propuesta se consideró el patrón Singleton y los patrones GRASP: bajo acoplamiento y alta cohesión. A continuación se describe el empleo de cada patrón.

Singleton: Se utilizó para crear una única instancia de la clase que contiene las funciones de acceso a la base de datos, con el objetivo de lograr mayor seguridad en la base de datos.

Bajo acoplamiento: Se tuvo en cuenta para establecer la menor cantidad de dependencias entre las clases y de esta forma obtener una mejor reutilización, puesto que si una clase depende muchas es difícil reutilizarla. El estilo arquitectónico en capas empleado en el diseño arquitectónico favorece a un bajo acoplamiento.

Alta cohesión: Se tuvo en cuenta para asignarle a las clases las responsabilidades exclusivas que esta debe desempeñar, con el objetivo de que no se vean afectadas por constantes cambios, sean reutilizables y fáciles de conservar.

2.7 Consideraciones de la arquitectura propuesta

El diseño arquitectónico, independientemente que está propuesto para ser usado de forma práctica con el motor de juegos Unity 3D, se elaboró lo más genérico posible como para ser usado con otros motores de juegos, adaptándolo a sus particularidades y sin que ello suponga una afectación en la integridad conceptual de la arquitectura.

Los estilos arquitectónicos basados en capas y en componentes utilizados para el diseño de la AS imponen sus propias restricciones. Cada capa debe contener exclusivamente los elementos que fueron descritos en el epígrafe 2.4 para obtener mayor independencia entre las mismas y por ende una mayor reusabilidad.

La arquitectura por si sola constituye una herramienta de ayuda para desarrollar el sistema, sin embargo se considera conveniente describir como se usa. Por lo que para el desarrollo del sistema teniendo como base la AS propuesta se propone seguir la estrategia de implementación basada en componentes. Un componente constituye una unidad modular con interfaces definidas que puede ser reutilizable en un contexto específico. Los componentes deben ser desarrollados por los integrantes del proyecto y deben poseer una descripción de sus dependencias y funcionalidades. Para implementarlos se debe tener en cuenta la capa a la cual se integrará, por lo que deben ser ajustados a las restricciones de la misma. Una vez desarrollados se prueban sus funcionalidades de forma independiente y acoplados a la arquitectura.

Aspectos que se deben considerar del diseño propuesto son los siguientes:

- La AS propuesta no garantiza un mecanismo para añadir funcionalidades a la aplicación que no sean controladas por los desarrolladores. Para añadir nuevas funcionalidades los desarrolladores deben implementarlas e integrarlas al sistema, el cual debe ser reemplazado en su nueva versión.

- El diseño incluye el uso de componentes dinámicos, como es el caso de SQLite3, el cual constituye una biblioteca para la gestión de la base de datos, tal como se explicó en el epígrafe 1.13 del Capítulo 1. El uso de componentes dinámicos implica que deban ser copiados donde quiera que se encuentre el ejecutable de la aplicación, de lo contrario se verá afectado el correcto funcionamiento del sistema.

Capítulo 3: Evaluación de la arquitectura propuesta

En el presente capítulo se analizan los atributos de calidad que son evaluados en una arquitectura de software. Además se hace un estudio de las técnicas y métodos de evaluación de arquitecturas para realizar una buena selección de los mismos y aplicarlo a la arquitectura propuesta, con el objetivo de determinar sus riesgos, fortalezas y debilidades.

3.1 Evaluación de arquitecturas de software

La arquitectura juega un papel de vital importancia para el proceso de desarrollo de software, por lo que en este sentido es primordial realizar una evaluación de la misma para determinar si cumple o no con los atributos de calidad requeridos. Para ello el primer paso es definir qué se desea evaluar y qué se puede evaluar, puesto que el diseño arquitectónico contempla unos atributos de calidad y excluye otros. Resulta importante aclarar que la evaluación de una arquitectura no asegura si la misma es buena o no, solo expresa donde se encuentran sus debilidades y fortalezas.

Una vez definidos los atributos de calidad que se evaluarán es necesario establecer el momento en que se realizará la evaluación de la arquitectura. Aunque esta es posible realizarla en cualquier momento, se proponen dos variantes que agrupan dos etapas distintas: temprano y tarde. Para la primera variante no es necesario que la arquitectura esté completamente especificada para efectuar la evaluación, y abarca desde las fases tempranas de diseño y a lo largo del desarrollo. La segunda variante propuesta consiste en realizar la evaluación de la arquitectura cuando ésta se encuentra establecida y la implementación se ha completado. (30)

3.1.1 Atributos de calidad

Los atributos de calidad son requerimientos adicionales del sistema que hacen referencia a características que este debe satisfacer diferentes a los requerimientos funcionales. Estos se definen específicamente como las propiedades de un servicio que presta el sistema a sus usuarios. Estos atributos se clasifican dos grupos: observables vía de ejecución (OVE) y no observables vía de ejecución. Los atributos que se clasifican como OVE determinan del comportamiento del sistema en tiempo de ejecución y los NOVE se establecen durante el desarrollo del sistema. A continuación se describen algunos de estos atributos. (31)

Tabla 14. Atributos de calidad observables vía ejecución

Atributo de calidad (OVE)	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.
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.

Tabla 15. Atributos de calidad no observables vía ejecución

Atributo de calidad (NOVE)	Descripción
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.
Interoperabilidad	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de integrabilidad.
Modificabilidad	Es la habilidad de realizar cambios futuros al sistema.
Mantenebilidad	Es la capacidad de someter a un sistema a reparaciones y evolución.
Portabilidad	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.
Reusabilidad	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.
Escalabilidad	Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental.
Capacidad de prueba	Es la medida de la facilidad con la que el software, al ser

	sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba.

3.2 Técnicas y Métodos de evaluación de arquitecturas

De los diversos planteamientos hechos en la actualidad para evaluar una AS, uno de los más usados resultan los planteados por Bosh y Kazman, de los cuales se tiene que la evaluación de las arquitecturas de software puede ser realizada mediante el uso de diversas técnicas y métodos. A continuación se analizan varias de estas opciones para realizar la evaluación de la AS propuesta.

3.2.1 Técnicas de evaluación

Las técnicas de evaluación de las arquitecturas tienen como principal objetivo crear especificaciones y predicciones respecto a la arquitectura propuesta para determinar si esta satisface o no los atributos de calidad requeridos. Estas técnicas requieren información del sistema a desarrollar que no está disponible durante el diseño arquitectónico, sino al principio del diseño detallado del sistema (32), las mismas se clasifican en cualitativas y cuantitativas como se muestra en la Figura 17.

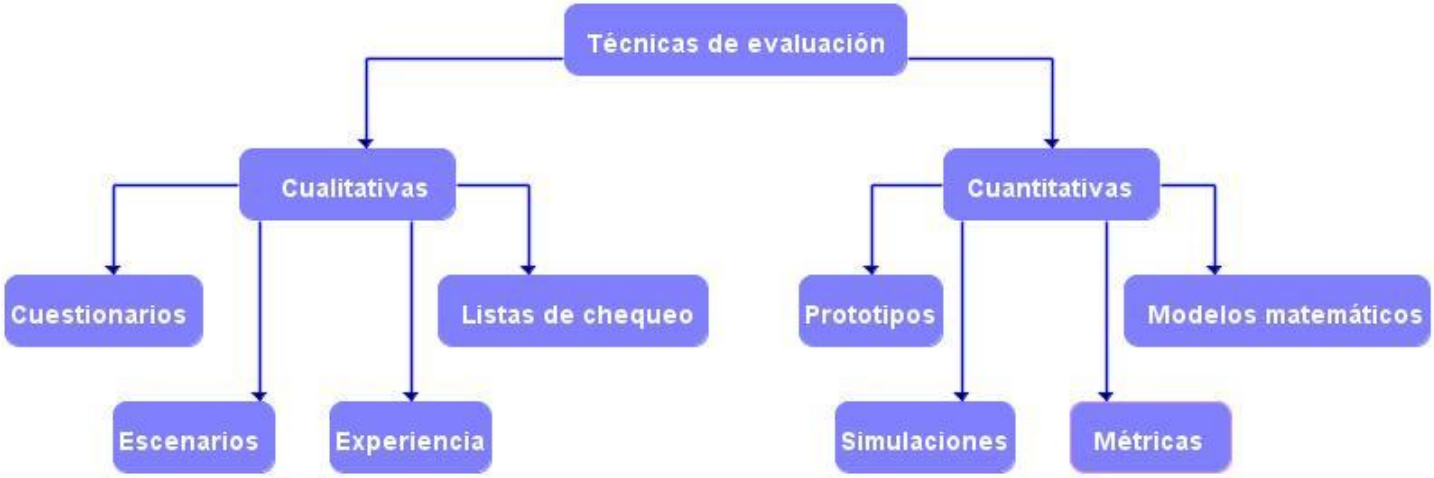

Figura 17. Técnicas de evaluación de la arquitectura

Las técnicas cualitativas son usadas cuando la arquitectura se encuentra en construcción, estas permiten tomar decisiones arquitectónicas en fases tempranas del desarrollo, requieren menor información detallada y pueden conducir a resultados relativamente precisos. (32) Las técnicas cuantitativas son más costosas de realizar y son usadas cuando la arquitectura ya ha sido implantada. A continuación se analizan varias de las técnicas más adecuadas a emplear en la evaluación la arquitectura propuesta.

3.2.1.1 Evaluación basada en escenarios

Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con éste. Consta de tres partes: estímulo, contexto y respuesta. El estímulo es la parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. El contexto describe qué sucede en el sistema al momento del estímulo. La respuesta describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado. (30) Esta técnica cuenta con dos instrumentos de evaluación de relevancia: *Utility Tree* (Árbol de utilidad) y *Profiles* (Perfiles).

Utility Tree (Árbol de utilidad): es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno. Su objetivo es la identificación de los atributos de calidad más importantes para un proyecto particular. Un árbol de utilidad tiene como nodo raíz la utilidad general del sistema, los atributos de calidad asociados al mismo constituyen el segundo nivel del árbol, los cuales se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado y otorgarle prioridad a cada atributo considerado. (30)

Profiles (Perfiles): es un conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de ellos. El uso de perfiles permite hacer especificaciones más precisas del requerimiento para un atributo de calidad. Para la definición de un perfil, es necesario seguir tres pasos: definición de las categorías del escenario, selección y definición de los escenarios para la categoría y asignación del peso a los escenarios. Los perfiles tienen asociados dos formas de especificación: perfiles completos y perfiles seleccionados. (32)

Los perfiles completos definen todos los escenarios relevantes como parte del perfil. Esto permite al ingeniero de software realizar un análisis de la arquitectura para el atributo de calidad estudiado de una manera completa, puesto que incluye todos los posibles casos. Su uso se reduce a sistemas relativamente pequeños y sólo es posible predecir conjuntos de escenarios completos para algunos atributos de calidad. (32)

Los perfiles seleccionados se asemejan a la selección de muestras sobre una población en los experimentos estadísticos. Se toma un conjunto de escenarios de forma aleatoria, de acuerdo a algunos requerimientos. La aleatoriedad no es totalmente cierta por limitaciones prácticas, por lo que se fuerza la realización de una selección estructurada de elementos para el conjunto de muestra. Si bien es informal, permite hacer proposiciones científicamente válidas. (32)

3.2.1.2 Evaluación basada en simulación

La evaluación basada en simulación utiliza una implementación de alto nivel de la arquitectura de software. El enfoque básico consiste en la implementación de componentes de la arquitectura y la implementación a cierto nivel de abstracción del contexto del sistema donde se supone va a ejecutarse. La finalidad es evaluar el comportamiento de la arquitectura bajo diversas circunstancias. Una vez disponibles estas implementaciones, pueden usarse los perfiles respectivos para evaluar los atributos de calidad. (32)

La exactitud de los resultados de la evaluación depende de la exactitud del perfil utilizado para evaluar el atributo de calidad y de la precisión con la que el contexto del sistema simula las condiciones del mundo real. En términos de los instrumentos asociados a las técnicas de evaluación basadas en simulación, se encuentran los lenguajes de descripción arquitectónica y los modelos de colas. (32)

3.2.1.3 Evaluación basada en experiencia

La evaluación basada en experiencia establece que en muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas de diseño. Aunque todas estas experiencias se basan en evidencia anecdótica; es decir, basada en factores subjetivos como la intuición y la experiencia. Sin embargo, la mayoría de ellas puede ser justificada por una línea lógica de razonamiento, y pueden ser la base de otros enfoques de evaluación. Existen dos tipos de evaluación basada en experiencia: la evaluación informal, que es realizada por los arquitectos de software durante el proceso de diseño, y la realizada por equipos externos de evaluación de arquitecturas. (32)

3.2.1.4 Evaluación basada en prototipos

Esta técnica implementa una parte de la arquitectura de software y la ejecuta en el contexto del sistema. Es utilizada para evaluar requerimientos de calidad operacional, como desempeño y confiabilidad. Para su uso se necesita mayor información sobre el desarrollo y disponibilidad del hardware, y otras partes que constituyen el contexto del sistema de software. Se obtiene un resultado de evaluación con mayor exactitud. (32)

Un prototipo es un modelo a escala o facsímil de lo real, pero no tan funcional para que equivalga a un producto final, ya que no lleva a cabo la totalidad de las funciones necesarias del sistema final, este proporciona una retroalimentación temprana por parte de los usuarios acerca del Sistema. Entre las ventajas de su uso se destacan la confiabilidad, pues se puede observar de forma directa que tanto se afecta o no, un atributo de calidad en la aplicación que se desarrolla, los usuarios finales pueden observar el resultado que se está obteniendo y evaluar junto al equipo de desarrollo si satisface o no sus expectativas. El éxito del uso del prototipo depende de qué tan pronto y con qué frecuencia se reciba la retroalimentación del usuario para hacer cambios y adecuarlos a las necesidades actuales.

3.2.2 Métodos de evaluación de arquitecturas

Los métodos de evaluación de la arquitectura se orientan hacia la mitigación de riesgos, ubicándolo donde un atributo de calidad se ve afectado por decisiones arquitectónicas, estos métodos se complementan con las técnicas de evaluación arquitectónica analizadas anteriormente.

La existencia de un método de análisis de arquitecturas de software hace que el proceso sea repetible, y ayuda a garantizar que las respuestas correctas con relación a la arquitectura pueden hacerse temprano, durante las fases tempranas de diseño. Es en este punto donde los problemas encontrados pueden ser solucionados de una forma relativamente poco costosa. De manera similar, un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema, en la búsqueda de conflictos que puede presentar una arquitectura, y sus soluciones. (30)

Actualmente existen diversos métodos de evaluación arquitectónica, algunos de los más reconocidos y usados son los propuestos por Kazman, estos son:

- Método de Análisis de Arquitectura de Software (*Software Architecture Analysis Method,* SAAM).
- Método de Revisión Intermedio de Diseño (*Active Reviews for Intermediate Designs*, ARID).
- Método de Análisis de Acuerdos de Arquitectura de Software (*Architecture Trade-off Analysis Method*, ATAM). (30)

3.2.2.1 Método de Análisis de la Arquitectura de Software (SAAM)

El método de evaluación SAAM 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. Es considerado muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad. 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. Como entrada principal, es necesaria alguna forma de descripción de la arquitectura a ser evaluada. Las salidas de la evaluación del método SAAM son las siguientes:

- Una proyección sobre la arquitectura de los escenarios que representan los cambios posibles ante los que puede estar expuesto el sistema.
- Entendimiento de la funcionalidad del sistema, e incluso una comparación de múltiples arquitecturas con respecto al nivel de funcionalidad que cada una soporta sin modificación. (30)

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 requerimientos de modificabilidad. Para el caso en el que se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite

observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones (33). En la Tabla 16 se muestran los pasos propuestos por el método SAAM.

Tabla 16. Pasos del método SAAM

Pasos del SAAM	Descripción
1. Desarrollo de escenarios	Un escenario es una breve descripción de usos anticipados o deseados del sistema. De igual forma, estos pueden incluir cambios a los que puede estar expuesto el sistema en el futuro.
2. Descripción de la arquitectura	La arquitectura (o las arquitecturas candidatas) debe ser descrita haciendo uso de alguna notación arquitectónica que sea común a todas las partes involucradas en el análisis. Deben incluirse los componentes de datos y conexiones relevantes, así como la descripción del comportamiento general del sistema. El desarrollo de escenarios y la descripción de la arquitectura son usualmente llevados a cabo de forma intercalada, o a través de varias iteraciones.
3. Clasificación y asignación de prioridad de los escenarios	La clasificación de los escenarios puede hacerse en dos clases: directos e indirectos. Un escenario directo es el que puede satisfacerse sin la necesidad de modificaciones en la arquitectura. Un escenario indirecto es el que requiere modificaciones en la arquitectura para poder satisfacerse. Los escenarios indirectos son de especial interés para SAAM, pues permiten medir el grado en el que una arquitectura puede ajustarse a los cambios de evolución que son importantes para los involucrados en el desarrollo.
4. Evaluación individual de los escenarios indirectos	Para cada escenario indirecto, se listan los cambios necesarios sobre la arquitectura, y se calcula su costo. Una modificación sobre la arquitectura significa que debe introducirse un nuevo componente o conector, o que alguno de los existentes requiere cambios en su especificación.
5. Evaluación de la interacción entre escenarios	Cuando dos o más escenarios indirectos proponen cambios sobre un mismo componente, se dice que interactúan sobre ese componente. De forma similar, puede verificarse si la arquitectura se encuentra documentada a un nivel correcto de descomposición estructural.
6. Creación de la evaluación global	Debe asignársele un peso a cada escenario, en términos de su importancia relativa al éxito del sistema. Esta asignación de

	peso suele hacerse con base en las metas del negocio que cada escenario soporta. En el caso de la evaluación de múltiples arquitecturas, la asignación de pesos puede ser utilizada para la determinación de una escala general.

3.2.2.2 Método de Revisión Intermedio de Diseño (ARID)

El método ARID es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. Este es un híbrido entre Active Design Review (ADR), utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos y Architecture Trade-Off Method (ATAM), el cual será analizado en el próximo epígrafe.

Tanto ADR como ATAM proveen características útiles para el problema de la evaluación de diseños preliminares, dado que ninguno por sí solo es conveniente. En el caso de ADR, los involucrados reciben documentación detallada y completan cuestionarios, cada uno por separado. En el caso de ATAM, está orientado a la evaluación de toda una arquitectura (30). Los pasos para la realización para evaluar las arquitecturas mediante ARID se resumen en la Tabla.

Tabla 17. Pasos del método ARID

Pasos del ARID	Descripción
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.
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, 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 explican 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 el pseudo-código que utiliza el diseño para proveer el servicio. 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	El facilitador recuenta la lista de puntos tratados y pide opiniones a los participantes sobre la eficiencia del ejercicio de revisión.

3.2.2.3 Método de Análisis de Acuerdos de Arquitectura de Software (ATAM)

El método ATAM está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. El nombre del método 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; esto es, los tipos de acuerdos que se establecen entre ellos (30). En la Tabla 18 se resumen los pasos de este método.

Tabla 18. Pasos del método ATAM

Pasos del ATAM	Descripción
Fase 1: Presentación	

1. Presentación del ATAM	El líder de evaluación describe el método a los participantes, trata de establecer las expectativas y responde las preguntas propuestas.
2. Presentación de las metas del negocio	Se realiza la descripción de las metas del negocio que motivan el esfuerzo y aclara que se persiguen objetivos de tipo arquitectónico.
3. Presentación de la arquitectura	El arquitecto describe la arquitectura, enfocándose en cómo ésta cumple con los objetivos del negocio.
Fase 2: Investigación y análisis	
4. Identificación de los enfoques arquitectónicos.	Se detectan los enfoques arquitectónicos pero no se analizan.
5. Generación del árbol de utilidad	Se especifican, en forma de escenarios, los atributos de calidad que engloban la "utilidad" del sistema. Se anotan los estímulos y respuestas, y se establece la prioridad entre ellos.
6. Análisis de los enfoques arquitectónicos	Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. En este paso se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance
Fase 3: Pruebas	
7. Lluvia de ideas y establecimiento de la prioridad de los escenarios	Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios.
8. Análisis de los enfoques arquitectónicos	Se repiten las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.
Fase 4: Reporte	
9. Presentación de los resultados	A partir de la información recolectada a lo largo de la evaluación del ATAM, se presentan los resultados a los participantes.

3.2.2.4 Comparación de los métodos de evaluación de arquitecturas

En la siguiente tabla se muestra un resumen de los principales aspectos de los métodos de evaluación de arquitecturas analizados anteriormente. (30)

Tabla 19. Comparación de los métodos ATAM, SAM y ARID

	ATAM	SAM	ARID
Atributos de calidad contemplados	-Modificabilidad -Seguridad -Confiabilidad -Desempeño	-Funcionalidad -Modificabilidad	Conveniencia del diseño evaluado

Objetos analizados	-Estilos Arquitectónicos -Documentación -Flujo de datos -Vistas Arquitectónicas	-Documentación -Vistas Arquitectónicas	Especificación de los componentes
Etapas del proyecto donde se aplica	Luego de que el diseño de la arquitectura ha sido establecido	Luego de que la arquitectura cuenta con funcionalidad ubicada en módulos	A lo largo del diseño de la arquitectura
Enfoques utilizados	-Utility Tree y lluvia de ideas para articular los requerimientos de calidad -Análisis arquitectónico que detecta puntos sensibles, puntos de balance	-Lluvia de ideas para escenarios y articular los requerimientos de calidad -Análisis de los escenarios para verificar funcionalidad o estimar el costo	Revisiones de diseños, lluvia de ideas para obtener escenarios

3.3 Evaluación de la arquitectura propuesta

Luego del estudio de las diferentes técnicas y métodos de evaluación de las arquitecturas de software se selecciona para evaluar la arquitectura propuesta la técnica basada en prototipos, ya que esta permite obtener un resultado más exacto y se puede observar de forma directa que tanto se afecta o no un atributo de calidad.

Como método de evaluación se selecciona ARID. Después de haber comparado los tres métodos estudiados se pudo constatar que el escogido resulta mucho más ligero que el resto, y es útil para ser aplicado en etapas tempranas del diseño arquitectónico e incluso, sobre partes del sistema y no en el sistema completo.

3.3.1 Evaluación mediante el método ARID

La evaluación mediante ARID se basa en detallar el funcionamiento del sistema en una serie de escenarios predefinidos, a través de lo cual se pueden identificar posibles puntos problemáticos de la arquitectura. Su utilidad se encuentra en que prepara el camino en sistemas en los que, por su complejidad, puedan ser necesarias revisiones de la arquitectura en etapas intermedias del diseño.

Los atributos de calidad seleccionados para la evaluación son los siguientes: Funcionalidad, Eficiencia, Portabilidad y Mantenebilidad.

3.3.1.1 Creación del árbol de utilidad

La Figura 18 muestra el árbol de utilidad con los atributos y los escenarios seleccionados para evaluar la arquitectura. El nodo raíz del árbol contiene la utilidad general del sistema. En el segundo y tercer nivel se corresponde a los atributos de calidad más importantes para la aplicación refinados hasta el establecimiento de los escenarios que corresponden a las hojas del árbol, los cuales deben ser lo suficientemente concretos para ser analizados y otorgarle una prioridad (Alta, Media o Baja), esta prioridad se refiere a la importancia que tiene el escenario para el sistema.

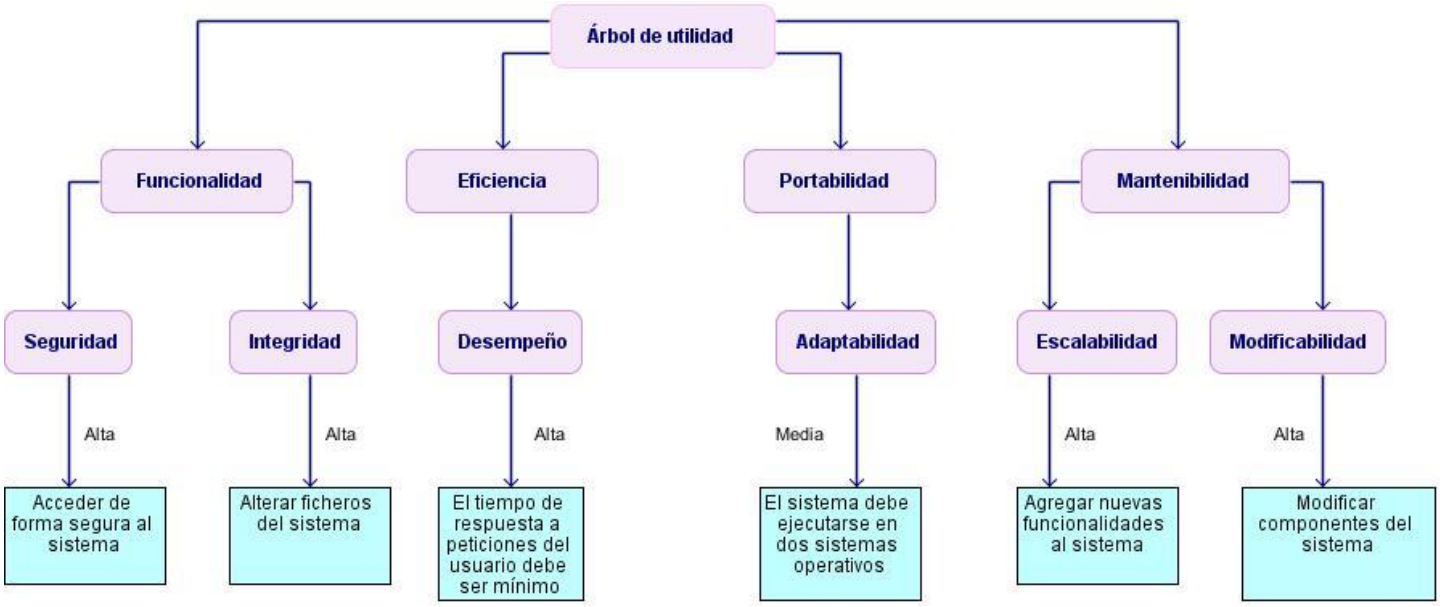

Figura 18. Árbol de utilidad

3.3.1.2 Descripción de los escenarios

A continuación se muestran las descripciones de los escenarios definidos en el árbol de utilidad, donde se muestra la relación de estos con los atributos de calidad seleccionados.

Tabla 20. Escenario # 1

Escenario # 1: Acceder de forma segura al sistema	
Atributo (s)	Funcionalidad (Seguridad)
Estímulo	Un usuario no registrado intenta autentificarse para acceder a las secciones del sistema que corresponden a usuarios previamente registrados.
Contexto	El sistema no permite el acceso y muestra el mensaje “Datos no válidos”.
Respuesta	Para acceder a las secciones del sistema definidas para los usuarios: estudiante, profesor o administrador, estos deben suministrarles al sistema su usuario único, y su contraseña. La clase **GUI_Autenticar** de la capa de presentación se encarga de capturar los datos y validarlos para enviárselo a la clase **Db_Access** ubicada en la capa lógica. Esta clase posee las sentencias SQL necesarias para acceder a la base de datos haciendo uso de las funcionalidades de la biblioteca **SQLite 3**, la cual, al igual que el

	fichero de la base de datos se encuentran ubicadas en la capa de soporte.

Tabla 21. Escenario # 2

Escenario # 2: Alterar ficheros del sistema	
Atributo (s)	Funcionalidad (Integridad)
Estímulo	Un usuario modifica o elimina algún fichero de la aplicación.
Contexto	El sistema debería mostrar un mensaje notificando un error por la ausencia o modificación de un fichero.
Respuesta	La arquitectura no tiene un mecanismo para asegurar la integridad de los ficheros usados por la aplicación. Un usuario con o sin los conocimientos adecuados puede modificar o eliminar estos ficheros y como consecuencia afectar el funcionamiento de la aplicación.

Tabla 22. Escenario # 3

Escenario # 3: El tiempo de respuesta del sistema a peticiones del usuario debe ser mínimo	
Atributo (s)	Eficiencia (Desempeño)
Estímulo	El usuario selecciona un objeto de la escena para manipularlo.
Contexto	El sistema responde de forma rápida a la petición hecha por el usuario.
Respuesta	La arquitectura garantiza que las clases sean lo más cohesivas posible para que tengan la menor cantidad de dependencias entre ellas y así de esta forma aumentar la rapidez con que este responde a peticiones del usuario. El otro factor que influye en la rapidez de respuesta depende del hardware donde esté desplegada la aplicación.

Tabla 23. Escenario # 4

Escenario # 4: El sistema debe ejecutarse en dos sistemas operativos	
Atributo (s)	Portabilidad (Adaptabilidad)
Estímulo	El usuario ejecuta la aplicación en una distribución de Linux o en alguna versión de Windows.
Contexto	La aplicación funciona correctamente.
Respuesta	La base de implementación de la arquitectura es el motor de juegos Unity 3D, el cual exporta para múltiples plataformas, incluyendo para distribuciones de Linux, por lo que la aplicación usa la misma arquitectura en cualquiera de los sistemas operativos donde se quiera ejecutar.

Tabla 24. Escenario # 5

Escenario # 5: Agregar nuevas funcionalidades al sistema	
Atributo (s)	Mantenibilidad (Escalabilidad)
Estímulo	Agregar un nuevo componente de software.
Contexto	La aplicación funciona correctamente.
Respuesta	La arquitectura está basada en capas lo que permite que, si se desea agregar un nuevo

	componente se seleccione la capa donde se quiere agregar y se realice la integración al sistema sin que se afecten los componentes de las otras capas.

Tabla 25. Escenario # 6

Escenario # 6: Modificar componentes del sistema	
Atributo (s)	Mantenibilidad (Modificabilidad)
Estímulo	Modificar un componente de software.
Contexto	El sistema se debe mantenerse funcional una vez que se modifique un componente.
Respuesta	La arquitectura está basada en capas y componentes, los componentes constituyen unidades de composición independientes que admiten ser modificados sin afectar el sistema.

3.3.2 Evaluación mediante técnica de prototipo

Para evaluar la arquitectura propuesta además de hacer uso del método ARID, tal como se hizo en el epígrafe anterior, se confeccionó un prototipo funcional donde se implementaron los principales CUAS, con el objetivo de ver de forma directa que tanto se ve afectado o no un atributo de calidad y proveer al equipo de desarrollo de una maqueta base para desarrollar la aplicación con la totalidad de sus funcionalidades. A continuación se muestran varias imágenes correspondientes a las principales interfaces del prototipo.

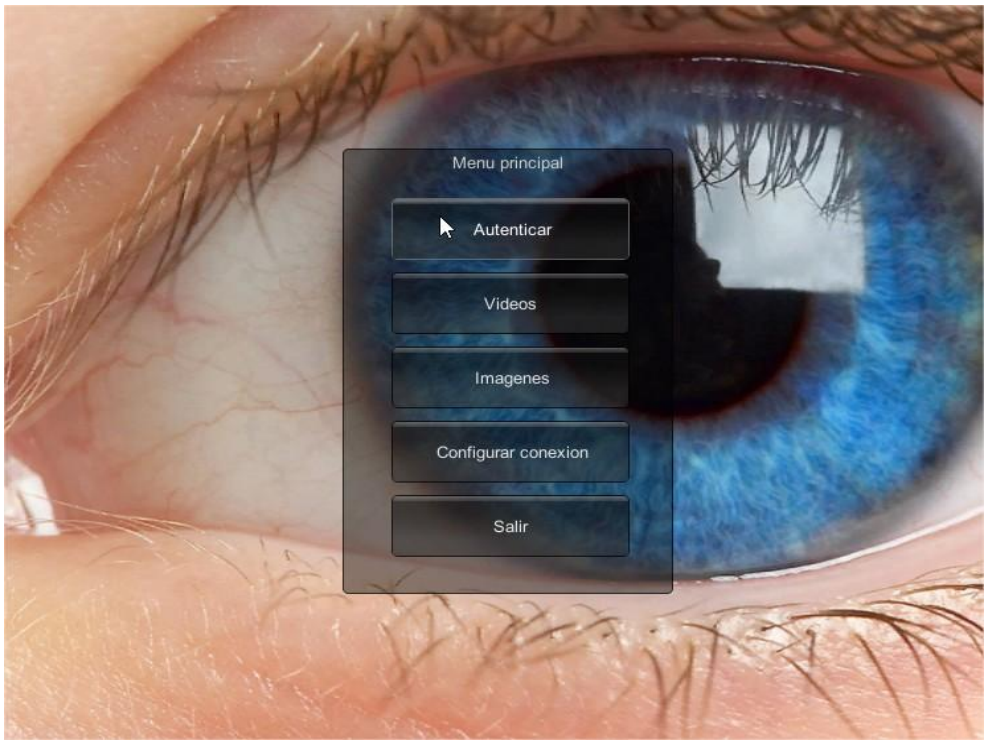

Figura 19. Prototipo funcional (Menú inicial)

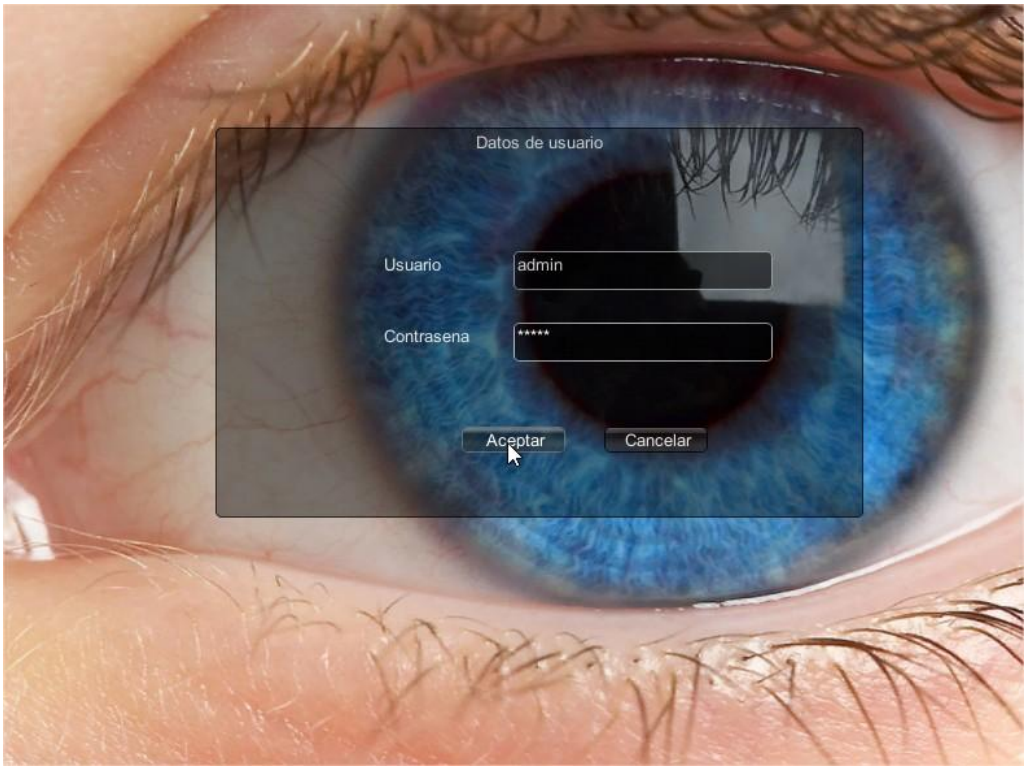

Figura 20. Prototipo funcional (Panel de autentificación)

Figura 21. Prototipo funcional (Panel para gestionar usuarios)

3.4 Resultados de la evaluación

Con la evaluación de la arquitectura propuesta mediante el método ARID se identificó que esta posee un riesgo en el escenario # 2. Este, en caso de ocurrir, puede producir afectaciones en el sistema, por los que se propone el tratamiento de este riesgo en una segunda iteración de la arquitectura. Los demás escenarios analizados arrojaron resultados positivos para el diseño arquitectónico, por lo que estos satisfacen los atributos de calidad correspondientes. La evaluación basada en la técnica de prototipo permitió observar de forma práctica los atributos de calidad observables vía ejecución de interés para los usuarios y desarrolladores tales como seguridad y desempeño.

Conclusiones generales

El estudio realizado en la presente investigación permitió desarrollar una arquitectura para guiar el proceso de desarrollo de la aplicación ProteSIM del proyecto UCI-CIMEQ. De esta forma se le dió cumplimiento al objetivo propuesto al inicio de la investigación, además se arribaron a las siguientes conclusiones:

- La arquitectura propuesta constituye una opción factible a utilizar para desarrollar el ambiente virtual de aprendizaje ProteSIM y puede servir como base para futuras aplicaciones similares que se realicen dentro del proyecto.
- El prototipo funcional confeccionado basado en la arquitectura propuesta como infraestructura básica puede servir como punto de partida para desarrollar ProteSIM a partir de él.
- Mediante el método ARID fue detectado un riesgo presente en la arquitectura, el cual puede ser mitigado en posteriores iteraciones de la misma.

Recomendaciones

A partir del presente trabajo se recomienda:

- Perfeccionar la arquitectura de software propuesta mediante el ciclo de desarrollo de la metodología RUP y mitigar el riesgo detectado.
- Valorar la factibilidad de la arquitectura propuesta para emplearla en el proceso de desarrollo de ambientes virtuales de aprendizaje con características similares a ProteSIM.

Bibliografía

1. **Avila, Patricia.** *Ambientes Virtuales de Aprendizaje, una nueva experiencia.* 1999.

2. **Marchena, Daniel.** *Entornos Virtuales de Aprendizaje.* Venezuela : s.n., 2008.

3. **Viciedo Caraballoso, Luis Gabriel y Gonzáles Campiztrus, Jaime.** Estrategias para la implementación de Laboratorios Virtuales con fines de aprendizaje con el manejo del lenguaje de programación script Lua. La Habana : s.n., 2010.

4. **Salinas, Silvia Alicia.** *Software para el Trabajo Colaborativo y Bibliotecas.* Monte Grande : s.n., 2008.

5. **Glinz Férez, Patricia Elízabeth.** *Un acercamiento al trabajo colaborativo.* ISSN: 1681-5653.

6. *Aprendizaje colaborativo en entornos virtuales: un modelo de diseño instruccional para la formación profesional continua.* **Touriñán López, José Manuel, y otros.** 2009.

7. **Vargas Torres, Osvel y Lázaro Suárez, Jesús .** *Educación a distancia en la Universidad de las Ciencias a través de escenarios virtuales 3D.* Universidad de las Ciencias Informáticas. La Habana : s.n., 2013. ISSN: 2306-2495.

8. **Perry, Dewayne y Wolf, Alexander.** *Foundations for the study of software architecture.* 1992. págs. 40-52. Vol. 17.

9. **Society, IEEE Computer.** *1471-2000 - Recommended Practice for Architectural Description for Software-Intensive Systems.* 2000.

10. **Rumbaugh, James, Jacobson, Ivar y Booch, Grady.** *El Proceso Unificado de Desarrollo de Software.* Madrid : Addison Wesley, 2000. ISBN: 84-7829-036-2.

11. **Garlan, David y Shaw, Mary.** *An Introduction to Software Architecture.* Carnegie Mellon University. Pittsburgh : s.n., 1994.

12. —. *Introduction to Software Architectures. New perspectives.* Prentice Hall. 1996.

13. **Shaw, Mary y Clements, Paul.** *A field guide to Boxology: Preliminary classification of architectural styles for software systems.* Carnegie Mellon University. 1997.

14. **Christopher, Alexander, y otros.** *A Pattern Language.* Oxford University Press. New York : s.n., 1977.

15. **Buschmann, Frank, y otros.** *Pattern-oriented software architecture – A system of patterns. John Wiley & Sons.* 1996.

16. **Reynoso, Carlos y Kiccillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Universidad de Buenos Aires. 2004.

17. **Shaw, Mary y Clements, Paul.** *A field guide to Boxology: Preliminary classification of architectural styles for software systems.* Computer Science Department and Software Engineering Institute, Carnegie Mellon University. 1996.

18. **Szyperski, Clemens Alden.** *Component software: Beyond Object-Oriented programming.* Segunda. s.l. : Addison-Wesley, 2002. ISBN: 0201745720.

19. **de la Torre, Cesar, Zorrilla, Unai y Ramos Javier Calvarro, Miguel Ángel.** *Guia de arquitectura N-capas orientada a dominio con .NET 4.0.* 2010.

20. **Gamma, Erich, y otros.** *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Addison Wesley, 1995. ISBN 0-201-63361-2.

21. **Larman, Craig.** *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso.* 2003.

22. **Méndez Pozo, Gonzalo.** *Una Arquitectura Software Basada en Agentes y Recomendaciones Metodológicas para el Desarrollo de Entornos Virtuales de Entrenamiento con Tutoría Inteligente.* UNIVERSIDAD POLITÉCNICA DE MADRID. Madrid : s.n., 2008. págs. 33-35 y 88-92.

23. **Merzeau Martínez, Alejandro David.** *Arquitectura de software para los Laboratorios Virtuales.* Universidad de Ciencias Informáticas. La Habana : s.n., 2012.

24. **Sanchez Mendoza, María.** *Metodologías de Desarrollo de Software.* 2004.

25. **Figueroa, Roberth G., Solís, Camilo J y Cabrera, Armando A.** *Metodologías Tradicionales vs Metodologías Agiles.* Universidad Técnica Particular de Loja. 2008.

26. **Kruchten, Philippe.** *The "4+1" View Model of Software Architecture.* 1994.

27. **Rumbaugh, James, Jacobson, Ivar y Booch, Grady .** *The Unified Modeling Language Reference Manual.* s.l. : ADDISON WESLEY, 1998. ISBN 0-201-30998-X.

28. **Sommerville, Ian.** *Ingeniería de Software.* Séptima. Madrid : s.n., 2005. págs. 109-110. ISBN 84-7829-074-5.

29. **Kruchten, Philippe.** *Architectural Blueprints. The "4+1" View.* Rational Software Corp. 1995.

30. **Kazman, R, Clements, P y Klein, M.** *Evaluating Software Architectures. Methods and case studies.* s.l. : Addison Wesley, 2001.

31. **Bass, L, Kazman, R y Clements, P.** *Software Architecture in practice.* s.l. : Addison-Wesley, 1998.

32. **Bosh, J.** *Design & Use of Software Architectures.* s.l. : Addison-Wesley, 2000.

33. **Camacho, Erika, Cardesco, Fabio y Núñez, Gabriel.** *Arquitecturas de software. Guía de estudio.* 2004.