Universidad de las Ciencias Informáticas

Facultad 4

"Arquitectura de software para la construcción de laboratorios virtuales usando HTML5"

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

Autores: Arelis Gómez Ramírez

Héctor Luis Rodríguez Pérez

Tutores: Ing. Gilberto Cao Tarrero

Ing. Arisbel Hechavarría Rojas

La Habana

Junio, 2012

Declaración de Autoría

Declaramos que somos los únicos autores del presente trabajo “Arquitectura de software para la construcción de laboratorios virtuales usando HTML5” y autorizamos a la Facultad (4) y a la Universidad de las Ciencias Informáticas (UCI) para que hagan el uso que estimen pertinente con este trabajo.

Para que así conste firmamos la presente a los ____ días del mes de _____del 2012.

_____ _____

Arelis Gómez Ramírez
Firma del Autor

Héctor Luis Rodríguez Pérez
Firma del Autor

_____ _____

Ing. Gilberto Cao Tarrero
Firma del Tutor

Ing. Arisbel Hechavarría Rojas
Firma del Tutor

Agradecimientos

A mis padres Graciela y Segundo por estar siempre ahí para mí, por apoyarme en cada momento, por sus esfuerzos para que nunca me faltase nada, por su comprensión y amor.

A mi tía Martha por el apoyo y amor que siempre me ha dado como mi segunda madre.

A mis hermanos Tito, Kendy y Juanca que siempre estuvieron pendientes de mis estudios en esta universidad y por su preocupación de que siguiera adelante y nunca abandonara aunque las cosas no salieran bien.

A mi novio, amigo y compañero te tesis que ha sido mi apoyo durante 3 años, apoyándome en todo y ayudándome a salir de los momentos difíciles que hemos tenido.

A mi familia inmensa y vecinos por siempre apoyarme y creer en mí, al igual que mis suegros que me han apreciado como hija y brindado ese cariño incondicional que siempre agradeceré.

A mi tutor Arisbel, por estar siempre pendiente de nuestro trabajo y brindarnos su apoyo de amigo.

A mi tutor Cao y a los profesores Osmany, Licet, Ismael, Yaismel, Yusimy y María que siempre han dado un poco de su tiempo para ayudar con la tesis.

A mis grandes amigas Yanela, Nidi, Niuris, Ana, Zaimar, Lien, María, Kare y Aniuska por siempre apoyarme y darme lo mejor de ellas, su amistad.

A mis compañeros (as) y amigas(os) de la universidad Liuba, Yanelis, Maday, Yailín, Yazmany, Pablo, Ramiro, Leczy, Lisi, Mayara, Henry, Mayelín y a todos en general que han compartido conmigo momentos felices, tristes y de tensión durante estos 5 años de mi carrera.

A todos los que de una forma y otra hicieron posible la realización de este trabajo muchas gracias.

Arelis.

Agradecimientos

A la UCI, a Fidel, Raúl, a esta maravillosa revolución por darte la oportunidad de estudiar y formarme como hombre justo y poder ser útil a la sociedad.

A mis padres, que con la elaboración de este trabajo no solo se cumple un sueño, sino un premio grande al esfuerzo que han dedicado a mi formación como persona y futuro profesional. A ellos les debo lo que soy, gracias por ser ejemplos, fuente de inspiración y orgullo en cada decisión que he tomado en mi vida. Gracias por todo.

A mi novia por ser compañera y amiga en momentos de tristeza y alegrías, por todo su amor, apoyo, comprensión y ser parte muy importante en el logro de este título.

A mi hermana Meylis, a mis tíos María, Francisco, Ramona, Pilar, Martha, Luis Modesto, Elena, Martí, Pacholo que de una forma u otra han puesto su granito de arena en este triunfo.

A mis suegros Graciela y Segundo por su calurosa acogida y tratarme como un hijo.

A Plinio por su amistad y su ayuda incondicional en mi formación.

A mis abuelos fallecidos Aracelis y Antonio que en paz descansen.

A los profesores del Pre en especial a Lisset, Yanet, Torres, Roberto.

A los Profesores Thaimy, Ismael, Licet, Osmany, Jacobo, Liana, Yaismel, Isyed y Leonardo.

A mis tutores Arisbel y Cao que sobre todas las cosas he aprendido algo con ellos, que tienes que luchar por lo que quieres aún sin saber por dónde empezar.

A todos mis compañeros durante estos cinco años de estudio y sacrificio, mencionarlos sería difícil por la posibilidad de que olvide algún nombre. A todos mis más sinceros agradecimientos.

A mis compañeros (as) y amigas(os) Cala, Lionel, Adriel, Madiel, Lismaidy, Daimarelis, Erick, Luisi, Nielvis, el Rojas, el Niño, Yumara, Jesús, Pablo, Ana Ilis, Eledio, Yoandry Cabrera, Laura, Ramón, Ariosmi, Yadeimy, Maday, Pablo E. Nory y Yasmany.

A los vecinos que se han preocupado por mis estudios Rafaela, Najarro, Eliseo, Osmany, Iván y su mujer, Eledio, Xiomara, Lucrecia, Rosa.

A todos lo que de una forma u otra han hecho posible este logro en mi vida, muchas gracias.

Héctor Luis.

Dedicatoria

El logro de mis esfuerzos va dedicado a mis padres Segundo y Graciela, por apoyarme no solo durante mi carrera sino durante todos estos años de mi vida, a los cuales les agradezco eternamente. Gracias mami y papi, los amo.

Arelis.

A mis padres Héctor Antonio y Cecilia María por estar ahí siempre a mi lado apoyándome , dándome ánimos , inspiración y ejemplo para seguir adelante no solo en la carrera sino en la vida, a ellos les debo todo y les voy a estar agradecido eternamente. Gracias Pipo, gracias Mami, son mi tesoro más preciado, los quiero y no encuentro palabras para describir lo que siento.

Héctor Luis.

Resumen

El avance de las Tecnologías de la Información y las Comunicaciones (TIC) ha permitido el incremento en el desarrollo del software educativo en el mejoramiento del proceso enseñanza-aprendizaje en todo el mundo. En Cuba se evidencia sobre ello con la colección de software educativo “El Navegante”, dirigida a estudiantes de la enseñanza secundaria con el objetivo de fomentar el aprendizaje de los mismos.

Entre los productos que integran la colección se encuentra “La Naturaleza y El Hombre” compuesto por varios módulos, entre ellos el módulo Mediateca que consta de diferentes componentes como el Laboratorio virtual, sobre el cual se basa la presente investigación definiendo la Arquitectura de software para el desarrollo de un nuevo laboratorio virtual haciendo uso de HTML5. Para ello se realizó un amplio estudio acerca de los principales conceptos, estilos y patrones arquitectónicos relacionados con la Arquitectura de software, así como las metodologías, lenguajes de programación y herramientas que se utilizará para el desarrollo del componente.

Para un mejor entendimiento se definieron los requerimientos del componente más significativos conjuntamente con la representación arquitectónica a través del modelo 4+1 vistas que propone la metodología RUP. Para finalizar se realiza el análisis sobre atributos de calidad, técnicas y métodos para la evaluación de la arquitectura, de ellos se utilizará la técnica basada en escenarios con el instrumento Árbol de Utilidades y el método de evaluación ATAM, con el objetivo de identificar riesgos y fortalezas además de comprobar si el componente cumple con los atributos de calidad definidos. Determinando así la factibilidad de la arquitectura para el desarrollo del componente.

Palabras Claves: Arquitectura de software, estilos, patrones, HTML5, ATAM.

Índice

Índice de tablas

Índice de figuras

Introducción

Las Tecnologías de la información y las comunicaciones (TIC) han propiciado gran desarrollo en la sociedad, facilitando para el hombre avances en diferentes áreas como salud, educación, comercio, publicidad, propaganda de instituciones, empresas y otras. En la educación el uso de las TIC ha elevado la calidad en el proceso docente educativo permitiendo una mejor comunicación entre profesores y estudiantes, despertando en estos últimos la motivación y el interés por la apropiación de los conocimientos a partir de los errores cometidos.

Con el avance de las TIC se ha incrementado en todo el mundo el uso del software educativo en el mejoramiento del proceso de enseñanza-aprendizaje. Cuba ha incorporado este método en los diferentes niveles de la educación, ejemplo de ello es la colección "El Navegante" software que está dirigido a estudiantes de la enseñanza secundaria. Inicialmente la colección fue desarrollada por el MINED; para su implementación se empleó la herramienta *ToolBook Instructor* 2004 que se encuentra bajo licencia propietaria. Debido a problemas que presentaba la colección inicial entre ellos su funcionamiento solamente en el sistema operativo Windows, el proyecto "Colecciones de Software Educativo Multisaber y El Navegante" perteneciente al Centro de Tecnologías para la Formación (FORTES) de la Universidad de las Ciencias Informáticas (UCI), se dio a la tarea de desarrollar una nueva versión multiplataforma haciendo uso de herramientas libres que se encuentra actualmente en desarrollo.

La colección El Navegante constituye un hiperentorno de enseñanza-aprendizaje, la cual tiene integrado 10 productos tales como: Encuentro con el pasado, Geoclío, Por los senderos de mi Patria, Elementos Matemáticos, Informática Básica, Aprende Construyendo, El fabuloso mundo de las palabras, La Naturaleza y el Hombre, Educarte y Rainbow, que abordan temas de los contenidos de las asignaturas impartidas en la enseñanza secundaria.

La Naturaleza y El Hombre es uno de los 10 productos que integran la colección. Se utiliza en apoyo a las asignaturas de Biología, Física, Química y Geografía. Está constituida por diferentes módulos entre ellos la Mediateca. La misma incluye los componentes: Videos, Animaciones, Imágenes, Diaporamas, Sonidos, Glosario, Personalidades y Laboratorios virtuales, este último desarrollado en el lenguaje ActionScript 3.0 y hace uso de la máquina virtual Flash Player para su visualización.

Debido a una posible comercialización de la colección fue realizado un estudio previo sobre la misma y productos de este fueron encontradas varias deficiencias en el componente Laboratorios virtuales.

La máquina virtual Flash Player no es de código abierto, el componente no cuenta con un diseño arquitectónico lo que implica que existan limitaciones en cuanto a: escalabilidad, resultando muy engorroso la modificación o una nueva versión que no conlleve a retrasos en el tiempo de desarrollo del componente sin tener previamente definida una arquitectura que guie el proceso de desarrollo; rendimiento, requiere de dos aplicaciones para la visualización: la máquina virtual y el navegador; reutilización, el código implementado no puede ser reutilizado debido a que se encuentra en el lenguaje ActionScript 3.0 lo cual no responde al uso de los lenguajes definidos en el proyecto, solo pueden ser reutilizadas las estructuras de las clases; presenta además carencia de estilos arquitectónicos, por tanto no cuenta con una organización arquitectónica que satisfaga algunos atributos de calidad.

La necesidad de corregir las dificultades encontradas originó un análisis entre los jefes de módulos del proyecto sobre nuevas tecnologías llegando a la conclusión de hacer uso del lenguaje *Hyper Text Markup Language* en su versión 5.0 (HTML5) por las ventajas y características que posee. Con la utilización de esta tecnología queda solucionada la deficiencia en cuanto a la restricción de comercialización, sin embargo no es solución para las restantes deficiencias encontradas. De lo anteriormente expuesto se originó el siguiente **problema a resolver:** ¿Cómo lograr que el componente Laboratorios virtuales haciendo uso de HTML5 tenga una estructura en el software que sea escalable, reutilizable y de alto rendimiento?

Teniendo en cuenta lo anteriormente expuesto se plantea como **objeto de estudio** La Arquitectura de software.

Como **objetivo general** describir la Arquitectura de software del componente Laboratorios virtuales haciendo uso de HTML5 que permita ser escalable, reutilizable y de alto rendimiento.

El **campo de acción** está enmarcado en la Arquitectura de software del componente Laboratorios virtuales haciendo uso de HTML5.

Como **idea a defender** se plantea que si se crea una Arquitectura de software para el desarrollo del componente Laboratorios virtuales haciendo uso de HTML5, se logrará la escalabilidad, reusabilidad y el aumento del rendimiento del componente.

Como **objetivos específicos** se definen los siguientes:

- Definir la arquitectura para el desarrollo de los Laboratorios virtuales usando HTML5.

- Validar la propuesta de arquitectura para la construcción de Laboratorios virtuales usando HTML5.

Para dar cumplimiento a los objetivos planteados se definen las siguientes **tareas de investigación:**

- Análisis del estado del arte relacionado con el desarrollo de aplicaciones con escenarios virtuales tridimensionales interactivos en HTML5.
- Análisis de los diferentes estilos y patrones arquitectónicos existentes que puedan usarse en la arquitectura.
- Selección de la metodología de desarrollo de software a emplear con la arquitectura.
- Selección de herramientas a utilizar para el desarrollo de los Laboratorios virtuales usando HTML5.
- Selección de framework a utilizar para el desarrollo de los Laboratorios virtuales usando HTML5.
- Caracterización de la Arquitectura de software.
- Implementación de los métodos y clases necesarias para validar la propuesta de arquitectura.
- Evaluación de la arquitectura usando técnicas y métodos de evaluación.

Para el desarrollo de la presente investigación se hará uso de métodos teóricos, los cuales permiten el conocimiento de las características del objeto de investigación que no son observables directamente y ayudan en el conocimiento del estado del arte, también serán utilizados métodos empíricos.

Métodos Teóricos:

Método Análisis Histórico- Lógico: Permitirá realizar un estudio de diferentes conceptos como Arquitectura de software, metodologías de desarrollo y estilos arquitectónicos, además de analizar información referente a la evolución de los laboratorios virtuales a nivel mundial.

Método Analítico-Sintético: Este método permitirá una profundización y selección de las herramientas, metodologías y conceptos planteados, posibilitando extraer elementos esenciales obtenidos luego de la investigación.

Método Modelación: Este método ayudará a la modelación de los diferentes diagramas que se especifican en las vistas arquitectónicas. Donde se describe el modelo de diseño del componente a desarrollar.

Método Empírico:

Método Observación: Este método será utilizado para el estudio de la nueva tecnología HTML5 y del alcance de la misma para el empleo en el desarrollo del componente Laboratorios virtuales del módulo Mediateca.

El presente trabajo está estructurado por 3 capítulos en los cuales se expone el contenido de la investigación.

Capítulo 1. Fundamentación teórica: En este capítulo se realiza un estudio del estado del arte sobre soluciones similares y tendencias actuales. Se describen conceptos y características sobre la Arquitectura de software. Selección de metodología de desarrollo, estilos y patrones arquitectónicos**,** herramientas, framework y lenguajes de programación que se emplearán en el desarrollo.

Capítulo 2. Descripción de la arquitectura: En este capítulo se realiza una especificación de la propuesta de arquitectura, se describen los requerimientos funcionales y no funcionales del sistema así como las características a través de las vistas arquitectónicas.

Capítulo 3. Evaluación de la arquitectura: En este capítulo se describe la importancia de la evaluación, se seleccionan los atributos de calidad más significativos, las técnicas y métodos de evaluación más adecuados. Se muestran los resultados obtenidos luego de validar la arquitectura propuesta mediante la técnica o método seleccionado.

Capítulo 1. Fundamentación Teórica

1.1 Introducción

En el presente capítulo se efectúa un análisis del estado del arte sobre nuevas tecnologías para el empleo en el desarrollo del componente Laboratorios virtuales, conceptos de Arquitectura de software y soluciones similares. Estudio y selección de herramientas, metodología, patrones de diseño, estilos y lenguajes de programación necesarios para el cumplimiento de las tareas trazadas.

1.2 Soluciones similares existentes

Durante el período de investigación se realizó la búsqueda de soluciones similares existentes, con el objetivo de encontrar apoyo para el desarrollo del trabajo, no fue encontrada ninguna referencia que hiciera alusión a una propuesta arquitectónica para la construcción de Laboratorio virtuales en HTML5, no obstante fueron encontrados trabajos de propuestas arquitectónicas para escenarios tridimensionales.

Primer documento: “Propuesta de Especificación y Arquitectura para aplicaciones de Interacción Multimodal en Escenarios 3D”, en él son expuestas las bases de una arquitectura para integrar componentes de aplicaciones que permitan la interacción multimodal (proceso en el cual dispositivos y personas llevan una interacción auditiva, visual, táctil y gestual conjunta desde cualquier sitio) en escenarios 3D reutilizando lenguajes de marca previamente definidos.

Segundo documento: “Propuesta de Arquitectura para la Generación Interactiva de Entornos Colaborativos de Realidad Virtual para el Campus Virtual del G9”, son expuestas las bases para la creación de un entorno interactivo en escenarios 3D, donde facilite el trabajo de diseño y gestión. Proponen el empleo del VRML (*Virtual Reality Modeling Language*), es un lenguaje de modelado de mundos virtuales en tres dimensiones que al igual que el HTML5 es usado para la creación de elementos tridimensionales en la web, sin embargo, requiere el uso de *plugins* para la interacción con los navegadores.

De los trabajos analizados se pudo observar el uso de las herramientas para el trabajo en espacios tridimensionales y el diseño presentado por cada documento independientemente del objetivo de cada uno de ellos.

1.3 Laboratorios virtuales

Los laboratorios virtuales simulan el ambiente de los laboratorios reales, tanto de Química, Física, Biología o cualquier otra especialidad. A través de ejercicios, explicaciones con ilustraciones, textos, guías de estudio, herramientas, objetos virtuales y otros componentes de forma digital de acuerdo al tema para el cual fue creado, muestran aquellos eventos que suceden en un laboratorio real, brindan la posibilidad de que el alumno interactúe con fenómenos y conceptos abstractos. En la mayoría de los casos están incluidos en algún software o integrados a internet. Facilitan en el estudiante la posibilidad de repetir el ejercicio hasta vencer el objetivo. Se apoyan en el uso de nuevas tecnologías como las computadoras e internet por lo que se diferencian de los tradicionales.

Algunas de las ventajas que presentan son:

- Es una herramienta de auto-aprendizaje, donde el alumno altera las variables de entrada, configura nuevos experimentos, aprende el manejo de instrumentos, personaliza el experimento, etc. La simulación en el laboratorio virtual, permite obtener una visión más intuitiva de aquellos fenómenos que en su realización manual no aportan suficiente claridad gráfica.
- Los estudiantes aprenden mediante prueba y error, sin miedo a sufrir o provocar un accidente, sin avergonzarse de realizar varias veces la misma práctica, pues pueden repetirlas sin límite, sin temor a dañar alguna herramienta o equipo.
- Reducen el coste del montaje y mantenimiento de los laboratorios tradicionales, siendo una alternativa barata y eficiente, donde el estudiante simula los fenómenos a estudiar como si los observase en el laboratorio tradicional (4).

Laboratorios de software: Son aquellos laboratorios desarrollados en forma de un programa de software independiente donde no requieren de servidores Web.

Laboratorios remotos: Son sistemas que se basan en la instrumentación de un laboratorio real donde permiten realizar actividades de forma local o remota.

Laboratorios para la Web: Se basan en un software donde dependen de los recursos de un servidor determinado.

El componente Laboratorios virtuales de la colección El Navegante se define dentro de la clasificación de laboratorios para la Web.

1.4 Tendencias y tecnologías actuales

HTML5 es la nueva especificación del lenguaje HTML, versión que cada vez gana más espacio en el desarrollo de sitios web. Actualmente muchos de los nuevos sitios que se publican son desarrollados con este lenguaje, debido a las ventajas y características que presenta. Más adelante en el documento se detallará sobre el mismo.

Aplicaciones que han sido desarrolladas con este lenguaje son por ejemplo: Cartoon Builder, Bespin, Sketchpad, CanvasPaint y Vimeo. Este último no es más que un portal de video que tiene incorporado un reproductor basado en HTML5 que permite integrar videos sin necesidad de tener instalado el *Adobe Flash Player* (5).

De esta manera también son creados juegos en 3D que hacen uso de este lenguaje, ejemplos de ellos son mostrados a continuación:

Tetris: Es un juego clásico y muy conocido, ahora sus nuevas versiones son desarrolladas en HTML5, utilizando capas para el uso de Canvas (nueva etiqueta que integra HTML5) y JavaScript (6).

Canvas Rider: Este juego aprovecha al máximo el nuevo elemento Canvas de HTML5, consiste en conducir una bicicleta a través de increíbles escenarios dibujados por los propios jugadores. En esencia es muy simple, pero entretenido y muestra el potencial de combinar HTML5 y JavaScript (7).

Grandes empresas como Google, Microsoft, Apple, Adobe, Facebook y otras muy reconocidas, emplean el HTML5 y sustituyen el uso de Flash por este. Apple por ejemplo sustituye el Flash integrado en los iPad e iPhone y al igual que otras empresas promueven el estudio y empleo de esta nueva tecnología.

WebGL es otra de las nuevas tecnologías que se emplea para el desarrollo tanto de sitios web como animaciones y juegos en 3D. Utiliza el elemento Canvas de HTML5 con el cual se logra renderizar gráficos 3D en el mismo navegador sin el uso de *plugins*. Más adelante en el documento se abordará con más profundidad sobre WebGL. Algunos ejemplos de aplicaciones y demos que han sido desarrollados con el lenguaje son:

ZygoteBody: Aplicación de exploración del cuerpo humano (8).

WebGL Bookcase: Estantería animada con decenas de libros, tomados desde la enorme biblioteca de Google Books. Al pulsar sobre alguno será mostrada una vista previa del libro con el enlace a GBooks.

1.5 Arquitectura de Software

El término “Arquitectura de Software” (en adelante AS) fue conocido por primera vez en la década de los años 60 en estudios realizados por Dewayne Perry y Alexander Wolf. Su definición es bastante amplia gracias a diferentes autores como Dijkstra y Pressman que aportan a la definición de la misma. A continuación definiciones expuestas por varios autores.

Hacia 1968 Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda propuso que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera (3).

Para Kazman (1996), la Arquitectura de software es una forma de representar sistemas complejos mediante el uso de la abstracción (2).

Pressman define como AS lo siguiente:

La arquitectura no es el software operacional. Más bien, es la representación que capacita al ingeniero del software para: analizar la efectividad del diseño para la consecución de los requisitos fijados, considerar las alternativas arquitectónicas en una etapa en la cual hacer cambios en el diseño es relativamente fácil y reducir los riesgos asociados a la construcción del software (1).

El documento de la IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) 1471-2000 define que: La AS es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución (3).

Al realizar un análisis de los conceptos abordados anteriormente, se determina que la Arquitectura de software es la base para la creación de cualquier software, a grandes rasgos define una estructura consistente en componentes y relaciones entre ellos en el sistema, establece una guía a seguir para cualquier equipo de trabajo, brinda una mejor comunicación entre ellos, permite la reutilización y organiza el trabajo facilitando el ahorro de tiempo, además de evitar en gran medida el costo de los riesgos, es una forma de diseño que comienza en los inicios del desarrollo de un sistema.

La presente investigación asume la Arquitectura de software de acuerdo a la definición expuesta por la IEEE 1471-2000 que define hasta ahora el concepto más completo sobre la misma siendo una de las fuentes más confiables en el mundo.

1.5.1 Importancia de la Arquitectura de Software

La AS juega un papel fundamental dentro del proceso de desarrollo de cualquier software. La rápida evolución y amplitud que toman los sistemas de software generan mucho trabajo, por lo que es aquí donde resulta fundamental. Define tanto elementos como componentes, que ayudan a la organización del trabajo y así a la definición de posibles riesgos que puedan ocurrir durante el ciclo de vida de un proyecto, generando además la forma de mitigarlos o reducir los daños que pueda ocasionar alguno de ellos. Su objetivo principal es lograr que el sistema o aplicación que se realice, sea lo más óptimo posible. Pressman en su libro Ingeniería del Software cita tres razones que identificaron Bass y sus colegas sobre la importancia de la AS.

- Las representaciones de la Arquitectura de software facilitan la comunicación entre todas las partes (partícipes) interesadas en el desarrollo de un sistema basado en computadora.
- La arquitectura destaca decisiones tempranas de diseño que tendrán un profundo impacto en todo el trabajo de ingeniería del software que sigue y es tan importante en el éxito final del sistema como una entidad operacional.
- La arquitectura constituye un modelo relativamente pequeño e intelectualmente comprensible de cómo está estructurado el sistema y de cómo trabajan juntos sus componentes (1).

1.6 Definiciones de Estilo y Patrón Arquitectónico

Shaw y Garlan utilizan indistintamente los términos estilo arquitectónico y patrón arquitectónico. Por otro lado, Buschmann establece diferencias sutiles entre ambos conceptos. De cualquier forma, los estilos y los patrones establecen un vocabulario común y brindan soporte a los ingenieros para conseguir una solución que haya sido aplicada con éxito anteriormente, ante ciertas situaciones de diseño. Además, su aplicación en el diseño de la arquitectura del sistema es determinante para la satisfacción de los requerimientos de calidad (2).

1.6.1 Estilos arquitectónicos

Un estilo arquitectónico describe una clase de arquitectura y sus propiedades básicas. La aplicación de algún tipo de ellos en el diseño de la arquitectura de un sistema, es fundamental para la satisfacción de los requerimientos de calidad, garantizando el éxito. Por

tanto constituye uno de los conceptos más importantes dentro de la Arquitectura de software.

- Sirven para sintetizar estructuras de soluciones.
- Define los patrones posibles de las aplicaciones.
- Permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales.

Diferentes autores definen los estilos arquitectónicos como una familia de sistemas de software, otros lo describen como categorías de sistemas. A continuación se presentan varias de las definiciones expuestas por ellos.

Shaw y Garlan definen estilo arquitectónico como una familia de sistemas de software en términos de un patrón de organización estructural, que define un vocabulario de componentes y tipos de conectores y un conjunto de restricciones de cómo pueden ser combinadas. Para muchos estilos puede existir uno o más modelos semánticos que especifiquen cómo determinar las propiedades generales del sistema partiendo de las propiedades de sus partes (2).

Pressman define sobre estilos arquitectónicos que, cada estilo describe una categoría del sistema que contiene: un conjunto de componentes (por ejemplo, base de datos, módulos computacionales) que realizan una función requerida por el sistema; un conjunto de conectores que posibilitan la comunicación, la coordinación y la cooperación entre los componentes; restricciones que definen como se pueden integrar los componentes que forman el sistema; y modelos semánticos que permiten al diseñador entender las propiedades globales de un sistema para analizar las propiedades conocidas de sus partes constituyentes (1).

De las definiciones estudiadas se concluye que, el estilo arquitectónico constituye la forma de organización arquitectónica que se expresa de manera formal y teórica, brinda el camino de cómo estructurar y relacionar todos los componentes y conectores que intervienen en el sistema, incluyendo las restricciones que puedan existir. El uso de ellos puede conducir a una reutilización significativa de código y de componentes entre sí, disminuyendo costos de desarrollo. Los mismos proporcionan legibilidad, entendimiento del sistema y de las partes que lo componen, mejoramiento o disminución de la posibilidad de satisfacer algunos de los atributos de calidad presentes en el sistema.

1.6.2 Patrones arquitectónicos

Los patrones arquitectónicos expresan el esquema de organización estructural fundamental para sistemas de software. Provee un subconjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos (20).

Los patrones arquitectónicos especifican propiedades generales de la estructura del sistema y tienen un impacto en la arquitectura de subsistemas. La selección de un patrón arquitectónico es una decisión fundamental de diseño en el desarrollo de un sistema de software. Quienes se ocupan de ellos están más cerca del diseño, la práctica, la implementación, el refinamiento, el código entre otros elementos.

1.7 Estilos y patrones arquitectónicos

Los estilos arquitectónicos se seleccionan de acuerdo a las características y lo que se quiera obtener de un sistema, luego de seleccionado el estilo este define los patrones arquitectónicos que deben ser implementados. Muchos autores definen varias clasificaciones de estilos y patrones arquitectónicos los cuales no constituyen todos los que se han propuesto, sino los más representativos y vigentes. La presente investigación se guiará por una clasificación que permita un mejor entendimiento de forma clara.

La notación se establecerá entonces en términos de lo que Shaw y Clements llaman “boxology” (20).

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

Estilos de Flujo de Datos: Este estilo se basa en la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. Según Pressman se aplica cuando los datos de entrada son transformados a través de una serie de componentes computacionales o manipulativos en los datos de salida.

Tubería y filtros: Es una tubería (pipeline) que conecta componentes computacionales (filtros) a través de conectores (pipes), de modo que las computaciones se ejecutan a la manera de un flujo.

Se encuentran diseñados para recibir entradas de datos y producir salidas de datos de manera específica. Este estilo se divisa como un conjunto de transformaciones sobre los datos de entrada. Debe ser usado cuando:

- Se puede especificar la secuencia de un número conocido de pasos.
- No se requiere esperar la respuesta asincrónica de cada paso.
- Se busca que todos los componentes situados corriente abajo sean capaces de inspeccionar y actuar sobre los datos que vienen de corriente arriba (pero no viceversa).

Ventajas que presenta este estilo

- Es simple de entender e implementar.
- Es posible implementar procesos complejos con editores gráficos de líneas de tuberías o con comandos de línea.
- Es fácil de envolver en una transacción atómica.
- Los filtros se pueden empaquetar y hacer paralelos o distribuidos.

Desventajas que presenta este estilo

- No es apto para manejar situaciones interactivas.
- No maneja con eficiencia construcciones condicionales, bucles y otras lógicas de control de flujo.

Estilos Centrados en Datos: Proporcionan la integridad de los datos, los componentes existentes pueden cambiar y añadirse nuevos componentes a la arquitectura sin que afecte a otros clientes o sea consta de un almacén de datos donde otros componentes acceden a actualizar, modificar o borrar datos dentro del almacén.

Arquitecturas de Pizarra o repositorio: Consta de una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. Se han usado en aplicaciones que requieren complejas interpretaciones de proceso de señales entre otras. Se definen dos subcategorías principales del estilo:

- Si los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar, el repositorio puede ser una base de datos tradicional (implícitamente no cliente-servidor).
- Si el estado actual de la estructura de datos dispara los procesos a ejecutar, el repositorio es lo que se llama una pizarra pura o un tablero de control (20).

Estilos de llamada y retorno: Los estilos de llamada y retorno propician los atributos de calidad: modificabilidad, escalabilidad y desempeño. Son los estilos más generalizados en un sistema a gran escala. Según Pressman, este estilo permite al arquitecto del sistema construir una estructura de programa relativamente fácil de modificar y escalar.

Modelo Vista Controlador (MVC): Es reconocido como un estilo arquitectónico por Taylor y Medvidovic. Se utiliza cuando se requiere modularizar la interfaz de usuario, las reglas del negocio y el control de eventos. Separa el modelo del dominio, la presentación y las entradas de datos por el usuario en tres clases diferentes. La vista y el controlador dependen del modelo, el mismo no depende de las otras clases, esto permite que el modelo sea probado de forma independiente de la representación visual.

- **Modelo:** Administra el comportamiento, los datos del dominio de aplicación y responde a requerimientos de información sobre su estado. No depende de la vista ni del controlador. Garantiza y asegura la integridad de los datos permitiendo derivar nuevos datos.
- **Vista:** Maneja la visualización de la información.

- **Controlador:** Interpreta las acciones del ratón y el teclado, informando al modelo y/o la vista para que cambien según sea apropiado. Guarda la lógica del sistema y realiza todas las acciones necesarias para generarla.

Ventajas que presenta este estilo

- **Soporte de vistas múltiples**. Debido a que no existe una dependencia directa entre el modelo y las vistas, la interfaz de usuario puede mostrar varias vistas simultáneamente de los datos.
- **Adaptación al cambio:** Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Debido a la independencia entre las vistas y el modelo pueden realizarse cambios en las opciones de presentación que generalmente no afectan al modelo.

Desventajas que presenta este estilo

- **Complejidad:** Introduce nuevos niveles de indirección y profundiza la orientación a eventos del código de la interfaz de usuario, que puede llegar a ser difícil de depurar.
- **Costo de actualizaciones frecuentes**: Desacoplar el modelo de la vista no significa que los desarrolladores del modelo puedan ignorar la naturaleza de las vistas. Si el modelo experimenta cambios frecuentes, podrían desbordar las vistas con una lluvia de requerimientos de actualización.

Arquitectura en Capas: Garlan y Shaw definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la capa inmediatamente inferior (20).

Ofrecen un diseño que maximiza la reutilización y mantenibilidad. La separación de las capas dentro de una aplicación ayuda a la disminución de la dependencia entre ellas, lo que permite el mantenimiento del código y de la información. Proporcionan una clara delimitación del lugar donde debe de estar cada tipo de componente funcional. Son capas constituidas por varios paquetes o subsistemas. El número mínimo de capas que puede existir es dos.

Ejemplo de arquitectura en dos capas

- **Cliente-Servidor:** Es un sub-estilo, que se encuentra mayormente en las aplicaciones en red. Donde un componente servidor ofrece ciertos servicios y el componente cliente realiza la solicitud de ese servicio mediante un conector. El servidor ejecuta o rechaza la petición del cliente a través de una respuesta.

La arquitectura en capas es una de las más usadas, constituidas por las capas de presentación, negocio y de acceso a datos.

- **Capa de Presentación:** Es la encargada de realizar la interacción del usuario con la aplicación. Esta organiza todos los elementos de la aplicación que interactúen con el usuario. Además realiza las validaciones de los datos ingresados por el usuario. Envía la información de los mismos a los servicios de negocios para su procesamiento y recibe el resultado de estos mostrándolos al usuario.
- **Capa de Negocio:** Encargada de la lógica del negocio o de la aplicación, interactúa con la capa de presentación, donde recibe datos, los procesa y envía respuestas a la misma, también se interrelaciona con la capa de acceso a datos donde envía peticiones solicitando, depositando, actualizando o eliminando información.
- **Capa de Acceso a Datos:** Es responsable de la integridad de los datos y de separar el acceso a los datos de la lógica del negocio, permite acciones como eliminar y modificar, gestiona el almacenamiento físico y recuperación de datos.

Ventajas que presenta este estilo

- Soporta un diseño basado en niveles de abstracción crecientes y permite a los implementadores la partición de un problema complejo.
- Admite optimizaciones y refinamientos.
- Proporciona amplia reutilización.

Desventajas que presenta este estilo

- Los cambios en las capas inferiores tienden a filtrarse hacia las de un nivel superior.
- Complica en ocasiones las aplicaciones simples.
- No todos los sistemas pueden estructurarse en capas.

Arquitecturas Orientadas a Objetos: Los componentes de este estilo son los objetos, o más bien instancias de los tipos de dato abstractos (20). Estos componentes se basan en el principio orientado a objetos. Un diseño orientado a objetos permite diseñar sistemas para encapsular los datos y los objetos proveen interfaces a otros objetos. Resulta insistente en la reutilización a través del encapsulamiento, modificabilidad, polimorfismo y la herencia.

Ventajas que presenta este estilo

- Permite realizar modificaciones en la implementación de algún objeto sin que se vea afectada la interfaz.

- El objeto en este estilo es ante todo una entidad reutilizable durante el desarrollo.

Desventajas que presenta este estilo

- Se debe conocer la identidad de un objeto antes de interactuar con el mismo.
- Presentan problemas de efectos colaterales en cascada.

Arquitectura basada en componentes: Los sistemas de software basados en componentes se basan en principios definidos por una ingeniería de software específica. Es una unidad de composición con interfaces especificadas contractualmente y dependencias del contexto explícitas (20).

Ventajas que presenta este estilo

- Las interfaces están separadas de las implementaciones y sus interacciones son el centro de incumbencias en el diseño arquitectónico.
- Permiten alcanzar un mayor nivel de reutilización de software.
- Simplifica el mantenimiento del sistema.

Desventajas que presenta este estilo

- Admite en ocasiones que una interfaz sea implementada por varias componentes.
- Los estados de un componente no son accesibles desde el exterior.

Estilos de Código Móvil: Este grupo de estilos resaltan la portabilidad. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando. Fuera de las máquinas virtuales y los intérpretes.

Arquitectura de Máquinas Virtuales: Incluye un pseudo-programa a interpretar y una máquina de interpretación, la misma incluye la definición del intérprete y la del estado actual de su ejecución. Consta de cuatro componentes:

- Una máquina de interpretación que lleva a cabo la tarea.
- Una memoria que contiene el pseudo-código a interpretar.
- Una representación del estado de control de la máquina de interpretación.
- Una representación del estado actual del programa que se simula.

Estilos heterogéneos: En este apartado podrían agregarse formas que aparecen esporádicamente en los censos de estilos, como los sistemas de control de procesos

industriales, sistemas de transición de estados, arquitecturas específicas de dominios o estilos derivados de otros estilos.

Sistemas de Control de Procesos: Se caracterizan por los tipos de componentes y por las relaciones que mantiene entre ellos. Su objetivo es mantener algunos valores en ciertos rangos que se denominan puntos fijos o valores de calibración. La ventaja de este estilo radica en su elasticidad ante perturbaciones externas (21).

Arquitecturas Basadas en Atributos: Propuesta por Klein y Kazman, se establece como una extensión de la noción de estilo arquitectónico, los autores proponen que este estilo incluye un razonamiento cualitativo o cuantitativo basado en modelos específicos de atributos de calidad que declaran el comportamiento de los componentes en interacción.

Estilos Peer-to-Peer: Este estilo se enfoca en la modificabilidad, consiste en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades pero no controlarlas directamente.

Arquitecturas Basadas en Eventos: Las arquitecturas basadas en eventos se vinculan históricamente con sistemas basados en actores y redes de conmutación de paquetes. Los conectores de estos sistemas incluyen procedimientos de llamada tradicionales y vínculos entre anuncios de eventos e invocación de procedimientos (21).

Entre sus ventajas se encuentra la simplicidad, modularidad y el mejoramiento de la eficiencia. Como desventajas, el potencial imprevisto de escalabilidad, pobre comprensibilidad, poco soporte de recuperación en caso de falla parcial entre otras.

Arquitecturas Orientadas a Servicios (SOA): Establece un marco de diseño para la integración de aplicaciones independientes de manera que desde la red pueda accederse a sus funcionalidades, las cuales se ofrecen como servicios. La forma más habitual de implementarla es mediante Servicios Web, una tecnología basada en estándares e independiente de la plataforma, con la que SOA puede descomponer aplicaciones monolíticas en un conjunto de servicios e implementar esta funcionalidad en forma modular. Utiliza los servicios para brindar soporte a los requisitos del negocio (28).

De manera arquitectónica:

- Los servicios son entidades que encapsulan funcionalidades de negocios y brindan estas funcionalidades a otras entidades mediante interfaces públicas.
- Los componentes del estilo se encuentran débilmente acoplados.

Arquitecturas Basadas en Recursos: Roy Fielding considera que la Arquitectura Basada en Recursos (REST por sus siglas en inglés) resulta de la composición de varios estilos más básicos, incluyendo repositorio replicado, cache, cliente-servidor, sistema en capas, sistema sin estado, máquina virtual, código a demanda. REST define recursos identificables y métodos para acceder y manipular el estado de esos recursos. El caso de referencia es nada menos que la *World Wide Web (*www*)*. Define recursos identificables y métodos para acceder y manipular el estado de esos recursos (20).

1.8 Entorno de desarrollo

Los ambientes de desarrollo de software son herramientas que ayudan a los programadores a desarrollar software sobre entornos más amigables. Es decir, aquellos en los que el programador puede acceder con el menor esfuerzo a diferentes recursos como editores, compiladores, herramientas de análisis entre otros (29). A continuación se exponen las características y ventajas de las herramientas y tecnologías a utilizar.

1.8.1 Metodologías de desarrollo

Las metodologías de desarrollo cumplen un papel fundamental durante el desarrollo de un software, estas brindan un conjunto de métodos, procesos y herramientas necesarias que posibilitan que se entienda mejor lo que se quiere a la hora de comenzar con la planificación de un software. Permiten que se puedan realizar cálculos muy aproximados sobre la duración del proyecto así como determinar costo, riesgos y otra serie de parámetros. Cuando se pretende iniciar un proyecto, es necesario que se estudie adecuadamente la metodología a utilizar; debido a que existen diferentes metodologías que son más factibles emplearlas en proyectos pequeños y otras donde la duración pueda ser de 1 o más años.

Extreme Programming

Extreme Programming (XP) surge como una nueva forma de encarar proyectos de software, proponiendo una metodología basada esencialmente en la simplicidad y agilidad (10).

Consiste en una programación rápida o extrema. Se basa en tres aspectos fundamentales, la simplicidad, reutilización del código desarrollado y en la comunicación. Propicia un mejor ambiente entre el equipo de trabajo mediante una comunicación fluida. El cliente trabaja directamente con los desarrolladores y forma parte también del equipo de trabajo, además está presente durante todo el ciclo de vida del proyecto. Garantizando de esta manera el éxito del software en desarrollo. XP se emplea principalmente en proyectos de muy corta duración y en proyectos donde exista mayor probabilidad de riesgo. Ayuda a los

desarrolladores a dar respuestas a los requerimientos cambiantes por parte de los clientes, aunque el ciclo de vida del desarrollo del sistema se encuentre en fases tardías. Es una metodología que consta de 4 fases: Planificación, Diseño, Desarrollo y Pruebas.

Algunas de las características principales son:

- **Pruebas Unitarias:** se basa en las pruebas realizadas a los principales procesos de tal manera que puedan realizarse pruebas de fallas que pudieran ocurrir en un futuro.
- **Refabricación:** se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- **Programación en pares:** una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa (11).

Metodología SCRUM

Es un modo de desarrollo de carácter adaptable más que predictivo. Orientado a las personas más que a los procesos. Emplea la estructura de desarrollo ágil: incremental basada en iteraciones y revisiones (12). SCRUM es una metodología que fue aplicada en 1993 por Jeff Sutherland en el desarrollo de software.

Surgió como modelo para el desarrollo de productos tecnológicos, se emplea en entornos que trabajan con requisitos inestables y que requieren rapidez y flexibilidad. Consta de tres reuniones que se pudiera decir también fases, en la que son determinadas cada una de las tareas y análisis del desarrollo del software que se realice. Una primera reunión es la de planificación del sprint, donde se determinan las diferentes tareas o también llamadas "*sprint backlog*". Además se determina el objetivo y el trabajo en cada iteración que se realice. Otra reunión es el seguimiento del *sprint*, en la cual se realiza un análisis de las tareas para determinar el avance de las mismas y el trabajo durante la jornada. Como última reunión o fase se plantea la revisión del *sprint*, donde es realizado un análisis y revisión del incremento generado.

Proceso Unificado de Desarrollo

El Proceso Unificado de Desarrollo (RUP) es un proceso de ingeniería del software. Proporciona un acercamiento disciplinado a la asignación de tareas y responsabilidades en una organización de desarrollo. Su propósito es asegurar la producción de software de alta

calidad que se ajuste a las necesidades de sus usuarios finales con unos costos y calendarios predecibles (13).

RUP es una metodología de desarrollo de software que define quién, qué, cuándo y cómo alcanzar un determinado objetivo. Se apoya en el Lenguaje Unificado de Modelado (UML). Está constituida por 4 fases:

- **Inicio**: etapa donde se determina la visión del proyecto.
- **Elaboración**: etapa donde es determina la arquitectura del proyecto.
- **Construcción**: etapa donde se obtiene la capacidad operacional inicial.
- **Transición**: etapa donde se obtiene el producto terminado.

Está conformada por 9 flujos de trabajo: modelamiento del negocio, requisitos, análisis y diseño, implementación, pruebas, despliegue, administración de configuración y cambios, administración de proyectos y ambiente. Sus características principales son: dirigido por casos de uso, centrado en la arquitectura e iterativo e incremental. Actualmente es la metodología más general, completa y la más utilizada por los grandes proyectos de larga duración porque permite el análisis, desarrollo y genera gran documentación orientada a objeto. RUP realiza un levantamiento exhaustivo de requerimientos. Busca detectar defectos en las fases iniciales e intenta reducir el número de cambios tanto como sea posible.

Está conformada por vistas arquitectónicas: vista de casos de uso, vista lógica, vista de despliegue, vista de proceso y vista de implementación, todas ellas conocidas como “modelo 4+1 vistas”. Una de las ventajas que propone la metodología RUP es que se pueda mitigar posibles riesgos que se detecten durante el ciclo de vida del proyecto. Además de poder aplicar el conocimiento que se haya adquirido con los errores de una iteración, en próximas iteraciones.

1.8.2 Entorno de Desarrollo Integrado

Un Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) es un entorno de programación constituido por un conjunto de herramientas. Incluye editor de código, compilador, depurador y un constructor de interfaz gráfica. Puede utilizar uno o varios lenguajes de programación.

Netbeans

Netbeans es un IDE multilenguaje de código abierto, gratis, completo y modular que permite desarrollar aplicaciones para escritorio, dispositivos portátiles y la web. Brinda soporte para

estructuras *spring*, lenguajes de programación como C/C++, JavaScript, Ruby, Groovy, Phython y PHP. Trabaja en los sistemas operativos OpenSolaris, Linux, Windows y Mac OS. Algunas de las principales características que presenta este IDE son las mejoras en el editor de texto, su instalación resulta muy simple y consta de buenas características visuales para el desarrollo web, posee un buen completamiento de código para cada lenguaje de programación que soporta.

Eclipse

Es un entorno de desarrollo integrado multiplataforma para desarrollar aplicaciones en Java y otros lenguajes originalmente desarrollado por IBM, actualmente está desarrollada por la Fundación Eclipse. Es de código abierto, creado de manera altamente modular, lo que permite que sea utilizado para cualquier lenguaje si son desarrollados los *plugins* correspondientes (Java, Diseño de GUIs, *plugins* para desarrollar *plugins* de Eclipse y otros). Entre las características que presenta se encuentran: editor de texto, compilación en tiempo real, etc. Es una plataforma que soporta la construcción de varias herramientas de desarrollo y su principal objetivo es proporcionar mecanismos y reglas donde los fabricantes puedan integrar sus herramientas.

1.8.3 Lenguajes de descripción de la arquitectura

Los lenguajes de descripción de la arquitectura (ADL por sus siglas en inglés) son lenguajes descriptivos de modelado que se centran en la estructura de alto nivel de la aplicación antes que en los detalles de implementación de sus módulos concretos. Sus principales características son: composición, configuración, flexibilidad, reutilización y análisis. Entre sus componentes integra restricciones, estilos, propiedades, propiedades no funcionales, componentes, conectores, dinamismo y derivación.

Lenguaje Unificado de Modelado

El Lenguaje Unificado de Modelado (UML por sus siglas en inglés) es un lenguaje de modelado visual para construir (partiendo de modelos que sean especificados en sistemas diseñados), visualizar (expresa de manera gráfica un sistema de forma que otro pueda entenderlo), especificar (extraídas las principales características antes de la construcción) y documentar los artefactos de un sistema de software (los propios elementos gráficos sirven de documentación y para posterior revisión). Define los diagramas de colaboración, estados, actividades, componentes, despliegue, casos de uso, actividades, objetos y clases. A través de los diagramas son descritos los sistemas que se pretenden construir.

1.8.4 Lenguajes de desarrollo

Un lenguaje de programación es un conjunto de sintaxis y reglas semánticas que son definidas por los programas de la computadora. A través de acciones o comandos es posible crear programas, algoritmos, controlar tanto el comportamiento físico como lógicos de las computadoras. Es la manera práctica para que el humano pueda dar instrucciones a un equipo.

HTML5

HTML5 es una actualización de HTML, no constituye solamente la nueva versión del lenguaje HTML si no que va más allá con nuevas características y funcionalidades que permitan mejorar las aplicaciones web y que la ejecución de las mismas en un navegador no implique la falta de facilidades para los desarrolladores, para esto se están creando APIs (*Aplication Programming Interface*) que posibilitan utilizar todos los elementos de las páginas sin necesidad de ninguna tecnología intermediaria para realizar estas acciones.

Algunas de las características que incorpora HTML5 son:

- **Canvas:** es un nuevo componente que permite dibujar, por medio de las funciones de un API, en la página todo tipo de formas, que pueden estar animadas y responder a interacción del usuario. Es algo así como las posibilidades que nos ofrece Flash, pero dentro de la especificación del HTML y sin la necesidad de tener instalado ningún *plugin*.
- **Etiquetas para contenido específico:** Hasta ahora se utilizaba una única etiqueta para incorporar diversos tipos de contenido enriquecido, como animaciones Flash o vídeo. Ahora se utilizarán etiquetas específicas para cada tipo de contenido en particular, como audio, vídeo, etc.
- **Nuevas APIs para interfaz de usuario:** temas tan utilizados como el "*drag & drop*" (arrastrar y soltar) en las interfaces de usuario de los programas convencionales, serán incorporadas al HTML5 por medio de un API (14).

HTML5 es la nueva tecnología que proporciona mejoras en el desarrollo de los sitios web en cuanto a animaciones, videos, componentes de interfaces complejas, audio entre otros componentes que requieren del uso de *plugins*. Ofrece la posibilidad de obtener un código más limpio y fácil de depurar, que los códigos de los estándares anteriores

JavaScript

JavaScript es un lenguaje de programación interpretado que básicamente se refiere a que

no necesita ningún programa intermediario para ejecutarse en un navegador web, es robusto y a la vez ligero, a pesar de no ser un lenguaje orientado a objetos implementa muchas de las características de este paradigma, se utiliza generalmente para crear páginas web dinámicas, que son aquellas que presentan textos, imágenes, animaciones, ventanas que interactúan entre ellos y con el usuario. Desde su aparición tuvo un impacto total en internet, muchos sitios lo utilizaban, pero con la aparición de Flash tuvo un descenso en la popularidad por algunas de las posibilidades que brindaba este último, de las cuales carecía JavaScript, con el surgimiento de una nueva tecnología como AJAX (*Asynchronous JavaScript and XML*), que está programada en JavaScript, ha alcanzado una popularidad sin límites que no ha perdido hasta los días de hoy (15).

WebGL

WebGL (*Web-based Graphics Library*) es una especificación estándar web multiplataforma basada en OpenGl ES.2.0 y expuesto a través del elemento Canvas de HTML5, aprovecha la potencia de este para poder renderizar gráficos 3D acelerados por hardware en tiempo real y en el mismo navegador web usando un API y sin la necesidad de utilizar *plugins*, complementos adicionales que se necesitan con el fin de añadir funcionalidades a algún sistema y que no siempre son útiles, en ocasiones por problemas de compatibilidad con el navegador.

El renderizado en WebGL es similar al de OpenGl 2.0, la diferencia es que este lo hace en un contexto HTML, por ejemplo haciendo uso del Canvas del HTML5. Los experimentos con WebGL iniciaron a mediados del 2006 con el Canvas 3D en Mozilla. Para el 2007 Mozilla y Opera habían hecho ya algunas implementaciones propias. En el 2009 Mozilla y Khronos dieron inicio al grupo de trabajo de WebGL. Como miembros de este grupo, están los principales proveedores de navegadores como Apple con Safari, Google con el Chrome, Mozilla con Firefox, Opera. WebGL es manejado por el consorcio de tecnología Khronos Group (16).

CSS

CSS (hoja de estilo en cascada) es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. Brinda a los sitios web un determinado estilo compuesto por varios elementos como color, imágenes, bordes, márgenes, etc. Describe como se mostrará un documento en los sitios o como se imprimirá. Permite que los desarrolladores web puedan controlar el estilo y formato de sus páginas. De las ventajas que presenta se encuentran la personalización de las

páginas, el uso y redefinición de etiquetas, tamaño, visibilidad, usabilidad, mejoras en la accesibilidad del documento entre otras.

1.8.5 Herramientas CASE

Las Herramientas CASE constituyen un conjunto de programas, métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases. Sirven de apoyo tanto a analistas como desarrolladores quienes pueden modelar el ciclo de vida de desarrollo de un sistema mediante diferentes diagramas que pueden ser construidos en estas herramientas, ayudan además en el ahorro de tiempo y la automatización del sistema.

Visual Paradigm

Visual Paradigm es una herramienta multiplataforma, gratuita y fácil de instalar que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue. Permite que puedan ser dibujados diferentes tipos de Diagramas de Clases, generar códigos a partir de diagramas, código inverso y generar documentación. Soporta el modelado en UML ayudando a que exista un menor costo y calidad en la construcción de aplicaciones. El modelo y código permanecen sincronizados en todo el ciclo de desarrollo. Presenta capacidad de realizar el proceso de ingeniería directa e inversa o sea a partir del código se pueden obtener información acerca del diseño del sistema.

Rational Rose

Es una herramienta de software para el modelado visual mediante UML que ayuda a analizar, diseñar y especificar sistemas antes de construirlos. Completa una gran parte de las disciplinas de la metodología de desarrollo RUP, Modelo de Negocio, Análisis y Diseño, captura de requisitos, implementación, etc. Su diseño es centrado en casos de uso con el objetivo de diseñar un software de gran calidad, tiene mecanismos que propician la ingeniería inversa. No es gratuito, solo puede ejecutarse en el sistema operativo Windows e integrarse a otras herramientas de desarrollo Rational. Soporta la generación de código de modelos en C++, Java, J2EE, Visual Basic, Visual C++ y otros lenguajes.

1.8.6 Framework de desarrollo

Un *framework* de desarrollo se utiliza como base para la programación avanzada de aplicaciones, que aporta una serie de funciones o códigos para realizar tareas habituales. Son librerías de código que contienen procesos o rutinas ya listos para usar (17).

Las aplicaciones web son cada vez más complejas, incluyen efectos e interacciones que hasta hace poco tiempo eran exclusivas de las aplicaciones de escritorio. Al mismo tiempo, la programación de estas aplicaciones avanzadas se complica por varios motivos. Por todo lo anterior, han surgido librerías y *framework* específicos para el desarrollo de aplicaciones con JavaScript. Utilizando estas librerías, se reduce el tiempo de desarrollo y se tiene la seguridad de que las aplicaciones funcionan igual de bien en cualquiera de los navegadores más populares (18).

JQuery

JQuery es un *framework* para el lenguaje JavaScript creado por John Resig**,** es un proyecto de código abierto que fue liberado en enero del 2006, permite la creación de interfaces de usuario, efectos dinámicos, aplicaciones que hacen uso de AJAX. Algunas de las nuevas características que presenta este popular *framework* son las siguientes:

- Permite acceder al documento HTML.
- Modificar la apariencia de la página.
- Modificar el contenido de la página.
- Manejar eventos de los elementos de la página.
- Crear efectos visuales.
- Manipular estilos CSS.
- Manipulación en JSON (*JavaScript Object Notation*).
- Un simple pero potente sistema de *plugins* que permite extender a jQuery.
- AJAX (19).

PhiloGL

PhiloGL es un *framework* de visualización de datos, codificación creativa y desarrollo de juegos para WebGL, permite modelar y representar de manera vistosa la información de diversas fuentes haciendo uso de la placa de video. Algunas de las carecterísticas que presenta son: *framework* de código abierto, cuenta con librerías agrupadas dentro de la aplicación y se encuentra escrito en JavaScript.

PhiloGL tiene documentación de la API completa con una descripción detallada de todos los módulos y los métodos de la clase. Otra cosa muy interesante es que todas las lecciones de aprendizaje WebGL se han portado a PhiloGL (24).

Three.js

Three.js es un *framework* para WebGL con gran número de características tales como vistas de cámaras, objetos, luces y texturas. Permite hacer escenarios en 3D haciendo uso del elemento Canvas, SVG (*Scalable Vector Graphic*) y WebGL. Es una aplicación útil y relativamente sencilla de código abierto (30).

El potencial de este motor para 3D en la Web parece ser muy amplio y es de gran ayuda para desarrolladores que ya tienen la posibildad de traer modelos desde otras aplicaciones.

Algunos de los componentes con que cuenta la librería son:

- **Escena**: contiene los datos de los objetos en 3D.
- **Cámara:** tiene una posición, rotación y campo de visión.
- **Procesador:** deduce los objetos que aparecen en la escena desde el punto de vista de la cámara.

Además de estos componentes se encuentran otros como: el núcleo, luces, materiales, objetos, texturas, animaciones extras etc.

SpiderGL

SpiderGL proporciona estructuras típicas y algoritmos para el procesado en tiempo real a los desarrolladores de aplicaciones 3D de gráficos Web, sin obligarlos a cumplir con un paradigma específico (es decir, no es un escenario gráfico), ni impedir el acceso a bajo nivel de la capa subyacente de gráficos WebGL. Entre sus principios de diseño se encuentran la eficiencia, simplicidad y flexibilidad. Proporciona estructuras típicas de gráficos en 3D y algoritmos para desarrolladores. Es una librería de código abierto según el sitio oficial.

Los componentes que presentan son:

- **MATH (Matemáticas):** representa las rutinas de bajo y alto nivel de funciones de álgebra lineal y clases.
- **SPACE (Espacio):** representa las estructuras geométricas, clases y algoritmos relacionadas con el espacio.
- **GL:** presenta un alto y bajo nivel de WebGL y acceso a los recursos.
- **MESH (Malla):** importadores, editan y renderizan mallas.
- **UI:** interfaz de usuario del manejo de eventos. El elemento Canvas levanta eventos que son interceptados por el módulo UI y este lo envia al objeto registrado.

- **DOM:** se encarga del acceso al Modelo de Objetos del Documento (25).

1.8.7 Lenguajes de marcado

Un "Lenguaje de marcado" o "Lenguaje de marcas" se puede definir como una forma de codificar un documento donde, junto con el texto, se incorporan etiquetas, marcas o anotaciones con información adicional relativa a la estructura del texto. Los lenguajes de marcado permiten hacer explícita la estructura de un documento, su contenido semántico o cualquier otra información lingüística o extralingüística que se quiera hacer patente (26).

Son lenguajes que describen como se representan los datos y pueden ser utilizados para el almacenamiento de los mismos si no se desea hacer uso de algún Sistema Gestor de Base de Datos.

XML

XML (*Extensible Markup Language,* Lenguaje de Marcas Extensible) es un lenguaje usado para estructurar información en un documento o en general en cualquier fichero que contenga texto, como ficheros de configuración de un programa o una tabla de datos. Es un estándar abierto y libre. Constituye un meta-lenguaje que permite definir lenguajes de marcado adecuados a usos determinados, entre sus ventajas destaca la variedad en las estructuras de los datos y la sencillez. Presentan estructuras jerárquicas, son fáciles de procesar tanto para el entendimiento del usuario como del software. Forman además un conjunto de reglas para definir etiquetas semánticas donde ayuda en la organización de un documento en diferentes partes.

Características

- Es una arquitectura abierta y extensible. No se necesita versiones para que pueda funcionar en futuros navegadores. Los identificadores pueden crearse de manera simple y ser adaptados en el acto en internet/Intranet por medio de un validador de documentos.
- Integración de los datos de las fuentes más dispares. Se podrá hacer el intercambio de documentos entre las aplicaciones tanto en el propio PC como en una red local.
- Datos compuestos de múltiples aplicaciones. La extensibilidad y flexibilidad de este lenguaje permitirá agrupar una variedad amplia de aplicaciones, desde páginas web hasta bases de datos.

- Los motores de búsqueda devolverán respuestas más adecuadas y precisas, debido a que la codificación del contenido web en XML consigue que la estructura de la información resulte más accesible (27).

JSON

JSON (*JavaScript Object Notation*) es un formato de datos muy ligero basado en un subconjunto de la sintaxis de JavaScript. Como hace uso de las mismas, las definiciones pueden incluirse dentro de archivos JavaScript y acceder a ellas sin ningún análisis adicional como los necesarios con lenguajes basados en XML. Debido a la sencillez de JSON, se ha proporcionado la generalización de uso como alternativa de XML en AJAX. Una de las ventajas que presenta sobre XML es la forma sencilla para escribir un analizador semántico. Para encerrar alguna lista de ciertos valores delimitados por comas de JavaScript se requiere del uso de corchetes.

JSON está constituido por dos estructuras:

- Una colección de pares de nombre/valor, que pueden ser una estructura, un registros, un objetos etc.
- Una lista ordenada de valores, donde en los lenguajes se representan como listas, arreglos o vectores.

Entre sus características se encuentra la compatibilidad con objetos nativos. Ofrece tipos de datos escalables y la capacidad de expresar datos estructurales a través de matrices y objetos. Proporciona una asignación mucho más directa para los datos de aplicación. No requiere de código adicional para analizar texto.

1.8.8 Servidor Web

Un servidor web es un programa que se ejecuta continuamente desde un computador. Se encuentra a la espera de peticiones realizadas por parte de los usuarios a través de los navegadores que se comunican con este mediante el protocolo HTTP (protocolo de transferencia de hipertexto), las peticiones ejecutadas por los clientes son respondidas con una página web o un programa en el mismo servidor. Es el encargado de permitir la conexión entre el usuario y las aplicaciones o sitios que se encuentran alojadas en él.

Apache 2.0

Apache 2.0 es el servidor definido con anterioridad por políticas establecidas en el proyecto. En él se encuentra la colección que tiene integrado el componente Laboratorios virtuales.

Por ello se hará uso del mismo y no se realiza un análisis de gran profundidad debido a que fue realizado con anterioridad. Entre sus características y ventajas se encuentra que es multiplataforma, estructurado en módulos con varias categorías que permite adicionar o quitar funcionalidades, es gratuito y de código abierto.

1.9 Conclusiones

En el capítulo desarrollado fue realizado un estudio sobre soluciones similares existentes que pudieran tomarse como guía para el desarrollado de la presente investigación. Al no encontrarse ninguna referencia que hiciera alusión a ello se analizaron temas referentes a la Arquitectura de software con el objetivo de definir los elementos más adecuados para la propuesta de arquitectura y así dar cumplimiento a tareas definidas en la investigación, por lo que se concluye la toma de las siguientes decisiones: teniendo en cuenta las características y necesidades que requiere el componente, se empleará de los estilos arquitectónicos analizados el estilo de Llamada y Retorno y dentro de él, el sub-estilo Modelo Vista Controlador que cuenta con características y ventajas factibles para el desarrollo del componente donde permitirá establecer las reglas de negocio, el control de eventos en la interacción del usuario con el laboratorio, la organización de la estructura del código, reusabilidad, escalabilidad y la adaptación a cambios que puedan realizarse. La metodología de desarrollo seleccionada es RUP. El IDE a utilizar, NetBeans en su versión 7.0 previamente definido en el proyecto y UML como lenguaje de modelado. HTML5, definido con anterioridad, JavaScript, CSS y WebGL como lenguajes de desarrollo, la herramienta CASE a emplear, Visual Paradigm en su versión 3.4. Entre los *frameworks*, jQuery en su versión 1.5 para el manejo del DOM en la interfaz y Three.js para la generación de gráficos 2D en el navegador; JSON será empleado para el almacenamiento de los datos del componente y el servidor web que será utilizado, es el Apache 2.0.

Capítulo 2. Descripción de la Arquitectura

2.1 Introducción

Para el diseño de un sistema informático se abordan los temas más importantes antes de comenzar con su desarrollo y se requiere que todos los involucrados estén claros de ellos. En el presente capítulo es descrita la arquitectura propuesta para proveer un mayor entendimiento del componente a desarrollar. Se representa el modelo de dominio con los conceptos más relevantes con los cuales el usuario interactuará. Se identifican tanto los requisitos funcionales como los no funcionales, la descripción de los casos de uso arquitectónicamente significativos por los cuales se guiará el desarrollo del componente Laboratorios virtuales, además de otros aspectos importantes.

2.2 Modelo de Dominio

Un modelo de dominio es una representación visual de clases conceptuales o de objetos reales en un dominio de interés. Ayuda a entender aquellos conceptos con los cuales el usuario se relacionará y trabajará para la realización de un sistema. Son modelados mediante un diagrama de clases de UML. La metodología RUP propone la realización de un modelo de dominio cuando los procesos de negocio no están claramente definidos. A continuación se exponen los conceptos y el diagrama de dominio para el componente Laboratorios virtuales.

2.2.1 Análisis de los conceptos del dominio

Laboratorio: Componente del módulo Mediateca que simula un laboratorio real, donde se exponen experimentos y ejercicios de las asignaturas de Biología, Física, Química y Geografía, para la enseñanza secundaria.

Actividades: Representa las diferentes prácticas de laboratorio a desarrollar de las asignaturas Biología, Física, Química y Geografía.

Utensilios: Es una representación visual de los utensilios con los cuales se trabajan en un laboratorio real para la realización de los experimentos.

MesaTrabajo: Es el área de trabajo donde se realizan los experimentos.

Estante: Representa el conjunto de estantes que integran el laboratorio virtual.

EstanteUtensiliosMaderaOtros: Estante que contiene la representación de los utensilios de tipos madera y otros para la realización de los experimentos.

EstanteUtensiliosPorcelanaOtros: Estante que contiene la representación de los utensilios de tipo porcelana y otros para la realización de los experimentos.

EstanteMaterialesBiológicos: Estante que contiene la representación de los materiales biológicos para la realización de los experimentos.

EstanteUtensiliosCristal: Estante que contiene la representación de los utensilios de tipo cristal para la realización de los experimentos.

EstanteInstrumentosMedición: Estante que contiene la representación de los instrumentos que se utilizan para medir en la realización de los experimentos.

EstanteUtensiliosMetal: Estante que contiene la representación de los utensilios tipos metal para la realización de los experimentos.

2.2.2 Diagrama Modelo de Dominio

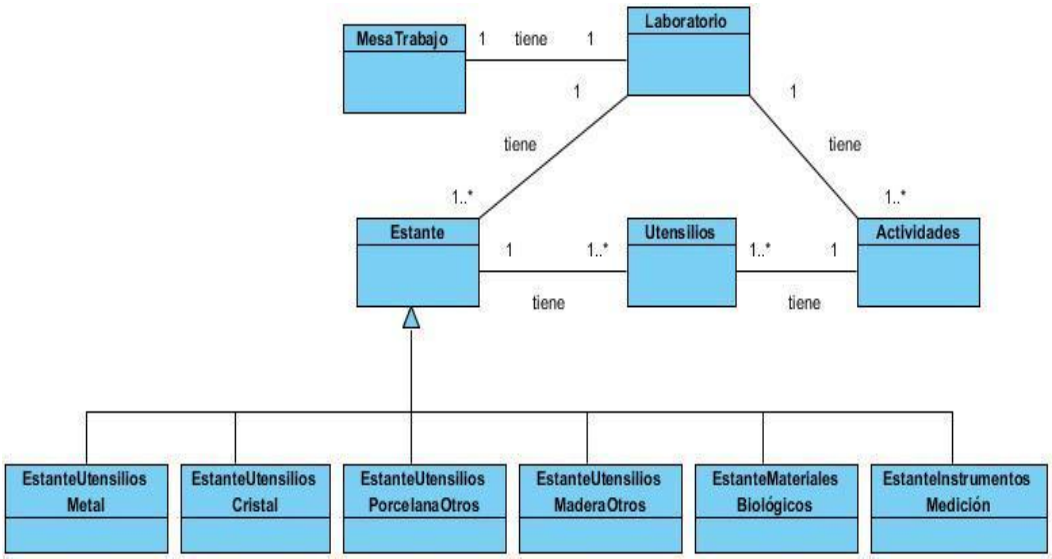

Figura 1. Modelo de Dominio

2.3 Descripción del sistema propuesto

Teniendo en cuenta el estudio realizado hasta el momento, fueron determinadas las funcionalidades principales y características que tendrá el componente Laboratorios virtuales. El mismo estará encargado de mostrar fenómenos y sucesos que ocurren en la naturaleza con el apoyo de imágenes, sonidos, videos, ejercicios y ejemplos, siendo el apoyo de varias asignaturas de la enseñanza secundaria.

En el componente estarán creados objetos y herramientas de trabajo para la realización de ejercicios, que ayudan al auto-aprendizaje del estudiante. Deberá ser desarrollado con

nuevas tecnologías libres para eliminar las restricciones privativas que presenta la versión actual, manteniendo las mismas normas, requerimientos y necesidades definidas al inicio de su desarrollo. Además de mantener los objetos y elementos actuales del componente debe ser posible agregar algunos otros en posterior desarrollo del mismo.

2.4 Requerimientos del sistema

Los requisitos de software son condiciones o capacidades que debe de cumplir el sistema, establecidos por un contrato que se define entre el cliente y el equipo de desarrollo, se dividen en los requisitos funcionales, que describen lo que el sistema debe de cumplir para su funcionamiento y los requisitos no funcionales, que son propiedades o cualidades que el producto debe de tener. A continuación para el componente Laboratorios virtuales perteneciente al módulo Mediateca de la colección El Navegante, serán expuestos solamente los requisitos funcionales más significativos definidos con anterioridad en el documento **"Desarrollo del componente Laboratorios Virtuales de la Colección El Navegante en su versión Multiplataforma"**.

2.4.1 Requisitos Funcionales

RF1: Seleccionar actividades virtuales.

RF2: Mover la cámara por la zona general de utensilios.

RF3: Seleccionar los utensilios a utilizar en la actividad virtual.

RF4: Mostrar la mesa o zona de trabajo.

RF5: Ejecutar la actividad virtual seleccionada.

2.4.2 Requisitos no Funcionales

Software

Para el componente Laboratorios virtuales se requiere de un servidor web Apache, navegador web Mozilla Firefox versión 10.0 o cualquiera superior a esta, para la visualización del mismo. Deberá ser capaz de trabajar en las plataformas Windows y Linux.

Restricciones en el diseño y la implementación

El componente Laboratorios virtuales debe ser implementado usando el lenguaje de programación JavaScript y HTML5. Para la implementación de las clases en la vista, se usará el framework de JavaScript jQuery en su versión 1.5 y para la representación de objetos 2D en la vista será utilizado el framework Three.js del lenguaje WebGL.

Usabilidad

El componente Laboratorios virtuales podrá ser usado por cualquier persona independientemente que está dirigido a estudiantes de la enseñanza secundaria. La información debe ser fácilmente accesible y contará con una interfaz amigable para lograr una mayor motivación en su utilización por los usuarios.

Funcionabilidad

El componente Laboratorios virtuales permitirá la interacción entre el usuario y los elementos que componen al mismo, para la realización de las diferentes actividades. Además de brindar la información necesaria para la comprensión de las actividades prácticas y la utilización de los objetos en cada una de ellas.

Portabilidad

El componente Laboratorios virtuales deberá ser multiplataforma, donde el usuario pueda acceder desde diferentes sistemas operativos que tengan disponible uno de los navegadores que soporte la colección, ya que el mismo deberá encontrarse integrada a esta.

Escalabilidad

El componte se construirá utilizando el estilo arquitectónico Modelo Vista Controlador, garantizando un mantenimiento ágil y seguro para futuras modificaciones o versiones del componente.

Rendimiento

El componente Laboratorios virtuales deberá ser desarrollado con un mínimo de aplicaciones para su visualización donde no dependa de *plugins*. Además de garantizar el consumo mínimo de recursos logrando un alto rendimiento.

2.5 Vistas arquitectónicas

Buschmann establece que una vista arquitectónica representa un aspecto parcial de una Arquitectura de software que muestra propiedades específicas del sistema (2).

Kruchten define una vista arquitectónica como una descripción simplificada o abstracción de un sistema desde una perspectiva específica, que cubre intereses particulares y omite entidades no relevantes a esta perspectiva (2).

Kazman describe las vistas arquitectónicas distinguiendo los componentes y las relaciones entre estos dentro de cada una, planteando que a menudo es útil combinar la información de dos o más vistas (2).

Para que exista una mejor compresión sobre el diseño del componente, es descrita la arquitectura a través de las 4+1 vistas que propone la metodología de desarrollo RUP, haciendo uso del lenguaje de modelado UML. Las vistas se basan tanto en la abstracción como en la descomposición y composición de estilos y estética. Utilizan elementos arquitectónicos para satisfacer los requisitos funcionales y no funcionales.

2.5.1 Vista de Casos de Uso

Esta vista contiene los casos de uso arquitectónicamente significativos, en la que es descrita la interacción entre objetos y procesos. Los casos de uso representan los requisitos funcionales, guían el diseño, la implementación y las pruebas. El componente en su desarrollo mantendrá la misma descripción de los casos de uso que se exponen en el documento **"Desarrollo del componente Laboratorios Virtuales de la Colección El Navegante en su versión Multiplataforma**", debido a que el mismo no presenta hasta el momento ninguna otra funcionalidad. Solo se expondrá en el presente trabajo los casos de uso que resultan arquitectónicamente significativos. La descripción más detallada de ellos se expone en el Anexo2.

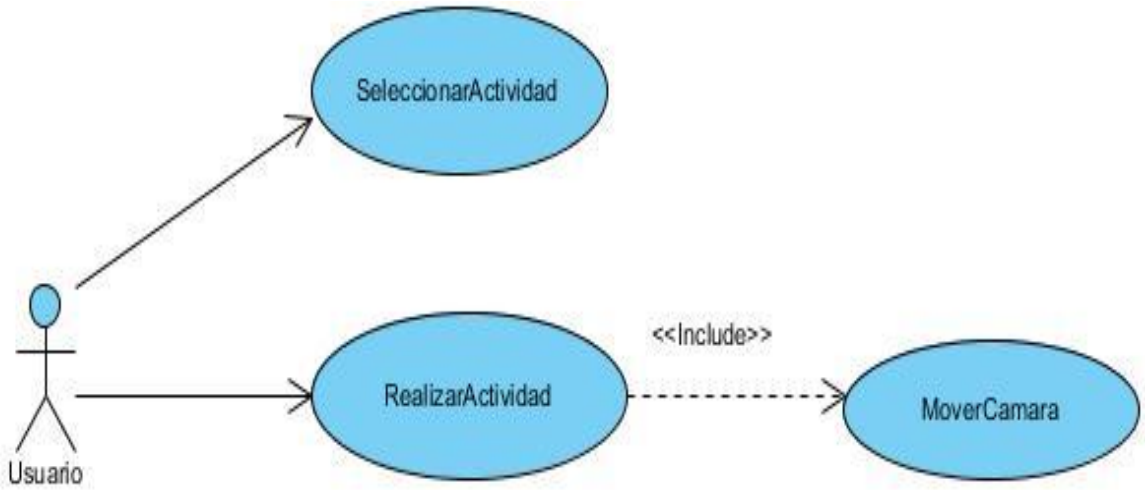

Figura 2. Diagrama de casos de uso arquitectónicamente significativos

Caso de uso SeleccionarActividad

Tabla 1. Resumen caso de uso SeleccionarActividad

Caso de Uso:	SeleccionarActividad
Actores:	Usuario
Resumen:	El caso de uso inicia cuando el actor selecciona una o todas las actividades de las que se proponen en un inicio, luego decide iniciar la actividad seleccionada finalizando así el caso de uso.
Precondiciones	Debe haberse generado el escritorio de trabajo del usuario autenticado y haberse generado el entorno de escritorio seleccionar actividad.
Referencias	RF1
Complejidad	Baja
Poscondiciones	Debe haberse seleccionado al menos una actividad experimental.

Caso de uso RealizarActividad

Tabla 2. Resumen caso de uso RealizarActividad

Caso de Uso:	RealizarActividad
Actores:	Usuario
Resumen:	El caso de uso inicia cuando el actor selecciona la acción realizar actividad, selecciona los utensilios que necesita para realizar la misma, luego monta los diferentes utensilios en la mesa de trabajo de manera que finalizando él pueda realizar el CU con la visualización del experimento.
Precondiciones	Debe haberse generado el escritorio de trabajo del usuario autenticado y haberse generado el entorno de escritorio de realizar actividad.
Referencias	RF2, RF3, RF4, RF5

Complejidad	Alta
Poscondiciones	Debe haberse visualizado la actividad.

Caso de uso MoverCamara

Tabla 3. Resumen caso de uso MoverCamara

Caso de Uso:	MoverCamara
Actores:	Usuario
Resumen:	El caso de uso inicia cuando el actor necesita de algún utensilio que no está disponible en el escenario donde se encuentra ubicado y necesita desplazarse al escenario siguiente. El usuario mueve la cámara hacia el escenario deseado finalizando así el caso de uso.
Precondiciones	Debe haberse generado el escritorio de trabajo del usuario autenticado y encontrarse en la zona general de utensilios.
Referencias	RF2
Complejidad	Baja
Poscondiciones	Haber pasado a otro escenario.

2.5.2 Vista Lógica

La vista lógica muestra una visión abstracta de cómo está diseñado el sistema. Se representan aquellos elementos de diseño más significativos de la arquitectura, las clases más importantes, su organización en paquetes y subsistemas. El objetivo del trabajo está enmarcado en brindar una vista lo más abstracta posible de cómo estará estructurado el componente Laboratorios virtuales, donde pueda ser utilizado en aplicaciones similares.

Vista: Se encuentra compuesta por tres paquetes: **Objetos de la Interfaz**, contiene las clases que representan los objetos de un laboratorio de forma virtual con los cuales el usuario interactuará para la realización de las actividades prácticas y así con las diferentes clases ventanas que visualizarán la información de los objetos. **Lib**, integra las librerías que brindan funcionalidades para el desarrollo del componente. **CSS** contendrá las funcionalidades que permitirán dar estilo a la interfaz del componente.

Controlador: Se encuentra compuesto por tres paquetes: **CEventos del Laboratorio**, contiene la clase que controlará todos los eventos que se realicen en el laboratorio. **Actividades**, integra las clases de las actividades a desarrollar dentro del componente Laboratorio virtuales. **Laboratorio**, constituido por la clase principal encargada de controlar todo la lógica, reglas del negocio, la implementación de las principales funcionalidades dentro de dicho componente así como el flujo de datos entre la vista y el modelo.

Modelo: Se encuentra compuesto por las clases que se encargarán de los datos. Debido a que el volumen de información es muy pequeño no se hará uso de ningún Sistema Gestor de Base de Datos, por lo que el almacenamiento de los mismos será a través de ficheros JSON.

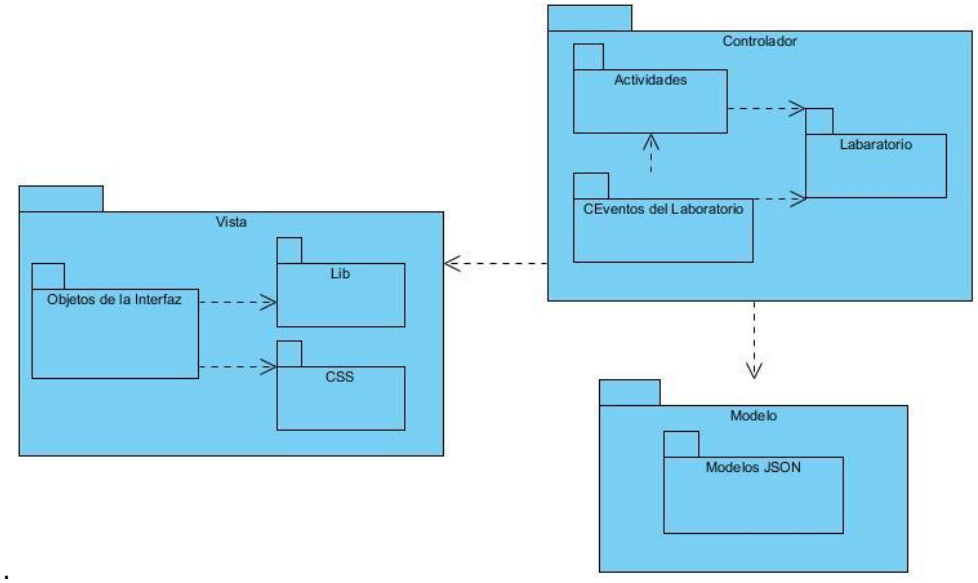

Figura 3. Distribución de paquetes más significativos

A continuación en la figura 4 se muestra un ejemplo del diseño de clase contenido dentro de los paquetes anteriormente descritos.

La Vista, contiene el paquete Objetos de la Interfaz integrado por la clase Estantes que incluye varios tipos de utensilios. Las clases MesaTrabajo y Ventana también se encuentran contenidas en el paquete, ellas brindan información necesaria al usuario para el trabajo dentro del laboratorio virtual.

El Controlador se encuentra compuesto por los paquetes CEventos del Laboratorio, Actividades y Laboratorio, los cuales contienen las clases EventosLab y Actividades que estarán relacionadas con la clase Laboratorio, siendo esta la controladora del negocio encargada de todas las funcionalidades que se desarrollen para el laboratorio.

El Modelo tiene presente un solo paquete, con las clases DatosJSON, que almacenará todos los ficheros JSON y ControladorJSON que realizará el control y trabajo con dichos ficheros.

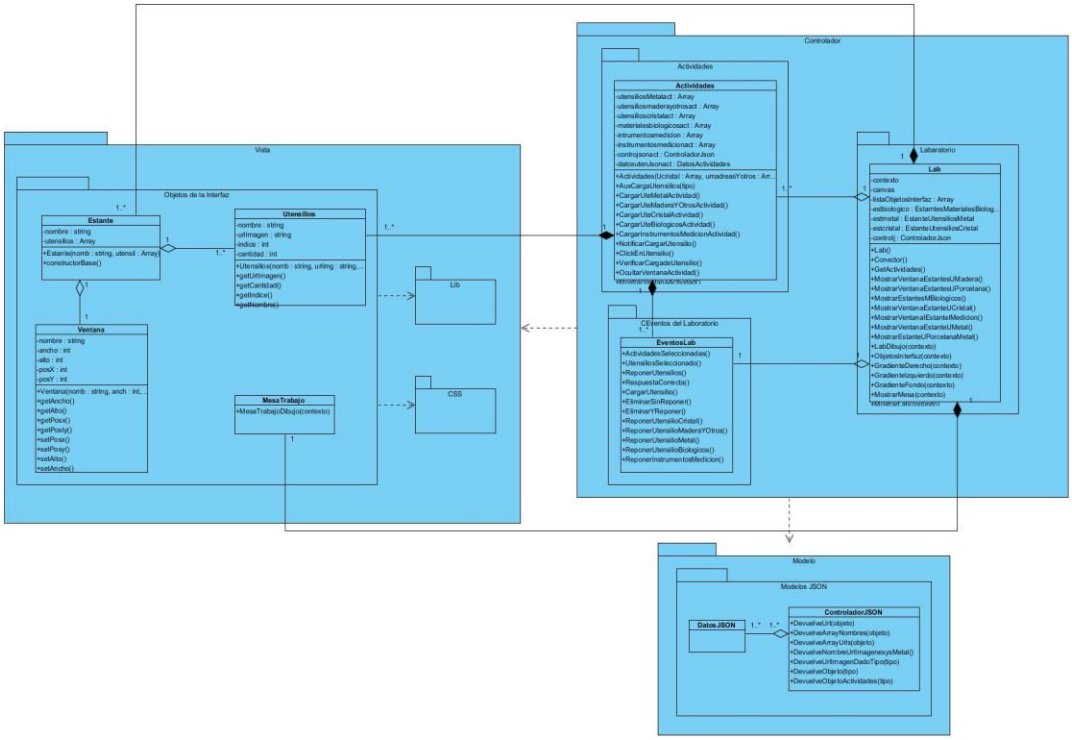

Figura 4. Ejemplo de diagrama de clases del caso de uso RealizarActividad

2.5.3 Vista de Despliegue

La vista de despliegue muestra una distribución física del sistema a través de nodos (recurso computacional que tiene memoria y capacidad de almacenamiento) y las conexiones entre ellos. Es utilizado cuando el sistema es distribuido. Cada componente físico es almacenado en los nodos por lo que existe una traza directa del modelo de implementación. El producto puede ser desplegado de dos formas para su explotación. A continuación se muestran ambas configuraciones definidas previamente en el documento **"Arquitectura de la versión multiplataforma de la colección de software educativo El Navegante",** debido a que el componente se encuentra integrado dentro de la colección el mismo se regirá por dichas condiciones.

Variante1: Instalarlo localmente en una PC. De esta forma, todas las características del software tienen que estar en todas las PC de los estudiantes.

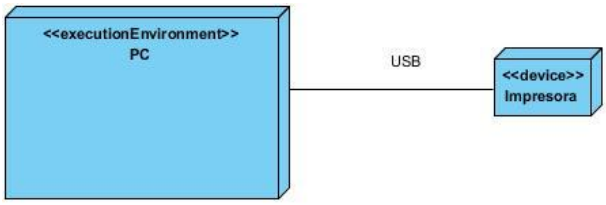

Figura 5. Diagrama de despliegue para la variante 1

Variante 2: Tener una PC como servidor central donde resida el Servidor Web y el de Bases de Datos. Desde cualquier PC se puede acceder al producto mediante la web sin tenerlo instalado.

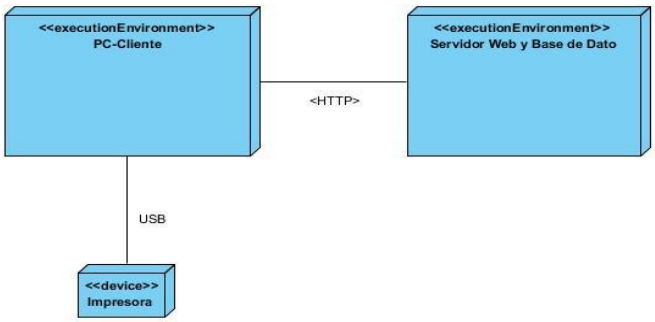

Figura 6. Diagrama de despliegue de la variante 2

Descripción de la capacidad que el dispositivo provee al sistema

El único dispositivo que utiliza el sistema es una impresora. El sistema tiene incluida la funcionalidad de impresión de los temas y el glosario, presentes en el software, por ello se necesita que la PC del usuario tenga conectada una impresora. No se necesita de una impresora en específico, solo requiere estar activa y lista para imprimir cuando el usuario esté trabajando con el software.

Descripción de Nodos

PC: Ordenador donde se instala la Colección, es decir, se instala un Servidor Web, uno de BD y todos los archivos que necesita el software para su correcto funcionamiento.

PC Cliente: Representa el nodo con el cual los usuarios interactuarán y donde se realizará la conexión a través de un navegador al servidor web que contiene la aplicación web con el componente.

Servidor Web: Contendrá la aplicación web que tiene incluido el componente a la que se conectarán los clientes a través de sus estaciones de trabajo y desde donde serán ejecutadas todas las funcionalidades del servidor web Apache.

Descripción del Protocolo de Comunicación:

<<HTTP>>: HTTP (protocolo de transferencia de hipertexto) es el protocolo que representa la comunicación entre la PC Cliente y el Servidor Web.

<<USB>>: Conexión que existe entre la impresora y la PC Cliente del usuario. Se modela con USB, pero puede existir otro tipo, todo depende del tipo de impresora que esté usando el usuario y el tipo de conexión que utilice ésta.

2.5.4 Vista de Implementación

La vista de implementación describe la descomposición de un software o sistema a desarrollar en capas y subsistemas de implementación. Los componentes son una parte modular en un sistema desplegable y reemplazable que encapsulan la implementación. Contienen clases y pueden ser implementados por ficheros ejecutables, binarios, entre otros artefactos. A continuación se representa la distribución de los componentes por cada paquete.

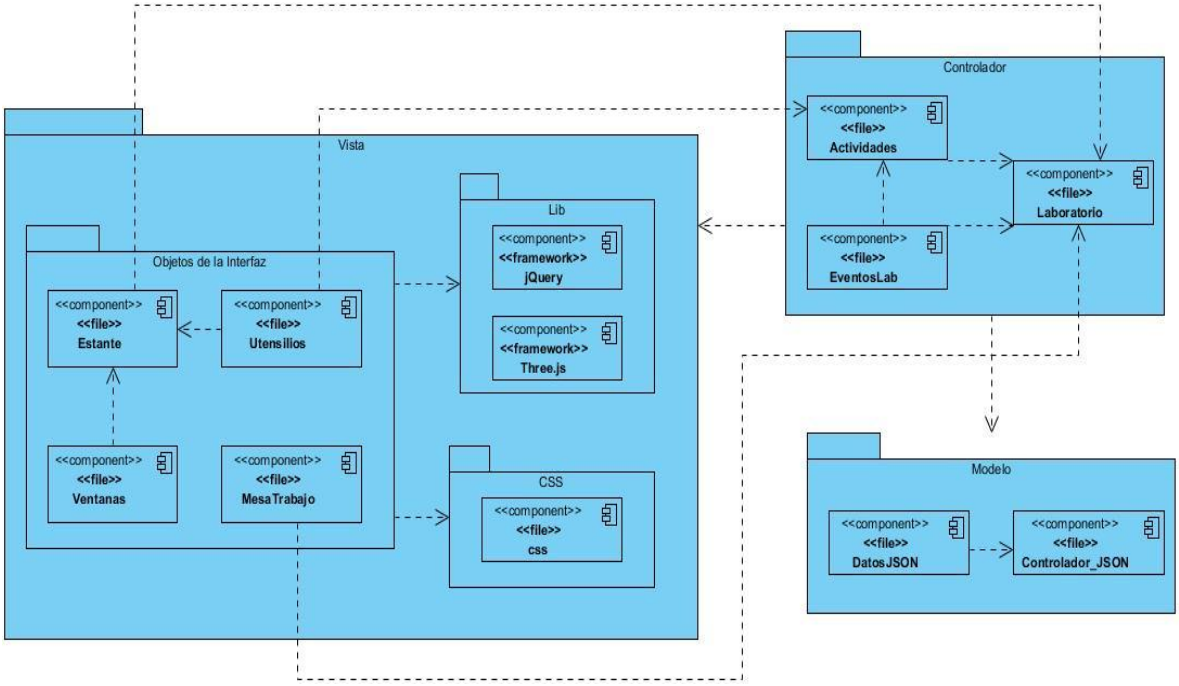

Figura 7. Diagrama de componentes

La vista representada en el diagrama de componentes contiene el paquete objetos de la Interfaz con los principales componentes. Dicho paquete se apoya de los componentes: framework Three.js, jQuery y la hoja de estilo CSS para el desarrollo de los mismos. El paquete controlador integra los componentes que controlan los eventos y la lógica del

negocio dentro del laboratorio. El modelo contiene los componentes para el almacenamiento de los ficheros JSON y el control de ellos.

2.5.5 Vista de Procesos

La vista de procesos describe los aspectos de concordancia y sincronización del diseño. Muestra los hilos y procesos de ejecución así como la comunicación entre ellos. Esta vista es usada cuando el sistema presenta procesos concurrentes o hilos. La solución que se propone no presenta procesos concurrentes por lo que no se realiza representación de la vista.

2.6 Conclusiones

En el capítulo desarrollado fue descrita la arquitectura propuesta a través del modelo 4+1 vistas que propone la metodología de desarrollo RUP, de esta forma se tiene una representación visual de cómo quedarían los componentes que integran la solución y las relaciones que existen entre ellos.

Capítulo 3. Evaluación de la Arquitectura

3.1 Introducción

El éxito o fracaso de un sistema se basa en varios factores, pero principalmente en su arquitectura. Haciendo uso de técnicas, métodos y atributos de calidad se evalúa la arquitectura con el objetivo de determinar que esta sea la más correcta para el sistema que vaya a ser desarrollado. Con una buena selección y definición de la misma se evitan posibles riesgos, por lo que el presente capítulo se basará en la evaluación de la arquitectura.

3.2 Evaluando la Arquitectura de Software

Cuando se define una correcta Arquitectura de software se garantiza que el sistema sea desarrollado con el mayor éxito posible. Para ello se requiere de la evaluación, que tiene como objetivo detectar posibles riesgos, buscar la forma de mitigarlos antes de la implementación del sistema y que la arquitectura satisfaga atributos de calidad que especifica el cliente. Las personas involucradas son: el equipo de evaluación y los stakeholders (grupo de personas o entidades que se ven afectadas por las decisiones de la empresa o institución), son los interesados en la arquitectura y el sistema que se construirá a partir de ella.

3.2.1 Importancia de evaluar la Arquitectura de Software

El objetivo de realizar evaluaciones en la arquitectura es la identificación de riesgos y la verificación de los requerimientos no funcionales del sistema. Un buen diseño arquitectónico depende del cumplimiento de los atributos de calidad. Es la forma más económica de evitar fracasos en el desarrollo. Que un sistema sea desarrollado con la mayor calidad posible y resulte exitoso, depende en gran medida de las decisiones arquitectónicas tomadas, debido a que es la arquitectura la que determina la estructura del proyecto, configuración, alcance, riesgos, presupuesto entre otros importantes aspectos, por ello su evaluación es fundamental, pues definirá donde se encuentran las vulnerabilidades, deficiencias y fortalezas de la misma y a partir de ahí tomar las decisiones pertinentes para la erradicación de los riesgos o vulnerabilidades en el sistema.

3.2.2 ¿Cuándo evaluar una arquitectura?

Generalmente la evolución de la arquitectura puede efectuarse luego de haber sido especificada y antes del inicio de la implementación, pero puede ser evaluada en varios

momentos dependiendo de su estado, si se encuentra en construcción o ha sido implementada. Estos momentos se identifican como **evaluación clásica**, es efectuada cuando la arquitectura se encuentra terminada y aún no se ha implementado; **evaluación temprana,** se lleva a cabo una o varias veces durante la etapa de construcción de la arquitectura y la **evaluación tardía**, es realizada una vez que la arquitectura ya existe y la implementación se ha completado**.**

Kazman expone que la evaluación de una arquitectura puede hacerse en cualquier momento y define dos variantes (2):

La primera variante, establece que no es necesario que la arquitectura esté completamente especificada para efectuar la evaluación y esto abarca desde las fases tempranas de diseño y a lo largo del desarrollo.

La segunda variante, consiste en realizar la evaluación de la arquitectura cuando ésta se encuentra establecida y la implementación se ha completado.

Reglas de Oro para determinar el momento de la evaluación de la arquitectura

- Realizar una evaluación cuando el equipo de desarrollo inicia a tomar decisiones que afectan directamente a la arquitectura.
- Cuando el costo de no tomar estas decisiones podrían pesar más que el costo de realizar una evaluación.

La evaluación de una arquitectura debe realizarse cuando la misma consta de suficientes elementos como para lograr justificarla. Un ejemplo sería, que el equipo de desarrollo comenzara a tomar decisiones que dependan de la arquitectura y el costo de las mismas sea mayor que el costo de evaluación.

3.3 Atributos de Calidad

Los atributos son requerimientos no funcionales del sistema, que referencian características que debe de satisfacer las necesidades del sistema. A dichas características de les denominan **atributos de calidad.**

De acuerdo al estándar IEEE un atributo de calidad es una característica que afecta la calidad de un elemento, en la anterior definición el término “característica” se refiere a aspectos no funcionales mientras que el término “elemento” se refiere a un componente o sistema (21).

Bosch establece una clasificación de los atributos de calidad en dos categorías (2).

Observables vía ejecución: son aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución.

No observables vía ejecución: son aquellos atributos que se establecen durante el desarrollo del sistema.

Observables vía ejecución

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

No observables vía ejecución

- **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.
- **Mantenibilidad:** Capacidad de modificar el sistema de manera rápida y a bajo costo.
- **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 Pruebas:** 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.

3.4 Técnicas de evaluación

Las técnicas existentes en la actualidad permiten realizar una evaluación sobre los atributos de calidad a nivel arquitectónico, las cuales pueden conducir a resultados precisos. Requieren de la información del sistema a desarrollar que se encuentra al principio del diseño debido a que no está disponible durante el proceso de diseño arquitectónico.

¿Cuándo usar las técnicas de evaluación?

- Las técnicas de evaluación cualitativas son usadas en la clasificación de evaluación clásica, cuando la arquitectura ha sido construida o cuando ésta se encuentra en construcción.
- Las técnicas de evaluación cuantitativas son usadas dentro de la evaluación tardía, una vez que la arquitectura ha sido implementada.

Las técnicas de evaluación para Arquitecturas de Software se clasifican en dos grupos, las cualitativas y cuantitativas. En las técnicas de evaluación cualitativas, se puede hacer uso de, escenarios, cuestionarios y listas de verificación y en las cuantitativas se emplean, métricas, simulaciones, modelos matemáticos, prototipos y experimentos. A continuación se realizó el estudio de algunas de las técnicas de ambos grupos con el objetivo de conocer y analizar las características de ellas.

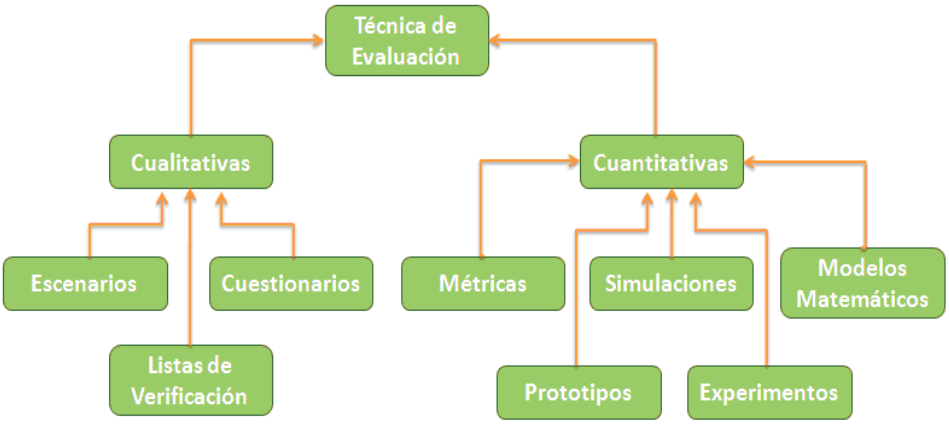

Figura 8. Clasificación de las técnicas de evaluación

3.4.1 Evaluación basada en escenarios

Los escenarios describen la interacción entre los stakeholders y el sistema, siendo los stakeholders cualquier involucrado en el desarrollo, desde clientes, desarrolladores, gerentes entre otros. Consta de tres partes: estimulo, contexto y respuesta.

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. Puede incluir ejecución de tareas, cambios en el sistema, ejecución de pruebas, configuración, etc. Permite establecer cuál es el atributo de calidad asociado.

Contexto: describe qué sucede en el sistema al momento del estímulo.

Respuesta: describe a través de la arquitectura, cómo debería responder el sistema ante el estímulo.

Estos distintos casos de escenarios se emplean para probar el sistema de puntos de vistas diferentes, de manera que sean analizadas todas las decisiones arquitectónicas que puedan representar algún riesgo para el sistema. Algunas de las ventajas que presentan los escenarios es que son simples de crear y entender, efectivos, no son muy costosos y no necesitan de gran entrenamiento. Proveen un vehículo que permite concretar y entender atributos de calidad. Esta técnica basada en escenarios consta de dos instrumentos para la evaluación, el Árbol de Utilidades (*Utility Tree*) que fue propuesto por Kazman y los Perfiles (*Profiles*) propuestos por Bosch.

Árbol de Utilidades

Para Kazman un Árbol de Utilidades son esquemas que se representan en forma de un árbol en el cual presentan ciertos atributos de calidad de un sistema de software, donde se

encuentran refinados hasta establecer escenarios con los cuales especifican detalladamente el nivel de prioridad que requiere o que tienen cada uno.

El Árbol de Utilidades es lo que realiza la identificación de los atributos de calidad más importantes para el sistema. Estos se definen por los involucrados en el desarrollo del sistema al momento en que se construye el árbol. Como nodo raíz contiene la utilidad del sistema. Los diferentes atributos conforman el segundo nivel los que se refinan hasta obtener los escenarios suficientemente concretos para analizarlos y otorgarle prioridad. Cada atributo del árbol contiene una serie de escenarios relacionados, una escala de importancia y dificultad asociada para la evaluación de la arquitectura.

Perfiles

Constituyen conjuntos de escenarios en la mayoría de los casos presentan importancia relativa relacionada con cada escenario. El uso de ellos permite realizar especificaciones de manera más precisa del requerimiento para atributos de calidad. Estos constan de dos calcificaciones:

- **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.
- **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.

3.4.2 Simulación

Bosch establece que 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 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 (22).

Los instrumentos de evaluación con que cuenta son, los lenguajes de descripción arquitectónica (ADL) y los modelos de datos.

Bosch define 5 pasos para este proceso de evaluación los cuales son:

- Definición e implementación del contexto.

- Implementación de los componentes arquitectónicos.
- Implementación del perfil.
- Simulación del sistema e inicio del perfil.
- Predicción de atributos de calidad.

3.4.3 Modelos Matemáticos

La evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. Permite una evaluación estática de los modelos de diseño arquitectónico y se presentan como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados, utilizando los resultados de uno como entrada para el otro (2).

Bosch define los siguientes pasos para este proceso de evaluación:

- Selección y adaptación del modelo matemático.
- Representación de la arquitectura en términos del modelo.
- Estimación de los datos de entrada requeridos.
- Predicción de atributos de calidad.

Cuenta con los instrumentos de evaluación, cadenas de Markov y los diagramas de bloque de fiabilidad. Una de sus desventajas es la carencia de modelos matemáticos para atributos de calidad relevantes para la Arquitectura de software.

3.4.4 Evaluación basada en experiencia

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 (2).

La evaluación basada en experiencia consta de dos tipos: la evaluación informal, que es la que se realiza por los arquitectos de software durante el proceso de diseño y la realizada por equipos externos de evaluación de arquitecturas. Los instrumentos de evaluación que presenta son: intuición y experiencia, tradición y proyectos similares.

3.4.5 Consideraciones sobre Técnicas de Evaluación

Las técnicas de evaluación cualitativa y cuantitativa de la arquitectura, permiten valorar el diseño arquitectónico desde el comienzo de su desarrollo, constituyen un medidor para medir cuán acertada es una Arquitectura de software. Son utilizadas principalmente en el apoyo de la evaluación realizada por un método. De las clasificaciones de técnicas estudiadas se concluye que las pertenecientes al grupo de las cualitativas constan de menor esfuerzo en su implementación que las técnicas cuantitativas, además están caracterizadas por resultar simples y entendibles por parte de todos los involucrados en el proceso de evaluación de la arquitectura. Debido a la simplicidad y facilidad de implementación, la técnica de evaluación basada en escenarios es considerada como la más difundida. Las mismas proveen una forma de concretizar cualidades en cuanto a tiempo de desarrollo y en la cual se apoyan los métodos de evaluación.

3.5 Métodos de evaluación

Existen varios métodos para la evaluación de la Arquitectura de software, ellos permiten la búsqueda de riesgos y soluciones, lo que ayuda a obtener una arquitectura bien definida. Entre los métodos existentes para evaluar arquitecturas, se encuentran: el Método de Análisis de Arquitectura de Software (SAAM), el Método de Análisis de Acuerdos de Arquitectura de Software (ATAM) y el Método de Revisión Intermedio de Diseño (ARID).

3.5.1 Método de Análisis de Arquitectura de Software

El Método de Análisis de Arquitectura de Software (SAAM) es un método de evaluación basado en escenarios, que representa probables cambios a los que el sistema se encontrará sometido. Permite evaluar o comparar una o varias arquitecturas. Es usado para evaluar de forma rápida atributos de calidad como modificabilidad, escalabilidad, portabilidad e integridad.

Cuando es evaluada una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, se culmina con un reporte que señala los componentes computacionales donde la arquitectura no logra alcanzar el nivel requerido. En el caso de ser múltiples arquitecturas, se produce una escala relativa que permite observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones. Culmina el análisis con una tabla indicando las fortalezas y debilidades de cada una en cada escenario.

Pasos a seguir del método SAAM

- **Paso 1:** Desarrollo de escenarios

- **Paso 2:** Descripción de la arquitectura
- **Paso 3:** Clasificación de escenarios
- **Paso 4:** Evaluación de escenarios
- **Paso 5:** Interacción de escenarios
- Paso 6: Evaluación general

3.5.2 Método de Análisis de Acuerdos de Arquitectura de Software

El Método de Análisis de Acuerdos de Arquitectura de Software (ATAM) es un método de evaluación que indica cuán bien una arquitectura particular satisface las metas de calidad y provee ideas de cómo esas metas de calidad interactúan entre ellas (23).

El método se basa en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM (2). Proporciona información sobre el cumplimiento de los atributos de calidad a través de elementos que representan los medios que utiliza la arquitectura para lograr alcanzar los atributos de calidad. Concentrándose principalmente en los estilos arquitectónicos.

Una de sus principales características consiste en identificar riesgos de la arquitectura diseñada a través de la identificación de puntos de sensibilidad, desventajas y riesgos. No cuenta con un modelo para el refinamiento de los atributos de calidad, pero si con el instrumento árbol de utilidad que presenta un nivel de refinamiento de los atributos más abarcadores. Este método es considerado como el más completo y es el encargado de analizar cómo la arquitectura logra satisfacer los atributos de calidad: modificabilidad, portabilidad, extensibilidad e integrabilidad y tiene en cuenta las dependencias que se crean entre ellos.

Este método consta de 9 pasos agrupados en 4 fases las cuales se muestran a continuación:

Fase 1.Presentación

- **Paso 1:** Presentación del ATAM
- **Paso 2:** Presentación de las metas del negocio
- **Paso 3:** Presentación de la arquitectura

Fase 2: Investigación y análisis

- **Paso4:** Identificación de los enfoques arquitectónicos
- **Paso 5:** Generación del *Utility Tree*
- **Paso 6:** Análisis de los enfoques arquitectónicos

Fase 3: Pruebas

- **Paso 7:** Lluvia de ideas y establecimiento de prioridad de escenarios
- **Paso 8:** Análisis de los enfoques arquitectónicos

Fase 4: Reporte

- Paso 9: Presentación de los resultados

3.5.3 Método de Revisión Intermedio de Diseño

El Método de Revisión Intermedio de Diseño (ARID) es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura.

Es un método de bajo costo considerado por los autores como 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). En ADR, los involucrados reciben una documentación detallada y completan cuestionarios, cada uno de ellos por separado.

Una de sus principales desventajas es en la manera de realizar la evaluación del diseño arquitectónico, donde se basa solamente en el análisis de los componentes de la arquitectura sin tener en cuenta las conexiones que se establecen entre ellos.

Este método consta de 9 pasos agrupados en dos fases que serán expuestas a continuación:

Fase 1: Actividades Previas

- **Paso 1:** Identificación de los encargados de la revisión
- **Paso 2:** Preparar el informe de diseño
- **Paso 3:** Preparar los escenarios base

- **Paso 4:** Preparar los materiales

Fase 2: Revisión

- **Paso 5:** Presentación del ARID
- **Paso 6:** Presentación del diseño
- **Paso 7:** Lluvia de ideas y establecimiento de prioridad de escenarios
- **Paso 8:** Aplicación de los escenarios
- Paso 9: Resumen

3.5.4 Comparación entre métodos de evaluación

La siguiente tabla expone una comparación entre los métodos de evaluación ATAM, SAAM y ARID (2).

Tabla 4. Comparación entre métodos de evaluación

	ATAM	**SAAM**	**ARID**
Atributos de Calidad Contemplados	Modificabilidad Seguridad Confiabilidad Desempeño	Modificabilidad Funcionabilidad	Conveniencia del diseño evaluado
Objetos Analizados	Estilos Arquitectónicos Documentación Flujo de Datos Vistas Arquitectónicas	Documentación Vistas Arquitectónicas	Especificación de los componentes
Etapas del Proyecto en las que se Aplica	Luego que el diseño de la arquitectura ha sido establecido	Luego que la arquitectura cuenta con funcionalidad ubicada en módulos	A lo largo del diseño de la arquitectura

Enfoques Utilizados	Árbol de Utilidad y lluvia de ideas para articular los requerimientos de calidad. Análisis arquitectónico que detecta puntos sensibles, puntos de balance y riesgos.	Lluvia de ideas para escenarios y articular los requerimientos de calidad. Análisis de los escenarios para verificar funcionalidad o estimar el costo de los cambios.	Revisiones de diseño, lluvia de ideas para obtener escenarios.

De la comparación expuesta en la tabla entre los métodos de evaluación se pudo apreciar que ATAM, es el método que más atributos de calidad y objetos abarcan para realizar la evaluación y dentro de todos ellos es el que mayor documentación presenta.

3.5.5 Método de evaluación seleccionado

Luego de analizados los métodos de evaluación se puede concluir que los mismos constituyen una guía para aquellos involucrados en el desarrollo de un sistema, ayudan en la búsqueda de debilidades y soluciones que puede presentar el diseño arquitectónico seleccionado. Cada uno independiente es utilizado para evaluar la arquitectura de acuerdo a lo que se requiera evaluar de ella, por ejemplo el método SAAM es factible aplicarlo cuando el atributo de calidad Modificabilidad es de mayor interés. ATAM es más profundo para evaluar aspectos que se encuentran más relacionados con la arquitectura como la confiabilidad o el rendimiento y ARID evalúa mejor la factibilidad de la arquitectura. Para la evaluación de la arquitectura propuesta el método más conveniente es el ATAM, debido a que es considerado el más completo, por la amplia documentación, muestra la manera de que la arquitectura satisfaga ciertos atributos de calidad especificados para el sistema e interactúen entre ellos.

3.6 Evaluación de la Arquitectura

Para el diseño arquitectónico propuesto anteriormente fue realizado un análisis sobre las técnicas y métodos de evaluación, en el cual se decidió utilizar, de las técnicas cualitativas, la evaluación basada en escenarios para la descripción de la interacción de los stakeholders

y el componente. Como instrumento de evaluación, el Árbol de Utilidades para la identificación de los atributos de calidad más significativos y de los métodos de evaluación de arquitecturas de software, ATAM para realizar la evaluación y determinar si dicha arquitectura satisface los atributos de calidad los cuales se determinaron de acuerdo a la utilidad del sistema o sea aquellos que poseen una alta prioridad.

Para la aplicación del método ATAM sobre la arquitectura propuesta, fue realizada la presentación del mismo donde se abordó sobre su funcionamiento, principales características, técnicas a emplear etc. Se expusieron las metas del negocio abordando principalmente los conceptos y el objetivo arquitectónico del mismo. Posteriormente se realizó la presentación de la arquitectura con el objetivo de verificar que la misma cumple con los atributos de calidad definidos para el componente. Se analizaron los atributos de calidad mediante los cuales se obtuvieron los escenarios que a continuación son mostrados en el Árbol de Utilidades.

Árbol de Utilidades

El Árbol de Utilidades se encuentra formado por los atributos de calidad más importantes para el componente Laboratorios virtuales, donde se encuentran refinados a escenarios para percibir su comportamiento según el diseño arquitectónico.

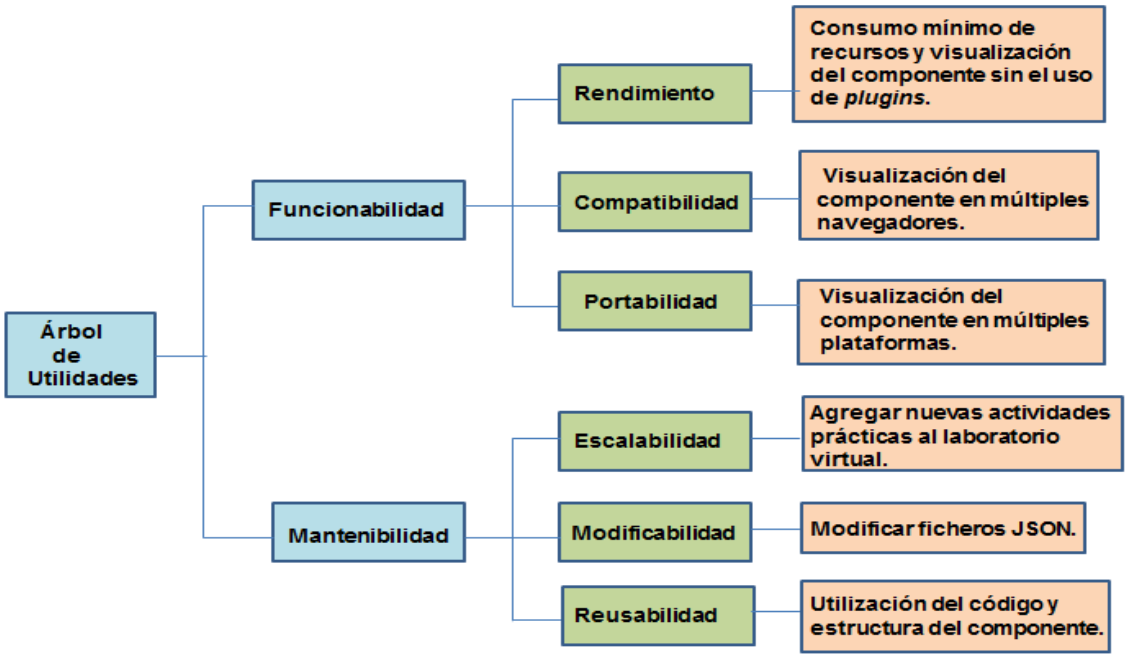

Figura 9. Árbol de Utilidades

Análisis de los escenarios

A continuación se muestran las diferentes tablas que resumen la relación entre los atributos de calidad y los escenarios definidos en el Árbol de Utilidades, brinda la posibilidad de confirmar todo el análisis desarrollado para la posterior implementación del componente Laboratorios virtuales.

Tabla 5. Análisis del escenario: "Consumo mínimo de recursos y visualización del componente sin el uso de *plugins*"

Escenario	Consumo mínimo de recursos y visualización del componente sin el uso de *plugins*.			
Atributo	Rendimiento.			
Ambiente	Trabajo sobre el componente.			
Estímulo	Utilización de nuevas tecnologías.			
Respuesta	El componente se visualiza solamente con el navegador.			
Decisiones arquitectónicas	**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
Uso del framework jQuery.				NR1
Uso del Canvas de HTML5 y el framework Three.				NR2
Explicación	El uso del framework jQuery permitirá la implementación de las ventanas y manipulación de las imágenes para la visualización de la interfaz del componente, además de ser el encargado de manejar todas las acciones en el DOM. Con el manejo del framework Three y el elemento Canvas de HTML5 se logra que el componente solo dependa para su visualización del navegador sin la ayuda de aplicaciones intermedias (*plugins*), mejorando el rendimiento. Las decisiones arquitectónicas tomadas constituyen un no riesgo debido a que no se			

	afectan otros elementos dentro del laboratorio.

Tabla 6. Análisis del escenario: "Visualización del componente en múltiples navegadores"

Escenario	Visualización del componente en múltiples navegadores.			
Atributo	Compatibilidad.			
Ambiente	Realización de actividades prácticas con el laboratorio.			
Estímulo	Ejecutar el componente.			
Respuesta	El componente se muestra con el mismo diseño en los principales navegadores.			
Decisiones arquitectónicas	**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
Uso del framework jQuery.				NR3
Uso del framework Three.				NR4
Uso del Canvas de HTML5.	R1			
Uso de CSS y JavaScript para la visualización.	R2			
Explicación	Para la visualización del componente en los navegadores, principalmente Mozilla Firefox en su versión 10.0 o cualquier versión superior, se utilizará jQuery, framework JavaScript donde todas sus funcionalidades son soportadas por todos los navegadores, su uso garantiza la compatibilidad del componente, representando un no riesgo para la implementación de otros subsistemas. El framework Three y el elemento Canvas de			

	HTML5 ayudarán a la visualización del componente, sin embargo este último constituye un riesgo porque no todos los navegadores tienen soporte para este elemento por lo que deberá tener presente para su desarrollo. Todos los navegadores no brindan soporte de igual manera para CSS y JavaScript lo que constituye un riesgo, por ello deberán ser implementadas las funcionalidades teniendo en cuenta esta restricción.

Tabla 7. Análisis del escenario: "Visualización en múltiples plataformas"

Escenario	Visualización en múltiples plataformas.			
Atributo	Portabilidad.			
Ambiente	El componente Laboratorios virtuales al encontrarse integrado dentro de la colección El Navegante siendo esta multiplataforma, debe cumplir los requerimientos para la portabilidad que esta define.			
Estímulo	Visualizar el componente en varios sistemas operativos.			
Respuesta	Se visualiza en varios sistemas operativos manteniendo los mismos requerimientos.			
Decisiones arquitectónicas	**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
Uso del lenguaje JavaScript para la visualización.				NR5
Uso del framework jQuery.				NR6
Uso del servidor web Apache.				NR7
Uso del fichero JSON				NR8

para el almacenamiento de datos.				
Explicación	La Portabilidad es la capacidad del sistema de ser ejecutado en ambientes como hardware o software. Para el desarrollo del componente Laboratorios virtuales se propuso como lenguaje de programación JavaScript, entre los framework jQuery, como servidor web Apache y el fichero JSON. Cada uno de estos elementos resulta compatible con la mayoría de los sistemas operativos más usados. Por ello se puede concluir que el componente puede adaptarse a diferentes ambientes como Linux y Windows. La utilización de dichos elementos no conlleva a ningún riesgo para el desarrollo o utilización del componente.			

Tabla 8. Análisis del escenario: "Agregar nuevas actividades prácticas al laboratorio virtual"

Escenario	Agregar nuevas actividades prácticas al laboratorio virtual.			
Atributo	Escalabilidad.			
Ambiente	Necesidad de incorporar otras actividades al componente.			
Estímulo	Se necesita incorporar otras actividades prácticas.			
Respuesta	Se agrega una nueva clase para la actividad práctica en el paquete encargado de guardar todas las clases actividades.			
Decisiones arquitectónicas	**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
Uso de JavaScript para agregar la nueva actividad.				NR9
Uso del framework jQuery para el manejo de los elementos del DOM				NR10

en la vista donde se visualizará la actividad.				
Explicación	Para agregar una nueva actividad práctica, se requiere añadir una nueva clase y para ello se emplea el lenguaje JavaScript, esta clase estará contenida dentro del paquete Actividades. El framework jQuery permitirá manipular los elementos del DOM para la ventana de visualización de la actividad. Las decisiones arquitectónicas tomadas no representan ningún riesgo que afecte al diseño, ni a la inclusión de los datos de la actividad en el fichero con formato JSON correspondiente.			

Tabla 9. Análisis del escenario: "Modificar ficheros JSON"

Escenario	Modificar ficheros JSON.			
Atributo	Modificabilidad.			
Ambiente	Necesidad de modificar la configuración de los ficheros JSON.			
Estímulo	Realizar modificaciones de los ficheros en el modelo.			
Respuesta	No deben ser afectados el resto de los ficheros JSON en el modelo.			
Decisiones arquitectónicas	**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
Uso del patrón Modelo Vista Controlador.				NR11
Explicación	La Modificabilidad es la habilidad de los sistemas para realizar cambios futuros. Al emplear el patrón Modelo Vista Controlador es separada la lógica del negocio de la manera en que serán representados los datos. Si se desea modificar o crear un fichero JSON, la vista y el controlador no recibirán cambios en caso de que no sea necesario. Por lo que su uso constituye un no riego.			

Tabla 10. Análisis del escenario: "Utilización del código del componente"

Escenario	Utilización del código del componente.			
Atributo	Reusabilidad.			
Ambiente	Se necesita de una nueva versión del componente.			
Estímulo	Desarrollar una nueva versión del componente.			
Respuesta	El código del componente debe responder a los lenguajes definidos en el proyecto.			
Decisiones arquitectónicas	**Riesgo**	**Sensibilidad**	**Tradeoff**	**No Riesgo**
Uso del lenguaje JavaScript y HTML5.				NR12
Uso del framework jQuery.				NR13
Explicación	La Reusabilidad es la capacidad de diseñar un sistema de manera que su estructura o componentes puedan ser utilizados en futuras aplicaciones. Con el uso del framework jQuery y el lenguaje de programación JavaScript, el componente permite que el código de sus clases puedan ser reutilizadas, debido a que constituyen lenguajes definidos para el desarrollo en el proyecto. Integrándose a estos el lenguaje HTML5 definido para el desarrollo del componente, el cual permite con la etiqueta Canvas definir un área de dibujo vectorial. Por tanto si se requiere de la creación de un nuevo componente laboratorios virtuales, no es necesario comenzar su desarrollo desde cero, pues el anterior deberá estar implementado con estos lenguajes y su código podrá ser reutilizado.			

La evaluación de la arquitectura demostró que la utilización de los elementos anteriormente detallados da cumplimiento a atributos de calidad como rendimiento, escalabilidad y reusabilidad sobre el componente Laboratorio virtuales. Con el uso de HTML5 y del *framework* Three no es necesario el uso de algún *plugin* para la visualización del componente, el mismo solo se apoyará del navegador, por lo que se logra un alto rendimiento. La utilización de jQuery, JavaScript y otros elementos facilitan la compatibilidad y la portabilidad. La modificabilidad es lograda a través del patrón Modelo Vista Controlador.

Una arquitectura para el desarrollo de un sistema resulta adecuada, cuando el sistema cumple con los atributos de calidad definidos sobre el mismo y cuando el sistema pueda ser construido con los recursos disponibles. Por lo anteriormente planteado se puede afirmar que el diseño de la arquitectura propuesta es adecuado para el desarrollo del componente Laboratorios virtuales, ya que cumple con los atributos de calidad más significativos identificados y puede ser construido con los recursos y condiciones creadas en el proyecto.

3.7 Conclusiones

En este capítulo se realizó la evaluación de la arquitectura, lo que permitió a través de la técnica escenario y el método de evaluación ATAM identificar, dos riesgos y trece no riesgos. Demostrándose así que el componente laboratorios virtuales cumple con los atributos de calidad definidos para el mismo y que la arquitectura propuesta es factible para su construcción.

Conclusiones Generales

Cuando se define una correcta arquitectura de software se garantiza que el sistema sea desarrollado con el mayor éxito posible. Por lo que resulta de gran importancia que todo sistema tenga definida una arquitectura sólida que sea capaz de organizar el desarrollo del sistema, fomentar la reutilización, escalabilidad y entendimiento del mismo.

Con la culminación del presente trabajo de diploma se concluye que la arquitectura propuesta para el desarrollo del componente Laboratorio virtuales usando HTML5 es factible. Para ello se logró la identificación y validación de dicha propuesta de arquitectura mediante el estudio de los puntos más importantes dentro de la Arquitectura de software, así como el análisis y selección de las herramientas y tecnologías más adecuadas. Realizándose posteriormente la estructura del componente para una mayor comprensión de la organización de los diferentes elementos del mismo, Con la utilización de la técnica de evaluación basada en escenarios y el método de evaluación ATAM se lograron identificar puntos de riesgos, además de comprobar que la arquitectura soluciona los problemas planteados en la situación problémica.

Recomendaciones

Con la culminación del presente trabajo de diploma se proponen las siguientes recomendaciones como parte del proceso de mejoras continuas:

- Desarrollar el componente Laboratorios virtuales a partir de la arquitectura propuesta.
- Durante el desarrollo del componente continuar con el refinamiento constante de la arquitectura propuesta.
- Una vez implementado el componente realizar la evaluación tardía de la arquitectura propuesta para la identificación de riesgos y debilidades que pueda presentar.

Referencias Bibliográficas

1. **Pressman, Roger S.** *Ingeniería del Software: Un enfoque práctico.* 2002.

2. **Camacho, Erica, Cardeso, Fabio y Nuñez, Gabriel.** *Arquitectura de Software: Guía de Estudio.* Buenos Aires : s.n., 2004.

3. **Reynoso, Carlos Billy.** *Introducción a la Arquitectura de Software.* Buenos Aires : s.n., 2004.

4. **Rosado, L. y J.R, Herreros.** *Nuevas aportaciones didácticas de los laboratorios virtuales y remotos en la enseñanza de la física.* Madrid : s.n., 2005.

5. **Pastor, Javier.** mycomputer. [En línea] 22 de Enero de 2010. [Citado el: 17 de Enero de 2012.] http://www.muycomputer.com/2010/01/22/actualidadnoticiasvimeo-tambien-con-html5_we9erk2xxddkoq50oc22itrxaaycsnpbjersv09-l3iocwtff6ydfrzm6zcihzju.

6. **Sánchez, Darío.** Redusers. *Top 10: juegos desarrollados en HTML5 .* [En línea] 12 de Diciembre de 2011. [Citado el: 16 de Enero de 2012.] http://www.redusers.com/noticias/top-10-juegos-desarrollados-en-html5.

7. **Briceño, Eduardo.** 8 juegos geniales desarrollados en HTML5. *bitelia.* [En línea] 25 de Febrero de 2012. [Citado el: 20 de Enero de 2012.] http://bitelia.com/2012/02/8-juegos-geniales-desarrollados-en-html5.

8. **Antonijoan, Marc.** Media Technologies. [En línea] [Citado el: 25 de Enero de 2012.] http://blogs.salleurl.edu/dtm-media-technology/3d-en-la-web-%C2%BFque-tecnologia-usar/.

9. **Penadés, Patricio Letier y M, Carmen.** *Metodologías ágiles para el desarrollo de software.* Valencia : s.n.

10. **Joskowicz, Ing. José.** *Reglas y Prácticas en eXtreme Programing.* 2008.

11. **Mendoza, María A.** *Metodologías De Desarrollo De Software.* 2004.

12 **Palacio, Juan.** *El modelo Scrum.* 2006.

13. **Martínez, Alejandro y Raúl, Martínez.** *Guía a Rational Unified Process.*

14. **Álvarez, Miguel Ángel.** Manual de Canvas del HTML5. *Desarrolloweb.* [En línea] 27 de Octubre de 2011. http://www.desarrolloweb.com.

15. **Pérez, Javier Aguíluz.** *Introducción a JavaScript.* 2009.

16. **Morales, Hugo Humberto y Mejía, Lucia.** *Desarrollo de un prototipo web para las etapas de transformación de coordenadas y proyección de animación 3D, de una librería gráfica basada en la especificación estándar WEBGL y HTML5.* 2011.

17. **Álvares, Miguel Ángel.** Manual de jQuery. *Desarrolloweb.* [En línea] 30 de Septiembre de 2010. [Citado el: 16 de Enero de 2011.] http://www.desarrolloweb.com/manuales/manual-jquery.html.

18. **Pérez, Javier Eguíluz.** *Introducción a AJAX.* 2008.

19. **Utreras, Victor.** Slideshare. *Introducción a JQuery.* [En línea] 2010. [Citado el: 16 de Enero de 2012.] http://www.slideshare.net/continuumslides/introduccin-a-jquery.

20. **Reynoso, Carlos y Kicillof, Nicolás.** *Estilos y Patrones en la Estrategía de Arquitectura de Microsoft.* Buenos Aires : s.n., 2004.

21. **Gómez, Omar S.** *Evaluando la arquitectura de software.*

22. **Bosch, Jan.** *Design and Use of Industrial Software Architectures.* 1999.

23. **Medina, Cesar Julio Bustacara.** *Evaluación de Arquitecturas de Software.* 2008.

24. **Belmonte, Nicolás.** *Sencha.* [En línea] 31 de Diciembre de 1969. [Citado el: 18 de Abril de 2012.] http://www.sencha.com/blog/introducing-philogl-a-webgl-javascript-library-from-sencha-labs/.

25. **Benedetto, Marco Di.** *SpiderGL: 3D Graphics for Next-Generation www.*

26. **Grimán, Anna, Pérez, María y Mendoza, Luis.** *Estudio de la influencia de mecanismos arquitectónicos en la calidad del software.*

27. **Montalvo, Marlene Melián.** *XML el nuevo lenguaje universal*

28. **Corporation, Microsoft.** *La Arquitectura Orientada a Servicios (SOA) de Microsoft aplicada al mundo real.* 2006.

29. **Dávila, José Alberto Vela.** *Ambientes de Desarrollo de Software Basado en Componentes.*

30. **Rodríguez, Txema.** Cinco proyectos JavaScript que deberías conocer y con los que deberías ponerte a trastear. *Genbetadev.* [En línea] 31 de Agosto de 2011. [Citado el: 18 de Abril de 2012.] http://www.genbetadev.com/javascript/cinco-proyectos-javascript-que-deberias-conocer-y-con-los-que-deberia-ponerte-a-trastear-ya.

Bibliografía Consultada

jQuery. [En línea] [Citado el: 18 de Diciembre de 2011.] http://jquey.com.

Jimmy. Tetris en 3D hecho en HTML5 con Canvas. *Html5 Fácil.* [En línea] 8 de SSeptiembre de 2011. [Citado el: 18 de Enero de 2012.] http://html5facil.com/informacion/tetris-en-3d-hecho-en-html5-con-canvas.

Visualizar datos con el framework WebGL PhiloGL. *0x27.* [En línea] [Citado el: 18 de Abril de 2012.] http://0x27.com.ar/2012/01/visualizar-datos-con-el-framework-webgl-philogl/.

Lorite, Juan Francisco Adame. *Características objeto-relacional y de soporte de documentos XML de Oracle Database.* 2003.

González, Carlos Samuel Cantero. *Web-based Graphics Library.*

Romero, Alfredo Reino. *INTRODUCCIÓN A XML EN CASTELLANO.* 2000.

Puebla, Yoan Arlet Carrascoso, Gómez, Enrique Chaviano y Vega, Anisleydi Céspedes. Procedimiento para la evaluación de arquitecturas de software basadas en componentes. *GestioPolis.* [En línea] 7 de Septiembre de 2009. [Citado el: 5 de Abril de 2012.] http://www.gestiopolis.com/administracion-estrategia/procedimiento-para-la-evolucion-de-las-arquitecturas-de-software.htm.

Puigdemunt, Eduard. Curso de XML. *Programación en Castellano.* [En línea] [Citado el: 13 de Abril de 2012.] http://www.programacion.com/articulo/curso_de_xml_164.

Parfeni, Lucian. Google demestra las ventajas de WebGL con bellas visualizaciones 3D para el volumen de las busquedas. *Softpedia.* [En línea] 6 de Mayo de 2011. [Citado el: 16 de Abril de 2012.] http://news.softpedia.es/Google-demuestra-las-ventajas-de-WebGL-con-bellas-visualizaciones-3D-para-el-volumen-de-las-busquedas-198802.html.

Reynoso, Carlos y Kicillof, Nicolás. *Lenguajes de Descripción de Arquitectura (ADL).* Buenos Aires : s.n., 2004.

Three.js. *Three.js.* [En línea] [Citado el: 28 de Abril de 2012.] http://mrdoob.github.com/three.js/.

ULTRAVISUAL. Three.js and webGL. *ULTRAVISUAL.* [En línea] 31 de Marzo de 2011. [Citado el: 29 de Abril de 2012.] http://ultravisual.co.uk/blog/2011/03/31/three-js-and-webgl.

HTML5Fácil. *TRabajando con Three.js.* [En línea] 20 de Abril de 2012. [Citado el: 2 de Mayo de 2012.] http://html5facil.com/tips/trabajando-con-three-js.

3D sin *plugins* en tu navegador con javascript: three.js. *Opsou.* [En línea] [Citado el: 29 de Abril de 2012.] http://www.opsou.com/blog/3d-sin-*plugins*-en-tu-navegador-con-javascript-three-js/.

Garrido, Jesús Manuel Montero. *Plataforma Eclipse Introducción Técnica.*

Msc. Caicedo, Isaac y otros y otros. *ARQUITECTURAS DE SOFTWARE.* Colombia : s.n.

Pérez, Javier Eguíluz. *Introducción a CSS.*

Belmonte, Nicolas Garcia. PhiloGL. *Senchalabs.* [En línea] [Citado el: 18 de Abril de 2012.] http://www.senchalabs.org/philogl/.

Briceño, Eduardo. 8 juegos geniales desarrollados en HTML5. *bitelia.* [En línea] 25 de Febrero de 2012. [Citado el: 20 de Enero de 2012.] http://bitelia.com/2012/02/8-juegos-geniales-desarrollados-en-html5.

Noguera, Bulmaro. Culturacion. [En línea] 4 de Julio de 2011. [Citado el: 15 de Enero de 2012.] http://culturacion.com/2011/07/ventajas-de-utilizar-html5.

Marquina, Ernesto y Parra, Jose David. *Guía de Patrones, Prácticas y Arquitectura .NET.* 2008.

Larman, Craig. *UML y Patrones: Introducción al análisis y diseño orientado a objetos.* 1999. Estándar: ISBN/970-17-0261-1.

Plasmando UML en Javascritp Parte I. [En línea] 22 de Septiembre de 2011. [Citado el: 7 de Mayo de 2012.] http://coderwar.com/2011/09/plasmando-uml-en-javascrit-parte-i/.

González, Damaris Batista y Acosta, Isael Bozán. *Desarrollo de una aplicación para la automatización de la evaluación de la Arquitectura de Software en los Proyectos Productivos de la UCI.* La Habana : s.n., 2010.

stackoverflow. SceneJS vs Three.JS vs others. *stackoverflow.* [En línea] [Citado el: 8 de Mayo de 2012.] http://stackoverflow.com/questions/6762726/scenejs-vs-three-js-vs-others.

Montilva, Jonás A, Arpé, Nelson y Colmenares, Juan Andrés. *Desarrollo de Software Basado en Componentes.* 2003.

Bibliografía Consultada

Amalgamas. *Amalgamas.* [En línea] 9 de Noviembre de 2004. [Citado el: 14 de Marzo de 2012.] http://homepage.mac.com/imaz/iblog/C612772037/E20050907222635/Media/Algunos%20Tipos%20de%20Arquitecturas.pdf.

Grimán, Anna y otros y otros. *TOWARDS A MAINTAINABILITY EVALUATION IN SOFTWARE ARCHITECTURES.* Caracas : Universidad Simón Bolívar.

Delgado, Andrea, Castro, Alberto y Germán, Martín. *Evaluación de Arquitecturas de Software con ATAM (Architecture Tradeoff Analysis Method): un caso de estudio.*

Kazman, Rick, Nord, Robert L. y Klein, Mark. *A Life-Cycle View of Architecture Analysis and Design Methods.* 2003. CMU/SEI-2003-TN-026.

IEEE. *IEEE Std. 1471-2000, IEEE Recommended Practice for Architectural Description of Software Systems.*

Suárez, Jose Manuel Sánchez. Introducción a JSON. *Atentia.* [En línea] 2008-07-22, 22 de Julio de 2008. [Citado el: 8 de Mayo de 2012.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=prototypejsAjaxJSON.

Villate, Jaime E. *Introducción al XML.* 2001.

Puigdemunt, Eduard. Curso de XML. *Programación en Castellano.* [En línea] [Citado el: 16 de Abril de 2012.] http://www.programacion.com/articulo/curso_de_xml_164.

Fernández, Víctor Fresno. *Tema 3. Lenguajes de marcado.*

Shaw, Mary y Clements, Paul. *A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems.* 1996.

Faúndez, Juan José González. *Análisis de las plataformas de redes sociales actuales para obtener patrones arquitectónicos comunes y rescatar las mejores prácticas.* Santiago de Chile : s.n., 2011.

Vega, John Freddy y Henst, Christian Van Der. *Guía HTML5.* 2011.

Pressman, Roger S. *Ingeniería del Software: Un enfoque práctico.* 2002.

Reynoso, Carlos y Kicillof, Nicolás. *Estilos y Patrones en la Estrategía de Arquitectura de Microsoft.* Buenos Aires : s.n., 2004.

Gómez, Omar S. *Evaluando la arquitectura de software.*

Glosario de Términos

Arquitectura de software: es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

Estilo Arquitectónico: Forma de organización arquitectónica que se expresa de forma formal y teórica, abarca un conjunto de componentes, conectores y un conjunto de restricciones de cómo pueden relacionarse entre ellos.

Patrón Arquitectónico: Expresan esquemas de organización estructural fundamentales para los sistemas de software.

Servicio Web: Sistema de software diseñado para soportar la interacción que existe máquina a máquina sobre una red. Posee una interfaz descripta en un formato procesable por máquina.

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

Requerimiento: Capacidades o características que debe tener el sistema o modelo desarrollo para satisfacer las necesidades del cliente.

Escenario: Descripción de la interacción de los stakeholders con el sistema.

Stakeholders: Cualquier involucrado en el desarrollo, desde clientes, desarrolladores, gerentes entre otros.

Portabilidad: Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación.

Modificabilidad: Es la habilidad de realizar cambios futuros al sistema.

Árbol de Utilidad: Esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican el nivel de prioridad de cada uno.

***Plugins*:** Aplicación informática que añade una característica o servicio específico a un sistema.