Propuesta de arquitectura basada en componentes para la producción de software con tecnologías multimedia para el centro FORTES.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS.

Autoras: Aimara Lores Samón.

Ana Leydis Tejeda Licea.

Tutores: Ing. Nilber Barbán Góngora.

Ing. Arlan Galvez Alonso.

La Habana, Junio del 2014

"Año 56 del Triunfo de la Revolución."

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes _____ del año 2014.

_____ _____

Aimara Lores Samón — Ana Leydis Tejeda Licea

Firma del Autor — Firma del Autor

_____ _____

Ing. Arlan Galvez Alonso — Ing. Nilber Barbán Góngora

Firma del Tutor — Firma del Tutor

Los hombres son como los astros, unos dan luz de sí y otros brillan con la que reciben.

José Martí.

Aimara

Le agradezco a una persona que siempre estuvo ahí cada vez que hacía falta, que soportaba mis malcriadeces, mis enojos y mis berrinches, aunque siempre me amenazaba con su brazo... Gracias por apoyarme siempre y darme tu amor, Ana Rosa.

Le agradezco a mi padre por todo lo que él ha hecho por mí y por todo el amor y cariño que me ha brindado... Gracias por tus atenciones y dedicación, Reinaldo.

Le agradezco a mi querida y amada abuela que ha luchado conmigo desde que nací y me dio todo de ella... Gracias abu por tu confianza y ternura, Gina.

Le agradezco a mi hermana que aunque seamos el aceite y vinagre yo siempre te querré... Gracias por todo lo que hiciste por mí, Mila.

Le agradezco a dos personas que llegaron un poco tarde a mi vida, pero también pusieron su granito de arena y espero que siempre lo hagan... Gracias por su amor y dedicación, Fino y Yamilka.

Le agradezco a mi compañera de tesis por haberme ayudado y apoyado en todo que sin ella este sueño no se hubiera hecho realidad... Gracias por ser más que una compañera sino también una amiga, Ana Leydis.

Le agradezco a mis tutores por ayudarnos y dedicarnos siempre un tiempo para nosotras... Gracias por sus atenciones, Arlan y Nilber.

Le agradezco a ZayPomo y a Lalo por su amistad, sus consejos y su ayuda... Gracias por ser las mejores del mundo las quiero, Zayli y Lizandra·

Le agradezco a vocecita, murici y a pelo suelto por estar ahí siempre y por su amistad... Gracias por todo el cariño que me brindaron, Rosalia, Dayneli y Gretel·

Le agradezco a todas mis amistades por apoyarme... Gracias por su tiempo, Michelle, Alberto, Elena, Mandy, Rosalia Martínez, Mena en fin a todos·

Ana Leydis

Le agradezco a Dios por permitirme avanzar en cada paso de mi vida·

Le agradezco a toda mi familia especialmente a mi hermano Rafael, mi mamá Gladis Elena y a mis abuelos paternos Arsenio Javier y Rafaela que me brindaron todo el apoyo, amor y abrigo·

Les agradezco a todos mis seres queridos que me han acompañado y que todavía siguen siendo parte de mi vida·

Agradezco a los tutores y consultantes Arlan, Nilber, Lizandra y Mayniuris por la dedicación e interés a nuestro trabajo investigativo· A mi compañera de tesis por la paciencia y dedicación que tuvo realizando nuestro trabajo investigativo, cuando yo no podía estar ella asumía cada tarea con valor y coraje·

A mis amistades, que me quieren, que no me dejan y que comparten, lo mismo los momentos malos y buenos· Le agradezco a Alejandro que me ha brindado tranquilidad, amor, ternura y me ha ayudado a convertirme en una mejor persona estos últimos años·

Aimara

Le dedico este trabajo de diploma a mi madre Ana Rosa, a mi padre Reinaldo, a mi querida abuela, Gina.

A mi adora hermana Milagros, a mis hermanos Fino, Chachi y a la mujer de mi hermano, Yamilka.

A todas mis amistades, conocidos y enemigos.

A toda mi familia por ser la mejor del mundo los quiero.

Ana Leydis

Les dedico este trabajo de diploma a Gladis Elena Licea Jiménez, Rafael Tejeda Licea,

Javier Arsenio Tejeda González,

Rafaela Sánchez Téllez y a mi papá.

Resumen

Propuesta de arquitectura basada en componentes para la producción de software con tecnología multimedia para el centro FORTES.

El Centro de Tecnología para la Formación (FORTES), perteneciente a la Universidad de las Ciencias Informáticas presenta deficiencias en el desarrollo de productos con tecnología multimedia debido a que no cuenta con una estructura adecuada que guie el proceso de elaboración de productos multimedia, de forma tal que sus componentes puedan ser reutilizados. Todo ello provoca un aumento en los gastos productivos y la desorganización en el proceso de desarrollo, influyendo en la ralentización de la entrega del software al cliente. El objetivo que persigue la presente investigación es diseñar una arquitectura basada en componentes para estandarizar los productos multimedia en el centro FORTES. Para la confección de la propuesta se realiza un estudio de los conceptos y elementos de la arquitectura de software, que tienden a tener los productos multimedia. Se seleccionan los atributos de calidad que deben tener estas tecnologías. Se escoge como patrón arquitectónico: Modelo Vista Controlador y el estilo llamada y retorno. Como metodología de desarrollo para la implementación del software se selecciona la metodología ágil SXP. Se utiliza para el proceso de desarrollo de aplicaciones con estas características, la herramienta de modelado Visual Paradigm y el lenguaje de modelado para aplicaciones educativas ApEM-L. Al finalizar, se prueba la arquitectura, donde se utiliza el método ATAM (Método de Análisis de Acuerdos de Arquitectura), con el objetivo de identificar los riesgos y fortalezas de la propuesta. Se obtiene el diseño de una arquitectura basada en componentes que sirve como base para el desarrollo de productos con tecnologías multimedia.

Palabras clave: arquitectura de software, centro FORTES, componentes, estructura, multimedia.

Contenido

Figuras

Tablas

Introducción

Cuba ha estado inmersa en un proceso de transformación educacional y social, en el contexto del programa Batalla de Ideas. Uno de esos proyectos a transformar ha sido la Universidad de las Ciencias Informáticas (UCI), que tiene como misión formar profesionales comprometidos con la patria y altamente calificados en la rama de las Ciencias Informáticas. La Universidad produce aplicaciones y servicios informáticos, a partir de la vinculación estudio-trabajo como un modelo de formación. Este modelo de formación utiliza las herramientas y Tecnologías de la Información y Comunicación (TIC) (1).

El centro FORTES perteneciente a la Facultad 4 de la Universidad, desarrolla tecnologías que permiten ofrecer servicios y productos para la implementación de soluciones de formación. El Centro cuenta con una gran actividad productiva que ha brindado servicios a algunas instituciones y empresas del país. Antes que se creara el Centro, existía un grupo de profesionales de la antigua Facultad 8, encargados de producir software atendiendo a diferentes polos. En el año 2010 el polo de Software Educativo (SWE) se convirtió en el Centro de Tecnologías para la Formación (FORTES). El desarrollo de productos multimedia se incorporó al Departamento de Materiales Educativos. En el año 2012, el Centro se reestructura y se distribuye las Líneas de Producción de Software (LPS). La creación de productos en una línea de producción de software exige que tengan características similares en su estructura. Por tanto la elaboración de una arquitectura es uno de los pasos más importantes (2).

Para una mejor organización estructural del Centro se crearon tres departamentos: Uno dedicado a la formación mediante la Práctica Profesional, otro para administrar la creación de los componentes de software y otro para el ensamblaje de estos componentes en diversas aplicaciones.

En el 2013 el centro FORTES, debido a la restructuración, crea cuatro LPS: Ambientes Integrados de Aprendizaje, Juegos y Simuladores Educativos, Recursos Educativos y Entornos Virtuales de Aprendizaje (enfocados en Moodle, principalmente). La distribución de las nuevas LPS constituye una ventaja y un paso de avance para la organización de los proyectos productivos dentro del Departamento de Componentes. Los componentes reutilizables se crean con el objetivo de ser ensamblados y así contribuir a la efectividad del proceso dentro de las LPS. Dicho departamento es el encargado de crear los componentes reutilizables con el objetivo de ser re-ensamblados en nuevas soluciones y así contribuir a la efectividad del proceso productivo con la calidad requerida.

El Departamento de Componentes cuenta, desde su formación con el proyecto “Producción de Recursos Educativos”, encargado de crear componentes reutilizables para ensamblar y desarrollar productos con tecnología multimedia. Dichos productos están enfocados principalmente al ámbito educativo, pero también se crean otros promocionales e informativos a solicitud de los clientes que se acercan al Centro en búsqueda de este tipo de soluciones.

Como resultado del proceso de desarrollo de productos con tecnología multimedia, se han ido elaborando diferentes componentes asociados a ciertas funcionalidades dentro de estos productos, esto permite que puedan ser reutilizados en otros con características similares. La reutilización de los componentes dentro del desarrollo de nuevos productos, no provee de una estructura adecuada, de tal manera que se comuniquen correctamente entre ellos. Tampoco se cuenta con una guía que centre el proceso de elaboración de productos en estas condiciones y con estas características. Todo ello causa el aumento de los gastos productivos y la desorganización en el proceso de desarrollo, influyendo en la ralentización de la entrega del software al cliente.

La situación anteriormente expuesta conlleva al siguiente **problema a resolver:** ¿Cómo estandarizar la estructura de los productos con tecnología multimedia en el centro FORTES?

Se propone como **objetivo general** diseñar una arquitectura basada en componentes para estandarizar la estructura de los productos con tecnología multimedia en el centro FORTES.

El objeto de estudio de la investigación es la arquitectura basada en componentes.

El campo de acción de la investigación está enmarcado en la arquitectura basada en componentes para la producción de software con tecnologías multimedia.

Del objetivo general se desglosan los siguientes **objetivos específicos**:

- Elaborar el marco teórico, que permita desarrollar el diseño de una arquitectura basada en componentes, para estandarizar la estructura de los productos multimedia en el centro FORTES.
- Diseñar la propuesta de arquitectura basada en componentes con tecnología multimedia.
- Probar la propuesta de arquitectura.

Para dar cumplimiento a los objetivos específicos y al objetivo general se han propuesto las siguientes **tareas de investigación**:

1. Análisis de los conceptos relacionados con la arquitectura basada en componentes y estudio de las existentes propuestas de arquitecturas en el mundo, Cuba y la UCI.
2. Identificación de los patrones y estilos arquitectónicos para los productos multimedia.
3. Generación de los artefactos necesarios para dar soporte a la propuesta de arquitectura.
4. Diseño del prototipo no funcional.
5. Análisis de los métodos existentes para la evaluación del prototipo.
6. Identificación y aplicación de los métodos de evaluación para la arquitectura de software propuesta.

Como **idea a defender** se plantea: el diseño de una arquitectura basada en componentes, permite estandarizar la estructura de los productos con tecnología multimedia en el proceso de desarrollo de este tipo dentro del centro FORTES

Para el cumplimiento de los **objetivos específicos** se utilizaron los **métodos científicos** que se describen a continuación.

Métodos teóricos

Analítico-sintético: Se descompone por partes los diferentes conceptos y definiciones que acapara una arquitectura de software y se hace una síntesis a partir de la bibliografía consultada acerca de los estilos, patrones, metodologías y herramientas de desarrollo.

Histórico-lógico: Este método es importante para conocer la evolución de la arquitectura de software para así ver los cambios que han venido ocurriendo desde su creación. Cómo se permite conocer el origen del Centro y la línea de producción de software al que se realiza la arquitectura.

Modelación: Este método es importante ya que a través de él se crean abstracciones, con el objetivo de explicar la realidad. Se utiliza en la creación de los diagramas de componentes, en las vistas del sistema, vista de proceso, vista lógica, vista de datos, vista de integración y la modelación de las interfaces de usuario.

Métodos Empíricos

El método empírico que se utiliza es la **observación,** durante toda fase del proceso de desarrollo de software, para ver los comportamientos principales del fenómeno**.** **Ver Anexo I.** Las técnicas que se utilizan para la recopilación de la información son **entrevista** y **cuestionario**, ya que se realizan reuniones con especialistas que tienen conocimiento sobre el tema y se formulan preguntas para obtener información acerca de la arquitectura de software para productos multimedia**.**

El presente documento se estructura en tres capítulos, cada uno con su introducción y conclusiones:

Capítulo 1: Fundamentación Teórica, en este capítulo se define el marco teórico de la investigación, se realiza un estudio del estado del arte de la arquitectura de software y conceptos relacionados con la misma. Se describen los patrones y los estilos arquitectónicos utilizados en la actualidad, así como metodologías de desarrollo, lenguajes de modelado, herramientas y tecnología a utilizar. Además, se realiza la selección de la metodología de desarrollo, como las herramientas y lenguajes de implementación y modelado de la propuesta.

Capítulo 2: Propuesta de arquitectura, en este capítulo se realiza la propuesta de solución al problema planteado en el diseño teórico de la investigación. Se establece la estructura arquitectónica, los lineamientos de diseño de la implementación, se definen los atributos de calidad de la propuesta y se implementa el prototipo no funcional de la arquitectura.

Capítulo 3: Evaluación de la arquitectura, en este capítulo se expone los principales métodos utilizados para evaluar la arquitectura, y se selecciona entre ellos el más óptimo de acuerdo a la propuesta.

Al terminar este trabajo se espera como **posibles resultados** obtener una arquitectura basada en componentes para estandarizar los productos multimedia en el centro FORTES.

Capítulo 1. Fundamentación Teórica

1.1. Introducción

La Arquitectura de Software (AS) constituye dentro de la fase de diseño, el diseño arquitectónico del software. En la actualidad se compone de numerosos conceptos que generan relevancia en la industria de productos informáticos. En este capítulo se realiza un estudio del estado del arte de la AS y definiciones relacionadas con la misma. Se describen los patrones y los estilos arquitectónicos utilizados, así como las metodologías de desarrollo, lenguajes de modelado, herramientas y tecnologías propuestas para el proceso de desarrollo.

1.2. Arquitectura de software

Hoy en día se definen diferentes conceptos sobre la arquitectura de software y los criterios planteados por los autores que han indagado acerca del tema. El Ingeniero de Software estadounidense reconocido internacionalmente y especialista en mejoras de procesos de software y en tecnologías: Roger Pressman, y el profesor de Ingeniería de Software en la Universidad de St Andrews Ian Somerville, que es uno de los padres de la Ingeniería de Software, han contribuido a publicar diferentes ejemplares que han servido como guía a los profesionales que estudian la Ingeniería de Software. Una de las primeras fases de desarrollo del software después del análisis de los requerimientos funcionales y no funcionales de un producto, es el diseño. En este se define la arquitectura con que va a contar el software. Por los autores anteriores se definen los siguientes conceptos:

Pressman en el año 2002 plantea: *"La Arquitectura del Software de un programa o sistema de cómputo es la estructura o las estructuras del sistema, que incluyen los componentes del software, las propiedades visibles externamente de esos componentes y las relaciones entre ellos"* (3).

Sommerville en el año 2005 plantea: *"…Un sistema está compuesto por subsistemas que a su vez estos proporcionan un conjunto de servicios relacionados. El proceso de diseño inicial que identifica estos subsistemas y que establece un marco para el control y comunicación de los subsistemas se llama diseño arquitectónico. El resultado de este proceso de diseño es una descripción de la arquitectura del software…" y"… se refiere a la estructura a grandes rasgos del sistema, estructura consistente en componentes y relaciones entre ellos…"* (4).

"La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución" (5).

Luego de las definiciones anteriormente planteadas se concluye que se toma el concepto planteado por el Ingeniero de Software Roger Pressman como base para la investigación, dado a la amplitud e integridad de la definición de la AS.

Los componentes son entidades, módulos de software, objetos y clases que residen en el interior de la arquitectura de software, estos deben comunicarse y colaborar con otros componentes externos (sistemas, dispositivos, personas) al software (5).

1.2.1-Arquitectura Basada en Componentes

La arquitectura basada en componentes consiste en una rama de la ingeniería de software en la cual se trata con énfasis la descomposición del software en componentes funcionales. Esta descomposición permite convertir componentes pre-existentes en piezas más grandes del software (6).
Este proceso de construcción de una pieza de software con componentes ya existentes, da origen al principio de reutilización del software, mediante el cual se promueve que los componentes sean implementados de una forma que permita su utilización funcional sobre diferentes sistemas en el futuro.

1.2.2-Pilares de la Arquitectura de Software

Los pilares de la Arquitectura de Software son una parte fundamental de la misma, puesto que alrededor de ella gira el estudio de la arquitectura de software. Estos pilares se entienden como las cuatro C de la arquitectura de software: componentes (*Components*), conectores (*Connectors*), configuraciones (*Configurations*) y restricciones *(Constraints).*

A continuación se describirán estos pilares:

Componentes: bloques de construcción que integran las partes de un sistema de software. Se pueden representar a nivel de lenguaje de programación como módulos, clases, objetos o un conjunto de funciones relacionadas. Se clasifican en tres tipos de elementos:

- Elementos de datos: contienen la información que será modificada.
- Elementos de procesos o procesamiento: transforman los elementos de datos.

- Elementos de conexión: estos pueden ser elementos de datos o procesos, los cuales mantienen unidas las diferentes piezas de la arquitectura.

La relación entre los componentes se define como una abstracción, que interactúan en el sistema a través de los elementos de conexión. Por lo tanto, los conectores se concretan en relaciones entre los componentes.

Las propiedades visibles son servicios que los componentes proveen, características de desempeño, manejos de fallas, uso de recursos compartidos, entre otros (5).

Conectores: un conector no es más que el elemento que relaciona cada componente dentro de la arquitectura, permitiendo la interacción entre ellos. Los conectores requieren de una infraestructura para realizar su función durante la ejecución. Además, los mismos promueven la reutilización compleja de las abstracciones. Muestran claramente las intenciones que tiene el arquitecto.

Tipos de conectores: pueden ser llamadas a procedimientos entre cliente y servidor o entre dos objetos. Además, se encuentran los conectores que permiten que los componentes se comuniquen mediante publicación-suscripción. Los que mandan mensajes asincrónicos y tubos que representan datos asincrónicos que preservan el orden. Estos conectores son los más sencillos. Los conectores más complejos son: los canales de comunicación orientada a las transacciones entre un servidor de base de datos y un cliente.

De forma general se define el tipo de interacción mediante un protocolo, esto puede generar patrones de eventos o acciones (7).

Configuraciones: constituyen grafos de componentes y conectores (8). Los valores de configuración son elementos del entorno de una aplicación o del entorno de alojamiento de aplicaciones que controlan el comportamiento de la aplicación así como el motor de tiempo de ejecución.

Existen varios tipos de configuraciones, dentro de las que se encuentran:

- Configuración personalizada: es un mecanismo que proporciona metadatos adicionales necesarios para modelar con precisión el entorno de desarrollo o implementación.
- Configuración en tiempo de ejecución: se implementa un sistema con un archivo de configuración de una aplicación que proporciona la información de versión del motor en tiempo de ejecución. El archivo de configuración es basado en el lenguaje XML, que crea el programador de la aplicación y que se suministra con este.

- ♣ Configuración compleja: es la que contiene más de un valor. Un ejemplo de configuración compleja incluye la configuración *ServerBinding* (Servidor de Encuadernación) (9).

Restricciones: Representan condiciones de diseño que deben acatarse incluso en el caso que el sistema evolucione en el tiempo (8). Son requisitos que establecen un valor de configuración. Se pueden crear restricciones contra opciones de configuración. Por ejemplo se establecen restricciones desde la capa de aplicación contra opciones de configuración en la capa que aloja las aplicaciones y viceversa (9).

Estos pilares son elementos fundamentales a tener en cuenta para la elaboración de una arquitectura, como también son importantes para su evaluación los atributos o requerimientos adicionales que debe tener un sistema. Estos atributos son denominados atributos de calidad o evaluación, ya que permiten según los estilos, patrones arquitectónicos y patrones de diseño contribuir al buen funcionamiento del sistema.

1.2.3-Componentes multimedia

Los productos con tecnología multimedia han tenido un impacto continuo en el proceso de enseñanza-aprendizaje. Estos combinan diversas medias como: videos, imágenes fijas, texto, animaciones y sonido que sirven como material añadido y no integrado en el proceso educativo. Los medios audiovisuales sirven como catalizador de experiencias o como dinamizador de la comunicación, son utilizados como material de apoyo en el proceso de enseñanza-aprendizaje (10) (11).

A partir de la bibliografía consultada se define los siguientes elementos, en un producto multimedia:

Texto: de acuerdo a González Cordero en el año 2001, *"puede escribirse en forma de párrafo o en columnas, recomendándose un tipo de letra sencilla y considerando además el color, tamaño y justificación"* (11).

Hipertexto: es una colección de palabras claves ligadas a información, permiten hacer vínculos o enlaces. Esto es según la navegabilidad del software informático.

Imagen: representación que se utiliza para expresar gráficamente algunas ideas.

Sonido: puede ser mensajes de voz, música y efectos.

Video: representa movimiento completo, compuesto por información binaria procesada por computadoras.

Animación: imágenes en movimiento utilizadas mayormente para dar una buena apariencia a los productos multimedia (11).

Otros elementos que son comúnmente utilizados e implementados previamente para integrar en el desarrollo de productos multimedia:

Reproductor de audio: es el medio donde se reproduce medios de audio (mp3). Contiene funciones básicas como: reproducir, pausar, detener, retroceder y avanzar.

Reproductor de video: es el medio donde se reproduce archivos en formato de video digital (flv, avi, mpg, webm). Contiene los controles: reproducir, pausa, detener, retroceder y avanzar.

Barras de desplazamiento: consta de una barra horizontal o vertical que tiene dos flechas en los extremos en sentido contrario, las cuales permiten el desplazamiento de la interfaz gráfica de un cuadro de texto. Esto sucede cuando existe mucha información en un área de texto (12).

Búsquedas o buscadores: son una forma sencilla de hallar un determinado tipo de información. Suelen aparecer como un área de texto donde se escribe el tipo de información que se desea encontrar.

Imprimir cambios en pantalla: este tipo de componente facilita la extracción de información que se visualiza en un producto multimedia.

1.2.4-Atributos de calidad

El proceso de diseño comienza con la construcción de un modelo de la arquitectura de software a partir de un conjunto de funcionalidades. El diseño preliminar obtenido es evaluado con respecto a ciertos atributos de calidad.

Barbacci en el año 1995 plantea que la calidad arquitectónica establece, que el desarrollo de forma sistemática para relacionar atributos de calidad de un sistema a su arquitectura, provee una base para la toma de decisiones objetivas sobre acuerdos de diseño y permite a los ingenieros realizar predicciones exactas sobre los atributos del sistema. Los atributos de calidad son importantes en el diseño de la arquitectura de software, ya que definen la calidad de un producto. Son requerimientos adicionales que deben satisfacer las características de un sistema; los cuales definen las propiedades de un servicio informático. Es importante esta relación porque cada decisión incorporada a una arquitectura de software puede afectar de forma impactante los atributos de calidad (5).

Los atributos de calidad se clasifican en dos formas:

- Son observables en vías de ejecución: se identifican aquellos atributos que se determinan en el comportamiento del sistema en tiempo de ejecución. **Ver Anexo 2**
- No observables en vías de ejecución: son los que se establecen durante el desarrollo del sistema. **Ver Anexo 3** (5).

1.2.5-Análisis de Arquitecturas similares

Para la propuesta de arquitectura basada en componentes, específicamente para tecnologías multimedia se realiza un estudio de diferentes arquitecturas similares. Por esa razón es necesario realizar un análisis de estas arquitecturas que existen a nivel nacional, internacional y en la universidad.

1.2.5.1-Propuesta de una arquitectura de software basada en componentes distribuidos para una célula CIM (*Computer Integrated Manufacturing*)

Esta propuesta se realiza con el objetivo de desarrollar un sistema de software para una célula CIM basada en una Arquitectura de Software para los componentes distribuidos CORBA.

Dentro de sus características se encuentran:

- Crear un software flexible ante las diferentes configuraciones físicas que puede adoptar una célula CIM.
- Fácilmente ampliable y transportable a distintos entornos industriales sin un coste excesivo de adaptación.
- Que los componentes tengan un nivel de reutilización alto que puedan definirse en una línea de productos dentro del campo de las células CIM.
- Conlleva a que el sistema tenga una capacidad inteligente y autónoma.
- Funcionamiento distribuido.
- Comportamiento en tiempo real (13).

1.2.5.2-Arquitectura de Sistemas de Información basados en Componentes sobre la Plataforma *Java 2 Enterprise Edition* (J2EE)

Esta arquitectura propone una correspondencia entre la arquitectura lógica en capas de un sistema de información creada independientemente de la tecnología, aplica una propuesta metodológica conocida, y las construcciones de la plataforma J2EE. Su objetivo es definir transformaciones entre los modelos independientes de la plataforma resultantes de la aplicación de la metodología mencionada.

Características

- Se utiliza la arquitectura por n capas o niveles de abstracción ya que es una arquitectura para sistemas de información.
- Se organiza según el estilo de capas.
- La cantidad de capas y la responsabilidad de cada una dependen del enfoque metodológico de desarrollo del sistema.
- Orientada a la especificación independiente de la tecnología de implementación de componentes como habitantes de algunas capas.
- Desarrollo basado en componentes.
- Utiliza la arquitectura basada en componentes.
- En el interior de cada capa utiliza el estilo cliente-servidor (14).

1.2.5.3-Propuesta de arquitectura basada en componentes para plataformas de Gestión de procesos desarrollados en Django

Esta propuesta desarrolla una arquitectura basada en componentes reutilizables, permite que las aplicaciones que se integren a la plataforma hagan uso de dichos componentes.

Características

- Es como un esqueleto que puede ser utilizado por cualquier sistema de gestión en que la aplicación necesite integrar algún componente nuevo o adicionar otras aplicaciones.
- La mayoría de los componentes son reutilizables de modo que puedan generar menús de servicios, recursos asociados por las aplicaciones (CSS (Estilo en Cascada),JS (Lenguaje de programación Java Script))
- Se implementa con el lenguaje de programación *Python* por ser dinámico, simple, orientado a objeto, proveer una sintaxis clara y legible y se caracteriza por la reutilización.
- Contiene los siguientes componentes reutilizables: *ContentProvider* (Proveedor de Contenido), *MenuAppProvider* (Proveedor de menú de servicio), *MenuProvider* (Provee generar el menú del portal), *ResourseProvider* (Genera recursos asociados a las aplicaciones), *PluginMount* (Punto de Montaje o Punto de Extensión para los *plugins* a generar) (15).

1.2.5.4-Propuesta de arquitectura de software para el producto "Mis Mejores Cuentos".

Esta propuesta de arquitectura se realiza con el objetivo de definir una arquitectura de software para el producto Mis Mejores Cuentos, estableciendo los elementos necesarios para el desarrollo de un producto educativo sobre plataforma web y con herramientas libres.

Características

- El funcionamiento del sistema dispone un servidor con el sistema operativo Linux y el Servidor Web Apache 2.2 instalado.
- Los usuarios del sistema con un navegador o browser.
- El sistema debe brindar servicio a cualquier usuario que tenga acceso al mismo.
- La información del sistema debe ser inédita o libre de derecho de autor.
- El sistema también será probado, instalado y configurado por los especialistas, que se ocuparán de su mantenimiento.
- El diseño de la interfaz debe ser sencillo y no estará sobrecargado de contenido, pero debe tener elementos visuales que capten la atención y que de forma metafórica represente una idea visual.
- El sistema se implementó con el lenguaje de programación JavaScript al utilizar la librería JQuery.
- Se utilizó como herramienta de modelado Visual Paradigm, como IDE de desarrollo Aptana (16).

1.2.5.5-Conclusiones del estudio de los sistemas similares

Después de realizar el estudio de las principales características arquitectónicas de los sistemas anteriormente planteados, se concluye que la propuesta de arquitectura debe ser fácilmente ampliable, y permitir que los productos sean transportables a distintas estaciones de trabajo. La arquitectura tiene que desarrollarse basada en componentes. Estos componentes deben ser reutilizables y que puedan definirse en los productos multimedia que se implementan. Para la implementación de la propuesta se utilizan componentes reutilizables creados con Hojas de Estilo en Cascada (CSS), Java Script (JS), además de la incorporación de páginas en formato HTML. Los usuarios que accedan a los sistemas implementados con los tipos de lenguajes anteriores, deben tener un navegador web para poder visualizar el contenido. El diseño de la interfaz debe ser sencillo y no estará sobrecargado de contenido, aunque debe contener elementos visuales que capten la atención. Además, estas soluciones similares no se pueden usar para el desarrollo de la arquitectura ya que las mismas tienen desventajas con respecto al diseño de la propuesta.

La desventaja que tienen estos sistemas es que se diseñan arquitecturas de software para un producto en específico, pero se escogen sus características más comunes que permitan servir de estrategia en el desarrollo de la propuesta. Otra desventaja es que la mayoría de los sistemas similares no son para productos multimedia, por lo que no tienen los mismos tipos de componentes y propiedades, aunque Mis Mejores Cuentos sí sea para tecnología de este tipo.

1.3. Estilos Arquitectónicos

El diseño arquitectónico empieza con el diseño de los datos y después procede a la derivación de una o más representaciones de la estructura arquitectónica del sistema. Los estilos arquitectónicos o patrones son analizados con el fin de obtener la estructura que mejor se ajuste a los requisitos del cliente y a las normas de calidad.

Según Reynoso en el año 2004 “Los estilos se manifiestan en una arquitectura teórica descriptiva con un alto nivel de abstracción” (8) . Estos se encuentran en el centro de la arquitectura.

Los estilos son entidades que ocurren en un nivel muy abstracto, puramente arquitectónico, que no coinciden con la fase de análisis (pero si coincide con la fase de diseño), ni con la fase de definición de paradigma de arquitectura, ni con los patrones arquitectónicos. De igual forma constituyen una representación abstracta que solapan los patrones de la arquitectura (8).

1.3.1-Tipos de estilos arquitectónicos

En la actualidad existen una gran variedad de estilos arquitectónicos de acuerdo con el diseño de la aplicación. Las familias de estilos más representativas hasta el momento son:

- Estilos de Flujo de Datos.
- Estilos Centrados en Datos.
- Estilos de Llamada y Retorno.
- Estilos de Código Móvil.
- Estilos Heterogéneos.
- Estilos Peer-to-Peer.

A continuación se expondrán las características y atributos de calidad que identifican estas familias de estilos.

1.3.1.1-Estilos de Flujo de Datos

Este tipo de estilos enfatiza la reutilización y modificación de los componentes arquitectónicos. Es apropiado para sistemas que implementan transformaciones de datos en pasos sucesivos. Dentro de esta familia de estilos se emplean las arquitecturas de tubería-filtros y las de proceso secuencial en lote (8).

Tubería y Filtros

Una tubería (*pipeline*) es una popular arquitectura que conecta componentes computacionales (filtros) a través de conectores *pipes*. Ha prevalecido el nombre de tubería-filtros por más que se sabe que los llamados filtros no realizan tareas de filtrado de manera forzosa, como podría ser eliminación de campos o registros, sino que ejecutan formas variables de transformación (8).

Los datos se transportan a través de las tuberías entre los filtros, donde transforma gradualmente las entradas en salidas. Dos de sus características son la simplicidad y facilidad para captar una funcionalidad.

El sistema tubería-filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes.

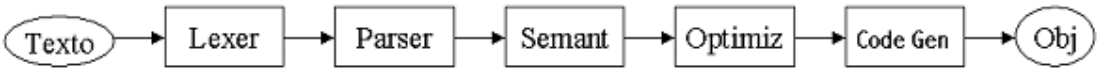


Fig. 1. Compilador en tubería y filtros (8).

Procesamiento secuencial en lote

En el estilo secuencial por lote (*Batch sequential*) los componentes son programas independientes. En los programas independientes cada paso que se ejecuta se completa antes que se inicie el paso siguiente. Según Garlan y Shaw la variante por lotes es un caso no generado del estilo, por eso las tuberías y filtros se han vuelto residuales (8).

1.3.1.2-Estilos Centrados en Datos

Esta familia de estilos enfatiza más en la integralidad de los datos. Es apropiada para sistemas que proveen el acceso y actualización de datos en estructuras de almacenamiento. Sub-estilos característicos son los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra.

Arquitecturas de Pizarra o Repositorio

En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre el mismo.

Estos sistemas integran dos subcategorías:

1. Si los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar, el repositorio puede ser una base de datos tradicional.
2. 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.

Estos sistemas se han usado en aplicaciones con complejas interpretaciones de proceso de señales (reconocimiento de voz, etc.) o en sistemas que involucran acceso compartido a datos con agentes débilmente acoplados (8).

1.3.1.3-Estilos de Llamada y Retorno

Esta familia de estilos se enfatiza en la modificación y escalabilidad de los elementos. Son los estilos más generalizados en sistemas en gran escala. Las arquitecturas que integran esta familia de estilos se encuentra la arquitectura de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.

Modelo Vista Controlador (MVC)

Reconocido como un estilo arquitectónico por Taylor y Medvidovic (17). Se creó antes que se generalizaran la arquitectura en capas múltiples. Se define más bien como una estrategia arquitectónica o como práctica recurrente.

Arquitectura por Capas

Según Garlan y Shaw en el año 1994 se define el estilo en capas como una organización jerárquica del sistema, tal que cada capa brinda servicio a la capa superior a ella, como también se sirve de las prestaciones que la capa inferior le provee, esto se hace de forma automática (18). Las capas pueden ser: entidades complejas, subsistemas o varios paquetes que la componen (8).

Arquitectura Orientada a Objetos

Este estilo está basado en objetos, organización de los objetos y abstracción de datos. Los componentes de este estilo son las instancias de los tipos de datos abstractos. Los componentes se basan en el encapsulamiento, herencia y polimorfismo (8).

Arquitecturas Basadas en Componentes

Se basa en elementos de software que marcan una unidad de composición y no de construcción, estos elementos llamados también componentes, se implementan antes de ser integrados en cualquier aplicación que se diseñe y se construya (8).

1.3.1.4-Estilos de Código Móvil

Esta familia de estilos enfatiza la portabilidad. Son utilizados para sistemas de intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando. La estructura arquitectónica que contiene este estilo es la arquitectura de máquinas virtuales. Tiene dos sub-estilos básicos, los intérpretes y los sistemas basados en reglas (8).

Arquitectura de Máquinas Virtuales son intérpretes que incluye un seudo-programa a interpretar y una máquina de interpretación. Se utiliza para la construcción de máquinas virtuales.

1.3.1.5-Estilos Heterogéneos

Este grupo de estilo incluye las formas compuestas o indóciles a la clasificación en las categorías habituales. Contiene dos estilos importantes: los sistemas de control de proceso y arquitecturas basadas en atributos.

Los sistemas de control de procesos se definen no solamente por los tipos de componentes, sino por las relaciones que mantienen entre ellos. La ventaja de este radica en su flexibilidad ante perturbaciones externas (8).

Las arquitecturas basadas en atributos tiene la intención de asociar a la definición del estilo arquitectónico un framework de razonamiento basados en modelos de atributos específicos (8).

1.3.1.6-Estilos Peer-to-Peer (Componentes Independientes)

Esta familia enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Los componentes son independientes y se comunican a través de mensajes, aunque estos

mensajes no son controlados directamente por ninguna otra entidad o proceso. Arquitecturas basadas en eventos, arquitectura orientada a servicios y arquitecturas basadas en recursos.

Se escoge para diseñar la propuesta la familia de estilos llamada y retorno por las propiedades de calidad que brinda y de acuerdo al patrón arquitectónico que presenta esta familia (**Ver Anexo12**).

Para llevar a cabo una estructura menos abstracta, en la mayoría de los sistemas se utilizan patrones arquitectónicos que ayudan a la organización y función entre las propiedades y relación arquitectónica de los elementos de software que se implementan (8).

1.4. Patrones arquitectónicos

Los patrones arquitectónicos pueden considerarse como estrategia de alto nivel, el cual incluye varios componentes a gran escala, propiedades y mecanismos del sistema. El cual tiene un conjunto de subsistemas, donde especifican sus responsabilidades y como se relacionan entre sí (8).

A continuación se explica con profundidad los patrones arquitectónicos que se pueden utilizar de acuerdo a las características de los sistemas multimedia:

1.4.1-Modelo – Vista –Controlador (MVC)

El patrón MCV divide la aplicación en tres niveles claramente identificables con funcionalidades bien definidas:

El Modelo: representa la información con la que trabaja la aplicación, es decir, su lógica de negocio (19).
La Vista: transforma el modelo en una página web que permite al usuario interactuar con ella.
El Controlador: se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista.
Este patrón arquitectónico fue diseñado para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos. Este modelo de arquitectura se puede emplear en sistemas de representación gráfica de datos, en donde se presentan partes del diseño con diferente escala de aumento, en ventanas separadas. La finalidad del modelo es mejorar la reusabilidad por medio del desacople entre la vista y el modelo (19).

Ventajas que trae al utilizar el MVC

1. Se puede tener diferentes vistas para un mismo modelo.

2. Facilita crear nuevas vistas sin que se modifique el modelo subyacente.
3. Contiene un mecanismo de configuración más tratable a componentes complejos que el basado en eventos (El modelo puede representarse como un estado de interacción con una forma estructurada) (20).
4. La conectividad del Modelo y sus Vistas es dinámica, ya que se produce en tiempo de ejecución, no de implementación (20).

Desventajas

1. Tener que concentrarse en una estructura predefinida, lo que puede incrementar la complejidad del sistema. Hay problemas que se pueden solucionar respetado el patrón.
2. La curva de aprendizaje de nuevos desarrolladores es mayor.
3. La distribución de componentes obliga a crear y mantener un mayor número de ficheros.

1.4.2-Modelo Tres Capas

Es un estilo de programación, su objetivo primordial es la separación de la capa de presentación, capa de negocio y la capa de datos.

Capa de presentación: es la capa que se le muestra al usuario que utiliza la aplicación, le comunica la información y captura la información del usuario en un mínimo de proceso. También es conocida como interfaz gráfica.

Capa de negocio: en esta capa se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Esta se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar datos de él.

Capa de datos: en ella residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de base de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa del negocio.

La ventaja principal es en el desarrollo, se puede llevar a cabo en varios niveles y, en caso de que se sobrevenga algún cambio. Permite distribuir el trabajo de creación de una aplicación por niveles, cada grupo

de trabajo está totalmente abstraído del resto de niveles, de forma que basta con conocer la *API* que existe entre niveles.

API (*Application Programming Interface*): es el conjunto de funciones y procedimientos o métodos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción (16).

Fig. 2. Ejemplo del modelo tres capa.

Dentro de los beneficios que trae desarrollar un sistema con este modelo se encuentran:

1. Las capas son priorizadas como un todo, sin considerar las otras.
2. Son sustituibles con implementaciones de los mismos servicios básicos.
3. Se hace menos las dependencias entre capas (21).

1.4.3-Arquitectura Cliente/Servidor

La Arquitectura Cliente/Servidor es uno de los modelos de arquitectura más predominantes para la distribución de los recursos de sistemas de información. Este tipo de patrón arquitectónico se define como un sistema distribuido entre varios procesadores, donde hay clientes que solicitan servicios y servidores que los proporcionan. Este patrón separa los servicios situándolos en su plataforma más adecuada.

Se implementa cuando presencian cambios estructurales y organizativos, cambios en organigramas, cuando se busca una respuesta dinámica de mercado. Además, ayuda a implementar las demandas de sistemas fáciles. Se pondera el precio con el rendimiento de estaciones y servidores. Separa los datos del programa haciéndoles más flexible y conlleva a la utilización de nuevas tecnologías de alta productividad (22).

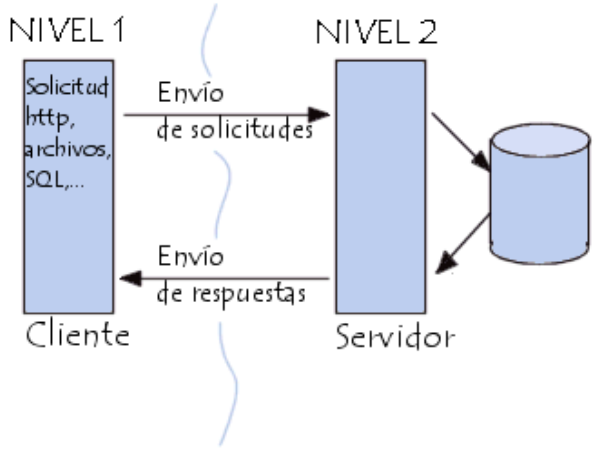


Fig. 3. Arquitectura cliente-servidor (22).

Los beneficios de esta arquitectura se basan en un mejor aprovechamiento de la potencia de cómputo, reduce el tráfico de la red. Opera bajo sistemas abiertos y permite el uso de una gran variedad de interfaces gráficas (22).

Entre los patrones arquitectónicos estudiados, se selecciona para estructurar los sistemas multimedia el modelo-vista-controlador por las ventajas de reutilización y flexibilidad que brinda a los sistemas.

En una arquitectura, los patrones arquitectónicos son simplemente una forma de estructurar los sistemas de acuerdo a las propiedades, mecanismos y las relaciones entre los componentes. Pero, ¿de qué forma se quiere implementar un sistema? Esa es la función de los patrones de diseños, ellos marcan una solución factible conjunto con las características, funcionalidades y problema presente en la implementación de un software.

1.5. Patrones de diseño

Los patrones de diseño son esquemas para refinar los subsistemas o componentes de un sistema de software o sus relaciones. Describe una estructura recurrente y común de componentes que se comunican y resuelven un problema de diseño dentro de un contexto. A continuación se exponen los patrones de diseño que se estudiaran para el modelado del software.

1.5.1-Patrones GoF

Los patrones GoF es un acrónimo que significa Gang of Four (Pandilla de Cuatro), estos patrones es una descripción de clases y objetos que se comunican entre sí, se dividen en tres grupos principales basados en su propósito (23). **Ver Anexo 4.**

1.5.2-Patrones GRASP

Es un acrónimo que significa General Responsibility Asignment Software (Patrones Generales para Asignar Responsabilidades). Los patrones GRASP describen principios fundamentales del diseño de objetos y la asignación de responsabilidades, expresados como patrones (24). **Ver Anexo 4**

De acuerdo con el estudio realizado se utilizará para el modelado del software los patrones GoF como: *Decorator* y *Observer*. Los patrones GRASP para la asignación de responsabilidades a los componentes del sistema se escogen Alta Cohesión y Bajo Acoplamiento. Estos dan una solución simple y también centra su desarrollo a aspectos estructurales, de comportamiento y de reutilización entre los componentes y archivos de los sistemas multimedia.

En todo desarrollo de software se ha necesitado crear formas para organizar, controlar y gestionar el proceso. Como solución se aplican las metodologías para el desarrollo y gestión de un proyecto. A continuación se habla de las metodologías más utilizadas.

1.6. Metodologías para el desarrollo de software multimedia

En la actualidad existen numerosas metodologías de desarrollo para aplicaciones multimedia. En la bibliografía consultada se realiza una comparación de las metodologías, donde se escogen las más óptimas y completas para el desarrollo de multimedia. De acuerdo a las características de las aplicaciones con este tipo de tecnología, se seleccionan las metodologías de desarrollo orientadas a objetos, después se compara la separación de navegabilidad con la interfaz de usuario y la separación conceptual de la navegación.

A partir de lo anterior se selecciona las metodologías para el desarrollo de software multimedia:

EORM (Modelo de Relación de Objetos Mejorado): dentro de su proceso de desarrollo se encuentra: realizar el análisis, realizar el diseño y realizar la implementación, por lo que no define todo el proceso de desarrollo de software. Se define un modelo de objetos, donde en la fase de diseño añade semántica suficiente a las relaciones para representar los enlaces. En el modelo se refleja la estructura de la información, así como

las posibilidades de navegación ofrecidas por el sistema. Debe, existir para crear este modelo, la presencia de un repositorio o librería de clases de enlaces (25).

OOHDM (Método de diseño de hipermedias orientadas a objetos): propone un modelo para representar a las aplicaciones multimedia. Este modelo es orientado a objetos conceptuales y se elaboran en cada fase de la metodología. Se construye el modelo de manera tal que represente el dominio de la aplicación al utilizar técnicas propias de la orientación a objetos (25).

SOHDM (Metodología de diseño de hipermedia orientadas a objetos y basadas en escenarios): genera artefactos de diseño como: Diseño de vistas; vistas bases, vistas de asociación, vistas de colaboración. El diseño de navegación y diseño de implementación (25).

HFPM (Estrategia de modelado de hipermedias flexibles): esta metodología es orientada a objeto. Genera los siguientes artefactos: modelado de los requisitos de software, planificación, modelado conceptual, modelado de navegación, modelado de la interfaz abstracta y diseño del entorno. Cumple con todo el ciclo de desarrollo del software. Esta metodología no cubre con las fases de diseño y codificación. Solo indica que se debe hacer y no cómo se debe hacer (25).

Luego del estudio realizado acerca de las metodologías de desarrollo para aplicaciones multimedia, se decide no utilizarlas, ya que los artefactos de diseño generados por las mismas, no son suficientes para la modelación de la estructura arquitectónica de los productos multimedia del centro FORTES. Además, se analiza que estas metodologías se utilizan para la programación orientada a objetos y algunas están diseñadas para materiales hipermedias, por lo que no es adaptable para todos los productos multimedia.

1.7. Metodología de desarrollo

Las metodologías de desarrollo surgieron con la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto de software. Como consecuencias del desarrollo de nuevos software, en la actualidad se han venido creando nuevas funcionalidades que resultan cambiantes y variadas. Por lo que es imprescindible la creación de numerosas metodologías de desarrollo para la confección de estos productos informáticos (26).

Las clasificaciones de estas metodologías de desarrollo están dadas por ser:

- ♣ Metodologías Pesadas o tradicionales.
- ♣ Metodologías ligeras o ágiles

En la siguiente tabla se muestran las principales diferencias de las metodologías ágiles con respecto a las tradicionales.

Tabla 1. Comparación entre las metodologías ágiles y las pesadas (27).

Criterios	Metodologías Tradicionales	Metodologías Agiles
Estándares de codificación.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.	Basadas en heurísticas provenientes de prácticas de producción de código.
Cambios en el sistema.	Cierta resistencia a los cambios.	Especialmente preparados para cambios durante el proyecto.
Modo de selección de la metodología.	Impuestas externamente.	Impuestas internamente (por el equipo).
Control de proceso.	Proceso mucho más controlado, con numerosas políticas/normas.	Proceso menos controlado, con pocos principios.
Interacción con el cliente.	El cliente interactúa con el equipo de desarrollo mediante reuniones.	El cliente es parte del equipo de desarrollo.
Cantidad de artefactos que generan.	Más artefactos.	Pocos artefactos.
Cantidad de Roles que genera.	Más roles.	Pocos roles.
Cantidad de miembros en el equipo de desarrollo.	Grupos grandes y posiblemente distribuidos.	Grupos pequeños (<10 integrantes) que trabajan en el mismo sitio.
Contrata.	Existe un contrato prefijado.	No existe contrato tradicional o al menos es bastante flexible.

Estas comparaciones se refieren a aspectos organizacionales y al proceso de desarrollo. Las metodologías ágiles están orientadas a pocas semanas de desarrollo y con bajos niveles de formalización en la documentación requerida. De acuerdo a las características de la propuesta, se realiza un estudio de las metodologías ágiles para ver cuál metodología se ajusta para guiar el proceso de desarrollo del software multimedia. A continuación se seleccionan las metodologías ágiles de acuerdo a su flexibilidad, a su rapidez, organización y por ser las metodologías más adaptables al proceso de desarrollo de software con tecnología multimedia por su diseño simple.

1.7.1-Metodologías Agiles

Para el análisis se seleccionaron las metodologías *Xtreme Programming, Scrum* y la unión de estas dos SXP; por ser las más utilizadas en el desarrollo de software y por ofrecer características y artefactos que son adaptables para la propuesta.

1.7.1.1-Xtreme Programming

XP se centra en consolidar las relaciones interpersonales, siendo así clave en el éxito de la producción de software. Promueve el trabajo en equipo y ayuda en el aprendizaje de los desarrolladores. Conlleva a una fuerte comunicación entre el cliente y el equipo de desarrollo. Una de sus características fundamentales son las simplicidades de las soluciones que se implementan y que está propensa a sufrir cambios. Se define para proyectos con requisitos imprecisos y muy cambiantes, y de donde exista un alto riesgo técnico (26).

Filosofía de XP:

Los principios y prácticas son de sentido común, pero llevadas al extremo. Kent Beck describe que su filosofía se basa en cubrir los detalles técnicos y de implementación de las prácticas.

Prácticas de XP:

- Diseño simple.
- Refactorización.
- Integración continua.
- Cuarenta horas a la semana de descanso.

Artefactos que genera la metodología XP (26):

- Historias de Usuario (HU).

- Roles.
- Procesos y prácticas.

Las Historias de Usuario: es la técnica utilizada por XP para especificar los requisitos del software. Se trata de tarjetas de papel donde el cliente describe las características que el sistema debe de poseer, sean funcionalidades o características funcionales.

Los roles de acuerdo Kent Beck (26):

- Programador.
- Cliente.
- Encargado de Pruebas.
- Encargado de seguimiento (*Tracker*).
- Entrenador (*Coach*).
- Consultor.
- Gestor *(Big Boss*).

Proceso:

El ciclo de vida de XP consiste en seis fases:

1. Exploración.
2. Planificación de la Entrega.
3. Iteraciones.
4. Producción.
5. Mantenimiento.
6. Muerte del Proyecto.

En la primera fase se prueba y se explora la posibilidad de la arquitectura del sistema, construyendo un prototipo y la creación de las tarjetas de Clase, Responsabilidad, Colaboración (CRC). En la segunda fase

el cliente establece la prioridad de cada HU, y correspondientemente, los programadores realizan una estimación del esfuerzo que se necesita en cada HU. En la tercera fase se incluyen varias iteraciones sobre el sistema antes de ser entregado. El plan de entrega está compuesto por iteraciones, por un tiempo de no más de tres semanas. En la primera iteración se puede establecer una arquitectura del sistema que pueda ser utilizada durante todo el proyecto. Esto se logra escogiendo las HU que hacen que se cree esta arquitectura, aunque esto casi siempre no sucede ya que mayormente es el cliente quien decide qué HU se va a implementar en cada iteración. En la última iteración el sistema estará listo para entrar en la cuarta fase que es la producción (26).

XP sobre arquitectura (26):
En esta metodología no se enfatiza la definición temprana de una arquitectura estable para el sistema. Dicha arquitectura se asume evolutiva y los posibles inconvenientes que se generan al no contar con la misma en el comienzo del proyecto, se solventan con la existencia de una metáfora.

XP sobre diseño simple:

Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto. La complejidad innecesaria y el código extra debe ser removido inmediatamente. Según Kent Beck dice que en cualquier momento el diseño adecuado para el software es aquel que: supera con éxito todas las pruebas, no tiene lógica duplicada, refleja claramente la intención de implementación de los programadores y tiene el menor número posible de clases y métodos. (28)

1.7.1.2-Melé *(Scrum)*

Esta metodología fue desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle en 1994. Las principales características de esta metodología se resumen en dos, la primera es que el desarrollo de software se realiza mediante iteraciones, definida con el nombre de *sprints*, con la duración de 30 días. La segunda característica importante son las reuniones a lo largo del proyecto.

Otras de sus características se encuentran: (28)

- Es realizada para equipos auto-dirigidos.
- Utiliza reglas para crear un entorno ágil de administración de proyectos.
- No percibe prácticas específicas de ingeniería.

- ♣ Los requerimientos se capturaron como ítems de la lista de reserva del producto y así el producto es construido gracias a una serie de *sprints*.
- ♣ Se puede encontrar tres fases en esta metodología, planificación del *sprint*, seguimiento del *sprint* y revisión del *sprint*.

Fases de la metodología Scrum:

1. Planificación del sprint: el documento de alto nivel del proyecto llamado *Product Backlog*, consiste en una lista priorizada de requisitos del sistema y es un documento vivo, que puede ser continuamente actualizado.
2. Seguimiento del sprint: se lleva a cabo breves reuniones diarias, para ver el alcance de las tareas y el trabajo que está previsto para la jornada.
3. Revisión del sprint: finalizado el sprint se realiza un análisis y revisión del incremento generado. En esta fase se representa los resultados finales y se recomienda tener siempre preparada una demo. (28)

   Artefactos que genera la metodología:

   - ♣ Pila de Producto *(Product Backlog).*
   - ♣ *Product Backlog Burndown.*
   - ♣ Iteraciones de la Pila de Producto (*Sprint Backlog*).
   - ♣ *Sprint Backlog Burndown.*
   - ♣ Reporte Delta de la Pila de Producto *(Product Backlog Delta Report).*
   - ♣ Lista de Implementación *(Implement list)* (28).

   Roles de Scrum:

   - ♣ Propietario del producto *(Product owner).*
   - ♣ Líder del Proyecto *(Scrum master).*
   - ♣ Miembros del Equipo *(Team members).*
   - ♣ Interesados en el proyecto *(Stakeholders).*
   - ♣ Usuarios *(Users)* (28).

1.7.2.3-Scrum-XP

Esta metodología propone procedimientos ágiles para el proceso de producción de software. Es una estrategia tecnológica especialmente dirigida para proyectos de pequeños grupos de trabajo, rápido cambio de requisitos o requisitos imprecisos, donde exista un alto riesgo técnico y se orienta a una entrega rápida

de resultados y una alta flexibilidad. SXP es un híbrido cubano de metodologías ágiles que tiene como base las metodologías SCRUM y XP que permiten actualizar los procesos de desarrollo de software para el mejoramiento de su producción. SCRUM es una forma de auto-gestión para los equipos de programadores y XP es la metodología de desarrollo de software más exitosa en la actualidad. Esta metodología ayuda a fortalecer el trabajo en equipo, enfocados en una misma dirección, con un objetivo claro, permite además, seguir de forma clara el avance de las tareas a realizar, de forma que los jefes pueden ver día a día cómo progresa el trabajo, a partir de la inserción de procedimientos ágiles que permitan actualizar los procesos de software para el mejoramiento de la producción, aumentando el nivel de interés del equipo (29). La metodología SXP durante sus años de despliegue y ejecución ha sido estudiada, refinada, adaptada de acuerdo al grupo de investigadores.

Esta metodología genera cuatro fases principales (29):

Planificación-Definición: se establece la visión, presupuesto y reserva del producto con estimaciones iniciales.

Artefactos que genera:

- Concepción del sistema.
- Historia de Usuarios.
- Lista de reserva del producto (LRP).
- Lista de riesgos.
- Modelo de diseño.
- Modelo de historia de usuario del negocio.

Desarrollo: se realiza la implementación del sistema hasta que esté listo para ser entregado.

Artefactos que genera:

- Caso de prueba de aceptación.
- Cronograma de producción.
- Estándar de código.
- Plan de reléase.
- Tarea de ingeniería.

Entrega: puesta en marcha.

Artefactos que genera:

- Manual de desarrollo.
- Manual de identidad.
- Manual de usuario.

Mantenimiento: donde se realiza el soporte para el cliente.

Artefactos que genera:

- Gestión de cambios.

El líder y su equipo se reúnen y controlan los avances e identificaran los posibles riesgos que afecten de una manera u otra la correcta ejecución del proyecto. También el cliente puede ser parte del equipo de desarrollo. Además, con el uso de esta metodología se logra un mejor proceso de desarrollo en la implementación de productos con tecnología multimedia de la línea de producción de software, ya que proporciona roles a cada integrante del proyecto, permitiendo así, la calidad, eficiencia y control del desarrollo de estas aplicaciones. (29)

Cada uno de estos artefactos generan los siguientes roles que son los responsables y supervisores del proyecto: (29)

- Líder del proyecto (*Scrum Master*): administra el proyecto.
- Gerente (*Management*): realiza tomas de decisiones finales.
- Especialistas: responsable de todo el proceso de desarrollo.
- Consultor: tiene conocimiento específico sobre algún tema que necesita el proyecto.
- Cliente (*Customer*).
- Miembros del proyecto
- Programadores (*Programmers*).
- Analista (*Analyst*).
- Diseñadores (*Designers*).
- Encargado de Pruebas (*Tester*).
- Arquitecto (*Architect*).

Se hacen entregas frecuentes y existe una refactorización continua, lo que permite un mejoramiento en el diseño cada vez que se añada una nueva funcionalidad. (29)

Después del estudio de las metodologías de desarrollo, los investigadores seleccionan la metodología ágil SXP, con el objetivo de poder diseñar una arquitectura basada en componentes, donde se utilizan elementos que son ensamblados para generar productos multimedia. El software no lleva una alta complejidad en sus estructuras, por lo que la metodología ágil seleccionada presenta un diseño simple y permite el enfoque en la implementación de los productos. Además, ofrece una entrega rápida del software al cliente. Esta metodología es especialmente indicada para proyectos pequeños y muy cambiantes, donde existe un alto riesgo técnico y se orienta a una entrega rápida de resultados y una alta flexibilidad (29).

1.8. Lenguaje de programación y lenguaje etiquetados

Un lenguaje de programación es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina. La función de él, es introducir una serie de códigos en lenguaje natural para que el programa lo interprete y realice la acción que se le pide. Un lenguaje etiquetado se utiliza para dar formato a los documentos de texto, define la estructura de los datos de un documento. Es un modo de codificar un documento y junto con el texto se incorporan etiquetas, marcas, anotaciones con información adicional relativa a la estructura del texto. En este epígrafe se abordará sobre los lenguajes de programación, librería y estilos más utilizados para la realización de los sistemas multimedia (30).

1.8.1-HTML

Es un lenguaje bastante sencillo que ha ayudado a promover el uso generalizado a Internet. La intención de HTML es incluir elementos que podrían utilizarse para marcar la información de un documento. Es un lenguaje etiquetado que está orientado del lado del cliente. Una de sus características es que posee una estructura no muy rígida, por eso es posible crear códigos en HTML que no están escrito correctamente, pero el navegador lo interpretará de forma correcta. (31)

Con la creación de HTML en su versión 5 se crearon nuevas etiquetas que provee que este lenguaje sea más semántico que los anteriores. Estas nuevas etiquetas afectaran la estructura de las páginas. Por ejemplo se encuentra: ***section*** utilizada para representar una sección dentro de un artículo, ***header*** se utiliza para definir una cabecera de un documento HTML o una sección (no tiene que ver con la etiqueta <head>). ***Footer***, utilizado para definir el pie de página de un documento HTML o una sección. ***Nav***, etiqueta utilizada

para definir secciones de navegación para dentro del mismo sitio web. Otras de las etiquetas es ***article***, que define una sección para incluir una entrada a un blog, revista, etc. Por las características que tiene integrada HTML 5 los buscadores podrán encontrar fácilmente el contenido principal, donde permite gastar menos recursos y tiempo en los menús, footer u otros elementos no relevantes de la página (31).

1.8.2-Lenguaje de Estilo Extensible (XML)

El lenguaje marcado XML al igual que HTML es una tecnología sencilla que tiene a su alrededor otras que las complementan y la hacen mucho más grande, permite la compatibilidad entre sistemas para compartir la información de forma segura, fácil y fiable. El XML se utiliza para crear documentos de textos que contiene formato estructurado. Además, de los datos puede incluir un conjunto de reglas que definen la estructura de dichos datos. El autor del documento XML define esas reglas (31). Se encuentra agrupado en tres grupos: uno se dedica a la visualización del contenido, otro dedicado a la recuperación de información contenida en él y el último permite los enlaces entre los documentos XML (32).

1.8.3-JavaScript

Es un lenguaje de programación utilizado para crear programas que se encarguen de realizar acciones dinámicas dentro del ámbito de una página web. JavaScript es un lenguaje interpretado, por lo que no es necesario compilar los programas para ejecutarlos. Permite crear efectos especiales en las páginas, lo que posibilita la interactividad con el usuario. Brinda facilidad en su código, de uso y aprendizaje (33).

1.8.4-Librería de Java Script (JQuery)

JQuery es una de las herramientas más populares y fáciles de trabajar en la implementación, esta librería ofrece una serie de funcionalidades, donde permite de manera simplificada trabajar con documentos de HTML. Con las funciones propias de esta biblioteca se logran grande resultados en menos tiempo y espacio, la misma ahorra mucho más líneas de código que otras herramientas. Contiene métodos que son implementados en JavaScript. Tiene un sistema expandible mediante plugins entre los cuales se encuentra el JQuery UI, encargado de la implementación de elementos visuales; y también provee de un mecanismo para la captura de eventos (34).

1.8.5-Lenguaje Action Script (AS)

Este lenguaje de programación ha sido utilizado por Flash desde sus orígenes y en la actualidad emplea Adobe Flash CS6. Este lenguaje permite al desarrollador tener el control absoluto de una película Flash. Es

un lenguaje de script, esto significa que no es necesario crear un programa completo para conseguir resultado. Hasta el momento este lenguaje tiene como última versión el AS 3.0. El lenguaje pone a disposición una biblioteca de funciones, clases, métodos ya implementadas que realizan lo que se busca, solo bastara con colocarla en el lugar indicado (37).

1.8.6-Hojas de Estilos en Cascadas

Las hojas de estilos permiten crear clases y pseudo clases. Estas permiten modificar e indicarle al navegador la forma en la que tiene que presentar cualquier elemento HTML. Cuando es creado un estilo, al definir las características de un elemento serán aplicables las mismas a todos aquellos elementos que se encuentren por debajo de él, atendiendo al criterio de herencia, por esta razón son llamados estilos en cascadas (35).

1.9. Herramientas de desarrollo para aplicaciones multimedia

Las herramientas de desarrollo surgieron con el propósito de ayudar a la implementación de los sistemas de información e intentar brindar soluciones a los problemas inherentes de los proyectos de generación de aplicaciones informáticas (36). Se utilizan para implementar la interactividad e interfaces del usuario, a fin de presentar el proyecto en pantalla y combinar los diferentes elementos multimedia en un solo proyecto cohesionado. También brindan el marco esencial para organizar y editar los elementos del proyecto multimedia, incluyendo gráficos, sonido, animaciones y secuencia de vídeo. En este epígrafe se realiza un estudio de las herramientas a utilizar para la realización de los productos con tecnología multimedia.

1.9.1-Adobe Flash CS6

Es un entorno de edición de gran alcance para crear animaciones y contenidos multimedia. Diseña experiencias interactivas que presentan constantemente a través de equipos de sobremesa y varios dispositivos, incluyendo tabletas, teléfonos inteligentes y televisores (37).

Esta herramienta combina fácilmente varios símbolos y secuencias de animación en una hoja de *sprites* optimizada, solo para un mejor flujo de trabajo, crea contenidos más atractivos con el uso de extensiones nativas para acceder a las capacidades específicas de los dispositivos, se puede utilizar el lenguaje etiquetado HTML 5 para la creación de activos y animaciones que se implementen en el sistema. Exporta como formato JavaScript para utilizar la arquitectura de Código abierto *CreateJS.*

Flash presenta ventajas como inconvenientes:

Flash es una tecnología propietaria de Adobe, por lo que su uso depende de lo que quiera hacer el equipo de proyecto con la herramienta. Para desarrollar en esta herramienta se debe pagar la licencia de permiso de utilización del software, por lo que se quiere migrar al software libre. No es recomendable la utilización de esta tecnología para la comercialización de los productos que se desarrollan por el costo que puede generar.

El tiempo de carga: mientras que una página HTML puede ocupar 10-20 KB como media, una animación Flash ocupa mucho más. Evidentemente depende del contenido que tenga, pero suelen superar los 100 KB con facilidad, y si además incorpora sonidos es fácil que la cifra se dispare. Al ocupar más espacio, el tiempo que tardan estar visible el contenido Flash es mayor y no todos los visitantes están dispuestos a esperar, simplemente, se irán a otra página (37).

Los buscadores: son capaces de indexar el contenido de la página, del texto, pero no el contenido del Flash, ya que no lo pueden leer, lo que afectará negativamente al posicionamiento de la página. Y hoy en día es crucial para una web encontrarse bien posicionada. No obstante, los buscadores trabajan para solucionar este problema, pero de momento la mejor forma de solucionarlo es crear un diseño paralelo sin Flash, lo que permite el aumento del trabajo (38).

1.9.2-NetBeans

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI) (39).

Netbeans es un entorno de desarrollo visual de código abierto para aplicaciones programadas mediante Java, uno de los lenguajes de programación más poderosos del momento. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE *NetBeans*. El IDE *NetBeans* es un producto libre y gratuito sin restricciones de uso. El editor de JavaScript está mejorado de forma significativa y tiene integrado el esquema de completamiento de código. Tiene un editor visual de CSS, un inspector de páginas y un navegador integrado, así como completa integración con *Chrome*. Se añade también la opción de archivos recientemente abiertos. Soportan los perfiles de Linux ARM. No importa que la máquina donde se instale sea un Linux, un Mac o un Windows, pues el funcionamiento del programa creado será igual (39). Esta herramienta es multiplataforma.

1.9.3-PhpStorm

Tiene un control de versiones propio, un servidor integrado con el editor para recuperar versiones antiguas del texto. Se puede integrar fácilmente con CVS, con GIT, con SV, es decir, con los sistemas de control de versiones más utilizados actualmente. No está basado en JAVA, con lo que es más rápido. A pesar de eso tiene clientes para Windows y para Linux, también para MAC. Integra de forma sencilla las librerías JavaScript más importantes. Permite refactorización de código.

Tiene un analizador de código con arquitectura de flujo de datos que mejora sustancialmente la precisión del análisis del código hasta el punto de que ya no se necesitarán anotaciones de código adicionales. Además, el nuevo modo de edición a tiempo real será de gran ayuda para los desarrolladores web que buscan acelerar la productividad de su código (40).

1.9.4-WebStorm

Es un editor de texto básico que se utiliza para implementar aplicaciones basadas en JavaScript y HTML. Es capaz de sugerir JavaScript optimizaciones específicas de código y rápidas soluciones y refactorizaciones. Se realiza un análisis exhaustivo del código de proyecto y ofrece a los desarrolladores la mejor en su clase de código de entendimiento, sugerencias sobre el código de terminación y las instalaciones del proyecto de navegación.

Proporciona soporte transparente para los sistemas de control más populares de la versión (Git, Mercurial, SVN) y varios gestores de incidencias, así como la implementación del proyecto y la sincronización a través de FTP / SFTP. La última versión soporta HTML5 WebStorm, JavaScript5, LESS y extensiones SASS para CSS.

El editor entiende su código, su estructura y realiza completamiento de código. Ayuda para la codificación completa se ofrece incluso para mezclas de idiomas tales como HTML dentro de las cadenas de Java Script.

Si bien se puede utilizar cualquier editor de texto básico, se usaría un IDE que ofrece un conjunto de características más robustas, tales como: sugerencias para el código, refactorización, gestión de proyectos, control de versiones, integración, y un depurador. JetBrains que tiene dos IDEs tanto para Java Script o desarrollo HTML5 y PHP. Si sólo piensa en hacer el desarrollo de JavaScript, se sugiere utilizar WebStorm (41).

1.10. Herramientas CASE y lenguaje de modelado

Las Herramientas CASE (Ingeniería de Software Asistida por Ordenador) permiten brindar a los analistas, ingenieros de software y desarrolladores; ayuda y asistencia durante todo el ciclo de vida de desarrollo de

un software. Esto lo hacen gracias a los lenguajes de modelado. En este epígrafe se menciona algunas herramientas y lenguajes que se pueden utilizar. Existen varias Herramientas CASE como **Bonita BPM *Community***, ***Crystal Report* 10**, **Erwin**, ***Rational Rose***, **Visual Paradigm**, entre otras. Las que se abordan a continuación: ***Rational Rose*** y **Visual Paradigm**.

Rational Rose: tiene el fin de modelar aspectos claves de todo el proceso de desarrollo de un sistema. Es una herramienta de diseño orientada a objetos que permite representar gráficamente el sistema, permitiendo hacer énfasis en los detalles más importantes. Proporciona mecanismos para realizar la Ingeniería Inversa, es decir, que a partir del código se puede obtener información de su diseño. Tiene la ventaja de que varias personas trabajen a la vez, ya que permite que cada desarrollador trabaje de forma privada y permite que tenga un control total sobre la propagación de los cambios en ese espacio de trabajo. Soporta los diagramas UML, excepto los Diagramas de Implementación (42)**.**

Visual Paradigm: utiliza el lenguaje de modelado UML, para modelar sistemas con gran escala de funcionalidades, necesidad de confiabilidad y modelación de diferentes diagramas para el negocio. Además, se pueden crear prototipo de interfaces de usuario con la herramienta. Esta herramienta ofrece: navegación intuitiva entre la escritura del código y su visualización, contiene un generador de informes en formato PDF/HTML potente, un ambiente visualmente superior de modelado, documentación automática Ad-hoc y un diagramador automático de disposición. Se puede exportar los diagramas como activo de imágenes en formato .jpg (43).

Lenguaje de modelado UML (Lenguaje Unificado de Modelado): es un lenguaje utilizado para representar y modelar la información con la que se quiere trabajar en las fases de análisis, y especialmente de diseño. Sirve para el modelado completo de sistemas complejos, tanto en el diseño de los sistemas, como para la arquitectura de hardware. Otra de sus metas es que pueda ser independiente del lenguaje de implementación, de tal forma que los diseños realizados en UML se puedan implementar en cualquier lenguaje que soporte las posibilidades UML (44).

Apen L (Lenguaje para la Modelación de Aplicaciones Educativas): este lenguaje de modelado es creado con el objetivo de modelar aplicaciones educativas. Toma elementos fundamentales de UML (Lenguaje Unificado de Modelado), OMMMA-L (Lenguaje Orientado a Objetos para la Modelación de Aplicaciones Multimedia) y OCL (Lenguaje de Modelación de Objetos). Puede utilizarse en Rational Rose, Visual Paradigm, Erwin. UML para la modelación el diagrama de componente y para la modelación de la

vista del sistema el diagrama de clases y para la vista de procesos se modela con el diagrama de proceso de negocio.

Consta de tres áreas fundamentales: la estructura lógica, comportamiento dinámico y gestión del modelo. Consta de cuatro vistas distribuidas en cada área. En la primera área consta de la vista de arquitectura y la vista estática. En la segunda se crea la vista de comportamiento. En la tercera se tiene la vista de presentación (45). **Ver Anexo 5**

OMMA-L (Lenguaje de Aplicaciones Multimedia Modelado de Objetos): Es una propuesta de extensión de UML, para la integración de especificaciones de sistemas multimedia basados en el paradigma orientado a objetos. No presenta cambios con respecto a UML en el flujo de requisitos y caso de uso, pero si en el modelo de clases de objetos. Estructura diagrama de presentación, diagrama de clases del análisis, diagrama de clases del diseño, diagrama de estado y diagrama de secuencia. Se modela la estructura a través de diagramas de objetos y clases. Describe el comportamiento a través de los diagramas de interacción (25).

1.11. Selección de las herramientas y lenguajes para implementar el prototipo no funcional de la propuesta de arquitectura

Para la implementación del prototipo no funcional se escogen las siguientes herramientas por la experiencia del equipo de desarrollo con estas tecnologías, la flexibilidad que brindan al desarrollar los sistemas multimedia y por ser tecnologías libres con la libertad de licencias para el desarrollo de futura comercialización.

NetBeans entorno de desarrollo en su versión 7.4 (IDE)

Para la implementación de la arquitectura de software se utiliza el IDE Netbeans en su versión 7.4 el cual es un entorno de desarrollo gratuito y de código abierto. Soporta los lenguajes utilizados en la implementación de los productos que se van a desarrollar bajo la arquitectura que se propone: HTML en su versión 5, XML en su versión 2, las hojas de estilos en cascada en la versión 3 y el framework de desarrollo de Java Script JQuery en su versión 2.0. El IDE puede usarse en varios sistemas operativos, ya que se tiene planteado como política de la institución la migración a software libre. Contiene un editor de código que realiza el completamiento, por lo que ofrece la facilidad de implementación. Además, el equipo de desarrollo tiene amplio dominio de esta herramienta.

HTML versión 5 (implementar las interfaces de usuario)

Se escoge HTML versión 5, ya que es un lenguaje de etiquetado, fácil de implementar y ayuda a crear las interfaces de usuarios que se proponen. Este lenguaje engloba en su versión etiquetas como: nav, section, audio y video, las cuales se utilizan para la implementación de la propuesta. HTML tiene como función la comunicación entre el usuario y el sistema.

XML en su versión 2 (gestor de datos)

Este lenguaje almacena las direcciones de los recursos multimedia. Permite la portabilidad de los sistemas, ya que no depende de otras tecnologías para la ejecución del software. Estructura el modelo de datos con etiquetas de forma estructural, definidas de acuerdo a las necesidades del programador. La utilización del XML permite la reusabilidad de los componentes de los sistemas que lo implementa. Es sencillo y se adapta con facilidad.

JQuery versión 2.0 (implementación de los archivos.js)

Es un framework de Java Script que ofrece varias funcionalidades que permite de forma simplificada, el uso de documentos HTML. Gracias a estas características de JQuery los documentos HTML cargan con la etiqueta <script> las funcionalidades de la librería. (45)Para la propuesta se utilizan archivos .js que implementan funcionalidades como el $.get () que permite cargar datos asociados con: imágenes, videos, entre otros. Dada las ventajas del framework de Java Script se puede crear una función que agrupe los elementos comunes de cada página y mostrarlos en ellas. Además, contiene plugins que han sido utilizados en el desarrollo de componentes reutilizables.

Hojas de Estilos en Cascadas en su versión 3 (diseño de las interfaces, clases .css)

Las clases css son aplicadas para indicarle al navegador la forma en la que va aparecer los elementos HTML. Algunos componentes integran clases .css para mejorar la apariencia visual. Las interfaces creadas en el prototipo no funcional, disponen de dos clases .css, donde una se encarga de darle estilo a los componentes de la aplicación y la otra a la apariencia. Se utilizan las hojas de estilos para hacer las interfaces más agradables, más intuitivas y sencillas. Facilita la reutilización de estilos en varias páginas, al permitir la agilización del proceso de desarrollo.

Herramientas y lenguajes para la modelación de la propuesta

Se emplea **ApEM-L** en su versión 1.0 es un lenguaje de modelado que ofrece diagramas necesarios para modelar multimedia. Este lenguaje consta de tres áreas de modelado, donde de estas tres áreas se utiliza dos de ellas para modelar la arquitectura: Estructura Lógica y Vista de Presentación (Modificada por Gestión

del Modelo). Donde en la estructura lógica se realiza el Diagrama de Componentes, junto con la Vista del Sistema. En la Vista de Presentación se tiene el diagrama de estructura de navegación y Diagrama de presentación.

Se utiliza **Visual Paradigm** en su versión 8.0, esta permite utilizar el lenguaje de modelado ApEM-L. Es una herramienta multiplataforma. Es utilizada para la creación de interfaces de usuario para el prototipo no funcional. Integra extensiones de diferentes lenguajes de modelado, entre los que están UML. Permite que todos los lenguajes extendidos de UML, puedan también utilizar la herramienta.

1.12. Conclusiones parciales

Después del estudio y análisis realizado de la arquitectura de software basada en componentes, apoyado en los métodos de investigación científico que se definieron, se pudo elaborar el marco teórico que soporta la investigación y se realiza la selección las herramientas, metodologías y lenguajes más utilizados en el desarrollo de productos con tecnología multimedia: se escoge como lenguaje de marcado HTML en su versión 5 para la creación de los componentes visuales del sistema y un XML para el manejo de datos. Para agregarle funcionalidades y dinamismo a las vistas se utiliza la librería JQuery 2.0 junto a java script como lenguaje de programación. Se selecciona para la modelación de los diagramas de componentes, las vistas de arquitectura e interfaces de usuario: Visual Paradigm en su versión 8.0 y el lenguaje de modelado ApEM-L en su versión 1.0. El desarrollo será controlado y dirigido mediante la metodología SXP. Además, se hace énfasis en los conceptos más importantes del objeto de investigación.

Capítulo 2. Propuesta de Arquitectura

2. Introducción

A partir del capítulo anterior donde se escogen las principales herramientas y tecnologías existentes con respecto a la Arquitectura de Software, se procede a representar según las características acordes a la propuesta, los estilos y patrones arquitectónicos, los patrones de diseño y la estructuración de los diagramas de componentes, la vista de proceso, vista del sistema, vista de datos, vista de integración y la vista de infraestructura y despliegue. También se representan las interfaces de usuarios del prototipo no funcional de la propuesta de arquitectura y se visualiza la estructura de carpetas que deben tener los productos multimedia en su implementación.

2.1. Estilo y Patrón Arquitectónico

La arquitectura se ha estructurado a partir del estilo arquitectónico de **llamada y retorno** ya que enfatiza en la modificabilidad, escalabilidad, reusabilidad y usabilidad del sistema. Presenta una jerarquía de control donde un programa principal es el que controla todo el sistema y se comunica con otros subprogramas a través de llamadas.

En la arquitectura basada en componente se utiliza el patrón **MVC** para el desarrollo de la propuesta de solución, con el propósito de aprovechar la separación de las responsabilidades de cada una de las capas que lo conforman.

El patrón arquitectónico MVC separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres formas diferentes (47).

Modelo: administra el comportamiento y los datos del dominio de la aplicación. Para las aplicaciones multimedia se elaboran de forma común un documento XML, donde permite cargar todos los elementos que intervienen en la capa modelo. La capa modelo tiene integrados por carpetas las imágenes, videos, animaciones y sonido, que son contenidos esenciales para los productos multimedia. Puede cambiar de estado desde el controlador.

Vista: maneja la visualización de la información. Provee todos los elementos visuales para mostrar a los usuarios, permite la interacción del sistema con el usuario. Esta capa tiene comunicación con el modelo mediante la clase controladora. Las relaciones de navegación de las páginas interfaces consta de una página de inicio que es la que presenta el producto y después carga la página principal que va tener acceso de acuerdo a los componentes de navegación a las demás interfaces.

Controlador: interpreta las acciones del ratón y el teclado, informa al modelo y/o a la vista para que cambien según el resultado apropiado. En esta capa se implementa el código para cargar desde el modelo todos los elementos multimedia que se quieran visualizar. La capa controladora contiene archivos que implementan las páginas interfaz. Estos archivos son implementados de acuerdo a los patrones de diseño seleccionados a continuación.

2.2. Patrones de diseño

Los patrones de diseño son buenas soluciones que pueden ser aplicadas a problemas recurrentes que surgen durante el diseño de un sistema o una aplicación. Para el desarrollo de la arquitectura de software basada en componente con tecnología multimedia se hace uso de los siguientes patrones:

Decorator (Envoltorio): este patrón de diseño es de tipo estructural que ofrece la solución de poder estructurar los elementos visuales de las páginas interfaces.

Problema: para los productos multimedia, se requiere mayormente que las aplicaciones sean interactivas, esto permite extender las funcionalidades del propio sistema.

Solución: este patrón añade de forma dinámica nuevas responsabilidades a un objeto, proporciona una alternativa flexible. Por lo que se usa en las aplicaciones para la comunicación de las clases vistas y controladora con respecto a la decoración de los elementos visuales, de forma que las funcionalidades se realicen de manera interactiva. La utilización del patrón se basa en crear una clase que integre la funcionalidad de cargar los elementos más comunes en todas las páginas de interfaz de los sistemas multimedia, desde un documento XML. Dígase banner (encabezado), footer (pie de página), menús, entre otros.

Consecuencias: ofrece más flexibilidad que la herencia estática. Se consiguen componentes pequeños. Se reduce el número de clases y el árbol de herencia de clases.

Observer (Observador): este patrón de diseño proporciona resolver problemas relacionados con el comportamiento de la aplicación, normalmente en tiempo de ejecución.

Problema: cuando se quiere realizar un cambio en una funcionalidad de algún archivo.js o .css, permite de forma dinámica la actualización de los otros archivos que existen en el sistema, todo esto en tiempo de ejecución.

Solución: define una dependencia de uno a muchos eventos, de forma que cuando uno de los eventos en el sistema cambia de estado se notifica y actualizan automáticamente todos los demás eventos de los archivos dependientes, sin que los otros archivos implicados en el sistema sean conscientes de los otros

archivos observables. Se aplica para cuando un evento depende de la ejecución de otro para poderse implementar. Se utiliza en el desarrollo con eventos en el sistema.

Consecuencias: cuando existen muchos *observer* registrados se ralentiza notablemente el envío de notificaciones. Un objeto observable puede ser a su vez observador respecto a otros.

Bajo acoplamiento: es un patrón de asignar responsabilidades a cada archivo o componente dentro del sistema. Se utiliza cuando hay pocas dependencias entre los archivos.js y .css. Ayuda a mejorar el diseño y organización de los elementos que integran un software.

Problema: resolver la interdependencia entre cada componente del sistema.

Solución: se evidencia en la separación que existe entre la capa vista, controladora y modelo. De tal manera que cada una contiene la información necesaria, lo cual implica que un cambio en una de las capas no afecta en las restantes.

Consecuencias: cuando existe mucha dependencia entre los elementos de un sistema se puede tener un mal diseño y un alto acoplamiento.

Alta cohesión: brinda responsabilidad a cada archivo del sistema de acuerdo a la función que realiza dentro del mismo, sin que contengan otras funcionalidades.

Problema: cada elemento del diseño debe realizar una labor única dentro del sistema.

Solución: plantea la solución en la propuesta cuando se le asigna a cada archivo información coherente, relacionada con la funcionalidad que realiza.

Consecuencias: este patrón Graf presenta como consecuencia la creación de numerosos archivos dentro del mismo sistema que implementen una funcionalidad en específico.

2.3. Lineamientos de diseño e implementación

Esta sección contiene las restricciones impuestas por el arquitecto de software para el correcto desarrollo y avance de la solución, tanto desde el punto de vista del diseño, como de la implementación y el mismo se encuentra en el documento de arquitectura que lleva la propuesta.

2.3.1-Estándares de codificación

Los estándares de codificación son reglas que se siguen para la escritura del código fuente, de tal manera que a otros programadores se les faciliten entender el código, (como identificar las variables, las funciones o métodos) (48).

A continuación se mostrará el estándar de codificación que se utiliza para implementar en el desarrollo de productos con tecnología multimedia.

2.3.1.1-Asignación de nombres

Cada tipo de elemento debe nombrarse con una serie de reglas determinadas.

Paquetes: todo en minúscula y empieza con la palabra *´paq_´* y seguido el nombre del paquete.

paq_nombredelpaquete, el nombre del paquete con una letra inicial mayúscula.

Archivos e interfaces: para las interfaces la letra inicial de cada palabra con mayúscula lo demás en minúscula y para los archivos se debe escribir la letra inicial con minúscula al igual que los caracteres restantes.

Funciones: el primer caracter de la primera palabra en minúscula y el resto de las palabras empiezan con mayúscula.

Variables: todo en minúscula, si son varias palabras unidas por el carácter ´_´.

Los nombres no pueden ser largos (deben de tener menos de 10 caracteres).

2.3.1.2-Nombres de ficheros

Código fuente: nombre del código.html

2.3.1.3-Declaraciones

Declaraciones de variables

Cada variable se declara en una línea distinta, así de esta forma se puede comentar por separado**.**

var suma;

var edad;

Inicializar cada variable en su declaración a menos que su valor inicial dependa de algún cálculo.

Suma = 10+6+59;

Los arreglos se deben declarar de este modo: var array_nombredelarreglo.

No se deben declarar variables con el mismo nombre que otras en una función.

Declaraciones de funciones

Debe haber no más ni menos de un espacio entre el nombre de la función, el paréntesis y la lista de parámetros.

Se abre la llave en la misma línea que la declaración del método.

La llave de cerrar debe aparecer en la línea aparte que el método cierra.

function sumaPromedio (suma) {

// Sentencias o código asociado

}

2.3.1.4-Sentencias

Cada línea debe tener una sola sentencia.

Todas las sentencias if, for, while, do while deberán tener llaves aunque contengan una sola sentencia, de esta forma se evita la introducción accidental de errores si se añaden posteriormente sentencias.

for (inicialización; condición; actualización) {

bloques de código

}

2.3.1.5-Comentarios

Existen dos tipos de comentarios, los de implementación y los de documentación. Los comentarios de implementación se identificará por // y los comentarios de documentación por: /**…..*/.

/**

@autor: nombre del autor.

@fecha: fecha.

@comentario: descripción.

*/

Los comentarios de documentación son comentarios para una visión de más alto nivel para desarrolladores que no tiene el código a mano y que solo quiere usarlo. Estos comentarios se ponen al comienzo de cada fichero.

Los comentarios de implementación son comentarios explicativos de una parte pequeña del código.

En el caso de comentarios en XML, HTML se utiliza <!----texto ---->.

2.3.2-Diseño

En este epígrafe se pretende establecer los elementos de arquitectura para los productos con tecnología multimedia. Se diseñó mediante la definición de las vistas de arquitectura del sistema, toma en cuenta los estilos y patrones arquitectónicos escogidos, así como las características más comunes.

2.3.2.1- Vista del sistema

Como se menciona anteriormente, la arquitectura propuesta será descrita mediante la vista de presentación, vista de modelo y la vista controladora. La Fig. 5 muestra una visión general del sistema, es suficientemente abstracto como para dar un bosquejo de la solución.

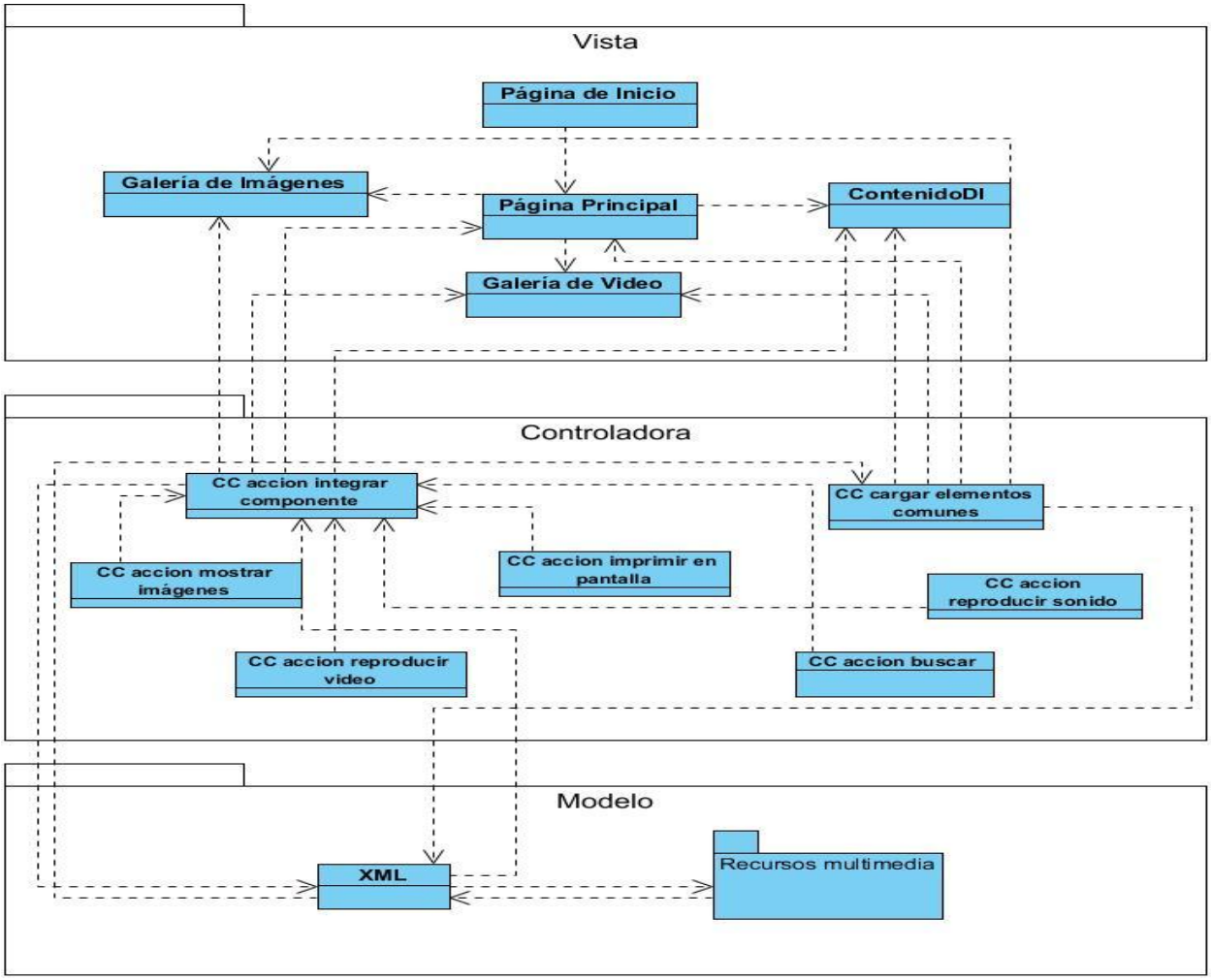


Fig. 4. Vista general de los sistemas multimedia.

A continuación se muestra los diagramas de componentes para la capa vista, capa controladora y capa modelo, que componen el diseño arquitectónico basado en componentes para la vista del sistema, en la Fig.5 se ilustra el diagrama de componentes para la capa Vista. Se crean las interfaces con código HTML y se implementan de acuerdo a librerías o framework de Java Script o CSS. Esta capa contiene las clases de presentación y las otras clases que integran mayormente los productos multimedia, se relacionan de acuerdo al flujo del sistema. En la parte de configuración se ilustra los componentes más comunes que deben llevar cada una de las interfaces creadas, desde el código hasta los elementos visuales que las integran. Estos elementos son implementados desde la clase controladora.

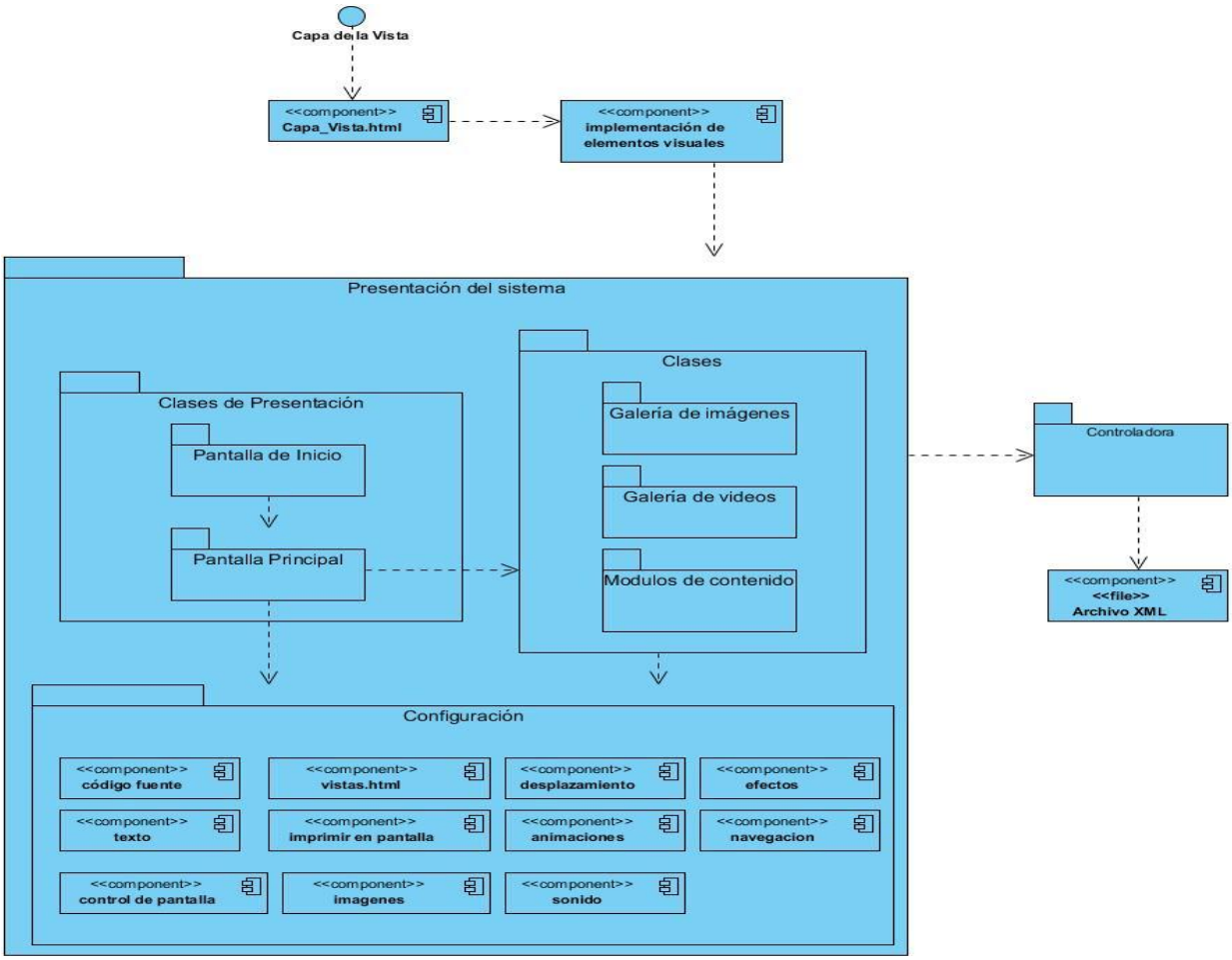


Fig. 5. Diagrama de componentes para la capa de la Vista.

La capa controladora contiene componentes externos e interno de la aplicación. Los componentes internos del sistema se encuentran dentro de los directorios JS y CSS, estos integran las clases .js y .css que

visualizan los elementos comunes entre todas las clases interfaces y permitirán organizar y controlar cada contenido dentro del sistema. Para los componentes que ya han sido implementados o componentes externos se crean nuevos directorios con el nombre de cada uno. El directorio JS debe contener una clase que permita integrar cada componente externo a la aplicación (utilizar funcionalidades de JQuery). Las medias que son visualizadas en el sistema se cargan a través de un XML que integra la dirección de cada media.

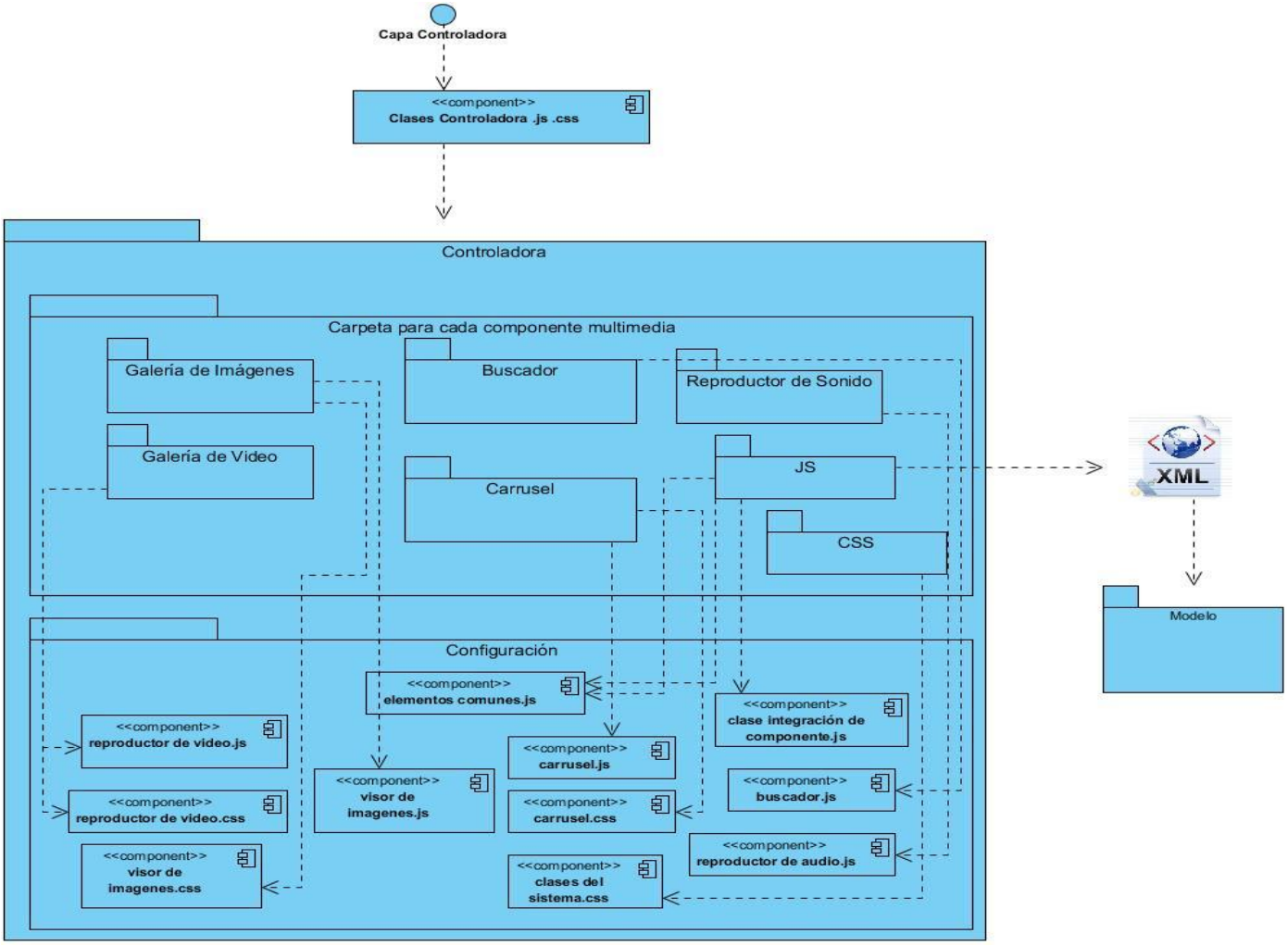


Fig. 6. Diagrama de componentes para la capa Controladora.

En el siguiente diagrama se estructura cada componente externo a la aplicación. La integración al sistema se realiza mediante la clase integrar componente CE.js. Dentro de cada directorio se encuentra el

componente, donde su estructura interna consta con: dos carpetas, una donde se guarda las clases .js y en la otra los .css. Además se integra dentro de cada componente una clase.html y un documento XML que permite la visualización del mismo.

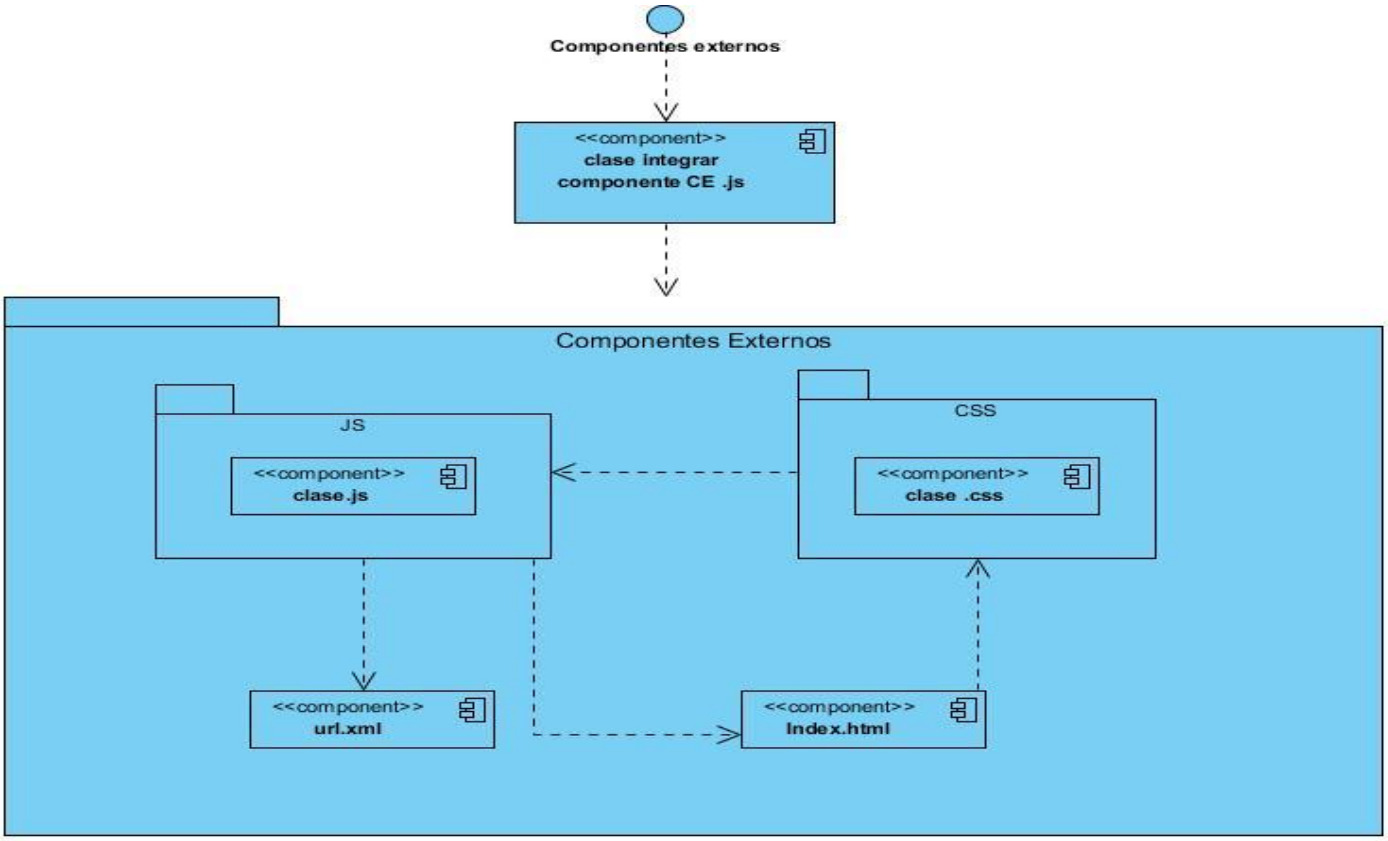


Fig. 7. Estructura de los componentes externos.

La estructura de componentes que contiene la capa Modelo consta de un archivo XML que almacena las url, la descripción y atributos de cada recurso multimedia. Cada media es almacenada por carpeta de acuerdo al tipo que sea.

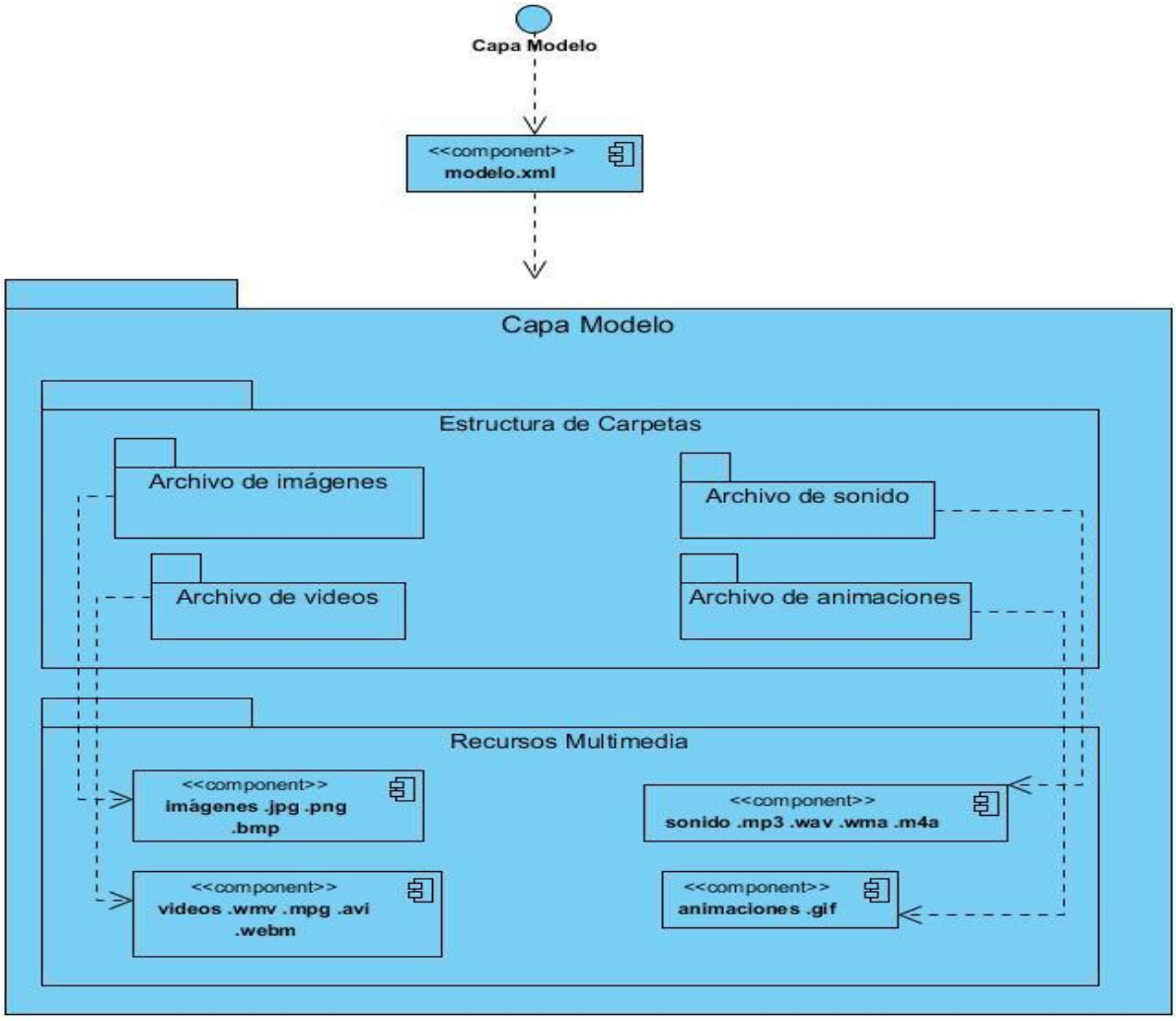


Fig. 8. Diagrama de componentes de la capa Modelo.

2.3.2.2-Vista de proceso

En la Fig. 9 se muestra el flujo de procesos que va a tener el desarrollo de los productos multimedia según la metodología SXP. Para representar las tareas que debe seguir el equipo de trabajo para desarrollar los productos multimedia, se tienen los macros procesos que incluye las fases de la metodología más la realización de las pruebas al sistema, genera el documento de aceptación. Primeramente se planifica y se define al sistemas: se crea el guion técnico de los productos multimedia, después se concibe el sistema, se describen las historias de usuario junto con el cliente, se realiza la lista de reserva del producto, la lista de riesgos, se confecciona el modelo de diseño y se crea un modelo de historias de usuario del negocio. Al concluir la actividad se procede a crear el prototipo del sistema. La segunda actividad es desarrollar el

sistema, donde se confecciona los casos de prueba de aceptación, el cronograma de producción, se establece el estándar de codificación generado por la propuesta de arquitectura. Se crea el plan de release y se establecen las tareas de ingeniería, después se implementan las historias de usuario. Luego se realizan las pruebas del sistema y se entrega con los artefactos siguientes: manual de desarrollo, manual de usuario, manual de identidad. Posteriormente se realiza el mantenimiento del sistema de acuerdo a los cambios que se establecieron y se procede a realizar de nuevo el proceso.

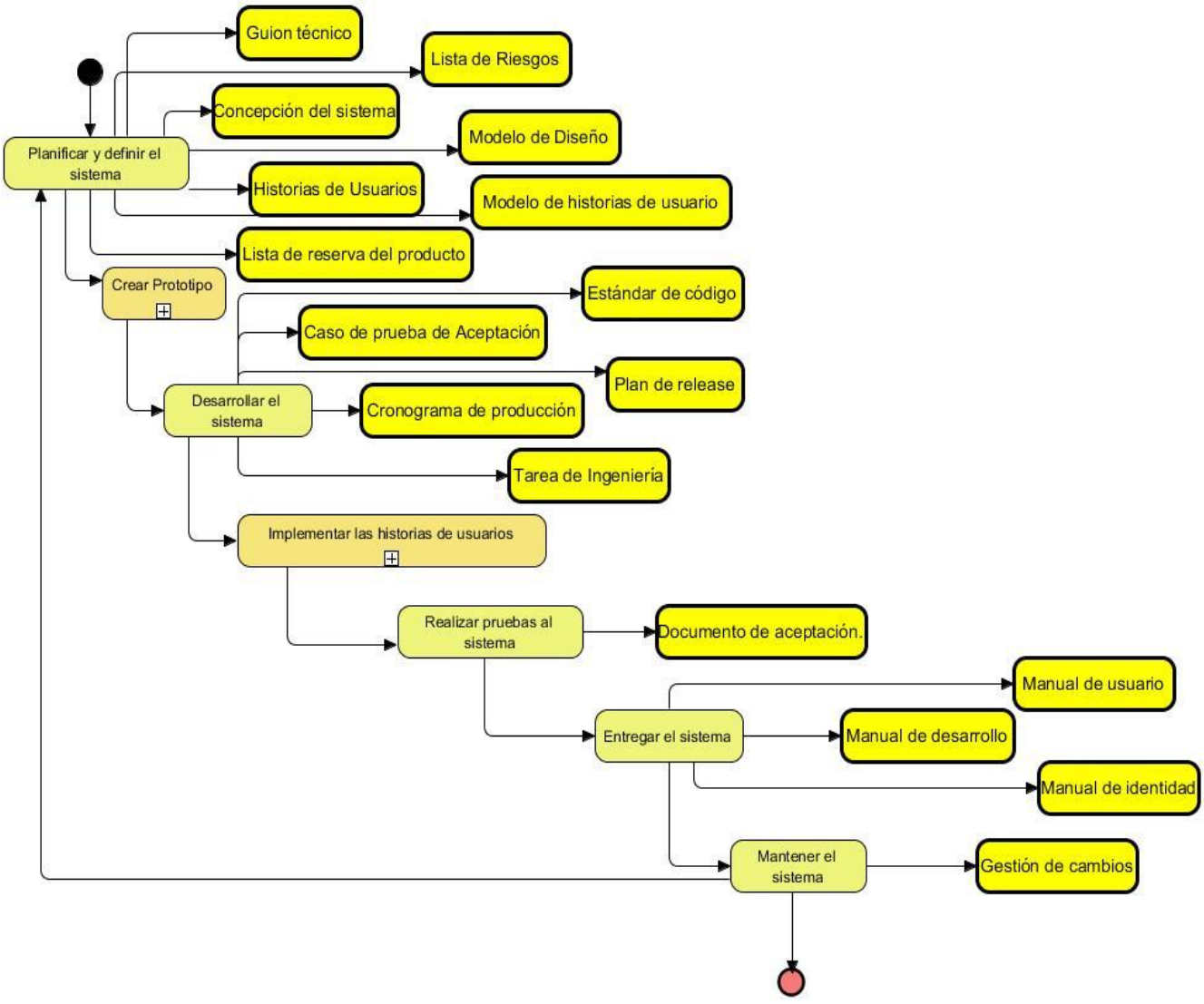


Fig. 9. Vista de proceso para multimedia según SXP.

2.3.2.3-Vista de Datos

En la vista de datos se incorpora la estructura del modelo de datos. La forma de almacenar las medias dentro del sistema. El cómo es cargado los datos externos del sistema. Cuenta con una clase .js que carga de acuerdo al tipo de media la dirección url en el XML, para poder ser mostrado posteriormente en la vista del sistema.

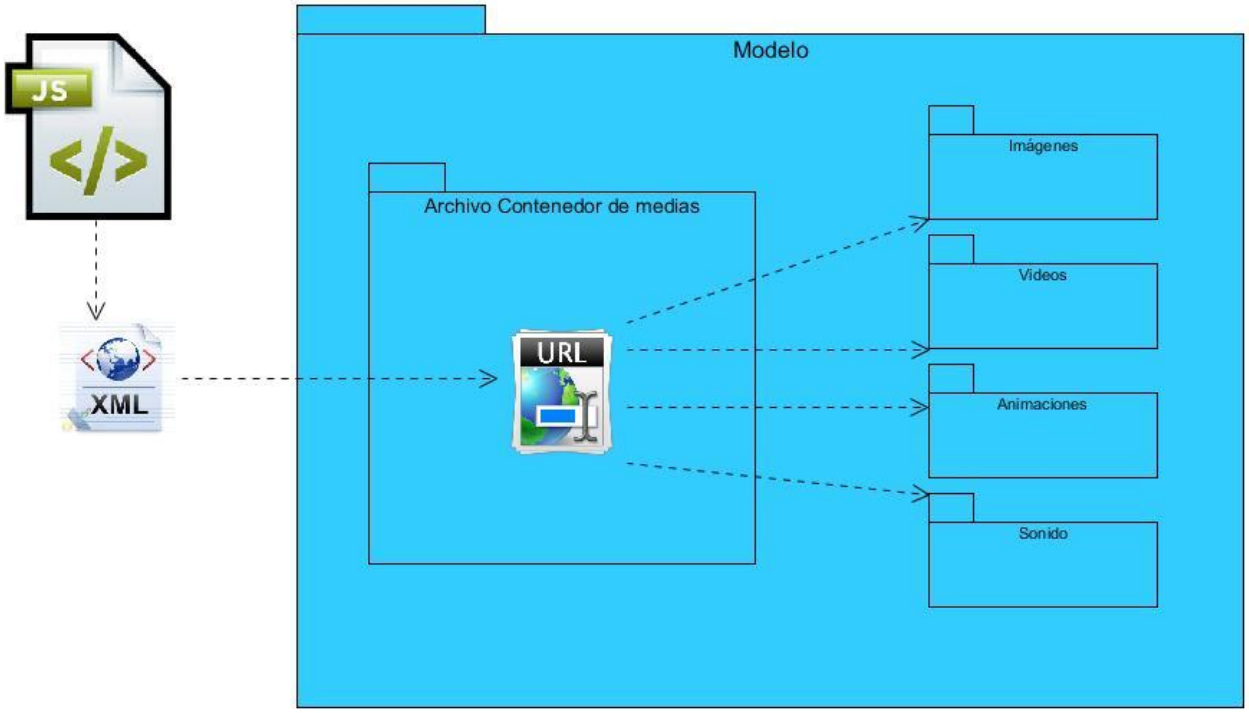


Fig. 10. Vista de datos.

Estructura del XML

<contenido>

//almacena todas las imágenes que integra el sistema, con la url y su descripción.

<imagenes>

<imagen href="../paq_Modelo/images/recursos/acuario.jpg">

<descripcion>Multimedia Acuario Nacional</descripcion>

</imagen>

</imagenes>

//almacena la url de los videos que va a contener el sistema.

<videos>

<video href="../paq_Modelo/video/drupal.webm" id="v1"></video>

<descripcion1>Tutorial de Drupal</descripcion1>

</videos>

// una lupa para un buscador, aparece en toda la aplicación.

<buscadores>

<buscador href="../paq_Modelo/images/lupa.jpg"></buscador>

</buscadores>

</contenido>

2.3.2.4-Vista de Integración

De acuerdo con las características comunes y los atributos de calidad que deben cumplir los sistemas multimedia se escoge de la familia de estilos más representativos mundialmente el estilo Llamada y Retorno, que a la vez son utilizados para sistemas que sean modificables y escalables. Como modelo arquitectónico se estableció dentro de esta familia el patrón Modelo-Vista-Controlador. Este patrón tiene como ventaja la adaptación al cambio, ya que los requisitos o peticiones del cliente pueden cambiar a medida del desarrollo del producto. Como estrategia de diseño arquitectónico se basa en confeccionar una arquitectura para una línea de producto. Esta estrategia comprende un conjunto de productos que comparten una colección de rasgos que satisfacen las necesidades de una determinada área de trabajo. En la arquitectura de Microsoft este modelo soporta un conjunto de lineamientos, herramientas y patrones arquitectónico específicos para aplicaciones de líneas de negocios. Las aplicaciones multimedia se integran de forma que los componentes ya sean clases, módulos, sistemas, objetos, entre otros, se puedan volver a utilizar y que cumplan con adaptarse a los otros sistemas.

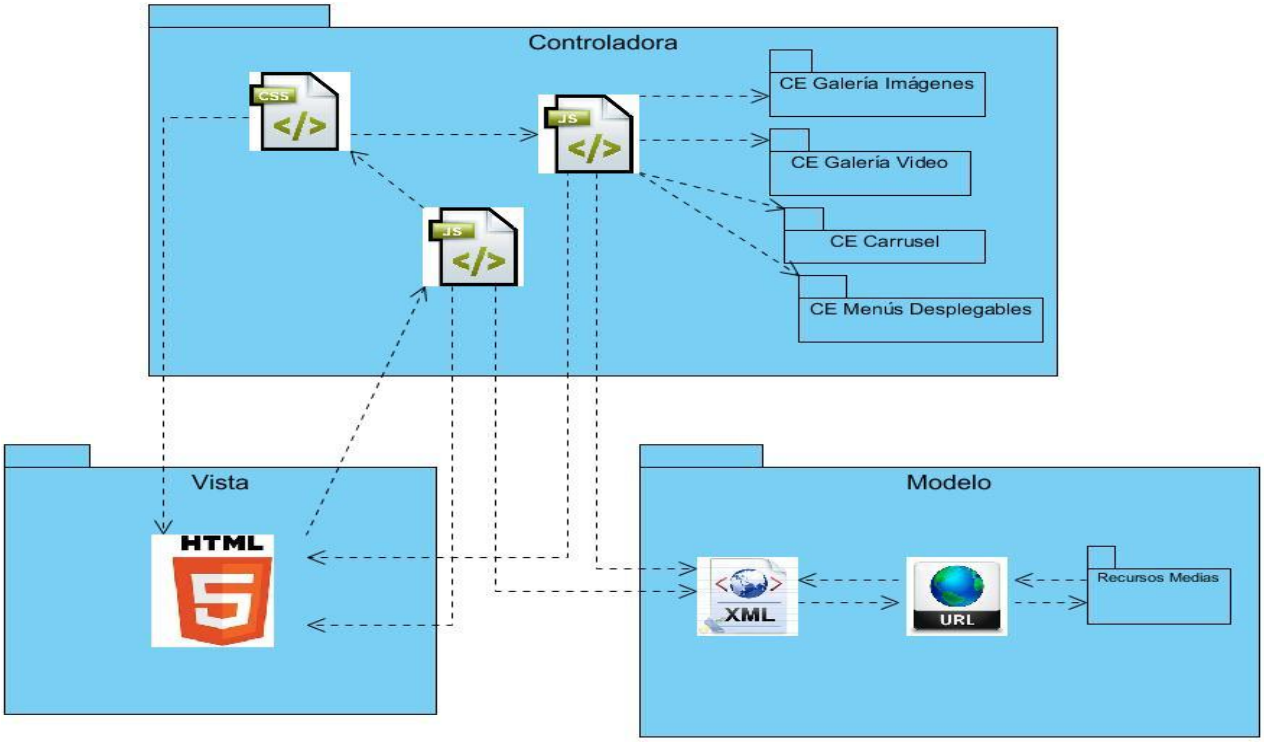


Fig. 11. Vista de integración.

2.3.2.5-Vista de Despliegue e Infraestructura

Los productos multimedia son sistemas centralizados, que solo necesitan de una estación de trabajo para ejecutarse correctamente. No es necesario realizar vista de despliegue, ya que si son aplicaciones web pero con tecnología multimedia solo es necesario un navegador para visualizarse. A continuación se explica la vista de infraestructura de forma general para todos los productos multimedia.

Con la tendencia de la migración del software propietario al software libre, se quiere efectuar un cambio en el plano tecnológico del proceso de desarrollo de los productos multimedia. Conjunto a las características de las herramientas de desarrollo multimedia utilizadas en el Centro, por tener licencia propietaria, se recomienda la incorporación de herramientas web para el desarrollo de este tipo de software. Las aplicaciones web son factibles porque se pueden desarrollar de forma fácil y permiten la facilidad de ejecución a la hora de utilizarlas como productos multimedia. El software multimedia no son productos distribuidos por lo que no es esencial la presencia de servidores y base de datos para almacenar el contenido.

2.3.2.6-Estructura de carpetas a tener en cuenta para el desarrollo de aplicaciones con tecnología multimedia

En la siguiente figura se encuentra la estructura de carpetas del prototipo no funcional implementado, la cual se realiza de acuerdo al patrón y estilo arquitectónico escogido. Se crean tres carpetas llamada paq_Vista, paq_Controladora y paq_Modelo, donde simula cada capa del Modelo-Vista-Controlador. La capa Vista (paq_Vista) contiene todas las clases .html que permite la comunicación entre el usuario y el sistema. En ella se cargan cada clase .js y .css mediante las etiquetas <script> y <link>. La capa Controlador (paq_Controladora) incluye por carpetas todos los componentes multimedia externo a la aplicación. A parte se crean dos directorios que son los que controlan el sistema, llamados: JS (almacena los .js) y CSS (almacena los .css), en ellos se integran las clases que decoran el prototipo y la clase que integra los componentes externos a la aplicación. En la capa Modelo (paq_Modelo) se almacenan todas las medias que va a contener el producto, mediante el documento .XML, que almacena las direcciones y atributos de cada una de las medias.

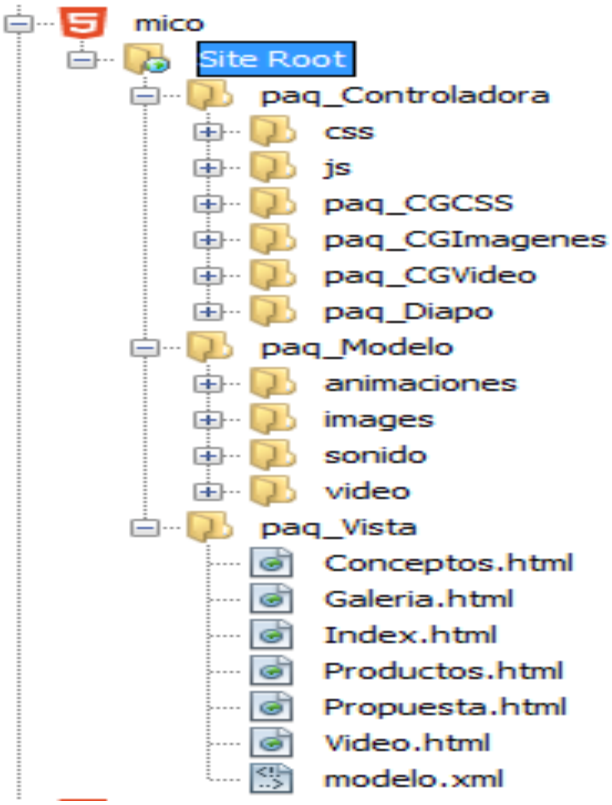


Fig. 12. Estructura de carpetas para los productos multimedia

2.3.2.7-Vista de presentación del prototipo no funcional para probar la propuesta de arquitectura

Un prototipo de interfaz de usuario es una representación visual de lo que se desea implementar, se utiliza para que el cliente pueda refinar sus necesidades y comunicarlas al desarrollador.

A continuación se presentan algunas interfaces de usuario del prototipo no funcional:

Fig. 13. Prototipo pantalla de Inicio.

Fig. 14. Prototipo pantalla Galería de Imágenes.

Fig. 15. Prototipo pantalla Introducción.

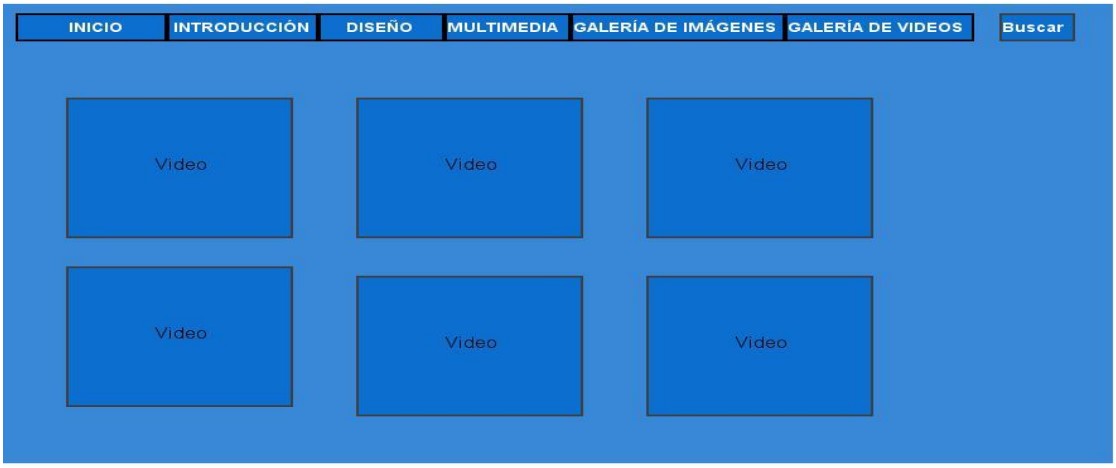


Fig. 16. Prototipo Galería de Video

2.4. Atributos de calidad

El diseño preliminar es evaluado con ciertos atributos de calidad. A partir de la búsqueda y análisis de la información correspondiente a la calidad de un software, se profundizan en los aspectos que definen las características principales que la arquitectura debe soportar. Para lograr una buena calidad de la propuesta se tuvo en cuenta la vinculación que hay entre patrones, estilos y las características más comunes que deben tener los sistemas con tecnología multimedia (5).

Los atributos definidos que se tienen en cuenta para la calidad de la propuesta son los siguientes:

Reusabilidad

Es la capacidad de diseñar un sistema de forma tal que su estructura o partes de sus componentes puedan ser reutilizados en futuras aplicaciones (5).

Modificabilidad

La Modificabilidad hace referencia a la habilidad de conocer cambios futuros del sistema. Es un atributo de calidad que asocia a los diferentes patrones arquitectónicos principalmente al Modelo-Vista-Controlador (5).

Escalabilidad

Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental (5).

Desempeño

Grado en el cual un sistema o componente cumple con su funcionalidad, dentro de ciertas restricciones dadas, como velocidad, exactitud, uso de memoria (5).

Interoperabilidad

Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema (5).

Portabilidad

Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación (5).

2.5. Conclusiones parciales

De acuerdo al diseño de la propuesta se concluye que para la realización de la arquitectura basada en componentes con tecnología multimedia para el centro FORTES, se hace uso del estilo arquitectónico llamada y retorno para representar el comportamiento de la estructura de los sistemas multimedia. El patrón arquitectónico Modelo-Vista-Controlador se escoge como estructura jerárquica. Como patrones de diseño se escogen el Decorator para la capa de las vistas y el Observer para la actualización o cambios de comportamiento entre eventos en la controladora. Se representan las cinco vistas de arquitecturas y los diagramas de componentes que integran los productos multimedia**.** Los atributos de calidad que deben tener en cuenta los productos multimedia son: la reusabilidad, modificabilidad, escalabilidad, portabilidad, el desempeño y la interoperabilidad. Además, se pone de manifiesto las reglas que se siguen para la escritura del código fuente y el diseño de la propuesta.

Capítulo 3. Evaluación de la Arquitectura Propuesta

3. Introducción

Al evaluar una arquitectura se tiene en cuenta la verificación de los factores de calidad desde una perspectiva arquitectónica. Se identifican los puntos críticos que presenta la arquitectura con respecto a los factores y peticiones del usuario (49). La evaluación de esta permite obtener una visión de las decisiones más riesgosas asociadas al software en cuestión y además, aquellas que son buenas y correctas. En el presente capítulo se abordará la evaluación de la arquitectura propuesta como solución a la problemática existente. Se describirán brevemente algunos métodos, definiendo elementos de vital importancia para el desarrollo de dicha evaluación, entre los que se encuentran las diferentes técnicas e instrumentos utilizados para evaluar la propuesta.

3.1. ¿Por qué evaluar una arquitectura?

Uno de los factores que determina el éxito o el fracaso de un software, es su arquitectura. Por lo tanto mientras más temprano se descubra el problema en un software, mejor. Es más factible arreglar un error durante la fase de requerimiento o diseño, que arreglarlo en la fase de verificación ya que el costo de arreglar ese error es menor. Dado que la arquitectura es un producto temprano de la fase de diseño, también determina la estructura del proyecto: configuración, agenda y presupuesto, alcance, entre otros aspectos. Por eso sería más factible cambiar la arquitectura, antes que otros artefactos basados en la misma se establezcan (50).

3.2. Etapas en que se evalúa una arquitectura

Generalmente, la evaluación de la arquitectura ocurre después que esta ha sido especificada, pero antes de comenzar la implementación. En un proceso iterativo y/o incremental, la evaluación se puede realizar al final de cada ciclo. Sin embargo, uno de los atractivos de la evaluación de arquitecturas es que se puede efectuar en cualquier etapa de la vida de la misma. En particular, existen dos variantes útiles: evaluación temprana y evaluación tardía (50).

Evaluación temprana: En esta evaluación no se tiene que esperar a que la arquitectura esté completamente especificada. La misma puede ser utilizada en cualquier etapa del proceso de creación del software (50).

Evaluación tardía: En esta variante se establece la evaluación cuando la arquitectura está definida y el proyecto se encuentra implementado. Este caso ocurre cuando la organización hereda un sistema legado. La técnica para evaluar un sistema legado es la misma que para evaluar un sistema recién nacido. Una evaluación es útil para entender el sistema legado, y saber si este cumple con los requerimientos de calidad y comportamiento (50).

3.3. Técnicas de evaluación de arquitecturas

Bosch en el año 2000, plantea que las técnicas de evaluación utilizadas para atributos de calidad requieren grandes esfuerzos por parte del ingeniero de software para crear especificaciones y predicciones. Estas técnicas requieren de información del sistema a desarrollar que no está disponible durante el diseño arquitectónico, sino al principio de diseño detallado del sistema (5).

En la actualidad existen diferentes técnicas para evaluar arquitecturas que permiten hacer una evaluación cuantitativa sobre los atributos de calidad a nivel arquitectónico, pero se tienen pocos medios para predecir el máximo (o mínimo) teórico para las arquitecturas de software. Sin embargo, debido al costo de realizar este tipo de evaluación, en muchos casos los arquitectos de software evalúan cuantitativamente, para decidir entre las alternativas de diseño. Bosch en el año 2000 propone diferentes técnicas de evaluación de arquitecturas de software, a saber: evaluación basada en escenarios, evaluación basada en simulación, evaluación basada en modelos matemáticos y evaluación basada en experiencia (5).

3.3.1-Evaluación basada en escenarios

De acuerdo con Kazman, un escenario es una breve descripción de la interacción entre algunos de los involucrados en el equipo de desarrollo y el sistema. Consta de tres partes: el estímulo, el contexto y la respuesta. El estímulo es la parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. Puede incluir ejecución de tareas, cambios en el sistema, ejecución de pruebas, configuración. El contexto describe lo que sucede en el sistema al momento del estímulo. La respuesta describe, a través de la arquitectura, como debería responder el sistema ante del estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado (5).

Los escenarios proveen un vehículo que permite concretar y entender atributos de calidad, entre las ventajas de su uso está:

- ♣ Son simples de crear y entender.

- ♣ No requieren de entrenamiento y su costo es muy pequeño.
- ♣ Son efectivos.

Actualmente las técnicas basadas en escenarios cuentan con dos instrumentos de evaluación relevantes, a saber: el *Utility Tree* (Árbol de Utilidad) propuesto por Kazman y los *Profiles* (Perfiles), propuesto por Bosch.

3.3.2-Evaluación basada en simulación

Bosch define que la evaluación basada en simulación utiliza una implementación de alto nivel de la arquitectura de software (5).

Este proceso de evaluación sigue los siguientes pasos:

- ♣ 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.

La exactitud de los resultados de evaluación depende, de la exactitud del perfil utilizado para evaluar el atributo de calidad y de la precisión con la que el contexto del sistema simula las condiciones del mundo real.

3.3.3-Evaluación basada en modelos matemáticos

Bosch expresa que la evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. Permitiendo así una evaluación estática de los modelos de diseño arquitectónico, y se representa como una alternativa a la simulación, dado que evalúan el mismo atributo (5).

El proceso de evaluación basada en modelos matemáticos sigue los siguientes pasos:

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

3.3.4-Evaluación basada en experiencia

Bosch plantea que en ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas del diseño. Aunque todas estas experiencias se basan en

evidencia anecdótica; es decir 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 (5).

Existen dos tipos de evaluación basada en experiencia, la evaluación informal, que es realizada por los arquitectos de software durante el proceso de diseño, y la realizada por equipos externos de evolución de arquitecturas.

3.4. Métodos de evaluación de arquitecturas

Los métodos han sido otras de las herramientas desarrolladas con el objetivo de evaluar una arquitectura de software (5).

Kazman expresa que no existían métodos de utilidad general para evaluar una arquitectura de software. De existir estos métodos eran incompletos y no repetibles, lo que no brindaba mucha confianza al utilizarlos. En virtud de esto, se han propuesto diferentes métodos de evaluación. A continuación se explican algunos de los más importantes (5).

3.4.1-Método de análisis de arquitectura de software (Software Architecture Analysis Method, SAAM)

El método SAAM constituye el primer método de evaluación de arquitectura ampliamente promulgado. Este fue creado para evaluar la modificabilidad de la arquitectura de software en todo su alcance (5).

Tiene como objetivo evaluar un conjunto de atributos de calidad que debe lograr la arquitectura de software, a través del uso de escenarios. En la práctica ha demostrado ser útil para la rápida evaluación de atributos de calidad como: modificabilidad, portabilidad y extensibilidad. Al evaluar una arquitectura, SAAM indica los puntos de fortalezas y debilidades de las arquitecturas que no cumple con los requisitos de modificabilidad. SAAM se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro. Como entrada principal, es necesaria alguna forma de descripción de la arquitectura a ser evaluada. Las salidas de la evaluación de SAAM son las siguientes (5):

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

Con la aplicación de este, si el objetivo es evaluar una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad. Para el caso en el que se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones.
SAAM comprende seis pasos que pueden ser consultados en el **Anexo 6** (5).

3.4.2-Método de análisis de interacciones de la arquitectura de software (Architecture Trade-off Analysis Method, ATAM)

Según Kazman, el Método ATAM está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. El nombre del método surge del hecho de que releva la forma en que la arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros; esto es, los tipos de acuerdos que establecen entre ellos (5).
El desarrollo del método ATAM se realizó sobre la base del SAAM. El propósito general de su diseño fue permitir la evaluación de un mayor número de atributos. Otro de los aspectos que abarca este método es que no se encuentra reflejado en su antecesor, es la exploración de las interdependencias que se establecen entre los distintos atributos.
ATAM analiza la arquitectura del software, satisfaciendo los atributos de calidad: modificabilidad, portabilidad, extensibilidad e integridad. Proporciona información sobre el cumplimiento de cada atributo de calidad teniendo en cuenta las interdependencias que se establecen entre ellos (49).
El método de evaluación ATAM comprende nueve pasos, agrupados en cuatro fases que pueden ser consultados en el **Anexo 7**.

3.4.3-Método de Análisis de Diseños Intermedios (Active Reviews for Intermediate Designs, ARID)

El método 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. Según los autores, ARID es un híbrido entre Active Reviews (ARD) y Architecture Trade-Off Method (ATAM) descrito anteriormente (5).

ADR es utilizado para la evaluación detallada de diseños de unidades del software como componentes o módulos. Este método utiliza preguntas que giran alrededor de la calidad, completitud de la documentación, suficiencia, ajuste y la conveniencia de los servicios que provee el diseño propuesto. Los métodos ADR y ATAM proveen características útiles para el problema de evaluación de diseños preliminares, dado que ninguno por si solo es conveniente. Con ADR, los involucrados reciben la documentación detallada y completan cuestionarios, cada uno por separado, donde las respuestas que se obtienen en el desarrollo son convenientemente confiables. El método ATAM, está orientado a la evaluación de toda una arquitectura. Donde el método ATAM hace uso de escenarios generados por los involucrados con el sistema (5).

De la combinación de ambos surge ARID, para efecto de la evaluación temprana de los diseños de una arquitectura de software. El método de evaluación ARID comprende nueve pasos, agrupados en dos fases que pueden ser consultados en el **Anexo 8.**

3.4.4-Método de Evaluación por Criterios de Expertos

El método de evaluación de expertos se emplea en investigaciones teóricas, con el fin de evaluar la calidad y efectividad del modelo teórico propuesto y comprobar la validez de los instrumentos de investigación que serán aplicados. También es empleado en las investigaciones experimentales, antes de someter a la prueba de la experiencia, el Modelo Teórico propuesto (51).

Los expertos que evalúan las investigaciones, deben cumplir con las siguientes características:

- La calificación científico-técnica.
- La experiencia profesional.
- La preparación, conocimiento y especialización en el tema objeto de investigación, gustos personales (51).

Las opiniones individuales de los expertos se analizan como magnitudes aleatorias, se analizan mediante métodos estadísticos. De esta manera la evaluación se realiza a través de un sistema de procedimientos organizativos, lógicos, estadísticos y matemáticos dirigidos a obtener información de los especialistas.

3.4.5-Método Diseño y Uso de Arquitecturas de Software propuesto por Bosch

Bosch plantea, en su método de diseño de arquitecturas de software, que el proceso de evaluación debe ser visto como una actividad iterativa, que forma parte del proceso de diseño, también iterativo. Una vez que la arquitectura es evaluada, pasa a una fase de transformación, asumiendo que no satisface todos los

requerimientos. Luego, la arquitectura transformada es evaluada de nuevo (5). El proceso de evaluación propuesto se divide en dos etapas, que son presentadas en el **Anexo 9**.

3.5. Evaluación de la arquitectura

Para la evaluación de la arquitectura de software se selecciona como método de evaluación: ATAM, que utiliza la técnica de evaluación basada en escenarios con el instrumento Árbol de Utilidad. Es el más completo y fácil de aplicar. A continuación se presenta la evaluación de la arquitectura propuesta según las fases del método de evaluación escogido.

Fase 1: Presentación

Paso 1. Presentación del ATAM: ATAM es un método de evaluación que permite identificar el cumplimiento de cada atributo de calidad teniendo en cuenta las relaciones que se establecen entre estos. Presenta un proceso de evaluación de forma organizada que establece una secuencia de actividades que involucran funcionalidades, requisitos de entradas y los resultados de cada actividad. No utiliza un modelo de calidad para medición de los atributos de calidad. Aunque, mediante el instrumento del árbol de utilidad, propone un nivel de refinamiento de esos atributos que contiene métricas con criterios de máximo y mínimo. Para mejora del método se realiza el manejo de la identificación de los riesgos de la arquitectura diseñada, mediante los puntos de sensibilidad, de desventajas y no riesgos que comprenden estos atributos en los sistemas.

Paso 2. Presentación de las metas del negocio: Estandarizar la estructura de los productos multimedia para el proceso de desarrollo de software. Diseñar una arquitectura que contemple componentes, que entre sus propiedades sean reutilizables y genéricos.

Restricciones:

- Los productos implementados deben contener al menos: imágenes, textos y videos para considerarse multimedia.
- Los productos deben de ser portables y adaptables a otros sistema.
- Los componentes creados deben de ser reutilizables e interoperables.
- Los sistemas deben de estar expuestos a sufrir cambios futuros.
- La arquitectura debe de comprender el atributo de calidad: la escalabilidad.

Paso 3. Presentación del negocio: Se presenta la propuesta de arquitectura diseñada en el Capítulo 2 de acuerdo a los procesos descritos. El objetivo esencial de la propuesta es diseñar una arquitectura que pueda estandarizar la estructura de los productos con tecnología multimedia del centro FORTES de la UCI. Se escogen los atributos de calidad por su prioridad de utilidad en el sistema.

Fase 2: Investigación y análisis

Paso 4. Identificación de los enfoques arquitectónicos: la evaluación de la arquitectura detecta los siguientes elementos que se analizarán más tarde: se utilizó para el desarrollo de la propuesta no funcional de la arquitectura el IDE de desarrollo Netbeans en su versión 7.4. Para la creación de las interfaces de usuario se utilizó el lenguaje HTML en su versión 5, para la carga de los datos un documento XML donde se almacenan las referencias de los contenidos multimedia. Se utilizó la librería o framework de Java Script JQuery en su versión 2.0. Dentro del patrón arquitectónico que se seleccionó en la propuesta se encuentra el Modelo-Vista-Controlador. Patrones de diseño propuestos se encuentran Decorator y Observer.

Paso 5. Generación del árbol de utilidad: se obtienen los atributos de calidad que engloban la utilidad del sistema especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establecen la prioridad entre ellos.

En este paso se obtienen los escenarios identificados en el paso 5 y las propuestas arquitectónicas que cumplen con estos escenarios, donde se documenta detalladamente la medida en la cual son adecuados el uno para el otro, y logra la meta del equipo de evaluación de estar convencidos que la propuesta instanciada en la arquitectura que se evalúa, es la apropiada para satisfacer los requerimientos de un atributo específico. A continuación se mostrará el Árbol de Utilidad Propuesto y algunos escenarios identificados.

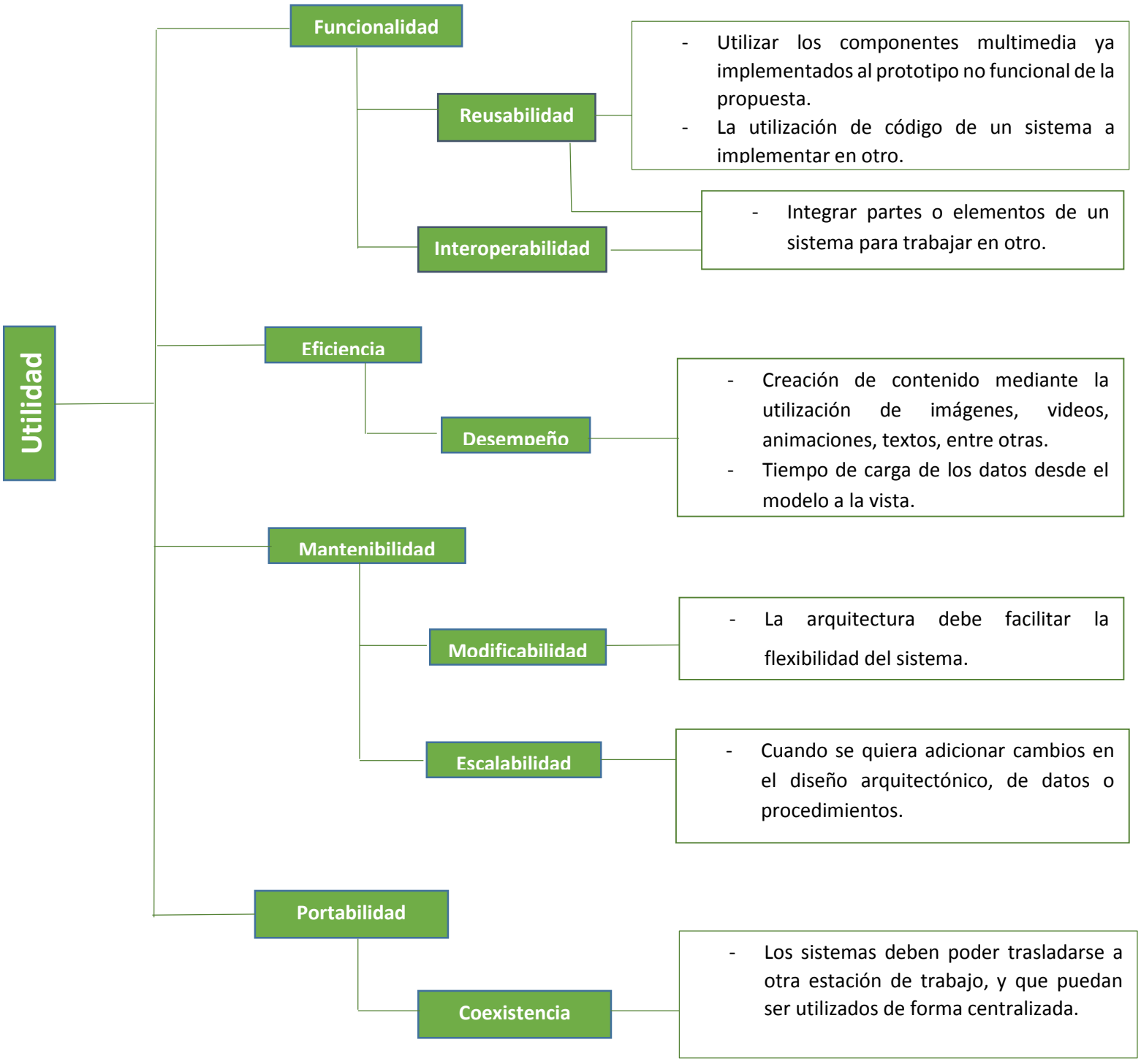

Utilidad
Funcionalidad
Reusabilidad
- Utilizar los componentes multimedia ya implementados al prototipo no funcional de la propuesta.
- La utilización de código de un sistema a implementar en otro.
Interoperabilidad
- Integrar partes o elementos de un sistema para trabajar en otro.
Eficiencia
Desempeño
- Creación de contenido mediante la utilización de imágenes, videos, animaciones, textos, entre otras.
- Tiempo de carga de los datos desde el modelo a la vista.
Mantenibilidad
Modificabilidad
- La arquitectura debe facilitar la flexibilidad del sistema.
Escalabilidad
- Cuando se quiera adicionar cambios en el diseño arquitectónico, de datos o procedimientos.
Portabilidad
Coexistencia
- Los sistemas deben poder trasladarse a otra estación de trabajo, y que puedan ser utilizados de forma centralizada.

Tabla 2. Escenario # 1.

Escenario # 1	**Utilizar los componentes multimedia ya implementados en prototipo no funcional de la propuesta.**			
Atributo(s)	Funcionalidad-Reutilización.			
Ambiente	Utilización de componentes ya implementados.			
Estímulo	La utilización de componentes externos para incorporar en la implementación de sistemas multimedia.			
Respuesta	Se integra un nuevo componente multimedia que pertenece a otro sistema y se reutiliza en la implementación de la propuesta.			
Decisiones arquitectónicas	Pto de sensibilidad Pto Trade-Off Riesgo No Riesgo			
Implementar un archivo para cada componente multimedia.				NR1
Explicación	La utilización de elementos ya creados, facilita el trabajo, reduce el tiempo y costo en el desarrollo de productos informáticos.			

Tabla 3. Escenario# 2.

Escenario # 2	**La utilización de código de un sistema a implementar en otro.**
Atributo(s)	Funcionalidad-Reutilización.
Ambiente	Refactorización de los sistemas.
Estímulo	La reutilización de códigos entre documentos .js, .css, .html, contribuyendo a la refactorización de los sistemas.
Respuesta	Se reutiliza el código de un archivo a otro, para lograr la optimización de las funcionalidades.

Decisiones arquitectónicas	Pto de sensibilidad	Pto de Trade-Off	Riesgo	No Riesgo
Implementación de archivos que se generan para las interfaces de usuario.		TR1		
Explicación	La utilización de un mismo código implementado en todas las páginas interfaces contribuye a la eficiencia del desarrollo de software, permitiendo englobar los elementos comunes de los sistemas y ahorra líneas de código a implementar.			

Paso 6. Análisis de los enfoques arquitectónicos

En base de los resultados del establecimiento de prioridades del paso anterior (**Ver Anexo 10 para los demás escenarios**), se analizan los elementos del paso 4. Se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance.

Riesgos Arquitectónicos

R1: Lo anteriormente planteado en S1 resulta ser un riesgo en esta decisión arquitectónica, ya que puede darse el caso que Flash como herramienta de aplicaciones multimedia no soporta funcionalidades hechas en Java Script, por lo que las aplicaciones no pueden reconocer este tipo de lenguaje.

R2: No permite la flexibilidad de los sistemas y al mismo tiempo presenta un riesgo para la aplicación, esto ocurre cuando se quiera agregar cambios futuros a la aplicación.

R3: Por la razón del punto de sensibilidad tres resulta ser un riesgo para cargar los datos desde la capa modelo hacia la vista, ya que si se realizan cambios de archivo CSS no se puede cargar los recursos multimedia contenidos en la capa modelo.

Puntos de sensibilidad

S1: Un punto de sensibilidad sería la implementación de componentes que presenten nuevas funcionalidades visuales de acuerdo a la utilización de los plugins de JQuery o Bootstrap en los sistemas. JQuery es más viejo, pero mejor desarrollado

S2: Un punto de sensibilidad es la utilización de componentes desarrollados en un lenguaje de programación, donde la aplicación que soporta el sistema no lo reconoce.

S3: La decisión arquitectónica descrita presenta punto de sensibilidad, ya que si se define un archivo CSS para la carga de los contenidos al hacer modificaciones a los componentes puede proveer errores en el sistema.

S4: Esta decisión arquitectónica tiene un punto de sensibilidad que es la utilización del documento XML en la capa modelo, esto dificulta lo que es el dinamismo de las aplicaciones.

S5: Cuando existen muchas actualizaciones, y el número de notificaciones que halla en el sistema es grande, esto pone lento los procedimientos de la aplicación.

S6: Esta decisión arquitectónica es un punto de sensibilidad. Provoca que el sistema pueda incluir elementos externos a la aplicación. Esto facilita el trabajo y ayuda a la integración y reutilización de los componentes.

Puntos de Trade Off

TR1: Esta decisión arquitectónica favorece a la reutilización del código en cada clase interfaz, optimiza el desarrollo de las aplicaciones. Sin embargo, puede ser poco aceptable en cuanto se vaya a integrar a otros sistemas en dependencia de las herramientas y lenguajes a desarrollar.

No Riesgos

NR1: Para añadir los componentes multimedia ya implementados al prototipo no funcional de la propuesta se implementará un componente multimedia para cada clase permitiendo reutilizar la misma en otros sistemas.

NR2: Esta decisión arquitectónica no representa un riesgo en el desarrollo de estos productos, sino que adiciona las funcionalidades y ayuda a la interactividad dinámica de los sistemas.

NR3: Este patrón permite crear menos archivos para lograr decorar la vista del sistema. Permite el dinamismo y contribuye a la refactorización en el desarrollo de los productos. Ofrece desarrollar la reutilización de componentes multimedia.

NR4: La decisión arquitectónica reflejada no tiene riesgos, ya que mejora y organiza el proceso y desarrollo de las aplicaciones. El uso de este patrón contribuye a que se pueda realizar cambios futuros en el sistema, permite la modificabilidad, ya que las capas de este modelo arquitectónico son independientes una de otras, aunque se comunican entre sí.

NR5: Esta decisión arquitectónica se utiliza cuando se hacen cambios en el comportamiento de eventos que están en un archivo que de él dependen otros. Estos archivos dependientes pueden actualizarse ellos mismos al recibir la notificación del evento actualizado en el Observer.

NR6: Esta decisión arquitectónica es importante, ya que permite por su estructura que se pueda modificar cualquier funcionalidad de alguna capa, sin tener que afectar las otras.

NR7: No es un riesgo por lo que al proponer una clase para cada componente es más fácil incorporarlo al sistema sin que haya algún error o riesgo para las otras partes.

NR8: Las tecnologías multimedia se basan en integrar muchas imágenes, textos, videos, animaciones. Para ser mejorar la implementación de estos productos y su ejecución, no es un riesgo tratar de implementarlas con tecnología web, ya que al programar estos sistemas con sólo tener un editor de texto en tu ordenador ya puedes incorporar código a la aplicación. Además, puede ser ejecutable sin tener un programa en específico, esto se logra a través de un navegador.

NR9: La creación de archivos .js y .css no representa un riesgo porque no dificulta los atributos de calidad planteados. Permite estandarizar la implementación de los productos y mejora la reutilización e interoperabilidad de los componentes al ser ensamblados.

N10: Esta decisión no presenta ser un riesgo en los sistemas, sino que mejora la eficiencia de desarrollo de los sistemas y ofrece organizar de forma tal que se entienda en donde se almacena cada subcomponente de cada elemento multimedia.

Fase 3: Pruebas

Paso 7. Lluvia de ideas y establecimiento de prioridad de escenarios

Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios. Mientras que la generación del árbol de utilidad es fundamental para entender cómo el arquitecto percibe y maneja las guías

arquitectónicas de los atributos de calidad, el propósito de la lluvia de ideas de escenarios es tomarle el pulso a un grupo de especialistas interesados en el tema.

Nro.	Escenario	Votos	Atributo de Calidad
9	Crear una clase .js para cargar los componentes externos que se integran a la capa controladora de la aplicación.	4	Interoperabilidad
10	En la capa controladora se distribuyen los componentes por carpetas.	3	Desempeño

Paso 8. Análisis de los enfoques arquitectónicos

Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento. Se incrementa el árbol de utilidad con los escenarios propuestos en el paso 7. **Ver Anexo 11.**

Tabla 4. Escenario # 9.

Escenario # 9	**Crear un archivo .js para cargar los componentes externos que se integran a la capa controladora de la aplicación.**			
Atributo(s)	Funcionalidad-Interoperabilidad.			
Ambiente	Integrar componentes externos al sistema.			
Estímulo	Para cargar componentes ya implementados para integrarlos a los sistemas.			
Respuesta	La implementación de una clase java script que permita controlar todos los elementos integrados al software.			
Decisiones arquitectónicas	Pto de sensibilidad Pto Trade-Off Riesgo No Riesgo			
Implementar un archivo .js que pueda integrar los componentes reutilizables en la aplicación.	S6			NR9

Explicación	El archivo .js permite controlar todos los componentes en la aplicación, incorpora las funcionalidades de estos elementos en el sistema.

Tabla 5. Escenario # 10.

Escenario # 10	**En la capa controladora se distribuyen los componentes por carpetas.**			
Atributo(s)	Eficiencia-Desempeño.			
Ambiente	La organización de los componentes multimedia.			
Estímulo	Se organiza las carpetas creadas para cada tipo de componente.			
Respuesta	Según el componente multimedia creado se almacena en una carpeta que incluye el nombre del componente y un número.			
Decisiones arquitectónicas	Pto de sensibilidad	Pto Trade-Off	Riesgo	No Riesgo
Crear carpetas en la capa controladora para cada tipo de componente multimedia.				NR10
Explicación	La creación de archivos que almacenen los componentes ya implementados, permite la organización de todos los elementos que integre la capa controladora. Permite organizar las carpetas de forma tal que se entienda el tipo de componente.			

Fase 4: Reporte

Paso 9. Presentación de los resultados: Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes. Finalmente, haciendo uso del método de evaluación seleccionado ATAM se resume y presenta; las más importantes salidas:

- El árbol de utilidad genera como atributos: la funcionalidad del sistema en que los componentes sean reutilizables e interoperable. La eficiencia de la arquitectura con respecto al desempeño de los sistemas. La mantenibilidad que genera la modificabilidad y escalabilidad de los componentes de la arquitectura y

permite la flexibilidad de los sistemas multimedia para los cambios. La portabilidad influye en la coexistencia de los sistemas dentro de las estaciones de trabajo.

- El conjunto de escenarios priorizados se definen de acuerdo a los atributos de calidad definidos en el árbol de utilidad y las lluvias de ideas que implican a los especialistas interesados en el tema.
- Los riesgos descubiertos: fueron en la segunda fase tres riesgos y después de la lluvia de ideas ningún riesgo.
- Los no riesgos descubiertos: en la segunda fase de detectaron ocho no riesgos y después de la lluvia de idea se obtuvieron dos más.
- Los puntos de sensibilidad y de trade-off encontrados: se detectaron cinco puntos de sensibilidad y después de la lluvia de ideas se detectan uno más. Se encuentra un punto de balance.

  En total se detectan tres riesgos, diez no riesgos, seis puntos de sensibilidad y un trade-off. Por lo que se concluye que la arquitectura presenta riesgos en las decisiones arquitectónicas con respecto a los atributos de calidad: funcionalidad-interoperabilidad, en integrar componentes externos al sistema, reutilizados de la biblioteca de componentes. Los atributos de eficiencia-desempeño, presenta el riesgo al implementar en un archivo con código CSS las direcciones de los datos multimedia. Esto es una desventaja porque si se decide no utilizar este archivo.css se perderían las direcciones de las medias y no se mostrarían en el sistema. Por seguridad y por estructura entre capas, el documento XML creado no puede encontrarse en la capa modelo por lo que dificulta en la flexibilidad y seguridad de los datos, al ser mostrado a los usuarios. Esto es un riesgo para el desempeño y eficiencia de la aplicación.

3.6. Conclusiones parciales

En este capítulo se expusieron las principales técnicas y métodos utilizados para evaluar una arquitectura de software. Tras el estudio de los métodos de evaluación se seleccionó el método ATAM. Los resultados expuestos fueron la creación del árbol de utilidad y la adicción de nuevos escenarios. En general se encontraron dentro de las decisiones arquitectónicas tomadas tres riesgos, seis puntos de sensibilidad, un punto de Trade-Off y diez no riesgos por lo que se plantea que la arquitectura propuesta cumple con los atributos de calidad que se definieron en la propuesta diseñada. El prototipo no funcional de la propuesta de arquitectura es implementado de acuerdo al estilo y patrón arquitectónico escogido. Se utilizan componentes de otro sistema ya implementados para reutilizar en el prototipo de la propuesta. Se integran estos componentes al sistema y se estructuran en la capa controladora por carpetas.

Conclusiones Generales

Con el propósito de darle cumplimento al objetivo general y a la problemática planteada en el presente trabajo, se han llevado a cabo satisfactoriamente cada uno de los objetivos que se expusieron en el marco teórico de la investigación, por lo que se arriba a las siguientes conclusiones:

- Con el uso de la metodología SXP se logra la efectividad con respecto a la ingeniería de software y la gestión de proyectos.
- Se definen las herramientas y lenguajes de implementación y modelado de acorde a las características de la propuesta de solución.
- Se estableció una arquitectura basada en componentes para productos con tecnología multimedia enfocados a la web que permite estandarizar el proceso de desarrollo de este tipo de productos en el centro FORTES.
- Se obtiene un prototipo no funcional con tecnología web, que valida la estructura de la arquitectura para los productos multimedia.
- A partir del método ATAM se evalúa la arquitectura, lo que permitió identificar la utilidad de los atributos de calidad planteados, y posibles puntos de riesgos, así como constatar que dicha arquitectura fuera óptima y que cumple con las metas trazadas como reutilizar componentes e integrarlos nuevamente en otro software. Mantener la política de utilizar software libre para la confección de los productos multimedia.

Recomendaciones

Después de concluida la investigación se recomienda lo siguiente:

- Utilizar la documentación generada como estándar o guía en el desarrollo de proyectos con características similares.
- Realizar una evaluación más detallada de la arquitectura para identificar nuevos riesgos y debilidades, contribuyendo al fortalecimiento de la misma.

Referencias bibliográfica

1. **Portal de laHistoria | Portal de la Universidad de las Ciencias Informáticas. Universidad de las Ciencias Informáticas. [En línea] 2012. [Citado el: 29 de Enero de 2014.] http://www.uci.cu/?q=historia.**

2. MERINO SALVIA, RAÚL ERNESTO. *Propuesta de procedimientos para la gestión organizacional dentro de las Líneas de Producción de Software.* La Habana : s.n., 2009.

3. PRESSMAN, ROGER S. *Ingeniería del software.* La Habana : Mc Graw Hill, 2005. Sexta Edición.

4. Sommerville, Ian. *Ingeniería del software.* La Habana : Pearson Educacion, 2005.

5. Camacho, Erika, Cardeso, Fabio y Nuñez, Gabriel. *Arquitectura de software.Guia de estudio.* 2004.

6. Gil, José A San. Arquitectura Basada en Componentes. [En línea] [Citado el: 20 de Mayo de 2014.] http://es.scribd.com/doc/14704374/Arquitectura-Basada-en-Componentes.

7. Estilos de Componetes y Conectores. [En línea] https://www.u-cursos.cl.

8. CARLOS REYNOSO, Nocolás Kicillof. *Lenguaje de descripcion de Arquitecturas(ADL).* Universidad de Buenos Aires : s.n., 2004.

9. MICROSOFT. Editor de restricciones y configuración. Developer Network. [En línea] Noviembre de 2007. [Citado el: 11 de Febrero de 2014.] http://msdn.microsoft.com/es-es/library/ms181963(v=vs.90).aspx.

10. Aparicio, Roberto. *LA EDUCACION PARA LOS MEDIOS DE COMUNICACION .* 2004.

11. ELORRIAGA, KOSTANTZE, Naranjo, Cecilia y BOHÓRQUEZ, DEYANIRA. *Arquitectura de un sitio web para la enseñanza-aprendizaje de la representación ortogonal de volúmenes.* 2008.

12. Definicion de Barra de desplazamiento - ¿qué es Barra de desplazamiento? [En línea] 1998-2013. [Citado el: 31 de Enero de 2014.] http://www.alegsa.com.ar/Dic/barra%20de%20desplazamiento.php.

13. GARCIA, V, y otros. *Propuesta de una arquitectura software basada en componentes distribuidos para una celula CIM.* Facultad de Ciencias-Universidad de Salamanca : s.n.

14. Perovich, Daniel, Rodríguez, Leonardo y Vignaga, Andres. *Arquitectura de Sistemas de Información basados en Componentes sobre la Plataforma J2EE.* Uruguay, Montevideo : Instituto de Computación-Facultad de Ingenieria-Universidad de la República : s.n.

15. MARTINEZ, ING. ELENA y Cancio., Ing. Sergio Andres Perez. *Propuesta_arquitectura_basada_componentes_plataformas_gestion_procesos_desarrolladas.* Universidad de las Ciencias Informáticas. : s.n.

16. BALLESTER JIMÉNEZ, LIANET. *Propuesta de arquitectura de software para el producto "Mis Mejores Cuentos".* La Habana : s.n., 2011.

17. Medvidovic, Nenad, y otros. *A Component- and Message-Based Architectural Style for GUI Software.* 1995.

18. GARLAN, DAVID y Shaw, Mary. *An introduction to software architecture.* 1994.

19. Symfony 1.2 la guía definitiva. [En línea] 2008. [Citado el: 2 de Febrero de 2014.]

20. Sebastián, Juan. Modelo Vista Controlador-Definicion y Caracteristicas. [En línea] 13 de Noviembre de 2010. [Citado el: 29 de Enero de 2014.] http://www.comusoft.com/modelo-vista-controlador-definicion-y-caracteristicasModelo Vista Controlador-Definición y Características.

21. Lizardo, Maria Eugenia Arevalo. Introduccion al patron de arquitectura por capas. [En línea] 12 de Febrero de 2010. [Citado el: 19 de Marzo de 2014.] http://arevalomaria.wordpress.com/2010/12/02/introduccion-al-patron-de-arquitectura-por-capas/.

22. Sistemas Informaticos Estructurado. Arquitectura Cliente/Servidor. [En línea] http://www.google.com.cu/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0CDAQFjAB&url=http%3A%2F%2Fwww.dlsi.ua.es%2Fasignaturas%2Fsid%2Fsid2001-t4.ppt&ei=atT6UozyEuGX0QH81IGQDw&usg=AFQjCNFfwSMbPkVDwy7hUXgZwyB5onBVeg&bvm=bv.61190604,d.dmQ&cad=rja.

23. Canarias, Iker. Patrones de diseño de software. [En línea] 22 de Octubre de 2012. [Citado el: 11 de Febrero de 2014.] http://www.slideshare.net/ikercanarias/patrones-de-diseo-de-software-14836338Software Developer at Universidad de Deusto.

24. Visconti, Marcello y Astudillo, Hernán. *08-Patrones de disenno.pdf.* 2011.

25. Rondon Robaul, Izary. *Metodología para el desarrollo de software multimedia. Análisis comparativo y propuesta.* La Habana. Universidad de las Ciencias Informáticas : s.n., 2007.

26. Perez, Isaias Carrillo, Gonzalez, Rodrigo Perez y Martinez, Aureliano David Rodriguez. *Metodologia de Desarrollo.* 2008.

27. Salazar Terrero, Lirisandra y Gnzález Rodríguez, Raúl. *Solución informática para el portal Web Preparación para la defensa Facultad # 4.* La Habana. Universidad de las Ciencias Informaticas. : s.n., 2012.

28. Metodologia Agil de desarrollo de software extremo y software libre. [En línea] 2007.

29. Peñalver G, Meneses A y S., García. *SXP,METODOLIGÍA ÁGIL PARA EL DESARROLLO DE SOFTWARE.* La Habana. Universidad de la Ciencias Informáticas : s.n., 2010.

30. ALMAGRO, CARLOS UREÑA. *Lenguajes de Programación.* 2011.

31. STURM, JACK. *Desarollo_de_Soluciones_XM.*

32. ENTRALGO, LIC. RAÚL y AGUILAR, LIC. JORGE DAYÁN. *Ventajas de la tecnología XML dentro de las Ciencias de la Información. .* 2009.

33. *Manual de JavaScript.* Alvarez, Miguel Angel. 2001.

34. ALVAREZ, MIGUEL ANGEL. *Manual de JQuery.* 2009.

35. Curso gratis de Flash CS5. aulaClic. 16 - Introducción a ActionScript 3.0. [En línea] Marzo de 2011. [Citado el: 20 de Marzo de 2014.] http://www.aulaclic.es/flash-cs5/t_16_1.htm.

36. Hojas de Estilo en Cascada (CSS). [En línea] [Citado el: 20 de Marzo de 2014.] http://www.chesco.info/curso/css.htm.

37. Bravo Yenifer, Hernández Simón, Jiménez Francelys. *Tecnicas y herramientas para el desarrollo de software.*

38. Aprendiendo con Adobe Photoshop CS6: Características para fotógrafos (Capítulo 1, primera parte). [En línea] 26 de Marzo de 2012. [Citado el: 12 de Febrero de 2014.] http://www.xatakafoto.com/edicion-digital/aprendiendo-con-adobe-photoshop-cs6-caracteristicas-para-fotografos-capitulo-1-primera-parteCurso de Photoshop CS6.

39. HTML5 debuta en NetBeans 7.3 (beta). [En línea] 5 de Octubre de 2012. [Citado el: 12 de Febrero de 2014.] http://www.unocero.com.

40. JetBrains PhpStorm 7.1.133.51 EAP. MagicDown. [En línea] [Citado el: 12 de Febrero de 2014.] http://www.magicdown-center.com/showthread.php?282-JetBrains-PhpStorm-7-1-133-51-EAP.

41. The best JavaScript IDE with HTML Editor for Web development?:: JetBrains WebStorm. [En línea] [Citado el: 11 de Febrero de 2014.] http://www.jetbrains.com/webstorm/.

42. Nobrega, Maria De. Herramientas CASE: Rational Rose. [En línea] [Citado el: 20 de Marzo de 2014.] http://curso_sin2.blogia.com/2005/060401-herramientas-case-rational-rose.-por-maria-de-nobrega.php.

43. Visual Parading. software.com.ar. [En línea] 2007-2013. [Citado el: 20 de Marzo de 2014.] http://www.software.com.ar/visual-paradigm-para-uml.html.

44. ORALLO, ENRIQUE HERNANDEZ. *El Lenguaje De Modelado Unificado UML.*

45. Campos, Yaime Ricardo y Lopez, Livan Muñoz. *Introduccion de ApEM-L 1.0 en proyectos productivos de la UCI.* La Habana. Universidad de las Ciencias Informáticas : s.n.

46. Steve burbeck, PH.D. How to use Model-View-Controller (MVC). Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC). [En línea] 1992. [Citado el: 21 de Marzo de 2014.] http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.

47. CALLEJA, MANUEL ARIAS. *Carmen. Estándares de codificación.*

48. Vigil Regalado, Yamila,. *Metodología de evaluacion de arquitecturas de software.* La Habana. Universidad de las Ciencias Informáticas : s.n., 2009.

49. Dávila, Mauricio, y otros. *Evaluacion de Arquitectura de Software.*

50. Aedo, Dr.C. Raúl Fernández. *Los Métodos de Evaluación de expertos para valorar resultados de las investigaciones. .*

51. HERNÁNDEZ LEÓN, ROLANDO ALFREDO y Coello González, Sayda. *El Proceso de Investigación Científica.* Ciudad de la Habana : Universitaria del Ministro de Educación Superior : s.n., 2011.

52. Nuñez, Gabriel, Camacho, Erika y Cardeso, Fabio. *Arquitecturas de Software.* 2004.

53. REYNOSO, CARLOS y Kicillof., Nicolás. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires : Universidad de Buenos Aires, 2004.

54. HEREDIA., ING. YANEISIS PÉREZ. *Metodologias para el desarrollo de software educativo: un estudio comparativo.* La Habana. Universidad de las Ciencias Informáticas : s.n., 2007.

55. BURGOS, DERLIS MIGUEL RUIZ DIAZ. Introducción a ActionScript. Programación en Castellano. Programacion en Castellanos. [En línea] 1998-2013. [Citado el: 20 de Marzo de 2014.] http://www.programacion.com/articulo/introduccion_a_actionscript_103.

56. Romero, Gladys Marsy Peñanlver. *SXP, metodología de desarrollo de software.* 2011.

57. Figueroa, Roberto G. *Metodologías tradicionales VS. Metodologías ágiles.*

Glosario de Términos

Arquitectónico: perteneciente o relativo a la arquitectura.

Desarrollo: evolución progresiva del proceso de implementación del software. Acción y efecto de desarrollar o desarrollarse.

Elementos: fundamento móvil o parte integrante de algo.

Estilo: modo, manera, forma de comportamiento que va a tomar el software.

Estructura: distribución y orden con que está compuesto un producto informático.

IDE: es un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Multimedia: una multimedia consiste en el uso de diversos tipos de medios para transmitir, administrar o presentar información.

Sistema: conjunto estructurado de unidades relacionadas entre sí que se definen por oposición; p. ej., un software o los distintos componentes de la descripción del software.

Software: conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.

Subsistemas: un subsistema es un sistema que es parte de otro sistema mayor. En otras palabras, un subsistema es un conjunto de elementos interrelacionados que, en sí mismo, es un sistema, pero a la vez es parte de un sistema superior**.**

Anexos

Anexo 1. Guía de observación

La guía de observación que se elabora en la propuesta se realiza con el objetivo de observar el comportamiento de los diferentes componentes y la relación que comúnmente existen entre ellos, que se implementan en los diferentes sistemas informáticos con tecnología multimedia que se producen en el centro FORTES.

El objeto de observación es la arquitectura de los diferentes productos multimedia creados hasta el momento, **el sujeto de observación** son los miembros que integran el equipo de desarrollo de productos con tecnología multimedia en el centro FORTES.

El **tipo de observación** que se utiliza es externa no incluida, ya que los observadores realizan desde fuera del grupo que se estudia; la observación de las diferentes estructuras que implican funcionalidades comunes en productos con tecnología multimedia en el Centro.

- Se observa el comportamiento de los productos y su estructura, y se definen las características más comunes de los componentes multimedia para así incorporarlos a la modelación de la propuesta. Se realiza encuestas y un cuestionario a los diferentes desarrolladores de productos con tecnología multimedia, y se formulan las siguientes preguntas del cuestionario:

1. ¿Qué elementos visuales comúnmente debe tener las aplicaciones con tecnología multimedia?
2. Diga las características fundamentales que debe tener un producto multimedia.
3. Cómo es la relación que existe entre todos los elementos de los sistemas multimedia, así como las propiedades de los mismos. Tales relaciones entre: clases, responsabilidad de cada componente, acceso a los datos, seguridad, flujo de navegación de las interfaces, entre otras.
4. ¿Qué patrones de diseño son implementados comúnmente para los productos multimedia del Centro?
5. ¿Qué tipo de codificación es el más utilizado en la implementación de las aplicaciones con tecnología multimedia?
6. ¿Qué metodología de desarrollo es la más utilizada frecuentemente para el proceso de desarrollo de los productos multimedia?

Resultados del cuestionario:

1) Tipos de componentes visuales que debe cumplir los productos con tecnología multimedia:

Reproductor de audio, Reproductor de video, Scroll, buscador, imprimir en pantalla, efectos de sonido y animaciones.

2) Las características fundamentales que debe tener un producto multimedia son: Debe poseer imágenes, video, sonido y animaciones. Debe de ser portable y centralizado en un solo entorno de trabajo. Debe poseer una interfaz agradable al usuario, un diseño simple. Debe tener un tiempo rápido de entrega del software. Las clases y código deben de ser reutilizable. Deben de ser software interactivo y dinámico en su funcionalidad.
3) El equipo de desarrollo que implementa sistemas multimedia en el Centro utilizan herramientas privativas y cada producto para ser ejecutado presencia de otras tecnologías como plugins y la instalación de la herramienta de desarrollo. Por lo que cada clase que se implementa es realizada con Action Script 3.0 y Adobe Flash CS6. Para acceder a los datos que brinda la información con la que trabaja la aplicación se accede a un archivo XML que contiene las url de todos los recursos multimedia. Se crean un documento XML para cada clase que contiene determinadas funcionalidades. Mayormente estos sistemas no comprenden con un alto nivel de seguridad por lo que no se centran en este requisito. El flujo de navegación es atreves de los componentes de navegación como menús, hipertextos, entre otros. Mayormente son software que brindan información.
4) No se ha establecido patrones de diseño para la implementación de los productos con tecnología multimedia en el centro FORTES.
5) No se ha establecido un estándar de codificación para las aplicaciones multimedia.
6) No se trabaja bajo una metodología de desarrollo que guie el proceso de desarrollo del software con tecnología multimedia del centro FORTES. Los productos multimedia son software muy pequeños que brindan servicios de información por lo que se trabaja con un plan de trabajo generado para los productos pequeños que se implementan en el Centro.

Se realiza una entrevista a diferentes especialistas del centro FORTES como: arquitectos, analistas y desarrolladores que han tenido experiencia en la rama de la arquitectura de software y en la confección de productos con tecnología multimedia. En la entrevista se tuvieron en cuenta las siguientes cuestiones para los arquitectos de software se realizaron preguntas tales como:

1. ¿Qué elementos fundamentales se deben tener en cuenta para el diseño de una arquitectura de software basada en componentes?
2. ¿Qué fuentes bibliográficas son las más recomendadas para investigar acerca del objeto de la investigación?

Resultados:

1. Se deben tener en cuenta primeramente los pilares de la arquitectura de software que son las bases en el diseño de la propuesta. Después definir los componentes multimedia de acuerdo a las características que establecen los productos multimedia en el centro FORTES. Luego de establecer la relación de cada componente y las características del software con este tipo de tecnología, se definen las restricciones del sistema. Acto seguido se establecen los estilos y patrones arquitectónicos que generan la estructura estándar de todos los productos con tecnología multimedia.
2. Las fuentes bibliográficas que son recomendables para el objeto de la investigación son: Guía de Arquitectura que se encuentra en eva.uci.cu, en la asignatura de Ingeniería de Software II, las tesis de propuesta de arquitectura para plataformas y la tesis “Propuesta de arquitectura basada en componentes para el producto Mis Mejores Cuentos”. La tesis de maestría del ingeniero Carlos Reynoso de la Universidad de Buenos Aires en Argentina. Ingeniería de Software Buenas Prácticas de Roger Pressman.

Para los analistas de este tipo de software se realizan preguntas como:

1. ¿Cuáles son las funcionalidades más comunes que establecen los clientes?
2. ¿Qué artefactos se generan para la modelación de las estructuras que contienen los productos multimedia?
3. ¿Qué artefactos se generan para documentar la información planteada por los clientes?

Resultados:

1. Las funcionalidades más comunes son los menús desplegables para la navegación en el sistema, la galería de imágenes y de videos para mostrar la información contenida en ilustraciones y acciones. Además de las opciones como control de pantalla, imprimir en pantalla, reproductor de sonido y video.
2. Se utiliza mayormente el patrón modelo-vista-controlador para estructurar el sistemas y se utiliza la modelación de la interfaz de usuario, pero no se crea otro artefacto de modelación.

3. Como no se utiliza una metodología de desarrollo se trabaja con un plan de trabajo que contienen todos los requisitos que piden los clientes y se estructuran los sistemas de acuerdo al diseño del programador.

Para los desarrolladores se establecen las preguntas planteadas en el cuestionario anterior.

- ♣ Para la evaluación de la arquitectura se implementa un prototipo no funcional donde se observa los resultados de la propuesta diseñada. Viendo la flexibilidad, reutilización de los componentes y la adaptabilidad de los mismos.

Anexo 2.

Ejemplos de atributos de calidad observables en vía de ejecución.

Atributos de calidad	Descripción
Disponibilidad	Es la medida de disponibilidad del sistema para ser usado.
Confidencialidad	Es el acceso único de la persona autorizada a la información.
Funcionalidad	Es la destreza del sistema para realizar la función para el cual fue creado.
Desempeño	Son aspectos temporales del comportamiento de un sistema. Se define además según Bass (1998) a la cantidad de comunicación e interacción existente entre los componentes del sistema. Es la manera en que un sistema o componente cumple con sus funciones que se le ha sido asignada.
Confiabilidad	Es la pericia de un sistema a mantenerse operativo a lo largo del tiempo.
Seguridad interna y externa	Son las medidas de seguridad interna y externa de un sistema para resistir amenazas. La seguridad externa pretende la ausencia de consecuencias catastróficas en el ambiente y de errores que generan pérdidas de información. La seguridad interna tiene la habilidad de velar por intentos de uso no autorizados y la negación del servicio mientras se sirven a usuarios legítimos.

Anexo 3.

Ejemplos de atributos de calidad no observable en vía de ejecución.

Atributos de Calidad	Descripción

Configurabilidad	Posibilidad que se le otorga a un usuario experto a realizar ciertos cambios al sistema.
Integración	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 un tipo especial de integrabilidad. Es la medida en que un grupo de partes del sistema trabajen con otro sistema.
Modificabilidad	Es la habilidad de realizar cambios futuros en el sistema.
Mantenibilidad	Es la capacidad de someter a un sistema a reparaciones y evolución. Realizarle nuevas versiones de acuerdo a los nuevos errores que se mitigan.
Portabilidad	Es la capacidad del sistema de poder ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser software, hardware 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 puede ampliar el diseño arquitectónico, de datos o procedimental.
Capacidad de Prueba	Es la medida con la que el software al ser sometido a una serie de pruebas pueda demostrar sus fallas fácilmente.

Anexo 4.

Patrones de diseño GoF

Patrones de creación: resuelven problemas relacionados con la creación de instancias de objetos, entre ellos están:

- ♣ Fábrica Abstracta (*Abstract Factory*): proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas.
- ♣ Método de Fábrica (*Factory Method*): define una interfaz para crear un objeto, pero deja que sean subclases quienes decidan que clases instanciar. Permite que una clase delegue en subclases la creación de objetos.
- ♣ *Singleton*: garantiza que una clase solo tenga una instancia y proporciona un punto de acceso global a ella.
- ♣ Prototipo (*Prototype*): especifica los tipos de objetos a crear por medio de una instancia prototípica, y crea nuevos objetos copiando de este prototipo.
- ♣ *Constructor* (*Builder*): separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones. (23)

Patrones estructurales: se centran en problemas relacionados con la forma de estructurar las clases, entre estos se encuentran:

- ♣ Adaptador (*Adapter*)*:* convierte la interfaz de una clase en otra distinta que es la que esperan los clientes.
- ♣ Decorador (*Decorator*): añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.
- ♣ Compuesto (*Composite*): permite combinar objetos en estructuras de árbol para representar jerarquías. Agrupa varios objetos como si fueran uno solo.
- ♣ Fachada (*Facade*): proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar.
- ♣ Peso Mosca (*Flyweight*): usa el comportamiento para permitir un gran número de objetos de grano fino de forma eficiente.
- ♣ *Proxy*: Proporciona un sustituto o representante de otro objeto para controlar el acceso a este. (23)

Patrones comportamiento: permiten resolver problemas relacionados con el comportamiento de la aplicación, normalmente en tiempo de ejecución, se pueden mencionar.

- ♣ Interprete (*Interpreter*): dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para representar las sentencias del lenguaje.
- ♣ Observador (*Observer*): define una dependencia de uno-a-muchos objetos, de forma que cuando una cambia de estado se notifica y actualizan automáticamente todos los objetos.

- ♣ Visitantes (*Visitor*): representa una operación sobre los elementos de una estructura de objetos. Define una nueva operación sin cambiar las clases de los elementos sobre los que opera.
- ♣ Mediador (*Mediator*): define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente, y permite variar la interacción entre ellos de forma independiente.
- ♣ Iterador (*Iterator*): proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna. (23)

Patrones de diseño GRASP

Patrón experto: asigna una responsabilidad al experto en la información. Conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Este patrón distribuye el comportamiento entre las clases que contiene la información requerida. (24)

Patrón Creador: es el que define quien debe ser responsable de crear una nueva instancia de alguna clase. También describe un patrón para definir objetos agregados la cual favorece la encapsulación de componentes. (24)

Patrón Controlador: se encarga de atender un evento del sistema, asigna la responsabilidad del manejo de mensajes de los eventos a una clase. (24)

Patrón Bajo Acoplamiento: asigna una responsabilidad de manera que el acoplamiento permanezca bajo. Define cómo dar soporte a una mínima dependencia y a un aumento de reutilización. Este patrón impulsa la asignación de responsabilidades de manera que su localización no incremente el acoplamiento hasta un nivel que nos lleve a los resultados negativos que puede producir un acoplamiento alto. También soporta el diseño de clases que son más independientes, no se puede dejar de mencionar que este patrón no puede considerarse de manera aislada a los patrones Experto o Alta Cohesión, sino que el mismo necesita incluirse como uno de los diferentes principios de diseño que incluyen en una elección al asignar una responsabilidad.

Patrón Alta Cohesión: asigna una responsabilidad de manera que la cohesión permanezca alta. Se encarga que cada elemento que se diseñe tenga una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable. (24)

Estos patrones de diseño proporcionan una estructura conocida por todos los programadores, de manera que la forma de trabajar no resulte distinta entre los mismos. Así la incorporación de un nuevo programador, no requerirá conocimiento de lo realizado anteriormente por otro. Permiten tener una estructura de código común a todos los proyectos que implemente una funcionalidad genérica. La utilización de patrones de diseño, permite ahorrar grandes cantidades de tiempo en la construcción de software.

Anexo 5.

Estructura lógica de ApEM-L.

Área	Vista	Diagramas	Modificaciones
Estructura Lógica	Vista estática	Diagrama de Clases.	Establece el patrón modelo-vista-controlador para la concepción del diseño de aplicaciones educativas. Plantea la semántica y los estereotipos restrictivos y descriptivos para las clases asociadas a las tecnologías multimedia e hipermedia.
		Diagrama de Casos de Uso.	No se establecen modificaciones por el lenguaje UML.
	Vista de arquitectura	Diagrama de Componentes.	No se modifica la semántica del lenguaje base, sino que se extiende al incorporar elementos de organización en paquetes asociados al patrón Modelo-Vista-Controlador.
		Diagrama de Despliegue.	No se establece modificaciones.

Anexo 6.

Pasos del método de evaluación SAAM.

1. Desarrollo de escenarios	Un escenario es una breve descripción de usos anticipados o deseados del sistema. De igual forma, estos pueden incluir cambios a los que puede estar expuesto el sistema en el futuro.
2. Descripción de la arquitectura	La arquitectura (o las candidatas) debe ser descrita haciendo uso de alguna notación arquitectónica que sea común a todas las partes involucradas en el análisis. Deben incluirse los componentes de datos y conexiones relevantes, así como la descripción del comportamiento general del sistema. El desarrollo de escenarios y la descripción de la arquitectura son usualmente llevados a cabo de forma intercalada, o a través de varias iteraciones.
3. Clasificación y asignación de prioridad de los escenarios	La clasificación de los escenarios puede hacerse en dos clases: directos e indirectos. Un escenario *directo* es el que puede satisfacerse sin la necesidad de modificaciones en la arquitectura. Un escenario *indirecto* es aquel que requiere modificaciones en la arquitectura para poder satisfacerse. Los escenarios indirectos son de especial interés para SAAM, pues son los que permiten medir el grado en el que una arquitectura puede ajustarse a los cambios de evolución que son importantes para los involucrados en el desarrollo.
4. Evaluación individual de los escenarios indirectos	Para cada escenario indirecto, se listan los cambios necesarios sobre la arquitectura, y se calcula su costo. Una modificación sobre la arquitectura significa que debe introducirse un nuevo componente o conector, o que alguno de los existentes requiere cambios en su especificación.
5. Evaluación de la interacción entre escenarios	Cuando dos o más escenarios indirectos proponen cambios sobre un mismo componente, se dice que *interactúan* sobre ese componente. Es necesario evaluar este hecho, puesto que la interacción de componentes semánticamente no relacionados revela que los componentes de la arquitectura efectúan funciones semánticamente distintas. De forma similar, puede verificarse si la arquitectura se encuentra documentada a un nivel correcto de descomposición estructural.
6. Creación de la evaluación global	Debe asignársele un peso a cada escenario, en términos de su importancia relativa al éxito del sistema. Esta asignación de peso suele hacerse con base en las metas del negocio que cada escenario soporta. En el caso de la evaluación de múltiples arquitecturas, la asignación de pesos puede ser utilizada para la determinación de una escala general.

Anexo 7.

Pasos del método de evaluación ATAM.

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

Anexo 8.

Pasos de método de evaluación ARID.

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

Anexo 9.

Etapas del método de evaluación de arquitecturas propuesto por Bosch.

Etapa I	
1. Selección de atributos de calidad	Deben seleccionarse aquellos atributos que se consideran cruciales para el éxito del sistema, y cuya satisfacción resulte poco clara a nivel de arquitectura. Resulta necesario porque es poco factible y poco útil evaluar todos los atributos de calidad, dado que requiere una gran cantidad de tiempo.
2. Definición de los perfiles	Para cada atributo de calidad seleccionado, se definen los perfiles respectivos para efectos de la evaluación.
3. Selección de una técnica de evaluación	Para la evaluación de los atributos de calidad dependientes del diseño de la arquitectura se recomienda utilizar la evaluación basada en escenarios, así como también los modelos basados en métricas o modelos matemáticos. Los atributos de calidad operacionales (observables vía ejecución) pueden evaluarse con técnicas de simulación o modelos matemáticos. La selección de la técnica, y la implementación concreta de ésta depende del objetivo y exactitud de la evaluación.
Etapa II	
4. Ejecución de la evaluación	Para cada atributo de calidad, las técnicas arrojan valores cuantitativos.
5. Obtención de resultados	Los resultados se resumen en una tabla que contiene el nivel requerido, el nivel predicho, y un indicador, que puede tener diversos significados: si el atributo se satisface o no, si necesita ser negociado con el cliente, o existencia de alguna relación negativa con otro atributo de calidad. El arquitecto puede decidir acerca de la realización de transformaciones sobre la arquitectura actual, y efectuar una nueva evaluación. Una vez que concluye el proceso de evaluación, con los resultados obtenidos es posible decidir entre la continuación, renegociación o cancelación del proyecto.

Anexo 10. Escenarios del método de evaluación ATAM.

Tabla 6. Escenario # 3.

Escenario # 3	**Integrar partes o elementos de un sistema para trabajar en otro.**			
Atributo(s)	Funcionalidad-Interoperabilidad.			
Ambiente	Integración de pluggins existentes.			
Estímulo	Utilización de componentes de sistemas ya implementados para integrarlo en uno nuevo.			
Respuesta	El empleo de componentes reutilizables de sistemas ya desarrollados para integrarlo a otro sistema.			
Decisiones arquitectónicas	Pto de sensibilidad	Pto Trade-Off	Riesgo	No Riesgo
Utilizar pluggins de las librerías JQuery y Bootstrap para la creación de elementos visuales de los sistemas.	S1			NR2
Integración de componentes galería de imágenes o videos de la biblioteca de componentes para utilizar en la propuesta.	S2		R1	
Explicación	La utilización de los pluggins para la integración de nuevas funcionalidades es una manera muy factible para la creación de elementos, permite ahorro de esfuerzo de desarrollo. Al igual que la integración de componentes multimedia de otros sistemas permite la reutilización y efectividad en el proceso de desarrollo.			

Tabla 7. Escenario # 4.

Escenario # 4	**Creación de contenido del sistema mediante la utilización de imágenes, videos, animaciones, textos, entre otras.**
Atributo(s)	Eficiencia-Desempeño.
Ambiente	Diseño de los elementos visuales.
Estímulo	Crear un conjunto de elementos que integren las funcionalidades visuales que va a contener el sistema.
Respuesta	Mediante la utilización de los componentes multimedia se trasmitirá un entorno interactivo, donde el usuario recibirá información mediante este.
Decisiones arquitectónicas	Pto de sensibilidad Pto Trade-Off Riesgo No Riesgo

Patrón de diseño Decorator.				NR3
Implementar archivos.css para darle estilos e interactividad a la vista.	S3		R2	
Explicación	El patrón Decorator es un patrón estructural que su función es brindar responsabilidades a los elementos visuales específicos y no a los archivos. Se vincula en la propuesta con el propósito de crear un archivo controlador, donde cargue los elementos más comunes de los sistemas desde el modelo a la vista. Esto hace los sistemas más dinámicos. Se implementa el archivo styles.css para albergar códigos CSS, que permita acomodar el contenido en el sistema.			

Tabla 8. Escenario # 5.

Escenario # 5	**Proceso de carga de datos desde el modelo a la vista.**			
Atributo(s)	Eficiencia-Desempeño.			
Ambiente	Comunicación entre capas.			
Estímulo	Procesar la información almacenada en el modelo hacia la vista de manera eficiente donde se evalúa la interactividad y dinamismo.			
Respuesta	Mediante la capa controladora se crea un acceso hacia los datos almacenados en la modelo.			
Decisiones arquitectónicas	Pto de sensibilidad Pto Trade-Off Riesgo No Riesgo			
Patrón arquitectónico modelo vista controlador.				NR4
Crear un documento XML que permita el almacenamiento de los datos.	S4		R3	
Patrón de diseño Observer.	S5			NR5
Explicación	El patrón arquitectónico modelo-vista-controlador brinda la interdependencia de las capas, permitiendo la comunicación entre la vista-controladora y controladora-modelo. Esto hace que al cargar los datos se realice de forma dinámica. Al almacenar las medias en el modelo se estructura carpetas donde se guarde cada uno de los recursos multimedia. Al ser cargados en el modelo se utiliza un archivo XML que contiene las url de cada componente multimedia.			

Tabla 9. Escenario # 6.

Escenario # 6	**La arquitectura debe facilitar la flexibilidad del sistema.**			
Atributo(s)	Mantenibilidad-Modificabilidad.			
Ambiente	Crear cambios futuros.			
Estímulo	Para poder modificar alguna funcionalidad que pueda ser amenaza al sistema. De acuerdo a las peticiones del cliente ser crear un producto donde es más fácil la modificabilidad de las funcionalidades o requisitos del producto.			
Respuesta	Implementar archivos donde se le puedan modificar dinámicamente las funcionalidades o componentes de los archivos que dependen de una modificación.			
Decisiones arquitectónicas	Pto de sensibilidad	Pto Trade-Off	Riesgo	No Riesgo
Patrón modelo vista controlador (MVC).				NR6
Explicación	La implementación del patrón Observer trabaja con una dependencia de uno a muchos, donde al actualizar un componente o evento, manda una notificación a los archivos dependientes a este componente, donde los mismos se actualizan automáticamente.			

Tabla 10. Escenario # 7.

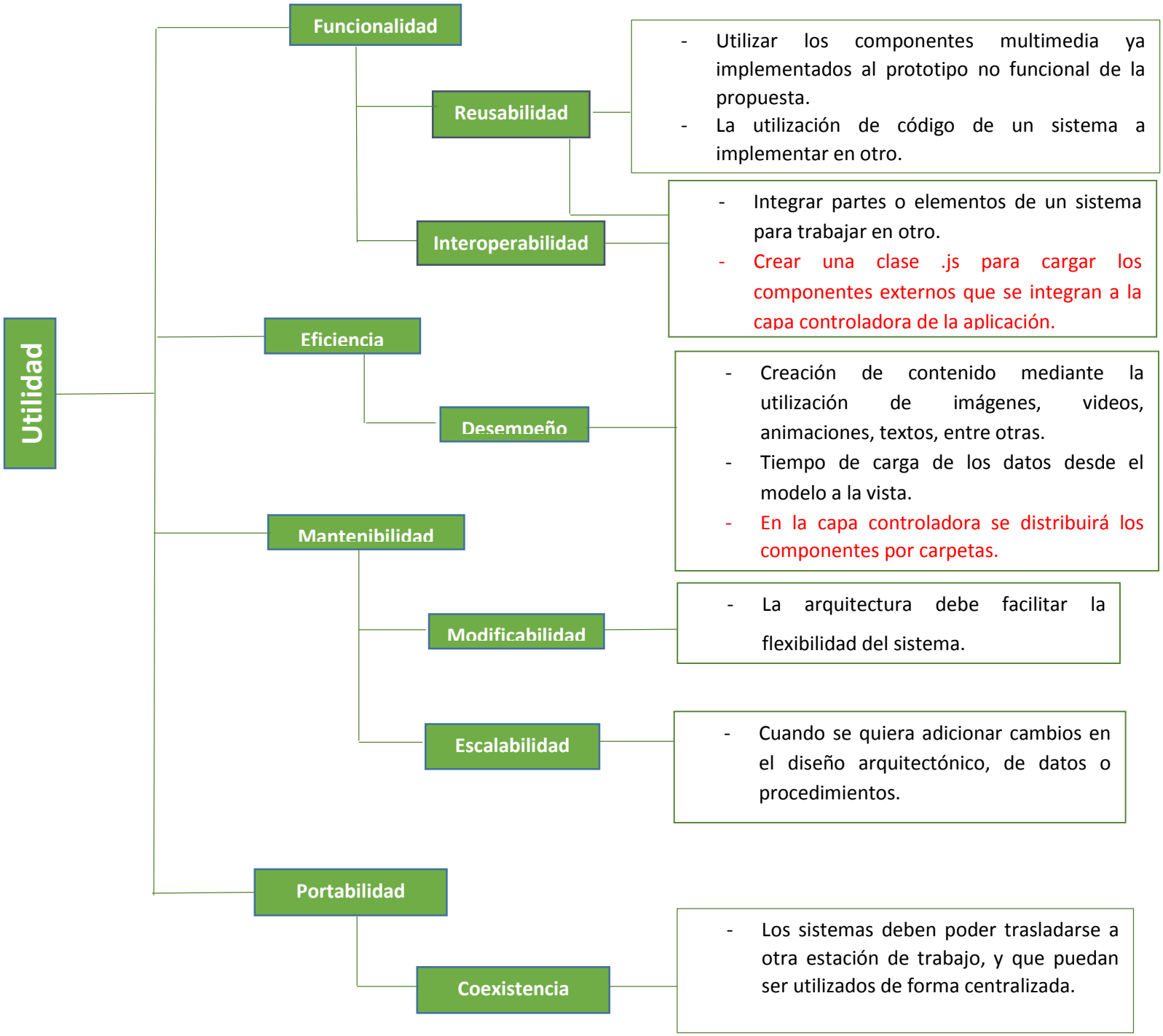
Escenario # 7	**Adicionar cambios en el diseño arquitectónico, de datos o procedimientos.**
Atributo(s)	Mantenibilidad-Escalabilidad.
Ambiente	Adición de elementos arquitectónicos.
Estímulo	Para incrementar el diseño de acuerdo a las peticiones que genera el cliente, se reconoce que los productos deben de ser escalables.
Respuesta	Se debe tener en cuenta el tipo de configuración, componentes, conectores y restricciones que deben de ajustarse a los sistemas multimedia.
Decisiones arquitectónicas	Pto de sensibilidad Pto Trade-Off Riesgo No Riesgo

Implementación de un archivo .js y .css para cada componente.				NR7
Explicación	Este modelo arquitectónico por sus características separa la vista de los datos permitiendo hacer cambios al sistema, a la hora de darle mantenimiento al software pueden haber cambios, los cuales pueden modificar el diseño arquitectónico, o sumarles nuevas funcionalidades al mismo.			

Tabla 11. Escenario # 8.

Escenario # 8	**Los sistemas deben poder trasladarse a otra estación de trabajo, y que puedan ser utilizados de forma centralizada.**			
Atributo(s)	Portabilidad-Coexistencia.			
Ambiente	Centralizado.			
Estímulo	En el caso de los productos multimedia se ejecutan de forma centralizada .El software tiene los datos en una misma estación de trabajo, y el sistema es ejecutado de acuerdo a la tecnología que se utilice en su implementación.			
Respuesta	Las aplicaciones multimedia se pueden trasladar mayormente en dispositivos portables. En el caso de la tecnología que utilice para realizar las aplicaciones va a ser el entorno de ejecución donde va a desempeñarse.			
Decisiones arquitectónicas	Pto de sensibilidad	Pto Trade-Off	Riesgo	No Riesgo
Implementar las aplicaciones multimedia con tecnologías web.				NR8
Explicación	Las aplicaciones multimedia desarrollada en HTML 5, XML, Java Script, son más portables ya que solo con un navegador que tenga tu sistema puedes visualizarlas, y al querer implementar o modificar algo, solamente puedes tener un editor de texto.			

Anexo 11.Integración de los nuevos escenarios.



Anexo 12.

Estilos y atributos de calidad.

Estilo	Descripción	Atributos Asociados	Atributos en conflictos
Datos Centralizados	Sistema en los que ciertos números de clientes acceden y actualiza datos compartidos de un repositorio de manera frecuente.	Integrabilidad Escalabilidad Modificabilidad	Desempeño
Flujo de Datos	El sistema es visto como una serie de transformaciones sobre piezas sucesivas de datos de entrada. El dato ingresa en el sistema, y fluye entre componentes, de uno en uno, hasta que se le asigne un destino final.	Reusabilidad Modificabilidad Mantenibilidad	Desempeño
Máquinas Virtuales	Simulan alguna funcionalidad que no es nativa al hardware o software sobre el que esta implementado.	Portabilidad	Desempeño
Llamada y Retorno	El sistema se constituye de un programa principal que tiene el control del sistema y varios subprogramas que se comunican con este	Modificabilidad Escalabilidad Desempeño	

	mediante el uso de llamadas.		
Componentes independientes	Consiste en un número de procesos u objetos independientes que se comunican a través de mensajes.	Modificabilidad Escalabilidad	