UNIVERSIDAD DE LAS CIENCIAS INFÓRMATICAS

Capa de presentación de una arquitectura para la construcción de aplicaciones enriquecidas de internet que incluyan escenarios tridimensionales interactivos.

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

Autora: Libeidy Hernández Martínez.

Tutor: Ing. Gilberto Cao Tarrero.

Ciudad de la Habana, Junio 2011

Declaración de Autoría.

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los días _____ del mes _____ del año 2011.

Libeidy Hernández Martínez Ing.: Gilberto Cao Tarrero

_____ _____

(Autor) (Tutor)

Datos de Contacto

Tutor: Ing. Gilberto Cao Tarrero.

Edad: 25

Ciudadanía: cubano

Institución: Universidad de las Ciencias Informáticas (UCI)

Título: Ingeniero en Ciencias Informáticas.

Categoría Docente:

E-mail: gcao@uci.cu

Graduado en la Universidad de las Ciencias Informáticas en el año 2009. Se desempeñó en el período 2009-2010 como líder de la línea de productos Visualización Web del Centro de Informática Industrial (CEDIN). Desde el 2010 dirige el desarrollo del módulo Laboratorio Virtual del software educativo “La naturaleza y el hombre” en el Centro de Tecnologías para la Formación (FORTES).

Agradecimientos

En primer lugar por ser la persona más especial que existe en este mundo, por su excelente forma de ser por preocuparse por mí, por haberme dado la libertad suficiente para tomar mis decisiones, por quererme y apoyarme tanto.

A mi Mamá.

Por sentirme tan orgullosa de ser su hija, por saber que tienes la mejor de las opiniones sobre mi y por nunca defraudarme.

A mi Papá.

Porque desde niña me daba consejos y estaba siempre pendiente a mis estudios, por haber sido mí ejemplo.

A mi hermana.

Por confiar en mí, por su cariño, por estar a mi lado y haber compartido la etapa más feliz que tiene uno en la vida.

A mi novio Yusniel.

Por haber contribuido de una forma u otra en mi formación profesional.

A toda mi Familia.

Por haberme dado ánimo y por su guía y ayuda en la realización de este trabajo.

A Yerandy y Ernesto.

Por haber compartido tantos momentos lindos estos 5 años y por estar siempre en los buenos y malos momentos.

A mis compañeras de cuarto Yoana, Liusba, Irenna, Liudmila, Dayanis, Sandra.

Por haber estado desde primer año siempre dispuesto a ayudarme en todo, por su disposición y ayuda incondicional y por haber hecho tantas cosas por mí, son tantas las razones que no me alcanzaría la hoja, unas gracias muy especiales para él.

A Adriel

Por haber pasado tantos sustos juntos y haber salido adelante nosotros solos.

A mis compañeros de tesis Liusba y Andy.

Por ser el grillito de la conciencia.

A Yurisel.

Por este tiempo que compartimos juntos y hacerme pasar momentos increíbles en el aula.

A mis compañeros de grupo y de año Dunia, Alejandro, Enrique, Yuniel, Antonio, Alexis, Juan Carlos, Alían, Palomino.

Porque siempre confiaron en mí y me decían: Tu veras que si lo vas a lograr.

A mis amigos del barrio.

Dedicatoria

A mis padres porque gracias a ellos estoy hoy aquí realizando mis sueños.

A mi abuelo Pedro que este donde este sé que se sentirá orgulloso de verme convertida en una profesional.

A mi sobrina Laura porque a pesar de su malcriadez la quiero mucho.

Resumen

A lo largo del tiempo, la tendencia a utilizar aplicaciones Web se ha ido incrementando de una manera sorprendente. A causa de los beneficios y la comodidad que éstas nos aportan, optamos, cada vez más, por utilizar este tipo de aplicaciones en lugar de las tradicionales.

Se ha llegado a un punto en el que las aplicaciones Web no sólo han de ofrecer multitud de funciones, sino que además han de ser lo más óptimas posibles y sus interfaces gráficas han de ser mucho más impactantes y cómodas para los usuarios. Para conseguir estos objetivos se han creado un nuevo tipo de aplicaciones llamadas Aplicaciones Enriquecidas de Internet *(*RIA*,* por sus siglas en inglés*),* cuyo objetivo es el de optimizar las comunicaciones de datos entre cliente y servidor, además de ofrecer interfaces mucho más atractivas para el usuario.

Cuba no se ha quedado atrás en el desarrollo tecnológico y ha despertado un gran interés en las comunidades de desarrollo del mundo, por lo que ha trabajado aceleradamente en la construcción de grandes y complejos sistemas de software siendo La Universidad de las Ciencias Informáticas (UCI) el centro que más se ha destacado en esta esfera. Las RIA desarrolladas en esta organización no cuentan con una arquitectura que guíe su proceso de desarrollo y sea capaz de lograr interfaces intuitivas, fáciles de usar y que enriquezcan la experiencia del usuario.

El presente trabajo se centra hacia el desarrollo de una estructura para la capa de presentación de una Arquitectura de Software de Dominio Específico (DSSA) de manera que sea funcional, mantenible y portable, que cumpla con todos los requisitos necesarios para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos.

Palabras Clave:

Aplicaciones Enriquecidas de Internet, componentes, interfaz.

Índice

DECLARACIÓN DE AUTORÍA.... II

DATOS DE CONTACTO I

AGRADECIMIENTOS.... II

DEDICATORIA IV

RESUMEN.... V

ÍNDICE.... VI

ÍNDICE DE FIGURAS.... X

ÍNDICE DE TABLAS XI

INTRODUCCIÓN.... 1

CAPÍTULO 1 FUNDAMENTACIÓN TEÓRICA.... 4

1.1 INTRODUCCIÓN.... 4

1.2 APLICACIONES ENRIQUECIDAS DE INTERNET. 4

1.3 ARQUITECTURA DE SOFTWARE 5

1.4 ESTILOS ARQUITECTÓNICOS 7

1.4.1 Tubería y filtros.... 7

1.4.2 Estilos de llamada y retorno.... 8

1.4.3 Arquitecturas Orientadas a Objetos 9

1.4.4 Arquitectura basada en componentes 10

1.5 PATRONES DE DISEÑO 10

1.5.1 Patrón Modelo Vista Controlador 10

1.5.2 Patrones GOF 12

1.5.3 Patrones GRASP.... 13

1.6 BASES TECNOLÓGICAS 14

1.6.1 Entorno de Desarrollo Integrado 14

1.6.2 Flex 4 SDK 15

1.6.3 Adobe Flash Builder 4 15

1.6.4 Lenguajes de Programación 16

1.6.5 Servidores Web 17

1.6.6 Metodologías de Desarrollo 18

1.7 HERRAMIENTAS DE MODELADO 22

1.7.1 Visual Paradigm 22

1.7.2 Rational Rose 22

1.8 BIBLIOTECAS 3D PARA FLASH 23

1.8.1 Away3D 23

1.8.2 Papervision3D 24

1.8.3 Sandy 3D 24

1.9 CONCLUSIONES PARCIALES 24

CAPÍTULO 2 SOLUCIÓN PROPUESTA 25

2.1 INTRODUCCIÓN 25

2.2 LÍNEA BASE DE LA ARQUITECTURA 25

2.3 SELECCIÓN DE LAS TECNOLOGÍAS 25

2.3.1 Estilo Arquitectónico 25

2.3.2 Patrones de diseño. 27

2.3.3 Lenguaje de programación y entorno de desarrollo seleccionado. 30

2.3.4 Servidor Seleccionado *30*

2.3.5 Metodología de desarrollo a utilizar *30*

2.3.6 Herramienta Case Seleccionada *30*

2.3.7 Biblioteca Seleccionada *30*

2.4 DESCRIPCIÓN DE LA ARQUITECTURA 31

2.4.1 Propósito *31*

2.4.2 Alcance *31*

2.4.3 Estructura de la Capa de presentación *31*

2.5 METAS Y LIMITACIONES DE LA ARQUITECTURA. 35

2.5.1 Requerimientos Funcionales *35*

2.5.2 Requerimientos no Funcionales *36*

2.6 MODELO 4+1 VISTA. 37

2.6.1 Vista de Casos de Uso *38*

2.6.2 Vista lógica *42*

2.6.3 Vista de Despliegue. *43*

2.6.4 Vista de Implementación. *44*

2.6.5 Vista de Procesos *45*

2.7 CONCLUSIONES PARCIALES 45

CAPÍTULO 3 EVALUACIÓN DE LA ARQUITECTURA. 46

3.1 INTRODUCCIÓN 46

3.2 ¿CUÁNDO EVALUAR UNA ARQUITECTURA? 46

3.3 NECESIDAD DE EVALUAR LA ARQUITECTURA 47

3.4 TÉCNICAS DE VALIDACIÓN 47

3.5 MÉTODOS DE EVALUACIÓN 48

3.6 ATRIBUTOS DE CALIDAD 52

3.7 EVALUACIÓN DE LA ARQUITECTURA DE SOFTWARE PROPUESTA 54

3.8 DEFINICIÓN DE LOS PRINCIPALES ESCENARIOS. 55

3.9 CONCLUSIONES DEL CAPÍTULO. 57

CONCLUSIONES GENERALES 58

REFERENCIAS BIBLIOGRÁFICAS 60

BIBLIOGRAFÍA 63

GLOSARIO DE TÉRMINOS 65

Índice de Figuras

Figura 1. Estructura de filtros y tuberías 7

Figura 2. Arquitectura en 3 capas 9

Figura 3. Patrón Modelo Vista Controlador 11

Figura 4. Ciclo de vida de AUP 21

Figura 5. Arquitectura en capas 26

Figura 6. Representación del MVC en la Capa de Presentación 28

Figura 7. Componentes de la Capa de Presentación 32

Figura 8. Componentes de Proceso de Interfaz de Usuario 34

Figura 9. Modelo 4+1 Vista 38

Figura 10. Diagrama de Casos de Uso. 39

Figura 11. Vista Lógica de la Capa de presentación 42

Figura 12. Vista de Despliegue 44

Figura 13. Vista De Implementación. 45

Figura 14. Clasificación de las técnicas de evaluación de arquitectura. 48

Índice de Tablas

Tabla 1. Comparación entre aplicaciones WEB, Escritorio y RIA. (ORT, 2008) 5

Tabla 2. Descripción textual del Caso de Uso Visualizar IU. 40

Tabla 3 .Descripción del Caso de Uso Administrar Modelo 3D. 41

Tabla 4. Descripción textual del Caso de Uso Administrar Eventos. 41

Tabla 5. Pasos del Método ARID. 52

Tabla 6. Atributos de Calidad 54

Tabla 7. Evaluación del atributo Portabilidad. 55

Tabla 8. Evaluación del atributo Mantenibilidad. 56

Tabla 9. Evaluación del atributo Reusabilidad. 56

Introducción

La expansión de las Tecnologías de la Información y las Comunicaciones (TIC) en todos los ámbitos y esferas de la sociedad se han producido a gran velocidad, y es un proceso continuo debido a la aparición incesante de nuevos elementos tecnológicos. La tendencia que presenta la informática dentro de este ámbito tiene su mayor auge debido al crecimiento alcanzado por internet. (Graells, 2008)

Con el desarrollo de Internet, surgen las aplicaciones Web, alcanzando en los últimos años una mejora considerable a través del enriquecimiento de la experiencia del usuario. Estas aplicaciones han logrado atraer más público accediendo a ellas desde cualquier lugar, pero los resultados obtenidos no fueron satisfactorios. El navegador Web obligaba a las aplicaciones a tener una interfaz estática lo que causaba una recarga de página para obtener datos del servidor. (Dr. Jon Kepa Gerrikagoitia, 2009)

Para contrarrestar estas desventajas surgen las Aplicaciones Enriquecidas de Internet (RIA).Estas pueden hacer que las interacciones de los clientes sean llamativas, dinámicas y útiles; en una palabra: atractivas (WebSchema, 2010).La utilización de las RIA combina las ventajas de las aplicaciones tradicionales de escritorio como copiar, cortar y pegar, redimensionar columnas, y ordenar, con el alcance y la flexibilidad de presentación y despliegue que presenta la Web. Constituyendo una de las alternativas para el desarrollo de estas aplicaciones, la inclusión de escenarios tridimensionales interactivos.

Cuba ha trabajado aceleradamente en el desarrollo de las TIC. En La Universidad de las Ciencias Informáticas (UCI) específicamente el Centro de Informática Industrial (CEDIN), cuenta con una línea de productos de software llamada Visualización Web, la cual se dedicó a desarrollar componentes para la representación de gráficos o escenarios 3D en la Web. Entre sus logros podemos destacar el desarrollo del “Paseo Virtual de la industria informática cubana”; que constituye la experiencia más completa que se ha tenido en este campo en la UCI. Llevar a cabo este proyecto aportó experiencias positivas, sin embargo, uno de los factores que influyó de manera negativa durante la implementación del mismo fue la no existencia de una arquitectura que guiara el proceso de desarrollo.

La Arquitectura de Software de Dominio Específico (DSSA) es una variante para la construcción de aplicaciones, permitiendo la reutilización de módulos de aplicaciones que se encuentren en el mismo dominio, de esta manera, se puede utilizar para su desarrollo el estilo arquitectónico basado en tres capas: Presentación, Lógica del Negocio y Acceso a Datos, donde cada capa se trata de forma independiente propiciando que si hay cambios de arquitectura se pueda acceder a los datos, cambiar el negocio o modificar la presentación y solo sería afectada la capa en cuestión.

Otro aspecto a tener en cuenta es que en nuestra organización no se han establecido requisitos de referencia para el diseño de interfaces, que facilite a los desarrolladores la creación de aplicaciones enriquecidas de internet, con altos niveles de usabilidad y eficiencia del usuario. Surgiendo así la necesidad de estructurar la Capa de presentación de manera tal que sea funcional, mantenible y portable, para brindar una solución genérica, estandarizada y basada en componentes del sistema logrando la satisfacción de los requerimientos del usuario.

Teniendo en cuenta lo expuesto se define como **problema científico**: ¿Cómo visualizar datos en una Arquitectura de Software de Dominio para Aplicaciones Enriquecidas de Internet? De esta manera, se define como **objeto de estudio** las Arquitecturas de Software de Dominio Específico. Se plantea como **objetivo general:** Propuesta de la Capa de presentación de una Arquitectura de Software de Dominio Específico para la construcción de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos. El **campo de acción** se enfoca en la Capa de presentación. Para dar respuesta a los objetivos planteados se realizaron las siguientes **tareas de investigación**:

- Selección de las herramientas y tecnologías adecuadas que cumplan los requisitos para ser utilizadas en el desarrollo de la Capa de presentación.
- Caracterización de los lenguajes de programación a emplear para la propuesta de la Capa de presentación.
- Descripción de los componentes que integran la propuesta de solución.
- Representación de las vistas de la arquitectura para su descripción.

- Validación de la propuesta de solución con la utilización de técnicas y métodos de evaluación de arquitectura.

- Definición de los atributos de calidad que se utilizaran en la evaluación de la arquitectura.

Mediante el desarrollo de esta investigación científica se espera tener como resultado el diseño de la estructura de la Capa de presentación de la arquitectura propuesta.

Los capítulos se conformaron de la siguiente manera:

Capítulo 1: Fundamentación Teórica. Se define todo lo relacionado con el objeto de estudio, así como los conceptos asociados al problema planteado. Además, se hace un análisis del estado actual de las tecnologías a utilizar, de los métodos y herramientas para llevar a cabo la investigación.

Capítulo 2: Solución Propuesta. Se definen los métodos, procedimientos y técnicas utilizadas para llevar a cabo la investigación. Se presenta los aspectos esenciales de la capa a desarrollar realizando una descripción detallada de la arquitectura y representando las características del sistema en las vistas arquitectónicas.

Capítulo 3: Evaluación de la Arquitectura. Se describe los métodos y técnicas existentes para evaluar la arquitectura, además de los atributos de calidad más adecuados que se tendrán en cuenta para realizar la evaluación.

Los métodos científicos utilizados para darle cumplimiento a las tareas de investigación fueron los siguientes:

Analítico-sintético: Fue utilizado para el desglose de la información, para facilitar su estudio, permitiendo la extracción de los elementos más importantes permitiendo la descripción de las RIA y la caracterización de las DSSA.

Análisis histórico – lógico: Para comprender los antecedentes y las tendencias de las Arquitecturas de Software Dominio Especifico además de conceptos, términos y vocabularios propios del campo como Aplicaciones Enriquecidas de Internet , Entornos Tridimensionales, Aplicaciones Web que contribuyen en gran medida al entendimiento del trabajo.

Capítulo 1 Fundamentación Teórica

1.1 Introducción

En el presente capítulo se realiza un enfoque general de los aspectos relacionado con el marco teórico de la investigación para estructurar la Capa de presentación de una Arquitectura de Software de Dominio Especifico para la construcción de RIA. Entre los temas que serán tratados se encuentra: las DSSA, las RIA, Arquitectura de software, además de las tecnologías a utilizar.

1.2 Aplicaciones Enriquecidas de Internet.

A lo largo de la historia del Internet se han desarrollado diferentes herramientas para crear aplicaciones propias para la Web. Una de las nuevas tecnologías para el desarrollo de aplicaciones web son las RIA, un nuevo tipo de aplicaciones óptimas e impactantes que las tradicionales aplicaciones Web, cuyo objetivo es incrementar, mejorar sus opciones y capacidades.

Las características más importantes de las RIA que la hacen superior a las aplicaciones Web tradicionales es que son altamente compatibles e interoperables con la mayoría de clientes, servidores de aplicaciones y herramientas de bases de datos, usan lenguaje XML y/o Java/JavaScript y otros estándares, además buscan optimizar y mejorar la interactividad entre el cliente y el servidor reduciendo el tráfico de red. (Elia Vite Villegas ITL, 2010)

Con la finalidad de mejorar la experiencia del usuario a través de interfaces propias de aplicaciones de escritorio, las RIA suelen ser más interactivas y con mayores capacidades gráficas y multimedia. Además, funcionan en cualquiera de los sistemas operativos que tenga instalado el usuario en su equipo (multiplataforma). (Dr. Jon Kepa Gerrikagoitia, 2009)

Entre los beneficios principales de aplicaciones RIA tenemos un mejoramiento importante en la experiencia visual, que hacen del uso de la aplicación algo muy sencillo, ofrece mejoras en la conectividad y despliegue instantáneo de la aplicación, agilizando su acceso, garantizan la desvinculación de la capa de presentación es decir,

acceso a la aplicación desde cualquier computador en cualquier lugar del mundo, solo es necesario para esto tener instalado el complemento Adobe Flash Player en el navegador y tener acceso a alguna red, de esta manera ,los programadores evitan tener que escribir códigos para diferentes navegadores. Estas aplicaciones presentan más capacidad de respuesta, ya que el usuario interactúa directamente con el servidor, sin necesidad de recargar la página. (Area3d, 2011)

En la tabla que se muestra a continuación podemos apreciar una comparación entre las aplicaciones web, las de escritorio y las RIA donde se demuestra las ventajas de estas últimas por encimas de las restantes.

Características	**Escritorio C/S**	**WEB**	**RIA**
Cliente Universal(navegador)	No	SI	SI
Instalación en cliente	Compleja	Simple	Simple
Capacidad de Interacción	Alta	Limitada	Alta
Lógica de negocios en servidor	SI	SI	SI
Lógica de negocios en cliente	SI	Limitada	SI
Actualización Completa de páginas	NO	SI	No
Requerimientos al servidor frecuentemente	NO	SI	NO
Comunicación Servidor-cliente	SI	NO	SI
Funcionalidad fuera de línea	SI	NO	SI

Tabla 1. Comparación entre aplicaciones WEB, Escritorio y RIA. (ORT, 2008)

1.3 Arquitectura de Software

Al consultar la bibliografía aparecen varias definiciones para Arquitectura de Software; en un sentido amplio se puede expresar como el diseño de más alto nivel de la estructura de un sistema, programa o aplicación. La definición oficial de Arquitectura de Software dada por el Instituto de Ingenieros en Electricidad y Electrónica2 (IEEE, por

sus siglas en inglés) plantea lo siguiente: "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." (IEEE, 2000)

La Arquitectura de Software establece las bases para que todo el equipo de desarrollo trabaje en una misma línea que les permita alcanzar los objetivos del sistema. Da una visión de lo que el sistema debe hacer y es el arquitecto el encargado de definir las tecnologías y herramientas, dígase patrones de diseño, estilos arquitectónicos, lenguajes de implementación y marcos de trabajo.

Existen tres razones por las cuales la Arquitectura de Software es importante para sistemas de software. Es un vínculo de comunicación entre los *Stakeholders* (interesados), una expresión de las decisiones tempranas del diseño y una abstracción del sistema reusable y transferible. Vale resaltar que la arquitectura tiene una relación directa con los requerimientos no funcionales de un sistema, los que permiten definir escenarios para la descripción arquitectónica. (Paul Clements, 2004)

En el CEDIN la arquitectura a utilizar para el desarrollo de RIA es la DSSA se basa en el concepto de una arquitectura de software genérica aceptada para un dominio, es un conjunto de componentes software especializado para dicho dominio. La arquitectura debe aplicarse a una amplia gama de sistemas en el dominio elegido, por lo que debe ser general y flexible. Una DSSA la compone un modelo de dominio a elaborar, los requisitos de referencia, una arquitectura de referencia, su infraestructura de soporte y un proceso y la metodología para crear una instancia, refinar y evaluarlo. (Tracz, 1995)

El objetivo principal de la creación de la DSSA es reutilizar la mayor cantidad de objetos como sea posible; además antes de buscar una solución para cada problema que surja es más factible la reutilización de módulos de aplicaciones que se encuentren en el mismo dominio de esta manera se estaría optimizando el desarrollo. (KAISER, 2005.)

Por todo lo definido anteriormente se puede decir que la arquitectura aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño Es el eslabón fundamental para que un software se desarrolle con la calidad requerida trayendo consigo una buena comunicación entre los diferentes participante.

1.4 Estilos arquitectónicos

Los estilos arquitectónicos “definen a una familia de sistemas en términos de un patrón de organización estructural. Específicamente, un estilo arquitectónico determina el vocabulario de componentes y conectores que puede ser usado así como un conjunto de restricciones de cómo pueden ser combinados”. (Pressman, 2001)

Los estilos generalmente proveen guía y análisis para crear una clase amplia de arquitecturas en un dominio específico donde los patrones se enfocan en solucionar los problemas más pequeños, más específicos dentro de un estilo dado. (Nicolás Kiccillof, 2004)

1.4.1 Tubería y filtros

Una tubería (pipeline) es una popular arquitectura que conecta componentes computacionales (filtros) a través de conectores (pipes), de modo que las computaciones se ejecutan a la manera de un flujo.

El estilo tubería y filtros está compuesto por componentes que son los filtros. Cada filtro lee flujos de datos en sus entradas y produce flujos de datos en sus salidas. Los conectores que son los tubos transmiten la salida de un filtro como entrada de otro y no se efectúa ningún otro procesamiento visible. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. 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. (Carlos Reynoso, 2004)

En la figura 1 se pueden apreciar los componentes más importantes en este estilo son: el origen de los datos, los filtros, las tuberías y los consumidores de los datos.

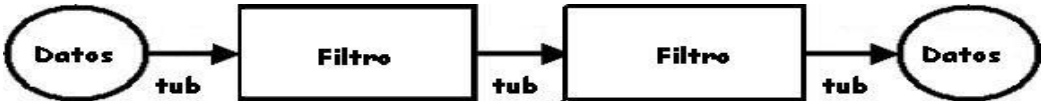

Figura 1. Estructura de filtros y tuberías

1.4.2 Estilos de llamada y retorno

Este estilo se centra en la interacción de un usuario con el propio sistema. Se puede dividir la arquitectura en dos partes, la primera representa la interfaz del usuario con el que este realiza la llamada al sistema, la segunda contiene la lógica del negocio que se realiza tras la correspondiente llamada del usuario. Dentro de esta familia de estilos se encuentran, las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas. (Carlos Reynoso, 2004)

1.4.2.1 Arquitecturas en Capas.

La arquitectura en capas es la vista conceptual de la estructura de una aplicación. Toda aplicación contiene código de presentación, de procesamiento de datos y de almacenamiento de dato. Este patrón se define como una organización jerárquica, donde cada capa se trata de forma independiente, de manera tal que cada una proporcione servicios a la capa superior y se sirve de las prestaciones que le brinda la inferior; de esta manera, al ocurrir un cambio, sólo se tendrían que realizar las correcciones necesarias en el nivel requerido sin tener que revisar el código de otros niveles. (Nicolás Kiccillof, 2004)

Son varias las ventajas que aporta el estilo en capas que lo hacen uno de los más usados en el mundo entre ellas encontramos las siguientes:

- Para la implementación permite la división de un problema complejo en una secuencia de pasos crecientes que lo hace más entendible y organizado.
- Permite optimizaciones y refinamientos sin afectar el sistema en conjunto.
- Permite la reutilización ya que se puede utilizar implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces ante las capas adyacentes.

En la figura Figura 2 se representa una arquitectura de tres capas en la que se puede observar la interacción de una capa superior con una inferior mediante interfaces que se encargarían de definir las funcionalidades que dicha capa debe brindar. Lo más frecuentemente es que haya varios ordenadores donde se encontraría la capa de

presentación y las capas de negocio podrían estar en el mismo o utilizar varios ordenadores dependiendo de las necesidades y la complejidad del sistema.

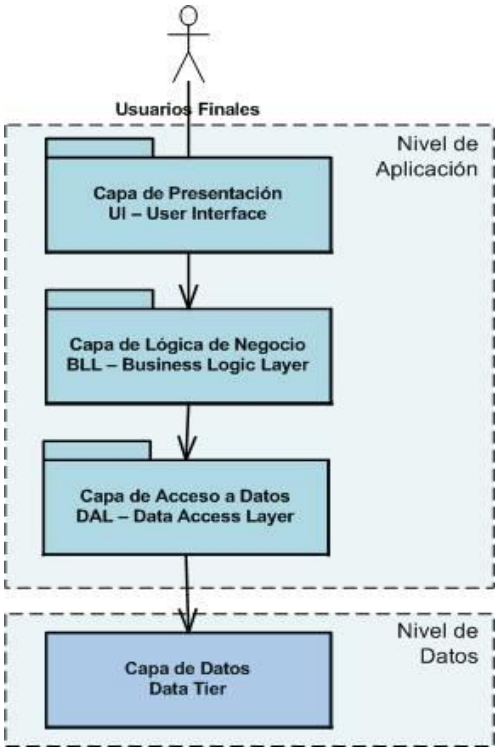

Figura 2. Arquitectura en 3 capas

1.4.3 Arquitecturas Orientadas a Objetos

Este estilo también es conocido como Arquitectura Basada en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos que se basan en los principios de la Programación Orientada a Objetos, encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación; los objetos y sus interacciones el centro de las incumbencias en el diseño de la arquitectura. Este estilo se caracteriza por la modificación la implementación de un objeto sin afectar a sus clientes, se hace posible descomponer problemas en colecciones de agentes en interacción y donde un objeto es ante todo una entidad reutilizable en el entorno de desarrollo. (Carlos Reynoso, 2004)

1.4.4 Arquitectura basada en componentes

Los sistemas de software basados en componentes se basan en principios definidos por una ingeniería de software específica y son las unidades de modelado, diseño e implementación, donde las interfaces y sus interacciones son el centro de responsabilidades en el diseño arquitectónico facilitando de forma general la reutilización del software, simplificación de las pruebas, simplificación del mantenimiento del sistema y una mayor calidad de la aplicación. (Garlan, 1994)

El uso de este estilo proporciona la reutilización en los procesos de ingeniería que están muy incluidos en el concepto de componente, pues un componente es una parte no trivial, casi independiente y reemplazable de un sistema que cumple una función dentro del contexto de una arquitectura bien definida. Un componente cumple con un conjunto de interfaces y provee la realización física de ellas (Brown, 1998).

1.5 Patrones de diseño

Un patrón de diseño describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular. (Cordeso.E, 2004)

Los patrones de diseño son soluciones simples a problemas específicos comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia, que se han demostrado que funcionan. Los patrones de diseño no son fáciles de entender, pero una vez entendido su funcionamiento, los diseños serán mucho más flexibles, modulares y reutilizables. Ha evolucionado el diseño orientado a objetos y todo buen arquitecto de software debería conocerlo. (García, 2005)

1.5.1 Patrón Modelo Vista Controlador

El patrón conocido como Modelo-Vista-Controlador (MVC) es un patrón arquitectónico, que separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes el Modelo, la Vista y el Controlador. (Kiccillof C. R., 2004)

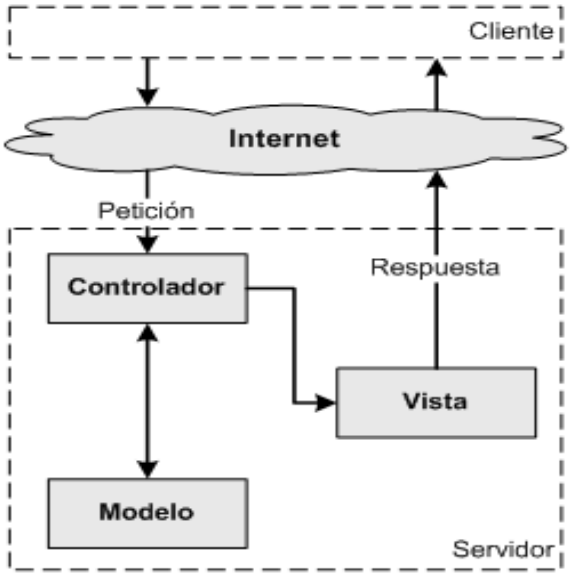

Figura 3. Patrón Modelo Vista Controlador

El Modelo: es el objeto que representa los datos del programa, es un conjunto de clases que representan la información del mundo real que el sistema debe procesar sin tomar en cuenta ni la forma en la que esa información va a ser mostrada ni los mecanismos que hacen que esos datos estén dentro del modelo, es decir, el Modelo no tiene conocimiento específico de los Controladores o de las Vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el Modelo y sus Vistas, y notificar a las Vistas cuando cambia el Modelo. Típicamente el modelo de clases contendrá funciones para consultar, insertar y actualizar información de la base de datos.

La Vista: es el objeto que maneja la presentación visual de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. Interactúa con el Modelo a través de una referencia al propio Modelo. Una vista está asociada a un modelo, pudiendo existir varias vistas asociadas al mismo modelo.

El Controlador: es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo. Cuando se realiza algún cambio, bien sea por cambios en la información del Modelo o por alteraciones de la Vista. Actúa como intermediario entre el Modelo, la Vista y cualquier otro recurso necesario para generar una página.

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo independientemente de la representación visual. La separación entre vista y controlador puede ser secundaria en aplicaciones de clientes ricos. En aplicaciones de Web, por otra parte, la separación entre la vista (el browser) y el controlador (los componentes del lado del servidor que manejan los requerimientos de HTTP) está mucho más específicamente definida.

1.5.2 Patrones GOF

Estos patrones son de gran utilidad en el diseño de una aplicación debido a las características imprescindibles que presentan, entre ellas se pueden mencionar: la independencia del resto de los patrones, se aplican en situaciones muy comunes ya que son soluciones simples que no indican cómo diseñar un sistema sino solo aspectos puntuales del mismo, el uso de un patrón no se refleja en el código ya que cuando se realiza la implementación es difícil determinar qué patrón de diseño se ha usado. Otra de las características es que es difícil reutilizar la implementación de un patrón ya que en la misma aparecen clases concretas, además suponen una sobrecarga de trabajo pues se usan más clases y es necesario delegar más mensajes.

1.5.2.1 Observador

Este patrón define una dependencia de uno-a-muchos entre objetos, de manera tal que al cambiar de estado un objeto se notifique y se actualicen automáticamente todos los objetos que dependen del mismo. (Blasi)

Este tipo de interacción también se conoce como publicar-suscribir, proporcionando un mecanismo de propósito general para la notificación de eventos y la comunicación indirecta en un sistema. Se origina así un nuevo método de diseñar los sistemas orientados a objetos. Un diseño basado en la notificación de eventos presenta algunas características que lo distinguen entre las que se encuentran:

- No se requiere el acoplamiento directo entre emisores y receptores.
- Un evento individual puede transmitirse a cualquier número de suscriptores.

- Puede generalizarse la reacción ante un evento en los objetos Respuesta**.**
- Es relativamente fácil lograr la concurrencia con solo ejecutar cada Respuesta en su propio subproceso múltiple.

Además de las ventajas de un menor acoplamiento, de la transmisión y de una concurrencia simple, algunas veces un sistema basado en la notificación de eventos alcanza mayor eficiencia cuando las cadenas largas de mensajes pueden compactarse en un nivel de indirección. (Wendy Boggs, 2002)

1.5.2.2 Singleton

Este patrón garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella. Es recomendable usarlo cuando varios elementos distintos precisan referenciar a un mismo elemento y se desea asegurar que no hay más de una instancia de ese elemento ,además la única instancia debería ser extensible mediante herencia y los clientes deberían ser capaces de utilizar una instancia extendida sin modificar su código. Este patrón determina un punto de acceso global a esta clase o instancia garantizando resolver el problema.

1.5.3 Patrones GRASP

Los Patrones Generales de Software para Asignación de Responsabilidades (GRASP, por sus siglas en inglés) son considerados buenas prácticas a seguir en la creación de una aplicación de cualquier tipo. Estos patrones describen principios básicos, simples y fundamentales de diseño de objetos para la asignación de responsabilidades, expresados en forma de patrones. (Larman)

1.5.3.1 Experto

Este patrón consiste en asignar una responsabilidad al experto en información que sería la clase que tiene la información necesaria para llevar a cabo la responsabilidad. Con este patrón se conserva el encapsulamiento de la información, ya que los objetos utilizan su propia información para llevar a cabo las tareas. Normalmente, esto conlleva un bajo acoplamiento, lo que da lugar a sistemas más robustos y más fáciles de mantener, distribuyendo el comportamiento entre las clases que contienen la información requerida, por lo tanto, se estimula las definiciones de clases más

cohesivas y “ligeras” que son más fáciles de entender y mantener. (Machorro Reyes, 2010)

1.5.3.2 Bajo Acoplamiento

Este patrón define como dar soporte a una dependencia escasa y a un aumento de la reutilización, enfocándose en asignar una responsabilidad para mantener bajo acoplamiento. El bajo acoplamiento estimula la asignación de responsabilidades de forma tal que, la inclusión de estas no incremente el acoplamiento, creando clases más independientes y con mayor resistencia al impacto de los cambios. (Wendy Boggs, 2002)

1.5.3.3 Alta Cohesión

Cada elemento del diseño debe realizar una labor única dentro del sistema. Este patrón indica que una clase tiene responsabilidades moderadas en un área funcional y colabora con las otras para llevar a cabo las tareas. Asignar una responsabilidad para mantener bajo el acoplamiento, de manera que una clase con bajo acoplamiento no dependerá de muchas otras trae consigo una mejor claridad y la facilidad con que se entiende el diseño, simplicidad en el mantenimiento y mejoras en funcionalidad, soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.

1.6 Bases Tecnológicas

Este epígrafe abarca las tecnologías a emplear para el desarrollo de la propuesta de capa de presentación ya sean las herramientas, lenguajes y ambientes de desarrollo.

1.6.1 Entorno de Desarrollo Integrado

Un entorno de desarrollo integrado (IDE, del inglés, *Integrated Development Environment*) es una aplicación compuesta por una serie de herramientas que utilizan los programadores para escribir código. Esta herramienta puede estar pensada para su utilización con un único lenguaje de programación o bien puede dar cabida a varios de estos. Las herramientas que normalmente componen un entorno de desarrollo integrado son las siguientes: un editor de texto, un compilador, un intérprete, herramientas para la

automatización, un depurador, un sistema de ayuda para la construcción de interfaces gráficas de usuario y, opcionalmente, un sistema de control de versiones.

Un excelente ambiente de desarrollo debe ser multiplataforma, soportar diversos lenguajes de programación y múltiples idiomas, debe estar integrado con sistemas de control de versiones y *framework* populares, reconocer sintaxis, tener extensiones y componentes para el IDE, un depurador, además debe importar y exportar proyectos.

1.6.2 Flex 4 SDK

Flex 4 SDK implementa una nueva arquitectura de componente *Skins*, que soporta un nivel de expresividad en los acuerdos de integración regional que no se habían visto anteriormente. Con esta nueva arquitectura de componentes y aspectos de aplicación, se separa la parte lógica de los componentes visuales de manera que la personalización, el comportamiento o la apariencia del componente sean mucho más sencillos. Además, en el SDK de Flex 4, hemos mejorado el rendimiento del compilador de Flex, el aumento de numerosos idiomas, además, el compilador ha mejorado también su desempeño y contiene soporte para nuevas capacidades del Flash Player 10. (Madeinflex)

El SDK de Flex, a partir de Flex 3, fue liberado bajo la licencia de software libre *Mozilla Public License*. Gracias a esto, se puede desarrollar de forma “libre” desde cualquier plataforma. El reproductor Flash, el *runtime* de Flex, y Adobe Flash Builder (el IDE de desarrollo construido sobre Eclipse) permanecen en su calidad de software privativo.

1.6.3 Adobe Flash Builder 4

Adobe Flash Builder antes conocido como Macromedia FLEX, es la evolución de los entornos para el desarrollo de aplicaciones FLASH programadas con ActionScript. Constituye una nueva tecnología creada por la empresa Macromedia, y en la actualidad forma parte del paquete de aplicaciones de Adobe. De manera simplificada se puede decir que es la combinación del SDK Flex 4 con el IDE Eclipse.

Es una herramienta profesional de desarrollo utilizada no solo para crear RIA, también podremos desarrollar aplicaciones de escritorio multiplataforma utilizando el marco de

trabajo de código abierto de Flex. Flash Builder 4 incluye compatibilidad con la codificación inteligente, la depuración y el diseño visual, y presenta potentes herramientas de prueba que agilizan el desarrollo y hacen que las aplicaciones tengan un rendimiento más elevado. Es compatible con los sistemas operativos Windows y Mac OS incluye editores de MXML, el lenguaje ActionScript. Permite la personalización de la apariencia de una aplicación utilizando CSS y editores de propiedades gráficas. (Adobe Corporation)

1.6.4 Lenguajes de Programación

Los lenguajes de programación son herramientas que permiten la comunicación entre una persona y un ordenador, por lo tanto, es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo.

1.6.4.1 MXML. Lenguaje de marcas de Flex

MXML, es un lenguaje descriptivo creado a partir de las reglas de XML, su acrónimo "Macromedia eXtensible Markup Language". Todas las normas en la construcción de un documento XML se aplican también al crear uno MXML, es decir, un documento MXML en uno XML bien estructurado. MXML permite, entre otras cosas, construir interfaces visuales de una forma muy intuitiva y ordenada, crear y extender componentes visuales, describe interfaces de usuario, crea modelos de datos. (MadeInFlex)

Como en la mayoría de lenguajes orientados a objetos, MXML permite el uso de interfaces para definir la funcionalidad de los componentes. Las interfaces son una buena manera de separar la definición de funciones de las implementaciones, son creadas para actuar como un modelo para cualquier objeto que los implementa. MXML se combina con ActionScript, lenguaje de scripting basado en el estándar ECMAScript (ActionScript, JScript), para obtener interactividad. En una aplicación Flex, se usa se usa MXML para establecer rápidamente la estructura/apariencia de la aplicación.

1.6.4.2 ActionScript

ActionScript es el lenguaje de programación propio de Flash que en su nueva versión ActionScript 3.0, pretende alcanzar un modelo de programación mejorado, más consistente y acorde con los estándares de la industria. Activa, entre otras muchas

cosas, la interactividad y la gestión de datos en el contenido y las aplicaciones de Flash. (Adobe Corporation, 2011). Aumenta las posibilidades de creación de scripts y la velocidad de ejecución de código de sus versiones anteriores, se ha diseñado para facilitar la creación de aplicaciones muy complejas con conjuntos de datos voluminosos y bases de código reutilizables y orientadas a objetos.

Ambos lenguajes de programación pueden ser vinculados para el desarrollo híbrido de aplicaciones haciendo uso del SDK Flex 4. El resultado de la compilación de este código genera un archivo binario (SWF), interpretado por la máquina virtual de Adobe, también conocida como Adobe Flash Player. Vale resaltar que esta máquina virtual solo reconoce el lenguaje de programación ActionScript en sus versiones 2.0 y 3.0; esto significa que cualquier código MXML que se escriba para una aplicación deberá ser transformado por el compilador MXML en ActionScript, elemento catalogado como una desventaja pero en realidad beneficia en gran medida a los desarrolladores que se abstraen al desarrollar interfaces completas en períodos de tiempo relativamente cortos gracias a los componentes que brinda MXML.

1.6.5 Servidores Web

1.6.5.1 Apache

Apache es un servidor web flexible, rápido y eficiente, continuamente actualizado y adaptado a los nuevos protocolos (HTTP/1.1). Entre sus características destacan que corre en varias plataforma WinXX, Netware, Unís y Linux es decir es multiplataforma. Este servidor HTTP es uno de los más utilizados y populares pues es compatible con múltiples sistemas operativos y es una tecnología libre de código abierto.

El diseño modular de Apache es altamente configurable, es un servidor quede ser adaptado a diferentes entornos y necesidades, con los diferentes módulos de apoyo que proporciona, y con la API de programación de módulos, para el desarrollo de módulos específicos, gracias a esto se han desarrollado diversas extensiones entre las que destaca PHP, un lenguaje de programación del lado del servidor .Entre las diferentes facilidades que nos ofrece Apache se puede mencionar su sistema de configuración y compilación, soporte nativo para Ipv6, su capacidad de tener múltiples lenguajes a la hora de mostrar los mensajes de error. (Ciberaula, 2010)

1.6.5.2 Microsoft Internet Information Services

Internet Information Servicies mejor conocido como IIS, sólo funciona bajo servidores Microsoft. Debe ser usado bajo licencia, es un servidor web propietario cuyos servicios se limitan a los ordenadores que trabajan con Windows que es la plataforma de servidores desarrollada por Microsoft .Ha tenido un ascenso significativo por su compatibilidad con los host y por sus nuevas innovaciones como .Net, Silverlight, entre otros. Soporta solo los lenguajes desarrollados por Microsoft y algunos otros estándares como Asp, Asp.net, Ajax, ms SQL, MySQL, XML.

1.6.6 Metodologías de Desarrollo

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos de software. Se plantea que una metodología impone un proceso disciplinado sobre el desarrollo de software con el objetivo de hacerlo más predecible y eficiente, por tanto, define un camino reproducible para obtener resultados confiables. (Paloma Cáceres, 2010)

Haciendo uso de una metodología, se debe ser capaz de dividir un proyecto en etapas, conocer qué tareas realizar en cada una de las etapas, así como identificar qué restricciones deben aplicarse, qué técnicas y herramientas se emplean y cómo se controla y gestiona un proyecto. Las metodologías se pueden clasificar en metodologías de desarrollo tradicionales y metodologías de desarrollo ágiles.

Las metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el objetivo de conseguir un software más eficiente y predecible, básicamente consisten en dividir el proceso de desarrollo en etapas, de una manera secuencial. Este tipo de metodologías proveen de un alto grado de ordenamiento. Las metodologías ágiles surgen como una extensión a las metodologías tradicionales para mejorar el desarrollo de sistemas, según el tipo de proyecto y empresa, añadiendo y mejorando (optimizando) las prácticas de desarrollo de software. (José H. Canos).

Con la utilización de una metodología de desarrollo de software se busca una corrección y control en cada etapa de desarrollo del software, lo que nos permitirá una

forma sistemática para obtener un producto correcto y libre de errores. A continuación se describen características de tres de ellas que propician la elección de una de ella.

1.6.6.1 Programación Extrema.

Mientras que el Proceso Unificado de Software (RUP, por sus siglas en inglés) intenta reducir a complejidad del software por medio de estructura y la preparación de las tareas pendientes en función de los objetivos de las fases y actividad actual, Programación Extrema (XP, por sus siglas en inglés), como toda metodología ágil lo intenta por medio de un trabajo centrado en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, originando el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores. Esta metodología se basa en realimentación continua entre el cliente y el equipo de desarrollo. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico (Gerónimo, 2008). En el ciclo de vida de XP se planifica, se desarrolla, se prueba y se diseña.

XP plantea la planificación como un permanente diálogo entre la parte empresarial y técnica del proyecto, en la que la primera decide el alcance, la prioridad, la composición de las versiones y la parte técnica seria los responsables de estimar la duración requerida para implementar las funcionalidades deseadas por el cliente. En el desarrollo el cliente debe estar disponible y todo el código al ser común se puede contribuir desde cualquier parte del proyecto, este código se debe desarrollar en pareja ya que es una de las particularidades; además se deja todas las optimizaciones para el final.

En la etapa de Pruebas se deben implementar pruebas de aceptación y publicar los resultados, estas son creadas a partir de las historias de usuario; todo el código debe ir acompañado de una unidad de prueba. XP establece recomendaciones o premisas para el diseño, debe ser lo más simple posible, todos los desarrolladores deber tener una visión de lo que el sistema hace, se crean soluciones puntuales para reducir riesgo y no añade funcionalidades en las primeras etapas.

1.6.6.2 Scrum

Scrum, más que una metodología de desarrollo software, es una forma de auto-gestión de los equipos de programadores. Un grupo de programadores decide cómo hacer sus

tareas y cuánto van a tardar en ello. Scrum ayuda a que trabajen todos juntos, en la misma dirección, con un objetivo claro. En esta metodología un proyecto se organiza en bloques temporales cortos y fijos que pueden durar un mes natural y hasta de dos semanas si se necesita llamados iteraciones. (Proyectoságiles, 2010)

Las actividades que se realizan en Scrum están dadas de la siguiente forma:

La planificación de la iteración que está dividido en dos partes Selección de requisitos y la Planificación de la iteración en la primera lo que se hace es una reunión con los clientes donde presentan al equipo de desarrollo una lista priorizada de requisitos del producto o proyecto, y sucesivamente se realiza la lista de tareas de la iteración necesarias para desarrollar los requisitos a los cuales se comprometieron.

Ejecución de la iteración: Diariamente se realizaran reuniones de sincronización donde cada miembro del equipo inspecciona el trabajo que el resto está realizando. Aquí se sacan las dependencias entre tareas, el progreso hacia el objetivo de la iteración, y los obstáculos que pueden impedir este objetivo.

Inspección y adaptación: El último día de la iteración se realiza la reunión de revisión de la iteración que también se divide en dos partes: La demostración en la cual el equipo presenta al cliente los requisitos completados en la iteración, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, re planificando el proyecto.

La retrospectiva: el equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar adecuadamente, mejorando de manera continua su productividad.

Si se lleva a cabo este proceso obtendríamos una gestión regular de las expectativas del cliente y basada en resultados tangibles, resultados, flexibilidad respecto a las necesidades del cliente, cambios en el mercado, gestión sistemática del Retorno de Inversión (ROI), mitigación sistemática de los riesgos del proyecto, productividad y por ultimo un alineamiento entre el cliente y el equipo de desarrollo.

1.6.6.3 AUP

Proceso Unificado Ágil (AUP, por sus siglas en inglés) es una versión simplificada de RUP, describe de forma simple y fácil la creación de software funcional usando técnicas ágiles y aplicando las mejores características de RUP. AUP es un refinamiento ágil que simplifica el RUP reduciendo los flujos de trabajo y el número de artefactos, en comparación AUP tiene solamente 7 los cuáles algunos son combinaciones de dos de los de RUP.

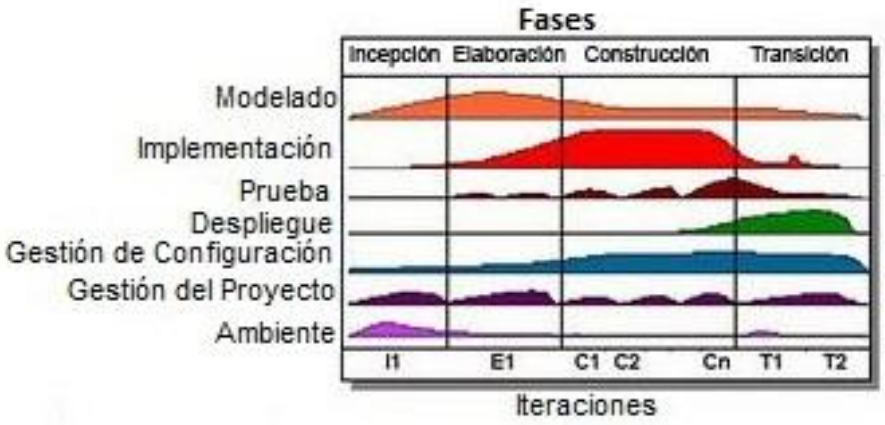

Figura 4. Ciclo de vida de AUP

Como se muestra en la figura los flujos de trabajo son diferentes aquí el modelo es una parte importante pues abarca el Modelado de Negocios, Requisitos y Análisis y Diseño del RUP. Otro de los cambios importantes es la agrupación de Configuración y Gestión del Cambio resultando Gestión de la configuración .Sin embargo a pesar de las diferencias AUP como RUP define también cuatro fases:

Inicio: Identifica el alcance inicial y dimensión del proyecto, una propuesta de arquitectura para el sistema y presupuesto del cliente.

Elaboración: Prueba la arquitectura del sistema.

Construcción: Construye el software sobre una base estable, las necesidades se van resolviendo de forma incremental comenzando por las de más alta prioridad para los clientes del proyecto.

Transición: Validación e implantación del sistema en el ambiente de producción.

El producto AUP proporciona enlaces a muchos de los detalles si uno está interesado pero no obliga seguir los detalles, es muy simple debido a que se describe consistentemente utilizando un número reducido de páginas, la atención se centra en las actividades que realmente cuentan. Cuando desarrolla un producto con AUP va a resultar fácil de manejar a través de cualquier herramienta de edición de HTML, ya que puede usar cualquier herramienta que se ajuste al trabajo que quiere realizar mayormente de código abierto.

De manera general desarrollar un producto usando AUP nos va a facilitar un tiempo de desarrollo relativamente corto ya que se ajusta a los valores y principios del modelo ágil.

1.7 Herramientas de Modelado

1.7.1 Visual Paradigm

Esta herramienta está desarrollada por Visual Paradigm Internacional una de las principales compañías de herramientas de Ingeniería de software asistida por computadora (CASE, por sus siglas en inglés), diseñada para Ingenieros de Software, Analistas de Sistemas, Arquitectos de Sistemas y otros que estén interesados en el diseño de software orientado a objetos. Su mayor éxito consiste en la capacidad de ejecutarse sobre diferentes sistemas operativos lo que le confiere la característica de ser multiplataforma. Visual Paradigm presenta un diseño centrado en casos de uso y enfocado al negocio que permite generar un software de mayor calidad, utilizando UML como lenguaje de modelado ofreciendo soluciones de software que permiten a las organizaciones desarrollar las aplicaciones de forma rápida, eficiente. Puede integrarse a los principales IDE como Eclipse, JBuilder, Oracle JDeveloper entre otros. Soporta aplicaciones Web y puede generar código C#, VB.NET, ActionScript, C/C++ y Delphi. (Visual Paradigm, 2011)

1.7.2 Rational Rose

Rational Rose es una de las más poderosas herramientas de modelado visual para el análisis y diseño de sistemas basados en objetos. Se utiliza para modelar un sistema

antes de proceder a construirlo. Rational Rose es una plataforma independiente que ayuda a la comunicación entre los miembros de equipo, a monitorear el tiempo de desarrollo y a entender el entorno de los sistemas, propone la utilización de cuatro tipos de modelos para realizar un diseño del sistema, utilizando una vista estática, otra dinámica de los modelos del sistema, una lógica y otra física; que permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software .El modelo Rose es una imagen de un sistema desde varias perspectivas. Estos incluyen todos los diagramas de UML, actores, casos de uso, objetos, clases, componentes y nodos de despliegues en un sistema .Rose permite la aplicación de la metodología RUP. Soporta el desarrollo iterativo incremental del proceso de desarrollo de software. Cubre todo el ciclo de vida de un proyecto con la concepción y formalización del modelo. (Wendy Boggs, 2002)

1.8 Bibliotecas 3D para Flash

La plataforma Flash surgió como un medio de dos dimensiones, en la cual las animaciones Flash que existían en la web eran a base de figuras planas, muy similares a los dibujos animados de la televisión. Sin embargo, gracias a las mejoras de los navegadores y las actuales versiones de Flash Player, capaces de utilizar la aceleración por Hardware para aumentar su rendimiento, han surgido algunas iniciativas de bibliotecas para crear entornos tridimensionales utilizando ActionScript. Esta progresión ha creado un verdadero espacio con profundidad dentro del navegador, logrando costosos cálculos matemáticos que son necesarios para recrear objetos en un espacio de tres dimensiones, darles texturas, moverlos e iluminarlos.

En el panorama de proyectos de bibliotecas 3D para ActionScript nos encontramos principalmente tres opciones:Papervision3D, Away3D Flash Engine y Sandy 3D Engine.

1.8.1 Away3D

Away3D comenzó su vida como una rama del motor Papervision3D en el 2007, pero rápidamente comenzó a desarrollarse hacia la estabilidad y facilidad de uso. Away3D posee una de las rutas de actualizaciones más consistente de cualquier motor 3D para Flash, con su licencia de código abierto que permite a cualquier persona aportar una corrección de errores o de mejora de funciones. Away3D es uno de los motores

3D en tiempo real más divulgados , lo que permite la creación de una amplia gama de aplicaciones, incluyendo la visualización detallada de entornos, visualización de modelos de animación y la creación de textos, todo en 3D, mostrando una gran variedad de efectos especiales. Con Away3D, un poco de ActionScript, y una gran imaginación las posibilidades son infinitas.

Away3D ofrece uno de los marcos de trabajo más poderosos y fáciles de usar. Además de ser gratuito, cuenta con una comunidad activa que proporciona de forma *online* ayuda técnica y gratuita.

1.8.2 Papervision3D

Papervision3D es una biblioteca legible por Flash. Combinando la programación de Flash con la increíble expresión de Papervision3D se pueden crear aplicaciones Web con entornos tridimensionales que dejen impresionado a cualquier usuario. Es por eso que los diseñadores gráficos que hacen webs están incluyendo este tipo de trabajos a su currículum. Papervision3D permite la importación de objetos 3d de los principales software (3dmax, Maya, Blender) y está en continua expansión.

1.8.3 Sandy 3D

Sandy 3D *Engine* es una potente librería de animación 3D en Flash. Lejos del típico cubo rotatorio o las letras en 3D, nos provee de un fantástico set de utilidades para dotar a cualquier aplicación o juego en Flash de interesantes efectos, permitiendo cargar modelos generados por herramientas de diseño 3D, entre las que se encuentran ASE, 3DS.

1.9 Conclusiones Parciales

En el capítulo se han abordado las principales temáticas relacionadas con el tema de investigación, permitiendo un mayor entendimiento de la descripción de arquitectura de software. Además, se describieron las metodologías e IDE de desarrollo, lenguajes de programación, se analizaron las características de un conjunto de tecnologías, herramientas y tendencias para determinar en capítulos posteriores las más adecuadas para la solución al problema científico planteado.

Capítulo 2 Solución Propuesta

2.1 Introducción

En el presente capítulo se abordaran aspectos relacionados DSSA propuesta para el desarrollo de RIA, además las principales herramientas y tecnologías seleccionadas, así como el dominio elegido que favorezcan el desarrollo de la capa propuesta para obtener un producto de buena calidad que dé solución al problema planteado.

2.2 Línea base de la Arquitectura

El uso de las líneas base en el desarrollo de una aplicación o producto de software viene dado por la definición generalmente de requisitos de usuario, requisitos de software, diseños, códigos fuente, planes o procedimientos, pruebas, funcionamiento, para lograr un mejor entendimiento del sistema y una máxima abstracción en el diseño arquitectónico de la aplicación, que ayudará a mantener la coherencia y calidad del producto.

En la línea base se agrupan elementos de configuración que comparten un mismo estado, se describen las tecnologías y herramientas de software que serán usadas en el desarrollo del sistema, se presentan los estilos arquitectónicos, elementos de la arquitectura, principales patrones de arquitectura utilizados. Dentro de las personas que se vinculan directamente con la línea base encontramos: el equipo de arquitectos, el equipo de desarrolladores, y los clientes utilizándola como guía en la toma de decisiones, en la implementación y como garantía de calidad respectivamente.

2.3 Selección de las tecnologías

Basado en la descripción de las tecnologías realizada en el capítulo anterior se hace una selección de tecnologías para la arquitectura propuesta.

2.3.1 Estilo Arquitectónico

El estilo arquitectónico que más se adapta a la solución de arquitectura es la Arquitectura en 3 Capas, teniendo en cuenta que este estilo en tres capas facilita que se pueda descomponer la aplicación en varios niveles de abstracción. De esta forma, proporciona la evolución del sistema, reutilización, optimización y refinamiento, ya que los cambios solo deben afectar a la capa donde se encuentre la modificación. En la

Figura 5 Arquitectura 3 capas, que se muestra a continuación, se pueden entender mejor la arquitectura de 3 capas que será nuestra propuesta de arquitectura.

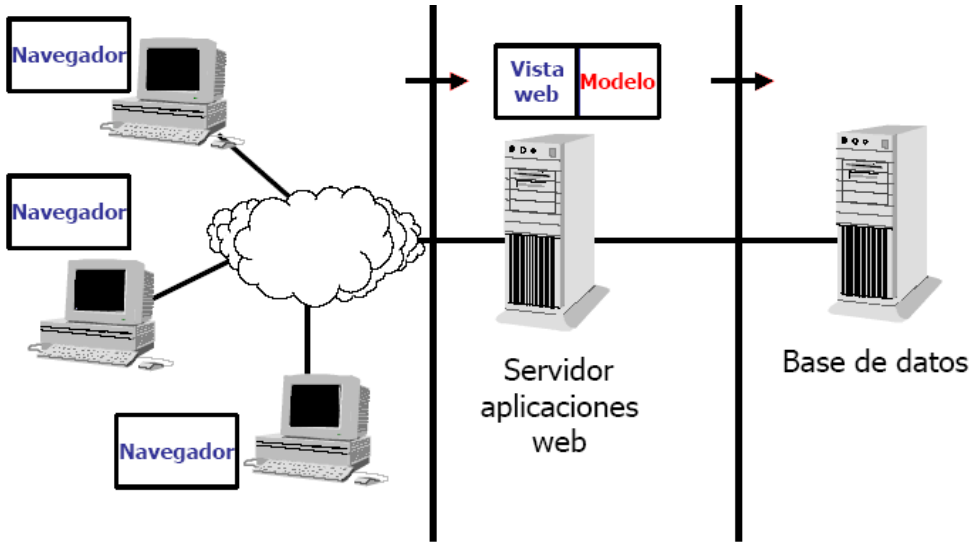

Figura 5. Arquitectura en capas

En este caso en particular describiremos la Capa de presentación la cual es la que se estructurará.

La Capa de presentación incluye diferentes elementos como Vistas, Controladores, Modelos, etc. Las responsabilidades de esta capa son las de presentar al usuario los conceptos de negocio mediante una interfaz de usuario (IU), facilitar la explotación de dichos procesos, informar sobre el estado de los mismos e implementar las reglas de validación de dicha interfaz, es decir, es aquí donde se le presenta el sistema al usuario, comunicándose con la capa de negocio. También es conocida como “capa de usuario” o “interfaz gráfica”.

Cuando se estructura una Capa de presentación hay que tener en cuenta habilidades que no están relacionadas con el desarrollador, como pueden ser las habilidades de diseño artístico, los conocimientos de accesibilidad y de usabilidad, y el control de la localización de las aplicaciones. Por tanto, lo más recomendable es que un profesional de este ámbito como puede ser un diseñador gráfico, trabaje junto al desarrollador para lograr un resultado de alta calidad. Es responsabilidad de esta capa el facilitar esta colaboración entre ambos roles.

La Capa de presentación también se ocupa de cifrado de datos y seguridad de los datos, pero la principal y más importante función que esta cumple es que los datos lleguen de manera reconocible. Esta capa presenta varias responsabilidades entre las que se encuentran:

- Navegabilidad del sistema.
- Formateo de los datos de salida.
- Validación de los datos de entrada.
- Interfaz gráfica de usuario.
- Enviar la información del usuario a los servicios de negocios para su procesamiento.
- Presentar al usuario los resultados del procesamiento de los servicios de negocios.

2.3.2 Patrones de diseño.

Se han seleccionado varios patrones arquitectónicos que definen la estructura de la Capa de presentación del sistema de software a desarrollar, estos patrones garantizarán una mayor comprensión de la arquitectura del sistema y agilizarán el posterior desarrollo e implementación del mismo.

2.3.2.1 Modelo Vista Controlador (MVC)

Analizando las características del MVC planteadas en el capítulo anterior se utilizara como patrón de diseño para la capa en cuestión, el cual forma parte del estilo arquitectónico de llamada y retorno. El MVC permite desarrollar el modelo, la vista y la controladora de forma independiente, de esta manera, cualquier modificación que se realice solamente se vería afectada esa parte.

Este patrón pertenece a un conjunto de patrones agrupados en el estilo arquitectural de presentación separada. Al utilizarlo se lograra separar el código responsable de la representación de los datos en pantalla, del código encargado de la ejecución de la

lógica de negocio. Para ello el patrón divide la Capa de presentación en tres tipos de objetos básicos: modelos, vistas y controladores como se muestra en la siguiente figura.

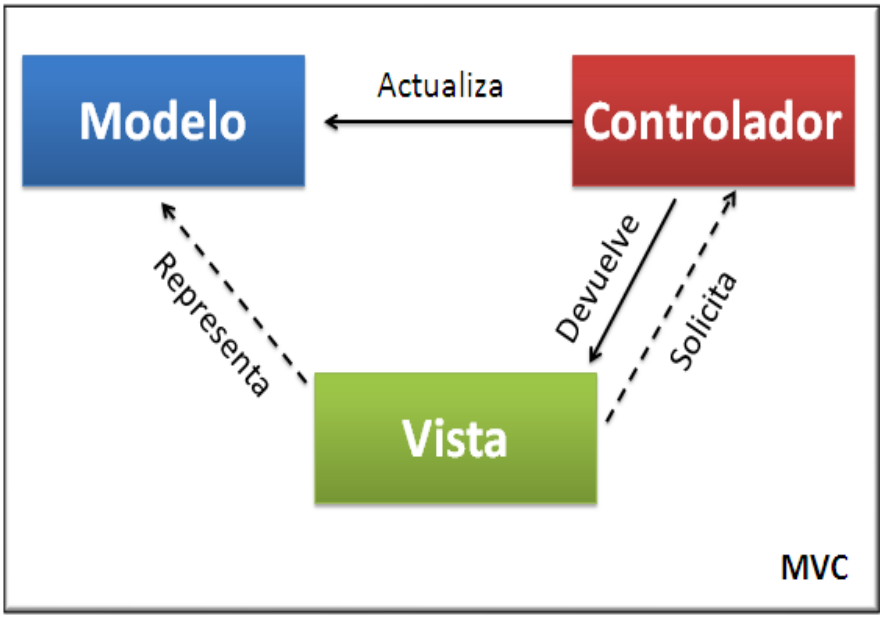

Figura 6. Representación del MVC en la Capa de Presentación

El modelo: Es el conjunto de clases encargadas de representar la información con que trabaja el usuario.

Las vistas: Son las encargadas de representar gráficamente el modelo y de ofrecer las acciones de los controladores para que el usuario pueda interactuar con este, no deben invocar ningún método que provoque un cambio de estado en el modelo.

El controlador: Organiza la interacción entre las vistas y el modelo. Recibe las peticiones del usuario, interactúa con el modelo realizando consultas y modificaciones a este, le proporciona los datos requeridos a la vista para su visualización.

2.3.2.2 Experto

La Capa de presentación utilizará este patrón para asignarle la responsabilidad correspondiente a las clases definidas expertas en la información, cuando el usuario interactúe con la aplicación sería la clase Gestionar Evento el experto en la información, que se encargaría de gestionar el evento correspondiente a la acción del usuario.

2.3.2.3 Alta cohesión

Este patrón nos permitirá delimitar las responsabilidades de cada objeto, para este caso en particular se pretende que todas las responsabilidades de las clases de interfaz, que está formada por diferentes funcionalidades, se encuentren estrechamente relacionadas, proporcionando que el software sea flexible frente a grandes cambios. Este patrón estará presente en la Capa de presentación pues al aplicar el patrón MVC cada una de las partes (Modelo, Vista, Controlador) son independientes, donde la comunicación entre ellas será mediante interfaces.

2.3.2.4 Bajo acoplamiento

Al utilizar este patrón de diseño estaremos reduciendo el principal problema que a menudo se encuentra en el acoplamiento de los componentes de la interfaz de usuario, cuando una capa realiza su trabajo se dice que las capas están acopladas y esto provoca que a medida que se vayan haciendo modificaciones en los requerimientos de la aplicación se tendrá que actualizar el código, y si por ejemplo hay un cambio en el modelo de datos tendremos que llevar esos cambios hasta la capa de Presentación cuando todo está muy acoplado, cualquier cambio en una parte de la aplicación puede provocar cambios en el resto del código y esto presenta un problema de complejidad y calidad del código.

2.3.2.5 Observador

Se utiliza para poder notificar a determinados objetos denominados suscriptores cuando otro ha sufrido un cambio de interés para estos. Permite de forma dinámica implementar dependencias entre objetos, de forma que los objetos dependientes sean notificados de los cambios que se producen en los objetos de los que dependen. En el caso particular de la Capa de presentación en una RIA este patrón estaría presente cuando se le envié una petición correspondiente a un evento generado por el usuario a la Capa de lógica de negocio, quien le enviaría una notificación si ocurre algún cambio o se modifique algún dato de manera que se actualicen las vistas con los datos actuales del sistema. Esta es la función del patrón observador y de ahí la importancia de su uso en el desarrollo del sistema.

2.3.3 Lenguaje de programación y entorno de desarrollo seleccionado.

Los lenguajes de desarrollo serán ActionScript 3.0 para el desarrollo de aplicaciones RIA utilizando el IDE Adobe Flash Builder 4, MXML para establecer rápidamente la estructura/apariencia de la aplicación ya que vinculado con ActionScript logra la interactividad en las aplicaciones que se desean desarrollar.

2.3.4 Servidor Seleccionado

Como servidor web se escogió Apache por ser una tecnología libre, y actualmente el servidor web más utilizado debido a las funcionalidades y eficiencia que brinda, además por ser un potente servidor HTTP.

2.3.5 Metodología de desarrollo a utilizar

El análisis de las características de cada una de las metodologías descritas con anterioridad trae como resultado que AUP es la metodología idónea para el desarrollo de RIA, al ajustarse al modelo ágil nos ofrece un producto de trabajo más rápido que el modelo de desarrollo lineal convencionales. Los cambios de última hora son posibles debido a la naturaleza adaptable de todo el proceso, la idea básica detrás de la mejora iterativa es desarrollar un sistema software de forma incremental, permitiendo al desarrollador aprovechar lo que va a ir aprendiendo durante el desarrollo de versiones anteriores, incrementales y entregables del sistema.

2.3.6 Herramienta Case Seleccionada

Visual Paradigm es la herramienta seleccionada pues emplea UML para el modelado. Permite crear tipos diferentes de diagramas en un ambiente totalmente visual. Es muy sencillo de usar, fácil de instalar y actualizar. La principal razón de su elección es que su licencia es legalmente pagada por la universidad y su uso es totalmente autorizado. También se tuvieron en cuenta sus características principales y las facilidades que brinda a los usuarios pues posee una interfaz amigable y fácil de utilizar.

2.3.7 Biblioteca Seleccionada

Se propone como biblioteca para la representación de gráficos 3D en la Web Aways3D por ser de código abierto y gratuito, además de contar con una robusta comunidad de desarrollo y una amplia documentación.

2.4 Descripción de la Arquitectura

En el apartado se ofrece el conjunto de modelos que describen la Línea Base de la Arquitectura, se brindará la explicación detallada de cada una de las vistas arquitectónicas de la aplicación a desarrollar con el fin de lograr un correcto entendimiento del equipo de desarrollo de las mismas, además se especifican los requerimientos no funcionales y funcionales que componen una arquitectura.

2.4.1 Propósito

La descripción de la arquitectura de un sistema tiene como propósito brindar visión arquitectónica del sistema mediante el uso de las distintas vistas. Además, proporciona una correcta evolución del mismo.

2.4.2 Alcance

Con la descripción de la arquitectura se brindará una visión general de la Capa de presentación para el desarrollo de RIA. La misma será generalizada a todos los que desarrollen este tipo de aplicaciones.

2.4.3 Estructura de la Capa de presentación

Los componentes de la Capa de presentación deben ser coherente y estar débilmente acoplados a simplificar la reutilización y el mantenimiento de la misma. En la Figura 7 se muestran los componentes que conformaran la capa.

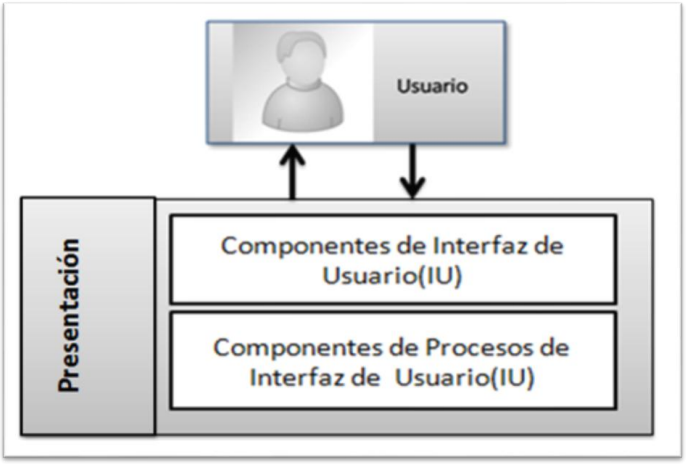

Figura 7. Componentes de la Capa de Presentación

2.4.3.1 Descripción de los Componentes

Componentes de Interfaz de Usuario

Cuando un usuario interactúa con un elemento de interfaz de usuario, se genera un evento que llama al código en una función controladora. Esta función, a su vez hace un llamado a los componentes de negocio, para implementar las acciones deseadas y obtener cualquier dato necesario que requiere ser mostrado. Finalmente, la función controladora actualiza la interfaz de usuario. De esta manera, estamos separando el código de la Vista y el Controlador. Esto ayuda mucho en el mantenimiento de esta capa, la Vista haría parte de los Componentes de Interfaz de Usuario, mientras que el controlador haría parte de los Componentes de Proceso de Interfaz.

Las siguientes son las principales responsabilidades de los componentes de interfaz de usuario:

- Comunicación con el usuario (entradas y visualización de los resultados)
- Realizar el formateo de datos (estén en el formato correcto).
- Interpretar los eventos y acciones del usuario, y llamar la función correspondiente del controlador.
- Limitar entradas y validaciones de información.

- Almacenamiento de datos en caché. En ASP.NET, se puede especificar el almacenamiento en caché de la salida de un componente determinado de la interfaz de usuario para evitar el continuo procesamiento del mismo.
- Paginación de información. Es frecuente, especialmente en las aplicaciones Web, mostrar largas listas de datos como conjuntos paginados.
- Personalizar el aspecto de la aplicación en función de las preferencias del usuario o el tipo de dispositivo de cliente utilizado.

Componentes de proceso de interfaz

Los componentes de proceso de interfaz resultan especialmente útiles cuando la interacción del usuario sigue una serie de pasos predecibles para realizar una tarea determinada. Las interacciones más complejas conllevan el diseño de componentes de proceso de interfaz que permiten organizar los elementos de la interfaz y controlar la interacción con el usuario. Para facilitar la coordinación del proceso de usuario y controlar el mantenimiento del estado requerido al visualizar varias páginas o formularios de la interfaz de usuario, es necesario crear componentes de proceso de interfaz.

La siguiente figura muestra los componentes de proceso de interfaz que pueden ser compartidos por cada uno de las implementaciones de interfaz de usuario que tiene la aplicación.

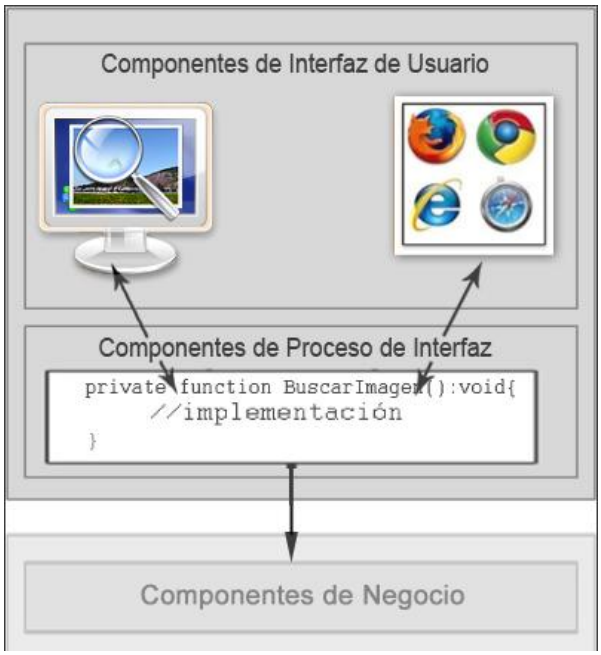

Figura 8. Componentes de Proceso de Interfaz de Usuario

Los componentes de proceso de interfaz se implementan normalmente como clases que exponen métodos a los cuales pueden llamar las interfaces de usuario. Cada método encapsula la lógica necesaria para realizar una acción específica en el proceso de usuario. La interfaz de usuario crea una instancia del componente del proceso de interfaz y la utiliza para efectuar la transición a través de los pasos del proceso.

Las siguientes son las principales responsabilidades de los componentes de proceso de interfaz:

- Separan el flujo de la interacción del usuario de la implementación o dispositivo en el que ocurre.
- Realizan el seguimiento del estado actual de la interacción del usuario.
- Encapsulan el modo en que las excepciones pueden afectar al flujo de proceso de usuario.
- Proporcionan un modo simple de combinar los elementos de la interfaz de usuario en los flujos de interacción del usuario sin que sea necesario volver a desarrollar el flujo de datos ni la lógica de control.

En resumen, la separación de la funcionalidad de interacción del usuario en componentes de interfaz y proceso de interfaz conlleva a mantener fácilmente el estado de la interacción de usuario de ejecución larga, lo que permite el abandono y la reanudación de la interacción, probablemente incluso utilizando una interfaz de usuario diferente. Igualmente varias interfaces de usuario pueden utilizar el mismo proceso.

2.5 Metas y Limitaciones de la Arquitectura.

A continuación se hace mención de los requerimientos funcionales y no funcionales arquitectónicamente significativos para el desarrollo de la Capa de presentación para las RIA que incluyan escenarios tridimensionales interactivos.

2.5.1 Requerimientos Funcionales

Los requisitos funcionales son las acciones u operaciones que la aplicación debe ser capaz de realizar. Basados en las funcionalidades básicas de las aplicaciones tratadas en esta investigación los seleccionados para la propuesta de solución son:

RF 1. El diseño de arquitectura propuesto debe permitirle a una aplicación que incluya escenarios tridimensionales interactivos visualizar interfaces gráficas.

RF 1.1 El diseño de arquitectura propuesto debe permitirle a una aplicación inicializar escenas 3D.

Rf 1.2 El diseño de arquitectura propuesto debe permitirle a una aplicación definir vistas de escenas 3D.

RF 1.3 El diseño de arquitectura propuesto debe permitirle a una aplicación definir cámaras para mostrar vistas de escenas 3D.

RF 2. El diseño de arquitectura propuesto debe permitirle a una aplicación que incluya escenarios tridimensionales interactivos administrar modelos 3D.

RF 2.1 El diseño de arquitectura propuesto debe permitir cargar un modelo 3D.

RF 2.2 El diseño de arquitectura propuesto debe permitir cargar y aplicar texturas a un modelo 3D.

RF 2.3 El diseño de arquitectura propuesto debe permitir definir los materiales a aplicar en un modelo 3D.

RF 2.4 El diseño de arquitectura propuesto debe permitir cargar un mapa de colisiones para un modelo 3D.

RF 3. El diseño de arquitectura propuesto debe permitirle a una aplicación que incluya escenarios tridimensionales interactivos administrar los eventos generados por un dispositivo de entrada.

2.5.2 Requerimientos no Funcionales.

En la mayoría de los casos los requerimientos no funcionales son esenciales en el éxito del producto. Son propiedades o cualidades que el producto debe tener. Estos se representan en varias categorías y para lograr que la Capa de presentación sea atractiva, usable, rápida y confiable hemos seleccionado los siguientes requisitos no funcionales.

Requerimientos de Apariencia o Interfaz Externa

Las interfaces de la aplicación deben ser diseñadas para que sean intuitivas, claras, familiares, con capacidad de respuesta, consistentes y sobre todo que tengan estética y sean eficientes, de forma general proporcionar una experiencia enriquecedora al usuario.

Restricciones en el Diseño y la Implementación

- MXML y ActionScript como lenguaje de programación.
- El patrón arquitectónico que se debe emplear en el desarrollo es el modelo-vista-controlador.

Requerimientos de Usabilidad

- La interfaz debe tener siempre visible la opción de Ayuda, lo que posibilitará un mejor aprovechamiento por parte de los usuarios de sus funcionalidades además, la utilización del sistema por personas que tengan conocimiento básico en el manejo de los términos de informática y de un ambiente Web en sentido general.

- La navegabilidad no debe ser muy compleja, todas las funcionalidades deben ser rápidas y fácilmente accesibles por los usuarios**.**

Requerimientos de Reusabilidad

- Los componentes deben ser utilizados por cualquier RIA con características similares en futuras aplicaciones.

Rendimiento

- El tiempo de visualización de una petición realizada por el usuario, debe relativamente rápido (menos de 5 segundos), aunque hay que tener en cuenta que este tipo de aplicaciones no manipula grandes cantidades de datos, pero si necesitan estar actualizándose constantemente, ya que en muchas ocasiones pueden ser utilizadas por varios clientes a la vez.

Requerimientos de Portabilidad

- Los componentes de la interfaz se utilizarán en cualquier sistema operativo ya que la arquitectura es multiplataforma.

Requerimientos de Software

- En las computadoras de los clientes se garantizará versiones de Windows, así como cualquier distribución de Linux.
- Se requiere tener instalado en los navegadores de la PC cliente, que pueden ser cualquier navegador de los conocidos (Internet Explorer, Mozilla Firefox, Safari, Netscape) el plug-in Flash Player 10.0 o versiones superiores.

Requerimientos de Hardware

- Se requiere para que la aplicación pueda ser mostrada que tenga como mínimo 512 de RAM aunque para un mejor funcionamiento de la misma se requiere como mínimo 1Gigabytes de RAM.

2.6 Modelo 4+1 Vista.

El modelo 4+1 vista fue desarrollado por Philippe Kruchten en 1995, para lograr un mayor entendimiento de la arquitectura del sistema a construir, tanto por las personas

involucradas en el desarrollo de la aplicación como por los desarrolladores, para ello es necesario agrupar cierto número de elementos arquitectónicos para satisfacer la funcionalidad y ejecución de los requisitos del sistema; así como los requisitos no funcionales del mismo: fiabilidad, escalabilidad, portabilidad, disponibilidad, etc. El modelo 4+1 vista describe la arquitectura del software usando cinco vistas concurrentes, la arquitectura evoluciona parcialmente a partir de estos escenarios que se muestran en la Figura 9 y más adelante se detallaran cada uno de ellos. (Kruchten, 1995)

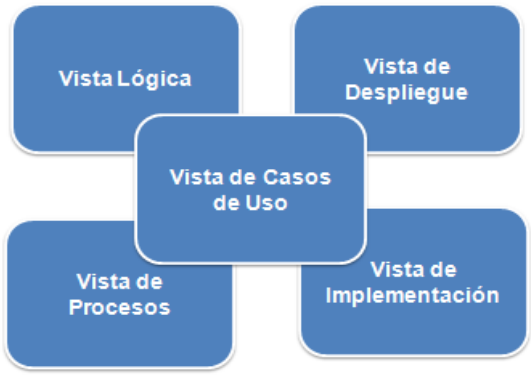

Figura 9. Modelo 4+1 Vista

2.6.1 Vista de Casos de Uso

La vista de casos de uso representa un subconjunto del artefacto Modelo de Casos de Uso. Contiene además los casos de usos significativos para la arquitectura, que son aquellos que describen funcionalidades imprescindibles para el sistema, y a través de los cuales se valida la arquitectura propuesta para el mismo, así como los diagramas de casos de uso.

En este caso en específico contamos con tres casos de usos que serían los críticos cada vez que se quiera desarrollar una RIA que incluya escenarios tridimensionales interactivos.

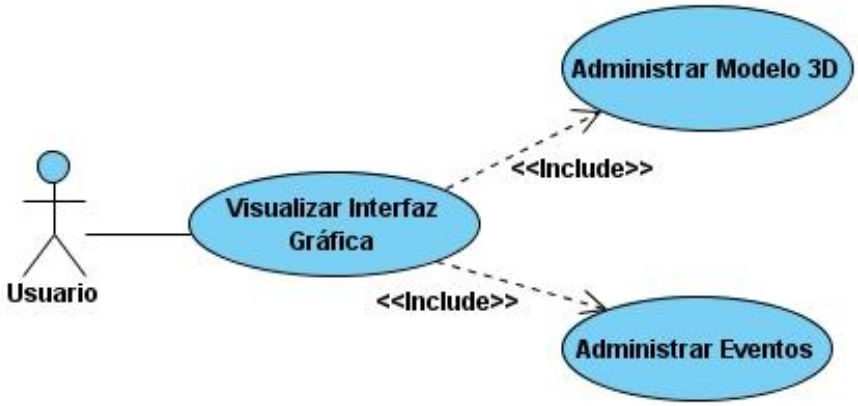

Figura 10. Diagrama de Casos de Uso.

A continuación se describirán los casos de usos significativos:

Caso de Uso:	Visualizar Interfaz Gráfica
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario solicita acceder a la aplicación o cuando realiza alguna acción que modifica la interfaz gráfica de la aplicación.
Precondiciones:	El usuario debe haber solicitado acceder a la aplicación.
Referencias	**RF 1, RF 1.1, RF 1.2, RF 1.3**
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El usuario solicita acceder a una aplicación enriquecida de internet que incluya escenarios tridimensionales interactivos.	1.1. El sistema inicializa una escena para mostrar en la aplicación. 1.2. El sistema define, dada la escena, la vista que será mostrada. 1.3. El sistema define la cámara con que se visualizará la vista.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**

2. El usuario solicita modificar la interfaz gráfica de la aplicación.		2.1 El sistema inicializa una escena para mostrar en la aplicación. 2.2 El sistema define, dada la escena, la vista que será mostrada. 2.3 El sistema define la cámara con que se visualizará la vista.
Relaciones	CU Incluidos	a) Administrar Modelos 3D b) Administrar Eventos
	CU Extendidos	
Poscondiciones	Se visualiza la interfaz gráfica de la aplicación Web.	

Tabla 2. Descripción textual del Caso de Uso Visualizar IU.

Caso de Uso:	Administrar Modelo 3D
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando se necesita cargar algún modelo 3D para ser mostrado en la interfaz gráfica.
Precondiciones:	El usuario debe haber solicitado mostrar una escena que contenga un modelo 3D.
Referencias	**RF 2, RF 2.1, RF 2.2, RF 2.3, RF 2.4**
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El usuario solicita visualizar una escena de la aplicación.	1.1. El sistema carga el modelo 3D. 1.2. El sistema carga las texturas a aplicar al modelo 3D. 1.3. El sistema define los materiales a aplicar al modelo 3D. 1.4. El sistema carga el mapa de colisiones para lograr una interacción en el modelo 3D.

Flujos Alternos		
Acción del Actor	**Respuesta del Sistema**	
Relaciones	CU Incluidos	
	CU Extendidos	
Poscondiciones	Se tiene un modelo 3D con las texturas, los materiales y las colisiones definidas.	

Tabla 3 .Descripción del Caso de Uso Administrar Modelo 3D.

Caso de Uso:	Administrar Eventos	
Actores:	Usuario	
Resumen:	El caso de uso se inicia cuando el usuario realiza algún evento a través del mouse o el teclado.	
Precondiciones:		
Referencias	**RF 3**	
Prioridad	Crítico	
Flujo Normal de Eventos		
Acción del Actor	**Respuesta del Sistema**	
1. El usuario al interactuar con la escena realiza un evento.	1.1 El sistema espera un evento realizado por el usuario.	
	1.2 Dependiendo del evento realizado el sistema realiza una acción.	
Flujos Alternos		
Acción del Actor	**Respuesta del Sistema**	
Relaciones	CU Incluidos	
	CU Extendidos	Gestionar Datos
Poscondiciones	El sistema después de captar el evento realiza una acción.	

Tabla 4. Descripción textual del Caso de Uso Administrar Eventos.

2.6.2 Vista lógica.

La vista lógica contiene las clases más importantes por las que se compone el sistema con su explicación detallada, la descomposición de las mismas en subsistemas y paquetes y las relaciones que se establecen entre ellas. Contiene los elementos del diseño más significativo para la arquitectura. Para un mayor entendimiento, se representa como quedaría lógicamente la Capa de presentación, basada en la estructura del MVC, teniendo en cuenta que es una representación lo más abstracta posible, ya que no es para una aplicación es específico, pero que si puede ser utilizada para cualquier RIA que incluya escenario tridimensional interactivo.

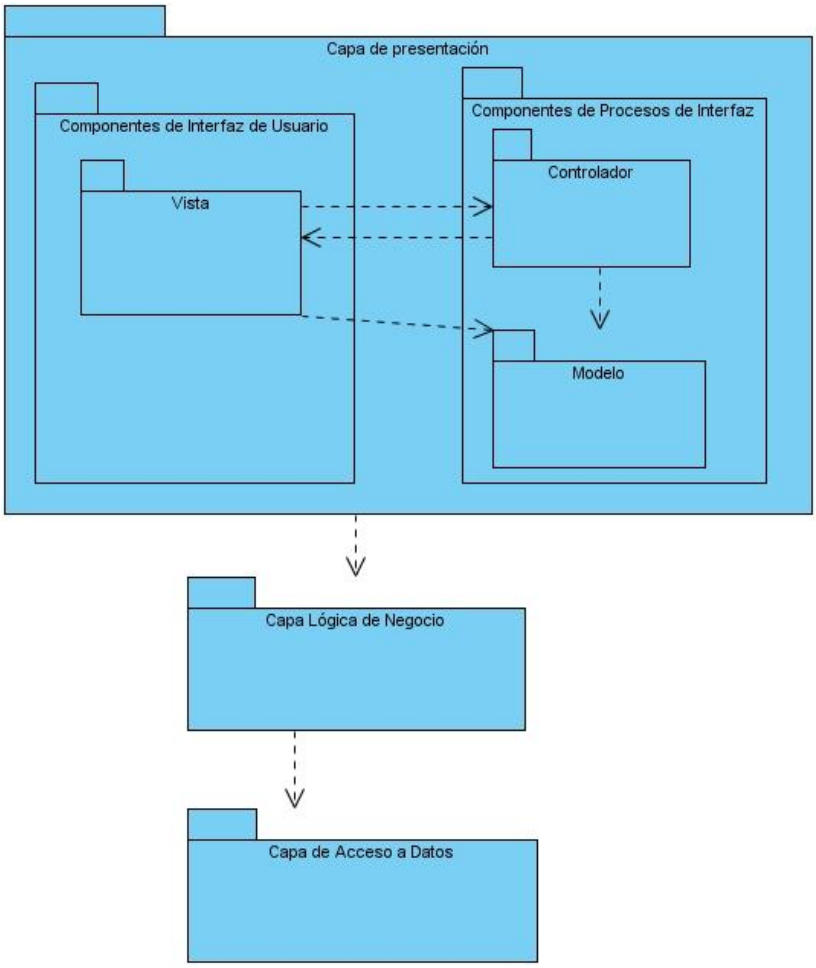

Figura 11. Vista Lógica de la Capa de presentación

En figura anterior se observa que la Capa de presentación está compuesta por tres paquetes fundamentales Vista, Modelo y Controlador y dentro de cada uno de ellos se representan las clases a utilizar en el desarrollo de la aplicación. En el paquete **Modelo** se encontrará las clases que se encargarán de contener los datos resultantes de la ejecución de la lógica de negocio de estas aplicaciones, en el paquete **Vista** estará las clases encargadas de mostrarle el modelo al usuario en respuesta de una petición y el paquete **Controlador** contendrá las clases responsables de procesar las entradas del usuario en forma de peticiones.

Se proponen tres clases que serán genéricas y fundamentales en todas las RIA que incluya escenarios 3D.

Interfaz: Esta clase se encontraría dentro del paquete Vista y será la clase encargada de la administración y gestión del contenido, además de inicializar el entorno de desplazamiento, es decir, la escena con la que interactuará el usuario.

Modelos: Se encontrará en el paquete Controlador, esta contendrá los métodos encargados de manipular los modelos, como las aplicaciones que se trabajaran incluirán escenarios tridimensionales, necesariamente hay que utilizar modelos 3D, de aquí la necesidad de tener organizado todas las acciones que se realizarán con estos modelos en esta clase ya sea cargarlos, modificarlos o utilizarlos desde cualquier dispositivo que disponga el usuario.

Gestionar Eventos: Esta clase se encontrará también en el paquete Controlador, en ella se registrarán e implementarán los métodos encargados de darle respuesta a las peticiones de los usuarios, es decir, manipular los eventos generados como resultados de la interacción entre el usuario y la aplicación.

2.6.3 Vista de Despliegue.

La vista de despliegue de un sistema contiene los nodos que forman la topología hardware sobre la que se ejecuta el sistema, la distribución, la entrega e instalación de las partes que constituyen el sistema físico. Se satisfacen además, parte de los requisitos no funcionales de hardware y software. Es un conjunto de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes software, objetos y procesos.

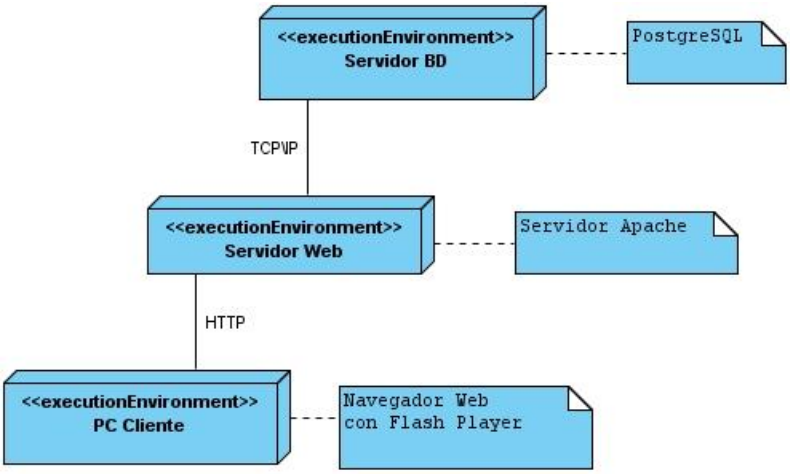

Figura 12. Vista de Despliegue

La Capa de presentación se encontrará en el nodo físico **PC Cliente**, que se requiere que tengan instalado el Flash Player para que soporte esta arquitectura y de esta manera permitir utilizar las aplicaciones de escenarios tridimensionales interactivos.

2.6.4 Vista de Implementación.

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

- Subsistemas de implementación.
- Diagramas de componentes.

A la hora de representar los componentes que estarán presentes en la Capa de presentación dentro de esta vista se hace lo más abstracto posible, ya que no es objetivo describir como estaría conformado los componentes de una aplicación en específico, sino se hace de manera tal que pueda utilizarse por cualquier aplicación del tipo para el cual está desarrollada la arquitectura.

A continuación veremos la representación de esta vista para la capa en cuestión.

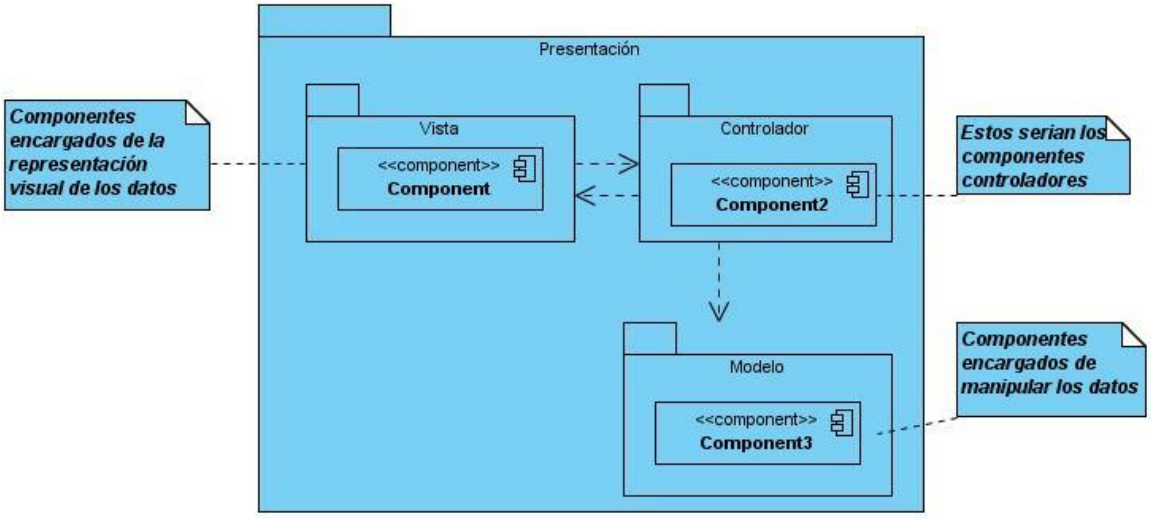

Figura 13. Vista De Implementación.

2.6.5 Vista de Procesos

La vista de procesos proporciona una base para comprender la organización de los procesos de un sistema. La misma solo suele usarse cuando el sistema presenta procesos concurrentes o hilos.

En este caso en particular no contamos con la representación de esta vista pues la solución propuesta no presenta procesos concurrentes.

2.7 Conclusiones Parciales

En este capítulo se han tratado específicamente todos los temas vinculados a la realización de la propuesta de arquitectura para la construcción de RIA que incluya escenarios tridimensionales interactivos. Se realizó una descripción del sistema a grandes rasgos, así como la especificación de las vistas de la arquitectura (Vista de CU, Lógica, Despliegue e Implementación). Se determinaron las herramientas, metodología, estilo arquitectónico y patrones de diseño que contribuirán al desarrollo de la Capa de presentación, así como la utilización del estilo MVC definido para establecer la línea base para la propuesta de la arquitectura. Se optó por el uso de la metodología AUP, Visual Paradigm como herramienta de modelado.

Capítulo 3 Evaluación de la Arquitectura.

3.1 Introducción

El presente capítulo se refiere a la evaluación de la arquitectura propuesta como solución a la problemática existente tratada durante el transcurso de la investigación. Se detallarán concisamente los métodos de evaluación de arquitectura, se describirán características y pasos de aplicación de los mismos, además los atributos de calidad en que se centran estos métodos.

3.2 ¿Cuándo evaluar una Arquitectura?

Un aspecto a tener en cuenta a la hora de realizar la evaluación de la arquitectura de software es el momento que se realiza. Para realizarla existen dos momentos, la evaluación temprana y la evaluación tardía. Se puede elegir cualquiera de las dos, la que más factible sea, de acuerdo con el proyecto que se esté desarrollando:

Evaluación Temprana: Para realizar este tipo de evaluación no es necesario que la arquitectura se encuentre completamente construida. Esta evaluación permite efectuar decisiones sobre la arquitectura en cualquier nivel, puesto que se pueden atribuir cambios arquitectónicos producto de una evaluación en función de los atributos de calidad esperados.

Evaluación Tardía: Para realizar este tipo de evaluación es necesario que la arquitectura del sistema se encuentre establecida y su implementación se ha completado, es decir, que el sistema ya esté terminado. Puede ser determinada en el momento que el equipo de desarrollo necesite tomar decisiones que dependen de la arquitectura propuesta y que el costo de no tenerlas en cuenta daría al trasto con un costo superior al de llevar a cabo una evaluación. Se considera que la evaluación en este punto es muy importante y útil, puesto que puede observarse el cumplimiento de los atributos de calidad asociados al sistema y su comportamiento general. (Gómez, 2007).

3.3 Necesidad de Evaluar la Arquitectura

El objetivo de evaluar una arquitectura es saber si puede habilitar los requerimientos especificados, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los *stakeholders*. (Brey, 2005).Realizar una evaluación de la arquitectura es la manera más económica de evitar desastres, ya que la arquitectura es un artefacto decisivo en la calidad de un software.

El propósito que se persigue al realizarle evaluaciones a la arquitectura, es el de identificar y mitigar los riesgos potenciales en su estructura y sus propiedades, asociados con el desarrollo del software, que pueden afectar su resultado, los cuales entre más temprano sean detectados menos costosos serán los mismos en términos económicos y a la hora de darle una solución.

Una arquitectura robusta debe cumplir con las funcionalidades del sistema y con los atributos de calidad esperados del mismo, además debe tener presente los requerimientos no funcionales que formarían parte de estos atributos de calidad, ya que especifican las características que se esperan del sistema, que sea robusto, fiable, que posea portabilidad, usabilidad, rendimiento, etc.

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

Para dar cumplimiento a estos objetivos se pueden aplicar varias técnicas y métodos para llegar a un mismo resultado.

3.4 Técnicas de validación

Las técnicas de evaluación se utilizan para la evaluación de atributos de calidad. Requieren grandes esfuerzos por parte de los ingenieros de software para crear especificaciones y predicciones respecto a si la propuesta arquitectónica satisface las cualidades del Software. Estas técnicas pueden ser clasificadas en cualitativas o cuantitativas Figura 13. Por lo regular, las técnicas de evaluación cualitativas son utilizadas cuando la arquitectura está en construcción, mientras que las técnicas de

evaluación cuantitativas, se usan cuando la arquitectura ya ha sido implantada (Gómez, 2007).

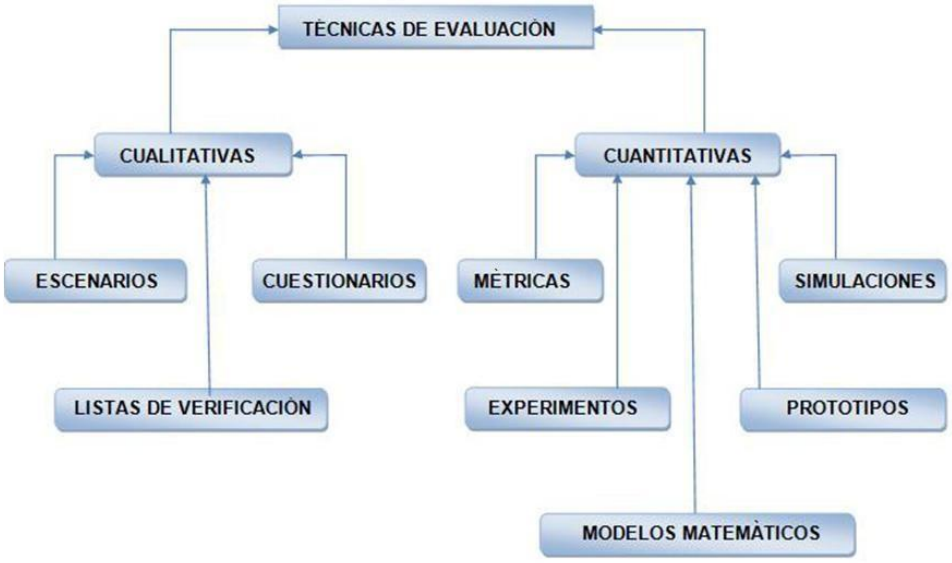

Figura 14. Clasificación de las técnicas de evaluación de arquitectura.

Estas técnicas requieren información del sistema a desarrollar que no está disponible durante el diseño arquitectónico, sino al principio del diseño detallado del sistema. El objetivo principal es tomar decisiones de tipo arquitectónico en las fases tempranas del desarrollo, es necesario utilizar aquellas que necesarias técnicas que exijan poca información detallada y conduzcan a resultados relativamente precisos. Las técnicas que posibilitan lo antes planteado son las cualitativas, de ahí que sean las técnicas de evaluación más empleadas por los arquitectos. (Camacho, 2004)

3.5 Métodos de Evaluación

Existen múltiples métodos para realizar pruebas a la arquitectura de software, cada uno con sus características específicas, las cuales son necesarias tener en cuenta para escoger el ideal, teniendo en cuenta las fortalezas y debilidades de los mismos. Entre los principales métodos se encuentran: SAAM (Software Architecture Analysis Method), ARD (Active Design Review), ATAM (Architecture Tradeoff Analysis Method) y ARID (Active Review Intermediate Designs).

Para la correcta aplicación de uno de estos métodos para evaluar la arquitectura, se verá a continuación una breve descripción de los que pudieran utilizarse en la arquitectura propuesta

3.5.1 SAAM

El método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) fue el primero ampliamente documentado entre los métodos de evaluación. En la práctica ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad. También evalúa la relación del diseño arquitectónico con los requerimientos del sistema.

Con la aplicación de este método en la evaluación de una sola arquitectura se obtienen los lugares en los que la misma puede fallar, partiendo 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. Consta de seis pasos básicos, donde se desarrollan los escenarios, se describe la arquitectura, los escenarios son priorizados y evaluados y finalmente se emite una evaluación de la interacción de los mismos.

3.5.2 ATAM

El método de Análisis de Acuerdos de Arquitectura de Software está inspirado en tres áreas distintas: Los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente.

La ATAM pretende ser un método de identificación de riesgo, un medio de detectar áreas de riesgo potencial dentro de la arquitectura de un sistema intensivo de software complejo, que puede aplicarse en fases tempranas del desarrollo del software.

La ATAM no presenta ninguna restricción con respecto a la característica de calidad a evaluar. El análisis de ATAM producirá resultados en consonancia con el nivel de detalle de la especificación de la arquitectura. Además, no es necesario producir

análisis detallados de cualquier atributo de calidad del sistema para tener éxito. (Kazman, 2000)

ATAM consta de nueve pasos, divididos en cuatro fases.

Fase 1: Presentación

En esta Fase se describe el método, las metas del negocio y la arquitectura.

1. Presentación del ATAM.
2. Presentación de las metas de negocio.
3. Presentación de la Arquitectura.

Fase 2: Investigación y análisis

Se especifican los atributos de calidad (desempeño, disponibilidad, seguridad, modificabilidad, usabilidad) en forma de escenarios así como se establece la prioridad entre ellos. Es en esta fase también donde se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance.

4. Identificación de los enfoques arquitectónicos.
5. Generación de Utility Tree.
6. Análisis de los enfoques arquitectónicos

Fase 3: Pruebas

Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios y se 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.

7. Lluvia de ideas y establecimiento de prioridad de escenarios.
8. Análisis de los enfoques arquitectónicos.

Fase 4: Reporte

Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

9. Presentación de los resultados

3.5.3 ARID

El método de Revisiones Activas para Diseños Intermedios (Active Reviews for Intermediate Designs, ARID) es un híbrido del método ARD y ATAM. ARID constituye un método conveniente para la evaluación de diseños parciales en las etapas tempranas del desarrollo usando técnicas de evaluación basada en escenario.

El método ARID evalúa el grado en que los atributos de calidad satisfacen en cada uno de los escenarios definidos. Como resultado de la aplicación de dicho procedimiento se obtiene un diseño de alta fidelidad acompañado de una alta familiarización con el diseño de los *stakeholders*. (Kazman, 2000)

Proveen características útiles para el problema de la evaluación de diseños preliminares, dado que ninguno por sí solo es conveniente. El método ARID Define los roles de Arquitecto, Equipo de verificación y *Stakeholders* y comprende nueve pasos agrupados en dos fases.

Fase 1: Actividades Previas comprende los pasos del (1 al 4)	**Fase 2: Revisión** comprende los pasos del (4 al 9)
Identificación de los encargados de la revisión.	Presentación del ARID.
Preparar el informe de diseño.	Presentación del diseño.
Preparar los escenarios base.	Lluvia de ideas y establecimiento de prioridad de escenarios.
Preparar los materiales.	Aplicación de los escenarios.

	Resumen

Tabla 5. Pasos del Método ARID.

3.5.4 ADR.

El método de Revisión de Diseño Activo o ARD (Active Design Review) se emplea en la evaluación de diseños bien detallados de unidades de software como los componentes. Se centra en la calidad y el nivel de completamiento de la documentación, y en el nivel de conveniencia y suficiencia de los servicios que contiene el diseño propuesto.

3.6 Atributos de Calidad.

Los atributos de calidad forman parte de los requerimientos no funcionales de una aplicación, y como tal deben ser específicos para satisfacer alguna necesidad dada o llegar a una meta específica. Se pueden tomar estos atributos de calidad en base a un punto de vista definido, por ejemplo a la adecuación de los requerimientos funcionales o la satisfacción del cliente. (Leticia Dávila Nicanor).Los atributos de calidad se clasifican en dos tipos: observables en vías de ejecución y no observables en vías de ejecución.

Observables mediante la ejecución: aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución.

No observables en la ejecución: aquellos atributos que se establecen durante el desarrollo del sistema.

Entre los atributos de calidad más relevantes por los cuales puede ser evaluada una arquitectura están los siguientes:

Observables vía ejecución	
Atributo	**Descripción**
Disponibilidad	Es la medida de disponibilidad del sistema para su uso.

Funcionalidad	Es la capacidad del software para proporcionar funciones que satisfacen las necesidades declaradas e implícitas cuándo el software se usa con las condiciones especificadas. (ISO, 2001)
Confidencialidad	Es la ausencia de acceso no autorizado a la información. Desempeño: El grado en que se satisfacen las necesidades, representado por un conjunto específico de valores para las características de calidad, como velocidad, exactitud o uso de memoria. (ISO, 2001)
Confiabilidad	La capacidad del producto de software para mantener un nivel de ejecución especificado.
Seguridad Externa	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información.
Seguridad interna	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación de servicio, mientras se sirve a usuarios legítimos
No observables vía ejecución	
Configurabilidad	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema. (Booch, 1999)
Modificabilidad	Es la habilidad de realizar cambios futuros al sistema.
Integrabilidad	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados
Integridad	Es la ausencia de alteraciones inapropiadas de la información.

Interoperabilidad	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de Integrabilidad.
Mantenibilidad	Es la capacidad de someter a un sistema a reparaciones y evolución. Capacidad de modificar el sistema de manera rápida y a bajo costo.
Portabilidad	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos. (Kazman, 2000)
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.
Escalabilidad	Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental. (Pressman, 2001)
Capacidad de prueba	Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba.

Tabla 6. Atributos de Calidad

3.7 Evaluación de la arquitectura de software propuesta

Dado el estado actual de la arquitectura y con vista a evaluar el diseño arquitectónico se decide utilizar el método ARID, ya que es conveniente cuando se posee un diseño parcialmente completo que se desea evaluar. Es un método que permite en las fases tempranas del diseño realizar una evaluación cualitativa basada en escenarios que permite analizar las debilidades en cuanto a riesgos de la arquitectura. Para ello se definen los escenarios a través de los cuales se evaluará la arquitectura propuesta con vista a evaluar los atributos de calidad considerados de mayor importancia según el Modelo ISO/IEC 9126-1:2001, teniendo en cuenta que en la presente investigación

los encargados de llevar a cabo la evaluación de la arquitectura propuesta serán los integrantes del grupo de arquitectura que definieron la misma, no se siguieron estrictamente los pasos propuestos por el ARID. Los arquitectos del sistema tienen un dominio completo del diseño arquitectónico del mismo, y presentan conocimientos del método a aplicar para la evaluación, por la lo que no se realizó Fase1, pasando directamente a aplicarse los tres últimos pasos de la Fase 2.

3.8 Definición de los principales escenarios.

Los principales escenarios serán aquellos que tengan según el arquitecto una prioridad alta y además poseen dificultad alta de implementación. A continuación se describen los atributos de calidad seleccionados para la evaluación de la Capa de presentación propuesta, además del perfil y escenario donde estos se enmarcan.

Atributo de calidad	**Perfil**	**Escenario**
Portabilidad	Adaptabilidad	Migración de sistemas operativos.
Relación atributo-escenario		
Portabilidad: La portabilidad de un sistema es un atributo que posee relevante importancia, ya que es la habilidad de un sistema para ser ejecutado en diferentes ambientes de computación. La Capa de presentación propuesta será desarrollada utilizando como lenguaje MXML y ActionScript, como servidor web el Apache. Cada elemento seleccionado para el desarrollo de la capa es multiplataforma, además que solamente requerirá de un navegador web utilizado por el cliente.		

Tabla 7. Evaluación del atributo Portabilidad.

Atributo de calidad	**Perfil**	**Escenario**
Mantenibilidad	Modificabilidad	Realizar una modificación en la interfaz.
Relación atributo-escenario		

Modificabilidad: Analizando que la modificabilidad es la habilidad de realizar cambios futuros al sistema, la capa debe ser modificable, permitir agregar nuevas funcionalidades, reutilizar los métodos definidos en las clases controladoras, teniendo en cuenta que la capa que se propone está estructurada por componentes genéricos que se adaptan a las necesidades específicas de cualquier aplicación desarrollada en el dominio especificado. Por esta parte el uso del patrón arquitectónico MVC, utilizado para su desarrollo, provee una clara separación entre los componentes, lo cual permite implementarlo por separado y así si se desea hacer algún cambio futuro es una ventaja que no influya en lo que ya está implementado. Además tiene como ventaja la flexibilidad con que se puede cambiar las vistas y los controladores, otro de los aspectos que muestra la modificabilidad del sistema.

Tabla 8. Evaluación del atributo Mantenibilidad.

Atributo de calidad	**Perfil**	**Escenario**
Reusabilidad	Reusabilidad	Reutilizar componentes.
Relación atributo-escenario		
Partiendo de lo que plantea la reusabilidad, que es la capacidad de diseñar un sistema de forma tal que su estructura o partes de sus componentes puedan ser reutilizados en futuras aplicaciones, la capa en cuestión al formar parte del estilo arquitectónico en tres capas se encuentra separada de la de lógica del negocio y del de acceso a datos brindando que los componentes sean lo más desacoplados posible ,siendo el factor fundamental la reusabilidad, ya que se puede reutilizar algunos componentes en otras aplicaciones del mismo tipo.		

Tabla 9. Evaluación del atributo Reusabilidad.

Atributo de calidad	Perfil	Escenario
Funcionalidad	Adecuación	Visualizar Modelo 3D.
Relación atributo-escenario		
Teniendo en cuenta que la Funcionalidad es la habilidad del sistema para realizar el trabajo para el cual fue concebido, la estructura de la capa propuesta permite que se pueda cargar modelos 3D, para esto se cuenta con la clase modelos que se encargará de realizar esta acción. Además se cumple con la adecuación, ya que se tuvieron en cuenta los requisitos funcionales y no funcionales, adecuándose al desarrollo de las RIA que incluyan escenarios 3D interactivos.		

Tabla 10. Evaluación del atributo Funcionalidad.

La evaluación cualitativa realizada refleja que la propuesta de Capa de presentación cumple con los atributos de calidad requeridos como son la portabilidad, mantenibilidad, reusabilidad y funcionalidad. Por lo antes planteado se puede afirmar que la arquitectura propuesta es adecuada para el desarrollo de la Capa de presentación de una RIA que incluya escenarios tridimensionales interactivos.

3.9 Conclusiones del capítulo.

En el presente capítulo se realizó la validación de la Capa de presentación de una arquitectura propuesta para el desarrollo de RIA., se destacaron las ventajas de realizar una evaluación ya que permite identificar el impacto que tiene la misma sobre los atributos de calidad, los cuales se analizaron y se valoró la aplicación de ellos a la solución propuesta. Para esto se ha hecho un análisis de las técnicas y métodos de evaluación, para escoger el que más resultados pueda ofrecer en la arquitectura. Quedando demostrado que estos se encuentran totalmente validados y que la arquitectura está apta para ser empleada en la aplicación que se quiera desarrollar.

Conclusiones Generales

- Al concluir esta investigación, se logró dar cumplimiento al objetivo trazado, logrando la propuesta de la Capa de presentación de una arquitectura para el desarrollo de RIA que incluya escenarios tridimensionales interactivos.

- Se definió la línea base de la arquitectura que tiene la importancia de definir las funcionalidades que va a tener el sistema que se quiera desarrollar, de esta forma se determinaron las herramientas y lenguajes de programación a utilizar para el desarrollo de la Capa de presentación.

- Se seleccionaron los métodos y las técnicas necesarias para la evaluación de la arquitectura, permitiendo afirmar que la capa cumple con los requerimientos no funcionales que deben poseer los sistemas que se desarrollen dentro del dominio especificado tratado en la investigación.

- Se obtuvieron componentes genéricos y reutilizables a partir de la propuesta de la Capa de presentación, brindando una solución estandarizada capaz de guiar los procesos de desarrollo de las RIA que incluyan escenarios tridimensionales interactivos.

Recomendaciones

- Integrar la Capa de presentación con las restantes capas para conformar la Arquitectura.
- Aplicar los resultados de esta investigación en el desarrollo de alguna aplicación enriquecida de internet que incluya escenario tridimensional interactivo en la línea de productos de software Visualización Web de la Universidad de las Ciencias Informáticas (UCI).

Referencias bibliográficas

[1].**Adobe Corporation.** [Online] http://www.adobe.com.

[2].**Area3d. 2011.** Area3d. [En línea] 2011. http://www.area3d.es/.

[3].**Brown, A. W. and C.Wallnau. 1998.** IEEE Software **:** s.n., 1998. 37-46

[4].**Camacho, E., F. Cordeso, and G. Núñez** (2004) Arquitectura de Software. Volumen, 1-54.

[5].**Carlos Reynoso, Nicolás Kiccillof. 2004.***Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* BUENOS AIRES : http://www.willydev.net/descargas/prev/Estiloypatron.pdf.

[6].**Ciberaula.**[Online] http://linux.ciberaula.com/articulo/linux_apache_intro/.

[7].**Dr. Jon Kepa Gerrikagoitia, Aitor Ariño. 2009.** Euskadi+innova. *Aplicaciones De Internet Enrriquecidas.* [En línea] 17 de Marzo de 2009.url: http://www.euskadinnova.net/es/enpresa-digitala/agenda/aplicaciones-internet-enriquecidas-nueva-experiencia-usuario/2676.aspx.

[8].**Elia Vite Villegas ITL, Armando Jaraleño Paloalto ITSUR. 2010.***DESARROLLO DE APLICACIONES WEB UTILIZANDO LAS RICH INTERNET APPLICATIONS-(RIAs).2010.* 2010.url: http://computomovil.files.wordpress.com/2009/10/elia-vite-y-j-armando-jaraleno.pdf

[9].**García, Juaquín. 2005.** [En línea] 27 de mayo de 2005. [Citado el: 18 de febrero de 2011.] http://www.ingenierosoftware.com.

[10].**Gómez, Omar Salvador. 2007.** E*valuando Arquitecturas de Software. Parte 1*. Panorama General. México: México: Brainworx S.A, 2007

[11].**Grady Booch, James Rumbaugh e Ivar Jacobson.** El Lenguaje Unificado de Modelado. Madrid: Addison-Wesley, 1999.

[12].**Graells, Dr. Perez Marquès. 2008.** LAS TIC Y SUS APORTACIONES A LA SOCIEDAD. [En línea] 2008. [Citado el: 25 de octubre de 2010.]http://peremarques.pangea.org.

[13].**Gustavo Andrés Brey, Gastón Escobar, Nicolas Passerini y Juan Arias. 2005.***Arquitectura de Proyectos de IT. Evaluación de Arquitecturas.* Universidad Tecnológica Nacional. Facultad Regional de Buenos Aires - Departamento de Sistemas : s.n., 2005.

[14].**IEEE. 2000.** Recommended Practice for Architecture Description of Software-Intensive Systems. [En línea] 2000. ANSI/IEEE Std 1471-2000, 2000.6.aspx.

[15].**ISO. (2001).** *INGENIERÍA DE SOFTWARE—CALIDAD DEL PRODUCTO—PARTE 1: MODELO DE LA CALIDAD.* La Habana.URL: http://calisoft.uci.cu/tmp/documentos/normas/iso/NC-ISO-IEC%209126-1.pdf

[16].**José H. Canós, P. L.** Metodologías Ágiles en el Desarrollo de Software. Valencia. URL: http://www.willydev.net/descargas/prev/TodoAgil.pdf

[17].**KAISER, S. H. 2005..***Software Paradigms.* 2005.

[18].**Kiccillof, Carlos Reynoso – Nicolás.** [En línea]

[19].**Kruchten, Philippe. Noviembre 1995.** *Architectural Blueprints—The "4+1" View Model of Software Architecture.* Noviembre 1995. pp. 42-50.995.URL: http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

[20].**Larman, Craig.***Uml y patrones: Introducción al análisis y programación orientada a objetos.*

[21].**Leticia Dávila Nicanor, P. M.** *Evaluación de la Calidad de Software en Sistemas de Información en Internet.* Zacatenco, Mexico: CINVESTAV-IPN. Sección de Computación.URl: http://delta.cs.cinvestav.mx/~pmejia/davila-mejia.pdf.

[22].**López, Xavier Farré. 2005.** Rich Internet Applications. [Online] julio 30, 2005. Url: http://upcommons.upc.edu/pfc/bitstream/2099.1/3720/4/40624-4.pdf.

[23].**Machorro Reyes, Ricardo Armando. 2010.** *Los Patrones como un Medio del Diseño Orientado a Objetos***. [Online] abril 2011.URL:** *http://www.revistaupiicsa.20m.com/Emilia/RevMayAgo04/Machorro1.pdf..3*

[24].**Madeinflex.**[Online] http://www.madeinflex.com/.

[25].**Nicolás Kiccillof, Carlos Reynoso. 2004.***Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires : s.n., 2004.

[26].**Paloma Cáceres, Esperanza Marcos Grupo Kybele. 2010**. Procesos giles para el Desarrollo de Aplicaciones Web. Móstoles, Madrid (España) : s.n., 2010. URL: http://www.dlsi.ua.es/webe01/articulos/s112.pdf

[27].**Paul Clements, Rick Kazman y Mark Klein. 2004.***Evaluating Software Architectures:Methods and Case Studies.* 2004. ISBN 0-201-70482-X.

[28].**Pressman, Roger S. 2001.***"Ingeniería de Software. Un enfoque práctico".5ta Edición.* Madrid : s.n., 2001.

[29].**Proyectosagiles.2010.**[Online] http://www.proyectosagiles.org/beneficios-de-scrum.

[30].**Shaw, Mary y Garlan, David.** *An introduction to software architecture.* 1994. CMU/SEI-94-TR-21, ESC-TR-94-21.

[31].**Tracz, Will. 1995.***DSSA Domain Analysis Tool.* 1995.

[32].**Universidad ORT. 2008.** *Introducción a Tecnologías Enriquecidas para Internet.* Uruguay : Freddy Veit – 129756, 2008.

[33].**Visual Paradigm.** [En línea] 2011.URL:http://www.visual-paradigm.com/.

[34].**WebSchema. 2010.** WebSchema. *Esquemas a la medida de tu proceso o negocio.* [En línea] 2010. [Citado el: 18 de febrero de 2010.]http://www.webschema.com.mx/ws1/.

[35].**Wendy Boggs, Michael Boggs. 2002.***UML whith Rational Rose 2002.*

URL:http://bibliodoc.uci.cu/pdf/0782140173.pdf.

Bibliografía

[1]. **Casperson, Matthew. 2011.** *Away3D 3.6 Essentials.* s.l. : Packt Publishing, 2011.

[2]. **Daniel M. Maldonado. 2010**. El CoDiGo K . Características de un excelente Entorno de Desarrollo Integrado. [Online] septiembre 2010. http://www.elcodigok.com.ar.

[3]. **Dávila, M., Germán, M., Crutas, D., & García, A**. Evaluación de Arquitecturas de Software.URL:http://www.fing.edu.uy/inco/cursos/gestsoft/Presentaciones/Evaluacion%20de%20Arquitecturas%20-%20G10/Evaluacion%20de%20Arquitecturas.doc

[4]. **Eclipse**. [Online]

[5]. **H. Olmedo-Rodríguez, D. Escudero-Mancebo, V. Cardeñoso-Payo. 2009.** *3D en las Rich Internet Applications: comparativa de opciones tecnológicas.* 2009. 47011.

[6]. **Junio18, 2008 GERONET** "Informática e Ingeniería. " disponible en: http://www.geronet.com.ar/?p=83

[7]. **LTC Erik Mettala and Marc H. Graham, e. (1992).**The Domain-Specific SoftwareArchitectureProgram.URL:http://www.sei.cmu.edu/library/abstracts/reports/92sr009.cfm

[8]. **Marcos Guglielmetti (2005, febrero 10**). Mastermagazine , Definición de Arquitectura Software URL:http://www.mastermagazine.info/termino/3916.php

[9]. **María Rosa Mas Camacho, J. P. (n.d.).** Experiencias de la Aplicación de la Ingeniería de Software en Sistema de Gestión. Revista Cubana de Informática Médica, 3-4.: URL: http://www.rcim.sld.cu/revista_1/articulo_1.htm.

[10].**Martínez, A. M. (n.d.)**. Guía a Rational Unified Process. Retrieved 2010, URL:http://www.dsi.uclm.es/asignaturas/42551/trabajosAnteriores/Trabajo-Guia%20RUP.pdf

[11].**Normalización, ININ/ Oficina Nacional de. 2001**. INGENIERÍA DE SOFTWARE— CALIDAD DEL PRODUCTO— PARTE 1: MODELO DE LA CALIDAD. S.l.: ISO, 2001. NC ISO/IEC 9126.URL: http://calisoft.uci.cu

[12].**Rodolfo Quispe-Otazu.** ¿Qué es la Ingenieria de Software? Blog de Rodolfo Quispe-Otazu [Internet]. Mayo 2007. Disponible en: http://www.rodolfoquispe.org/blog/que-es-la-ingenieria-de-software.php

[13].**Romero, D.** CAPAS DE SESIÓN, Presentación y Aplicación. Disponible en: http://www.elrinconcito.com/articulos/Sesiones/sesiones.pdf

[14].**Suárez, Agueda González. 2006**. Monografias.com. [En línea] marzo de 2006. [Citado el: 25 de octubre de 2010.] http://www.monografias.com.

[15].**Olsson, Rob Bateman & Richard. 2010.** *The Essential Guide to 3D.* United States of America : s.n., 2010.

Glosario de Términos

Capa de lógica del negocio: Es en esta capa donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso**.**

Capa de acceso a datos: En esta capa es donde residen los datos y es la encargada de acceder a ellos. Está formada por uno o más gestores de bases de datos que realizan todo su almacenamiento, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

CASE: (Computer Aided Software Engineering): se incluyen una serie de herramientas, lenguajes y técnicas de programación que permiten la generación de aplicaciones de manera semiautomática.

Escenario tridimensional: es el espacio destinado para la representación de un dominio de aplicación en las tres dimensiones ancho largo y profundidad.

Framework: es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado.

GoF: Gang of Four, "pandilla de los cuatro".

HTTP (Protocolo de Transferencia de Hipertexto): es un estándar para el intercambio de archivos (texto, gráficos y multimedia) a través de Internet. Es un protocolo sin estado de tipo cliente-servidor.

Navegador web: es un programa que permite visualizar la información que contiene una página web (ya esté está alojada en un servidor dentro de la WWW o en uno local).

PC: (en inglés Personal Computer) Computadora Personal.

Plug-in : aplicación que interactúa con otra para agregarle una funcionalidad específica y es ejecutada por la aplicación principal.

RIA (Aplicaciones Ricas en Internet): Una RIA (*Rich Internet Application)* es un nuevo tipo de aplicación Web cuyo objetivo es el de incrementar y mejorar las opciones y

capacidades de las aplicaciones Web tradicionales. Este nuevo tipo de aplicaciones son desarrolladas, en la mayoría de los casos, utilizando lenguajes de marcado propios y son ejecutadas utilizando unos servidores de presentación también propios.

Runtime: el tiempo de ejecución de Flex.

Skins: Símbolos (“revestimientos”) que definen el aspecto del componente. La Nueva Arquitectura de personalización que ha aportado Flex 4, que proporciona una separación clara y precisa entre la lógica y los elementos visuales de un componente.

***Stakeholders*:** Son aquellas personas o entidades que están interesadas en la realización de un proyecto o tarea.

XML: es el acrónimo del inglés eXtensible Markup Language (lenguaje de marcado ampliable o extensible) desarrollado por el World Wide Web Consortium (W3C).