UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 5

Capa de Acceso a Datos 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.

Autor: Andersson Pérez Fernández.

Tutor: Ing**.** Gilberto Cao Tarrero.

Ciudad de la Habana, junio del 2011

Año 53 de la Revolución

“Somos arquitectos de nuestro propio destino.”

Albert Einstein

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 de _____ del año _____.

_____ _____

Firma del Autor: Andersson Pérez Fernández **Firma del Tutor:** Gilberto Cao Tarrero

DATOS DEL CONTACTO

Nombre y apellidos: Ing. Gilberto Cao Tarrero

Edad: 25

Ciudadanía: Cubano

Categoría docente:

Centro de trabajo: Universidad de las Ciencias Informáticas

Título de la especialidad de graduado: Ingeniero en Ciencias Informáticas

Año de graduación: 2009

Correo electrónico: gcao@uci.cu

Agradecimientos

A mi abuelo Tomás por sus sabios consejos y su apoyo en todos los momentos de mi vida. Por ser la persona que más admiro en la vida. Quisiera que estuvieras aquí y pudieras compartir con el “porfiado” de tu nieto este momento para el cual me educaste y guiaste desde pequeño. Gracias por haber estado siempre ahí para mis malcriadeces, todo lo que soy ahora te lo debo a ti.

A mi abuela Cuquita por aguantar todas mis malcriadeces y por quererme tanto.

A mi mamá Lariza por ser la madre más preocupada del mundo.

A mi papá Pupo por su apoyo y sus consejos cuando los he necesitado.

A mi tío Jorge por sus regaños cuando hacía algo mal, espero que no me tengas que regañar más en lo adelante.

A mi abuela Dolores por preocuparse por mí.

A mi tío abuelo Gerardo por sus conversaciones todas las tardes cuando me obstinaba de estudiar o no tenía nada que hacer en la casa.

A mi hermana Lizandra por mortificarme, tata aunque te regañe mucho lo hago porque quiero lo mejor para ti.

A mi tía Marilyn por brindarme su ayuda cuando lo necesito.

A mi padrastro Jesús por tratarme como un hijo.

A toda mi familia.

Agradecerles a mis amigos Siles, Julio, Adriel y Tejeda por las ayudas y los buenos y malos momentos a los que hemos tenido que enfrentarnos en estos 5 años. A mis compañeros de apartamento y de aula. A Lia por todas las ayudas en estos últimos 3 años. A mis compañeras de tesis Liusba, Libeidy y Maybe por los momentos difíciles que hemos pasado en este último año. A mis amigos de Artemisa, en especial a Danae por estar siempre ahí cuando te necesito. A todos los amigos y amigas que he conocido en estos 5 años.

Dedicatoria

A toda mi familia por apoyarme y por los sacrificios que hicieron para que pudiera llegar hasta aquí.

En especial a mi abuelo Tomas que ya no está con nosotros, a mi abuela Cuca y a mi mamá Lariza que son los que han estado más cerca día a día junto mi sufriendo cuando salía mal en alguna prueba o tenía algún problema.

A mi tío Jorge que, aunque lleva 4 años cumpliendo misión internacionalista y no ha podido estar junto a nosotros, me ha apoyado y se ha preocupado por mí como si fuera su hijo.

A mi papá Pupo por la confianza que siempre ha depositado en mí, por su ayuda y sus consejos cuando los he necesitado.

Resumen

El vertiginoso avance que han tenido las aplicaciones Web en los últimos años ha propiciado el surgimiento de nuevas tecnologías para su desarrollo, logrando así que los usuarios de Internet puedan tener nuevas experiencias al visitar estas aplicaciones Web. En el presente trabajo de diploma se describe una propuesta de arquitectura de la capa de Acceso a Datos para desarrollar Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos. Para alcanzar este objetivo se realiza un estudio del arte, en el cual se analizan los principales conceptos que se deben conocer referente a esta área de conocimiento, los diferentes patrones y estilos arquitectónicos que pueden ser usados en el desarrollo de la capa, conjuntamente se investiga sobre las principales herramientas tecnológicas y metodologías de desarrollo que existen en la actualidad para realizar este tipo de software. La arquitectura propuesta es descrita por una serie de vistas arquitectónicas que ayudan a los implicados en la construcción del sistema a comprender su estructura, además se valida la propuesta utilizando métodos y técnicas que posibilitan determinar el cumplimiento de los atributos de calidad establecidos a partir de escenarios definidos.

Palabras Clave: Arquitectura de Software, Arquitectura de Software de Dominio Específico, Aplicaciones Enriquecidas de Internet.

Índice

Agradecimientos II
Dedicatoria III
Resumen IV
Índice V
Introducción IX
1 Capítulo 1: Fundamentación Teórica ..... - 1 -
1.1 Aplicaciones Enriquecidas de Internet ..... - 1 -
1.2 Arquitectura de Software ..... - 3 -
1.2.1 Arquitectura de Software de Dominio Específico ..... - 4 -
1.3 Estilos arquitectónicos ..... - 5 -
1.3.1 Arquitecturas en Capas ..... - 5 -
1.4 Patrones de diseño ..... - 7 -
1.4.1 Modelo-Vista-Controlador ..... - 7 -
1.4.2 Patrones GRASP ..... - 8 -
1.4.3 Patrones GoF ..... - 9 -
1.5 Entorno Integrado de Desarrollo ..... - 9 -
1.5.1 Eclipse ..... - 10 -
1.5.2 Adobe Flash Builder 4 ..... - 10 -
1.5.3 Zend Studio ..... - 10 -
1.6 Lenguajes de Programación ..... - 11 -
1.6.1 PHP ..... - 11 -
1.6.2 Action Script 3.0 ..... - 12 -
1.6.3 XML ..... - 12 -
1.7 Framework de comunicación ..... - 13 -
1.7.1 AMFPHP ..... - 13 -
1.7.2 WebOrb ..... - 13 -
1.7.3 ZendAMF ..... - 14 -
1.8 Servidores Web ..... - 14 -

1.8.1 Apache ..... - 14 -
1.8.2 Internet Information Services (IIS) ..... - 15 -
1.9 Sistemas de Gestión de Base de Datos ..... - 15 -
1.9.1 MySQL ..... - 15 -
1.9.2 PostgresSQL ..... - 16 -
1.10 Metodología de Desarrollo de Software ..... - 16 -
1.10.1 Rational Unified Process (RUP) ..... - 17 -
1.10.2 Agile Unified Process (AUP) ..... - 19 -
1.10.3 Programación Extrema (XP) ..... - 21 -
1.11 Herramientas CASE ..... - 22 -
1.11.1 Rational Rose Enterprise Edition ..... - 22 -
1.11.2 Visual Paradigm ..... - 22 -
2 Capítulo 2: Propuesta de Solución ..... - 24 -
2.1 Línea base de la arquitectura ..... - 24 -
2.1.1 Propósito ..... - 24 -
2.2 Selección de las tecnologías a utilizar ..... - 25 -
2.2.1 Patrones de diseño ..... - 25 -
2.2.2 Entorno de desarrollo ..... - 25 -
2.2.3 Lenguaje de desarrollo. ..... - 25 -
2.2.4 Framework de comunicación ..... - 26 -
2.2.5 Servidor Web ..... - 26 -
2.2.6 Metodología de desarrollo ..... - 26 -
2.2.7 Herramienta de modelado ..... - 26 -
2.2.8 Gestor de Base de Datos ..... - 27 -
2.3 Requerimientos del sistema ..... - 27 -
2.3.1 Requerimientos Funcionales ..... - 27 -
2.3.2 Requisitos No Funcionales ..... - 28 -
2.4 Descripción de la Arquitectura de la Capa de Acceso a Datos ..... - 29 -
2.4.1 Vista de Casos de Uso ..... - 30 -

2.4.2 Vista lógica ..... - 33 -
2.4.3 Vista de implementación ..... - 35 -
2.4.4 Vista de despliegue ..... - 38 -
2.4.5 Vista de Procesos ..... - 39 -
3 Capítulo 3: Validación de la propuesta de solución. ..... - 40 -
3.1 Evaluación de la Arquitectura de Software ..... - 40 -
3.2 Atributos de calidad ..... - 41 -
3.3 Técnicas de evaluación de la Arquitectura de Software ..... - 41 -
3.3.1 Evaluación Basada en Escenarios ..... - 42 -
3.3.2 Evaluación basada en simulación ..... - 43 -
3.3.3 Evaluación basada en modelos matemáticos ..... - 44 -
3.4 Métodos de evaluación de Arquitecturas de Software ..... - 44 -
3.4.1 Architecture Trade-off Analysis Method (ATAM) ..... - 44 -
3.4.2 Software Architecture Analysis Method (SAAM) ..... - 45 -
3.4.3 Active Intermediate Designs Review (ARID) ..... - 46 -
3.5 Evaluación de la propuesta de solución. ..... - 47 -
Conclusiones ..... - 52 -
Recomendaciones ..... - 53 -
Bibliografía ..... - 54 -
Anexos ..... - 57 -
Glosario de Términos y Siglas ..... - 61 -

Índice de Figuras

Figura 1: Usuarios de Internet IX
Figura 2: Arquitectura en 3 Capas ..... - 6 -
Figura 3: Modelo-Vista-Controlador ..... - 7 -
Figura 4: Fases e iteraciones de RUP ..... - 17 -
Figura 5: Modelo de vista 4+1 ..... - 18 -
Figura 6: Disciplina, fases, iteraciones del AUP ..... - 19 -
Figura 7: Metodología Extreme Programing ..... - 21 -
Figura 8: Interacción entre las capas ..... - 30 -
Figura 9: Vista de Casos de Uso. ..... - 30 -
Figura 10: Vista Lógica ..... - 34 -
Figura 11: Vista de Implementación ..... - 35 -
Figura 12: Diagrama de clases ..... - 37 -
Figura 13: Vista de despliegue ..... - 39 -
Figura 14: Técnicas de Evaluación de Arquitecturas. ..... - 42 -

Índice de Tablas

Tabla 1: Descripción CU Visualizar Interfaz Gráfica de Usuario ..... - 31 -
Tabla 2: Descripción del CU Administrar Modelo 3D ..... - 32 -
Tabla 3: Descripción CU Administrar Eventos ..... - 32 -
Tabla 4: Pasos del método de evaluación ARID ..... - 46 -
Tabla 5: Migración del Sistema Gestor de Base de Datos ..... - 48 -
Tabla 6: Migración de Sistema Operativo ..... - 48 -
Tabla 7: Aumento del modelo de datos ..... - 49 -
Tabla 8: Realizar una consulta a una fuente de almacenamiento de datos ..... - 49 -
Tabla 9: Interacción con otro sistema. ..... - 50 -

Índice de Anexos

Anexo 1: Comparación entre Servidores Web. ..... - 57 -
Anexo 2: Descripción de atributos de calidad observables vía ejecución. ..... - 57 -
Anexo 3: Descripción de atributos de calidad no observables vía ejecución. ..... - 58 -
Anexo 4: Características y subcaracterísticas de calidad – Modelo ISO/IEC 9126 ..... - 59 -
Anexo 5: Comparación entre 3 métodos de evaluación (Pressman, 2002). ..... - 60 -

Introducción

Las Tecnologías de la Información y las Comunicaciones (TIC) han ganado un espacio cada vez mayor en todos los aspectos de la vida. Brindan nuevos espacios para el desarrollo de servicios, la salud, la educación, el comercio, la industria y las relaciones entre personas. Durante el transcurso de los años se ha visto el descubrimiento creciente de nuevas tecnologías y con ella los cambios, oportunidades y amenazas que experimenta el espacio social en el que se desenvuelven los individuos.

En los años 90 con el desarrollo de Internet, las aplicaciones Web fueron tomando el espacio que antes ocupaban las aplicaciones de escritorio. La razón de este cambio estuvo en la facilidad que ofrecían estas nuevas aplicaciones para su distribución y mantenimiento, pues conseguían llegar a más público utilizando un único cliente (navegador Web) (Gerrikagoitia, 2009). Las facilidades que Internet brinda propician que sea cada vez mayor el número de usuarios que se interesen en buscar información o servicios en la Web. Un estudio realizado por la Unión Internacional de Telecomunicaciones (ITU por sus siglas en inglés) en el año 2010 con el nombre "*The World in 2010*" demuestra esta afirmación. Como podemos observar en el figura 1 el número de usuarios en los últimos 5 años se ha duplicado.

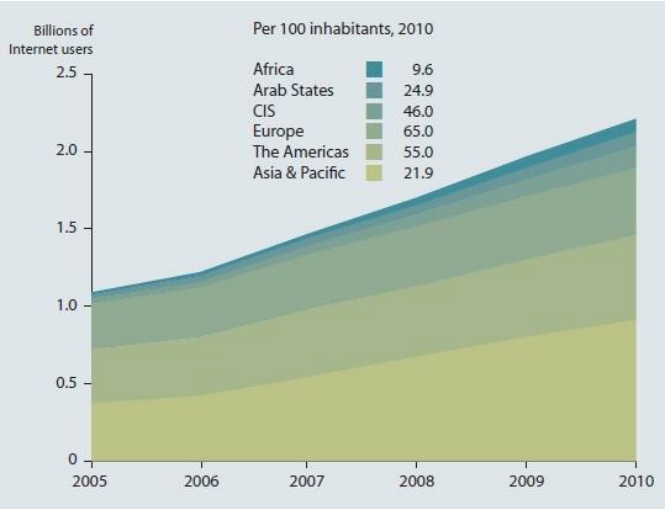

Figura 1: Usuarios de Internet

Los usuarios de Internet ganan cada día mayor cultura en los temas relacionados con la navegación Web, esto trae como consecuencia que sean más exigentes con el rendimiento que estas aplicaciones les brindan, propiciando así que los desarrolladores de este tipo de software busquen constantemente nuevas

soluciones en aras de mejorar las experiencias de los usuarios al visitar una página Web. Una de estas soluciones es la creación de Aplicaciones Enriquecidas de Internet (RIA por sus siglas en inglés) que incluyan escenarios tridimensionales interactivos. Las RIA combinan la capacidad de respuesta y las interfaces de usuarios enriquecidas de las aplicaciones de escritorio con el amplio alcance de las aplicaciones Web (Gerrikagoitia, 2009), por su parte los escenarios tridimensionales interactivos están compuestos por escenas en tres dimensiones que incluyen elementos multimedia. Esta combinación brinda nuevas experiencias a los usuarios transmitiéndoles una cantidad de información adicional que no es posible en la Web tradicional.

La Universidad de las Ciencias Informáticas y en particular el Centro de Desarrollo de Informática Industrial (CEDIN), ha venido investigando la evolución de las RIA, esto ha posibilitado la creación de aplicaciones Web que incluyen escenarios tridimensionales interactivos como por ejemplo el Paseo Virtual del salón de la Industria Informática Cubana (Showroom) realizado en el curso 2009-2010 por la línea de desarrollo Visualización Web.

Las RIA pueden hacer uso de Servicios Web externos al dominio de la aplicación, de Bases de Datos (BD) o de otros tipos de ficheros para adquirir información o para almacenarla, según las características del software que se desea desarrollar. El desarrollo de aplicaciones que tengan estas características en la universidad presenta dificultades, pues la forma en que se accede a los datos es cerrada a cambios. Si en algún momento se necesita cambiar la fuente de datos se hace muy costoso para el equipo de desarrollo en tiempo y esfuerzo realizar esta modificación. Otro de los problemas que enfrentan los desarrolladores es no poder reutilizar las clases encargadas del acceso a los datos, al no contar con un diseño arquitectónico que permita la reutilización. Estos problemas pueden retrasar el tiempo de desarrollo del software y dificultar la mantenibilidad de la aplicación después de creada.

Para mejorar los problemas antes expuestos se necesita diseñar una Capa de Acceso a Datos (*DAL* por su siglas en inglés) reutilizable que abstraiga a los desarrolladores de los tipos de archivos o gestores de BD que puedan ser utilizados para guardar información, mejorando así la mantenibilidad y la calidad de los productos que se desarrollen utilizando este diseño, se minimiza además el tiempo de desarrollo de estos software.

De la situación anterior se identificó el siguiente **problema científico**: ¿Cómo lograr un diseño arquitectónico que permita la modificación y reutilización de la Capa de Acceso a Datos de una Arquitectura de Software de Dominio Específico?

El **objeto de estudio** se enmarca dentro la Arquitectura de Software de Dominio Específico.

Se plantea como **objetivo general**: Diseñar la capa de Acceso a Datos de una Arquitectura de Software de Dominio Específico para desarrollar RIA que incluyan escenarios tridimensionales interactivos.

El **campo de acción** se enfoca en la capa de Acceso a Datos de una Arquitectura de Software de Dominio Específico.

Para dar respuesta a los objetivos planteados se realizaran las siguientes **tareas de investigación**:

- Elaboración del marco teórico de la investigación en el que está enmarcado el sistema.
- Caracterización de las diferentes tecnologías empleadas a nivel mundial para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos.
- Descripción de los patrones y estilos arquitectónicos que puedan ser usados en el diseño arquitectónico de la *DAL*.
- Descripción de los framework de comunicación para seleccionar el que mejor se ajuste a la arquitectura propuesta.
- Descripción de los componentes que integran el diseño propuesto.
- Validación de la propuesta de solución utilizando técnicas de evaluación de Arquitecturas de Software.

Para darle cumplimiento a las tareas de investigación propuestas, haremos uso de los métodos científicos de investigación:

Método teórico:

Análisis histórico–lógico: Este método permitirá realizar un análisis de las características y evolución histórica de diferentes conceptos relacionados con la Arquitectura de Software, entre ellos se estudiaran los estilos y patrones arquitectónicos, metodologías de desarrollo, técnicas y métodos para la evaluación de arquitecturas. Este análisis aportará un punto de referencia para lograr una estructura con calidad de la *DAL*.

Métodos Empíricos:

Observación: A través de este método se indaga sobre las diferentes tecnologías más comunes utilizadas en el mundo para el Acceso a Datos en aplicaciones RIA.

A continuación se hace una pequeña descripción de cada uno de los capítulos para conocer brevemente el contenido de este trabajo.

Capítulo 1: "Fundamentación Teórica", en este capítulo se describen características y funcionalidades de las RIA, se profundiza en los conceptos de Arquitectura de Software y Arquitectura de Software de Dominio Específico, se mencionan y se describen estilos y patrones arquitectónicos así como las metodologías, herramientas, framework y lenguajes de desarrollo que pueden ser usados en la creación de la *DAL*.

Capítulo 2: "Propuesta de Solución", en este capítulo se realiza la selección y justificación de la metodología, herramientas, framework y lenguajes que junto a los estilos y patrones arquitectónicos serán usados en la estructuración de la *DAL*. Se describe además la propuesta de diseño de la *DAL*.

Capítulo 3: "Validación de la Propuesta de Solución", en este capítulo se describen las técnicas, métodos y atributos de calidad que pueden ser usados para evaluar la arquitectura. Se selecciona además uno de estos métodos para realizar la evaluación y dejar descritos los resultados de la misma.

1 Capítulo 1: Fundamentación Teórica

Introducción

En este capítulo se realiza un estudio de los aspectos relacionados con las Aplicaciones Enriquecidas de Internet, se exponen los conceptos de Arquitectura de Software y Arquitectura de Software de Dominio Específico. Se mencionan y se describen patrones y estilos arquitectónicos candidatos a usar en la propuesta de solución, además se identifican los diferentes Entornos Integrados de Desarrollo, lenguajes de programación, framework de comunicación, servidores Web, Sistemas Gestores de Base de Datos y metodologías de desarrollo.

1.1 Aplicaciones Enriquecidas de Internet

En los comienzos de Internet, la Web era simplemente una colección de páginas estáticas que ofrecían textos, imágenes y videos que podían consultarse y/o descargarse. Con el transcurso del tiempo la Web ha ido evolucionando y con ella los contenidos y servicios que se ofrecen, el siguiente paso en su evolución fue la confección de páginas Web con contenidos dinámicos que permiten que lo mostrado varíe en función de las peticiones enviadas al servidor Web (López, 2005).

Hasta este momento las páginas Web presentaban varias limitaciones, entre ellas podemos encontrar:

- Los navegadores interpretan lenguajes basados en scripts, como Java Script, de diferentes maneras, a veces incompatibles entre ellos, esto provoca que los desarrolladores tengan que escribir el código múltiples veces acomodando la programación para cada navegador.
- HTML (Hypertext Transfer Protocol) es un lenguaje estático basado en etiquetas muy limitado, que no puede ser ampliado, efectos como arrastrar, soltar y cambio de tamaño de elementos son imposibles de realizar, al igual que la actualización de zonas específicas de la página sin la necesidad de tener que cargar una nueva página HTML.
- En estas aplicaciones hay una recarga continúa de páginas cada vez que el usuario pulsa sobre un enlace, de esta forma, se produce un tráfico muy alto entre el cliente y el servidor que podría llegar a saturarlo.

La necesidad de eliminar estas limitaciones y crear un nuevo tipo de aplicación Web que no sólo ofrezca multitud de funciones, sino que además sean óptimas y sus interfaces gráficas sean mucho más impactantes y cómodas para los usuarios, ha llevado al surgimiento de las llamadas *Aplicaciones Enriquecidas de Internet.*

Este término fue acuñado por la compañía desarrolladora de software Macromedia, según su definición, una RIA es participativa, interactiva, ligera y flexible, "*las RIA ofrecen la flexibilidad y facilidad de uso inteligente de una aplicación de escritorio y añade el amplio alcance de las aplicaciones Web tradicionales*…" (Fain, 2007).

Tienen como ventajas un mayor tráfico entre cliente y servidor con un menor tiempo de respuesta para el cliente. Son una variante de la arquitectura Cliente-Servidor. Pueden ejecutarse *offline.* Se puede modificar sin intervención del servidor. Efectúa peticiones asincrónicas al servidor, tienen menor necesidad de comunicación entre cliente y servidor que las aplicaciones tradicionales, presentan una lógica de negocios repartida entre ambas facetas, mejoran la interacción en el cliente y carga menos el servidor (Rivero, et al., 2008).

Las RIA son lanzadas desde (o incluso contenida dentro de) una página Web. A través de estas aplicaciones enriquecidas se evita tener que mostrar la pantalla en blanco del navegador hasta que se acaba de cargar la nueva página Web. Permite a los usuarios realizar operaciones que son comunes en clientes pesados como arrastrar, soltar, cambio de tamaño o animación de objetos. El modelo petición/respuesta no es necesario para cada acción realizada en la interfaz de usuario. Con las RIA, el usuario interactúa con la interfaz de usuario y sólo se realizan comunicaciones con el servidor cuando son necesarias. Se requiere la utilización de un complemento (Plug ins en inglés) en el lado del cliente, que será el motor de la tecnología RIA, para esto se utiliza el Adobe Flash Player. Las RIA permiten desvincular la presentación de la aplicación de la lógica de ésta. La complejidad de desarrollo de las aplicaciones no difiere mucho de las aplicaciones Web tradicionales. La aplicación realiza todo lo anteriormente dicho sin la necesidad de grandes plataformas y es compatible con todos los navegadores siempre que estos tengan el complemento Adobe Flash Player instalado (López, 2005).

Las RIA aportan una nueva forma de ver y usar las páginas Web, es una tecnología que permite el desarrollo de múltiples escenarios que pueden ser utilizados de distintas formas en el mercado. La

aceptación que han tenido en el mundo por sus características avanzadas ha logrado que cada día sean más los que se interesen por su uso y desarrollo.

1.2 Arquitectura de Software

Cada vez que se narra la historia de la Arquitectura de Software (AS), se reconoce que en 1968, Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda, propuso que se estableciera una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de forma artesanal, aunque Dijkstra no utiliza el término arquitectura para describir el diseño conceptual del software. El primer estudio en que aparece la expresión "Arquitectura de Software" en el sentido en que hoy lo conocemos fue realizado por Perry y Wolf en 1992, aunque este trabajo se fue gestando desde 1989 (Reynoso, 2004).

Desde esos momentos son muchas las definiciones que se han dado sobre el concepto de AS, según Pressman *"la arquitectura no es el software operacional, más bien, es la representación que capacita al ingeniero del software para: analizar la efectividad del diseño para la consecución de los requisitos fijados; considerar las alternativas arquitectónicas en una etapa en la cual hacer cambios en el diseño es relativamente fácil; y reducir los riesgos asociados a la construcción del software…"* (Pressman, 2002).

La definición de este concepto planteada por Bass, Clements y Kazman precisa que "*la Arquitectura de Software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos…*" (Bass, et al., 1998).

En el sitio oficial del Instituto de Ingeniería de Software (SEI por siglas en inglés) se encuentra publicada una definición en la que se expone que la Arquitectura de Software sirve para modelar el sistema y el proyecto en desarrollo, definiendo las asignaciones de trabajo que deben llevarse a cabo por los equipos de diseño e implementación. La arquitectura es el soporte principal de las cualidades del sistema, tales como el rendimiento, modificabilidad y la seguridad, ninguno de los cuales se puede lograr sin una visión arquitectónica unificadora. La arquitectura es un artefacto para el análisis temprano de un proyecto, permite asegurarse de que el enfoque de diseño de un sistema tiene un rendimiento aceptable. En

resumen, la arquitectura es el pegamento conceptual que sostiene todas las fases del proyecto junto a todas las partes interesadas (SEI, 2011).

Por su parte el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE por sus siglas en inglés) define que una AS es la organización fundamental de un sistema, compuesta por sus componentes, las relaciones entre ellos y su ambiente y los principios que gobiernan su diseño y evolución (IEEE, 2000).

El concepto dado por el SEI es uno de los más completo y al unirlo con los demás conceptos analizados se puede concluir que la arquitectura es el pegamento conceptual que sostiene todas las fases del proyecto junto a todas las partes interesadas, en ella se incluyen los principales componentes, cómo se comportan y cómo interactúan entre ellos. La AS aporta grandes beneficios en cuanto al ahorro de tiempo, la organización del trabajo, el entendimiento entre las partes que intervienen en el desarrollo y posterior mantenimiento del software, además brinda una visión de cómo quedará estructurado el sistema y el modo en que sus componentes interactúan entre sí.

1.2.1 Arquitectura de Software de Dominio Específico

La Arquitectura de Software de Dominio Específico (DSSA por sus siglas en inglés) nació en la Agencia de Investigación de Proyectos Avanzados de Defensa (DARPA por sus siglas en inglés). El desarrollo de esta arquitectura trae beneficios para la creación de proyectos grandes, ya que a través de ésta se puede optimizar el desarrollo en un ámbito específico, se puede reutilizar tantos objetos como sea posible. (KAISER, 2005).

Las DSSA son diseñadas para cubrir un sistema o una familia de sistemas muy centrados en un área o dominio determinado y enfocados a la reutilización, éstas deben ser generales y flexibles dentro del dominio de aplicación. Según (Tracz, 1995) los principales artefactos de una DSSA son: el modelo de dominio, los requerimientos de referencia y la arquitectura de referencia.

El propósito del modelo de dominio es proveer a las personas que desarrollen o mantengan actualizado el producto de un conocimiento inequívoco de varios aspectos del dominio. Los requerimientos de referencia definen los principales requisitos funcionales y no funcionales que son aplicables a todo el dominio, mientras que la arquitectura de referencia brinda una arquitectura genérica que describe el sistema dentro de todo el dominio, satisfaciendo así los requerimientos de referencia.

La utilización de una DSSA tiene como objetivo principal poder reutilizar partes de una arquitectura, modelos arquitectónicos o componentes de la misma, de manera que no sea necesario volver a diseñar el modelo completo. De esta forma, es posible obtener un ahorro en costos y tiempo en la creación de nuevos sistemas de software.

1.3 Estilos arquitectónicos

El término estilo arquitectónico surge por la aparición con regularidad de ciertos comportamientos comunes en el desarrollo de software. Se define el nombre de estilos haciendo analogía con el uso del término en la arquitectura de edificaciones. El estilo arquitectónico constituye uno de los conceptos más importantes dentro de la AS, pues a través de estos se describen y proporcionan las propiedades básicas de una arquitectura y son los encargados de imponer los límites de su evolución.

Según Pressman (Pressman, 2002) un estilo describe una categoría de sistema que abarca un conjunto de componentes que realizan una función requerida por el sistema, un conjunto de conectores que posibilitan la comunicación, la coordinación y cooperación entre los componentes, las restricciones que definen cómo se integran los componentes para conformar el sistema, y los modelos semánticos que facilitan al diseñador el entendimiento de todas las partes del sistema. El estilo arquitectónico es también un patrón de construcción.

Se puede concluir que un estilo arquitectónico brinda una estructura para conocer cómo debe estar organizada la aplicación en cuanto a los componentes, los conectores, las configuraciones y las restricciones de estos elementos y las relaciones que pueden existir entre ellos. Los mismos sirven para sintetizar estructuras de soluciones, así como definir los posibles patrones de las aplicaciones.

1.3.1 Arquitecturas en Capas

El estilo en capas, según Garlan y Shaw (Garlan, et al., 1994), es una organización jerárquica tal que cada capa proporciona servicios a la capa inmediata superior y se sirve de las prestaciones que le brinda la inmediata inferior. Las capas de la aplicación pueden residir tanto en el mismo nodo físico como en nodos separados. Cada capa suele ser una entidad compleja, formada por un conjunto de paquetes o subsistemas. Contribuye a la disminución del acoplamiento, favorece la portabilidad y la sustitución de componentes y proporciona un alto nivel de abstracción. Una especialización muy usada de la arquitectura

en capas es la arquitectura en 3 Capas donde se observan delimitadas las responsabilidades de cada funcionalidad en la aplicación como se puede observar en la figura 2.

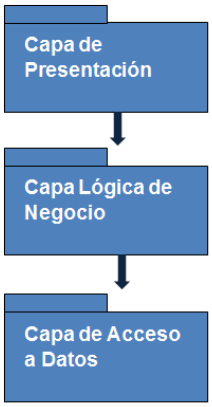

Figura 2: Arquitectura en 3 Capas

La responsabilidad de cada capa queda definida de la siguiente manera:

- **Capa de Presentación:** Esta capa contiene todos los elementos de la interfaz de la aplicación, permitiendo la interacción entre los usuarios y el software. Su función principal es gestionar los datos con el usuario.
- **Capa Lógica de Negocio:** Aquí se establecen todas las reglas del negocio que deben ser cumplidas por el software. Ésta interactúa con la capa de Presentación para recibir los datos provenientes del usuario, así como para enviar respuestas a dichas peticiones. Además, se relaciona con la capa de Acceso a Datos para gestionar la información que se guarda en los medios de almacenamiento.
- **Capa de Acceso a Datos:** Esta capa sirve de enlace entre la Capa Lógica de Negocio y las fuentes de datos. En esta capa existen componentes que hacen transparente el acceso a las fuentes de datos siendo en ésta el mejor lugar para implementar los objetos de acceso a datos permitiendo así las funcionalidades de insertar, eliminar, leer y actualizar la información de las fuentes de datos.

Entre las ventajas de la utilización de este tipo de estilo arquitectónico está que soporta un diseño basado en niveles de abstracción crecientes, lo cual permite a los implementadores la partición de un problema complejo en una secuencia de pasos crecientes que lo hace más entendible y organizado, permite además la optimización y refinamientos del sistema afectando solo áreas específicas, y proporciona un

amplio nivel de reutilización. Por estas características este estilo es frecuentemente utilizado en desarrollos de aplicaciones Web.

1.4 Patrones de diseño

Un patrón de diseño provee un esquema para refinar los subsistemas, componentes de un sistema de software o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en cuanto a comunicación, resuelve además un problema general de diseño en un contexto particular. Son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. A continuación se describen algunos de ellos.

1.4.1 Modelo-Vista-Controlador

Modelo-Vista-Controlador (MVC), como se muestra en la figura 3, separa el modelo de datos, la interfaz y la lógica de negocio de un sistema en tres componentes distintos. Esta división de las responsabilidades posibilita que sea fácil de modificar ante algún cambio que ocurra en el sistema. A continuación se describe cada uno de los componentes (Kiccillof, et al., 2004):

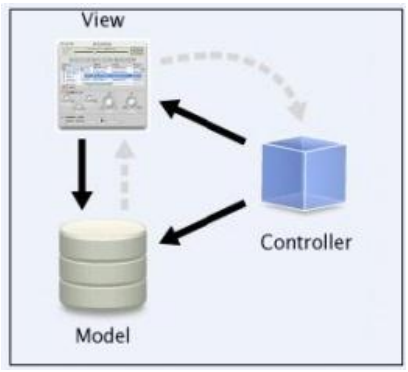

Figura 3: Modelo-Vista-Controlador

Vistas: Se encarga de presentar la información necesaria para interactuar con los usuarios.

Controlador: Responde a eventos, usualmente acciones del usuario que pueden provocar cambios en el modelo y probablemente en la vista.

Modelo: El modelo encapsula la manipulación de los datos y es independiente de las representaciones específicas o del comportamiento de entrada. En él la lógica de datos garantiza y asegura la integridad de

los mismos permitiendo derivar nuevos datos. El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

1.4.2 Patrones GRASP

Los Patrones Generales de Software de Asignación de Responsabilidades (GRASP por sus siglas en inglés) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. A continuación se describen los patrones básicos de asignación de responsabilidades:

Experto: Surge como principio fundamental que hay que tener en cuenta cuando se esté asignando una responsabilidad a una clase. Con la utilización de este patrón se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Se le debe asignar las responsabilidades a la clase que contengan la información necesaria para cumplir las tareas, o sea, la clase debe ser la experta en la información, alentando con ello definiciones de clases sencillas y más cohesivas, que son más fáciles de comprender y mantener.

Creador: Define quien debería ser el responsable de la creación de una nueva instancia de alguna clase. Guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento.

Bajo Acoplamiento: Este patrón plantea la idea de tener las clases lo menos posibles atadas entre sí. De tal forma que en caso de producirse una modificación en alguna de ellas, la repercusión en el resto de las clases sea lo menor posible, potenciando así la reutilización y disminuyendo la dependencia entre las clases. El acoplamiento es una medida de la fuerza con que una clase está relacionada a otras clases, una clase con bajo o débil acoplamiento no depende de muchas otras.

Alta Cohesión: Este patrón indica que cada clase debe realizar una labor única dentro del sistema y que debe colaborar con las demás para llevar a cabo las tareas. La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase.

Controlador: Este patrón ayuda a definir quién será el responsable de atender un evento del sistema. Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente un controlador de fachada.

1.4.3 Patrones GoF

Los patrones GoF fueron descritos en el libro ¨Design Patterns: Elements of Reusable Object-Oriented Software¨, también conocido como el libro GoF (Gang-Of-Four o "pandilla de los 4") escrito por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Según el libro GoF estos patrones se clasifican según su propósito en creacionales, estructurales y de composición. En este libro fueron recopilados y documentados 23 patrones, a continuación se describen algunos de ellos (Gamma, et al., 1998).

Fachada: Este patrón sirve para proveer una interfaz unificada que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Es utilizado para reducir la dependencia entre clases, ya que ofrece un punto de acceso al resto de las clases, si éstas cambian o se sustituyen por otras solo hay que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones clientes. Fachada no oculta las clases sino que ofrece una forma más sencilla de acceder a ellas.

Singleton: Este patrón garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella. Se usa cuando debe haber exactamente una instancia de una clase y ésta debe ser accesible a los clientes desde un punto de acceso conocido. La única instancia debería ser extensible mediante herencia y los clientes deberían ser capaces de usar una instancia extendida sin modificar su código.

Abstract Factory: Proporciona una interfaz para crear familias de objetos relacionados o dependientes sin especificar clases concretas. Permite trabajar con objetos de distintas familias de manera que éstas no se mezclen entre sí, haciendo transparente el tipo de familia concreta que se esté usando.

1.5 Entorno Integrado de Desarrollo

Un Entorno Integrado de Desarrollo (IDE por sus siglas en inglés) es una aplicación informática compuesta por un conjunto de herramientas de programación. Puede dedicarse en exclusivo a un solo lenguaje de programación o bien, poder utilizarse para varios. Un IDE es un entorno de programación que

ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de Interfaz Gráfica del Usuario (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Estos proveen además un marco de trabajo amigable para la mayoría de los lenguajes de programación.

1.5.1 Eclipse

El IDE Eclipse es una herramienta universal, multiplataforma, de código abierto (open source) y extensible para todo, que permite a través de la incorporación de complementos extenderse para usar varios lenguajes de programación como C/C++, Cobol, PHP, C# y Action Script. Es administrado y dirigido por un consorcio de compañías de desarrollo de software sin fines de lucro liderado por IBM (Fundación Eclipse , 2011). Con la adición al Eclipse del Framework Flex 4 SDK se logra crear un IDE para el desarrollo de RIA, esta unión es recomendable para aquellos desarrolladores que necesiten un entorno de desarrollo de código abierto y tengan experiencia en el trabajo con el Eclipse.

1.5.2 Adobe Flash Builder 4

El desarrollo de RIA también se puede realizar utilizando el IDE Adobe Flash Builder 4, este IDE no es más que un empaquetado del Eclipse y Flex SDK realizado por la compañía Adobe la cual posee su licencia. Presenta un entorno gráfico donde se previsualiza la aplicación 100% de cómo quedaría una vez compilada, además de adelantar esta tarea en la parte de la implementación; acelera además el desarrollo y pruebas de las RIAs, ofreciendo capacidades de codificación y prueba, nuevas y mejoradas; permite a los diseñadores y desarrolladores colaborar de forma más eficiente; logra una integración con una variedad de servidores incluyendo PHP, ColdFusion, J2EE y ASP.NET (Adobe Corporation, 2011).

1.5.3 Zend Studio

Zend Studio es un IDE de programación que proporciona un buen número de ayudas desde la creación y gestión de proyectos hasta la depuración del código. Mediante herramientas de edición, análisis y optimización incrementa la velocidad de los ciclos de desarrollos y hace más fácil los proyectos de gran complejidad. Zend Studio puede correr tanto en Linux como en Windows. Se distribuye comercialmente como software privativo. Soporta varios lenguajes de programación, aunque fue construido especialmente para PHP.

Zend Studio está disponible en dos ediciones: Standard y Professional. Este IDE fue concebido con el fin de crear aplicaciones altamente fiables, proporciona una facilidad de uso inigualable, escalabilidad, fiabilidad y la extensión que los programadores profesionales y de empresas requieren para desarrollar, distribuir, depurar y administrar aplicaciones desarrolladas en PHP. Zend Studio proporciona la visualización, edición y la capacidad de ejecución para BD populares SQL incluyendo MySQL, Oracle, IBM DB2 y Cloudscape, Microsoft SQL Server, SQLite y PostgreSQL (Zend, 2011).

1.6 Lenguajes de Programación

Un lenguaje de programación es un idioma artificial diseñado para describir el conjunto de acciones que equipos como las computadoras pueden ejecutar. Se usan para crear programas que controlen el comportamiento físico y lógico de una máquina. Un lenguaje de programación está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual se escribe, se prueba, se depura, se compila y se mantiene el código fuente de un programa informático se le llama programación.

1.6.1 PHP

PHP (acrónimo de Hypertext Preprocessor) es un lenguaje script de código abierto que es procesado en el lado del servidor y el resultado es enviado al navegador. Entre sus principales características se destaca que es libre, por lo que es de fácil acceso para todos. Es multiplataforma ya que es soportado por Windows y Linux, además que cuenta con muchas funcionalidades. La meta del lenguaje es permitir rápidamente a los desarrolladores la generación dinámica de páginas. Al ser PHP un lenguaje que se ejecuta en el servidor no es necesario que su navegador lo soporte, es independiente del navegador, sin embargo, para que sus páginas PHP funcionen el servidor donde están alojadas debe soportar PHP. Tiene soporte para una gran cantidad de BD: MySQL, PostgreSQL, Oracle, MS SQL Server, Sybase mSQL, Informix, entre otras. Permite el empleo de Programación Orientada a Objeto (POO), no requiere que se le especifique el tipo de datos de las variables y maneja excepciones a partir de la versión 5.0 (Southwell, et al., 2005).

1.6.2 Action Script 3.0

Action Script es el lenguaje de programación para los entornos en tiempo de ejecución de Adobe Flash Player y Adobe AIR. Action Script se ejecuta mediante la máquina virtual Action Script (AVM), que forma parte de Flash Player y AIR. El código Action Script suele transformarse en formato de código de bytes mediante un compilador (Bytecode un tipo de lenguaje que los ordenadores pueden escribir y comprender). El código de bytes está incorporado en los archivos SWF ejecutados por Flash Player y AIR. Entre los ejemplos de compiladores se incluyen el incorporado en Adobe Flash Professional, en Adobe Flash Builder y el SDK de Adobe Flex. Action Script 3.0 ofrece un modelo de programación robusto que resultará familiar a los desarrolladores con conocimientos básicos sobre POO. 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 (Jacobs, et al., 2008). Estas características junto a otras hacen que este lenguaje sea unos de los más usados en desarrollos de RIA.

1.6.3 XML

El Lenguaje de Marcado Extensible (XML por sus siglas en inglés) está basado en el Lenguaje de Marcado Generalizado Estándar (SGML por sus siglas en inglés). XML es una de las tecnologías más importantes que respaldan el desarrollo Web moderno.

XML crea lenguajes basados en texto que trabajan con contenidos estructurados. Se puede utilizar las reglas de la construcción de XML para inventar etiquetas que describen un conjunto particular de datos y las estructuras de datos. También hay muchos vocabularios XML estandarizados para su uso en las industrias y situaciones específicas, uno de estos lenguajes creados a partir de las reglas de XML es MXML.

Cualquier archivo que su contenido esté marcado por etiquetas que sigan las reglas de XML se denomina documento XML. Este tipo de archivo puede ser un documento físico o puede ser una corriente de datos digitales, puede ser creado a partir de una llamada a una BD o a través de otro Software. Un Servicio Web o una fuente RSS (Really Simple Syndication) son ejemplos de datos XML que son proporcionados por una corriente digital. Un documento XML es una forma de almacenar datos en un archivo de texto. XML permite describir la estructura y contenido de modo que un procesador XML o un software pueda leer el

contenido de una manera específica. Un navegador Web es un ejemplo de un analizador XML, ya que puede leer y representar el contenido XHTML. Del mismo modo, Flash Builder contiene un procesador de XML que hace que las etiquetas MXML sean visualizadas en la vista Diseño (Jacobs, et al., 2008).

1.7 Framework de comunicación

En este epígrafe se describen diferentes framework que pueden ser usados para comunicar Action Script con la BD a través de PHP y poder así hacer uso de los datos almacenados en BD cuando sea necesario para la creación de una RIA.

1.7.1 AMFPHP

AMFPHP (Action Message Format) es uno de los framework más usados y unos de los primeros creados que permite la serialización binaria de objetos y tipos nativos de Action Script 2 y Action Script 3 para ser enviados a servicios del lado del servidor. Es multiplataforma, gratuito y de código abierto. Brinda la posibilidad además de que aplicaciones realizadas en Flash, Flex y AIR puedan hacer uso de llamadas a procesos remotos para comunicarse directamente con objetos de clases PHP.

La forma en la que funciona se resume en que se pueden crear clases en PHP que son las encargadas de conectarse a la BD y realizar las consultas que se necesiten, después se acceden a los métodos de esas clases por medio de llamadas remotas desde Action Script y se obtienen así los datos retornados por los métodos PHP. Tiene como ventajas que es ligero, incluye una gestión de roles bastante simple, no es necesaria su instalación, soporta mapeo de clases y por último, incluye un navegador de servicios para que se puedan probar rápidamente los métodos implementados en PHP así como para conocer varias informaciones sobre el tiempo de respuesta, el resultado que se deben retornar, entre otros datos que facilitan mucho el desarrollo de aplicaciones RIA (AMFPHP, 2011).

1.7.2 WebOrb

WebOrb es una alternativa gratis y libre para conectar Action Script con PHP y hacer uso de las fuentes de datos. Bridan soporte al protocolo en Java, .NET, PHP y Ruby, con un rendimiento muy alto y estable. WebOrb es una serie de archivos del lado del servidor (en PHP) que permiten a Flash o Flex conectarse directamente con PHP. Según la bibliografía consultada se puede decir que es el más completo de los framework estudiados, pero esto trae como consecuencia que sea un poco difícil de entender y utilizar por

desarrolladores novatos. Entre sus características está que presenta un administrador de servicios junto con opciones pensadas para ahorrarle al desarrollador la necesidad de escribir código, incluye su propio administrador de acceso a servicios y métodos, además de un administrador centralizado para el mapeo de clases.

1.7.3 ZendAMF

ZendAMF es un framework desarrollado por Zend que proporciona soporte al AMF de Adobe para permitir la comunicación entre el Flash Builder y PHP. Específicamente, provee una implementación del servidor de puerta de enlace para tramitar las solicitudes enviadas desde el Flash Builder hasta el servidor. ZendAMF no cuenta con un navegador de servicios, haciendo más difícil la depuración del código PHP en comparación con los framework antes descritos. Sin embargo, tiene la gran ventaja de ser parte de Zend Framework permitiéndole ajustarse perfectamente a todo su conjunto de herramientas e integrarse de manera sencilla a cualquier proyecto de PHP. Es un framework de código abierto utilizado por el Adobe Flash Builder que permite a los desarrolladores de Flex hacer llamadas a procedimientos remotos desde Flex hacia una clase PHP, además utiliza AMF para serializar los mensajes entre el servidor y el cliente Flex, ofrece la posibilidad de asignar una clase de Action Script a una clase de PHP y provee una pasarela en el servidor para manejar las solicitudes enviadas por el Flash Player mapeándolas a objetos y métodos arbitrarios (Zend Technologies Ltd, 2011).

1.8 Servidores Web

Un Servidor Web es un programa diseñado para alojar las páginas Web que serán mostradas tras la petición realizada por un navegador Web mediante el protocolo HTTP. Navegadores Web como Microsoft Internet Explorer, Apple Safari o Mozilla Firefox, son algunos de los navegadores más comunes que muestran las salidas del servidor Web tras la petición realizada por un usuario. A continuación se describen algunos de los servidores más importantes.

1.8.1 Apache

El servidor Apache es desarrollado por la Apache Software Fundation. Es considerado la plataforma de servidores Web de código abierto más potente y utilizada del mundo, pudiendo ejecutarse en sistemas operativos como Linux, Macintosh y Windows. Las capacidades de este servidor pueden ser ampliadas

incorporándole nuevos módulos, pues su diseño modular es altamente configurable. Por su sencillez, robustez, flexibilidad, estabilidad y eficiencia ha logrado ser el servidor Web más utilizado en el mundo, muy superior a los demás existentes, esto es demostrado por la empresa Netcraf, pues éste afirma que un 61.13 % de las aplicaciones publicadas en Internet en abril del 2011 usan Apache como su servidor Web. Se señala como desventaja que no posee una interfaz gráfica que facilite su configuración (Apache Foundation, 2011).

1.8.2 Internet Information Services (IIS)

Es un servidor Web desarrollado y licenciado por la corporación Microsoft Corporation. Está limitado a usarse solamente en plataforma Windows lo cual es una limitante. Ofrece servicios FTP, SMTP, NNTP y HTTP/HTTPS, también proporciona las herramientas y funciones necesarias para su administración de forma sencilla y segura. Está basado en varios módulos que le dan la posibilidad de poder procesar distintos tipos de páginas. Su principal problema radicaba en la pobre seguridad de sus primeras versiones, aspecto que fue resuelto en la versión 6.0. Según los estudios realizados por la empresa Netcraf es el segundo servidor Web más usado en el mundo con un por ciento de uso de 18.83 % en abril del 2011 (Microsoft Corporation, 2011).

1.9 Sistemas de Gestión de Base de Datos

Los Sistemas de Gestión de Bases de Datos (en inglés Data Base Management System, SGBD) son un tipo de software muy específico, dedicado a servir de interfaz entre la BD, el usuario y las aplicaciones que la utilizan. Un SGBD permite definir los datos a distintos niveles de abstracción, crear y mantener una BD, asegurando su integridad, confidencialidad y seguridad.

1.9.1 MySQL

MySQL es uno de los SGBD más utilizado para el desarrollo de páginas Web y aplicaciones de escritorio. Funciona en varias plataformas, es fácil de usar, instalar y mantener. Existe una amplia y variada documentación sobre él, tiene un buen soporte a través de grupo de usuarios y también ofrece soporte comercial. Es rápido, puede administrar los vínculos entre filas 1.5 miles de millones de datos en varias tablas virtuales. Puede manejar cientos de clientes que se conectan al servidor y el uso de varias BD simultáneamente. Ofrece tolerancia a fallos, equilibrio de carga y la seguridad a través de la replicación.

Su conectividad, velocidad y seguridad hacen que MySQL Server sea altamente apropiado para acceder BD en Internet. (Giacomo, 2005)

1.9.2 PostgresSQL

PostgreSQL es un SGBD, desarrollado en la Universidad de California en Berkeley por el Departamento de Ciencias de la Computación. PostgresSQL fue pionero en muchos conceptos los cuales estuvieron disponibles en algunos SGBD comerciales más tarde. Los desarrolladores de PostgreSQL han luchado por mejorar la integridad de los datos, el rendimiento y otras optimizaciones. Presenta compatibilidad con más de una docena de lenguajes diferentes, incluyendo C, SQL, PL, pgSQL, PL, Perl, PL, PHP, PL, Tcl, Ruby. Soporta una amplia gama de protocolos de seguridad y opciones de configuración, además puede ser ampliado por el usuario en muchos aspectos, entre otras se encuentra la adición de nuevos tipos de datos y nuevas funciones. Una de las características más relevantes de este SGBD es la gestión de transacciones simultáneas, ya que utiliza un sistema conocido como Control de Concurrencia Multiversión (MVCC) para evitar dificultades cuando haya muchos usuarios realizando lecturas o escrituras o se estén ejecutando muchas sentencias SQL simultáneamente en la BD. PostgreSQL está patentado bajo una licencia BSD, lo que significa que puede ser utilizado tanto en código abierto como en aplicaciones comerciales de forma gratuita y es multiplataforma (Gilmore, 2006).

1.10 Metodología de Desarrollo de Software

Las Metodologías de Desarrollo de Software brindan un marco de trabajo para definir una serie de procedimientos, técnicas, herramientas y soporte documental necesario para llevar a cabo con altas posibilidades de éxito el desarrollo de un software. Las metodologías pretenden guiar a los desarrolladores al crear un nuevo software, pero los requisitos de un software a otro son tan variados y cambiantes que ha dado lugar a que existan una gran variedad de metodologías. Se pueden identificar dos grandes grupos: las metodologías orientadas al control de los procesos, llamadas Metodologías Pesadas; y las metodologías orientadas a la interacción con el cliente y el desarrollo incremental del software, llamadas Metodologías ligeras o ágiles. A continuación se describen algunas de las más utilizadas.

1.10.1 Rational Unified Process (RUP)

RUP es una de las metodologías pesadas más conocidas y utilizadas en el desarrollo de software. Esta metodología divide el desarrollo en 4 fases que definen su ciclo de vida:

- **Inicio:** El objetivo es determinar la visión del proyecto y definir lo que se desea realizar.
- **Elaboración:** Etapa en la que se determina la arquitectura óptima del proyecto.
- **Construcción:** Se obtiene la capacidad operacional inicial.
- **Transición:** Obtener el producto acabado y definido.

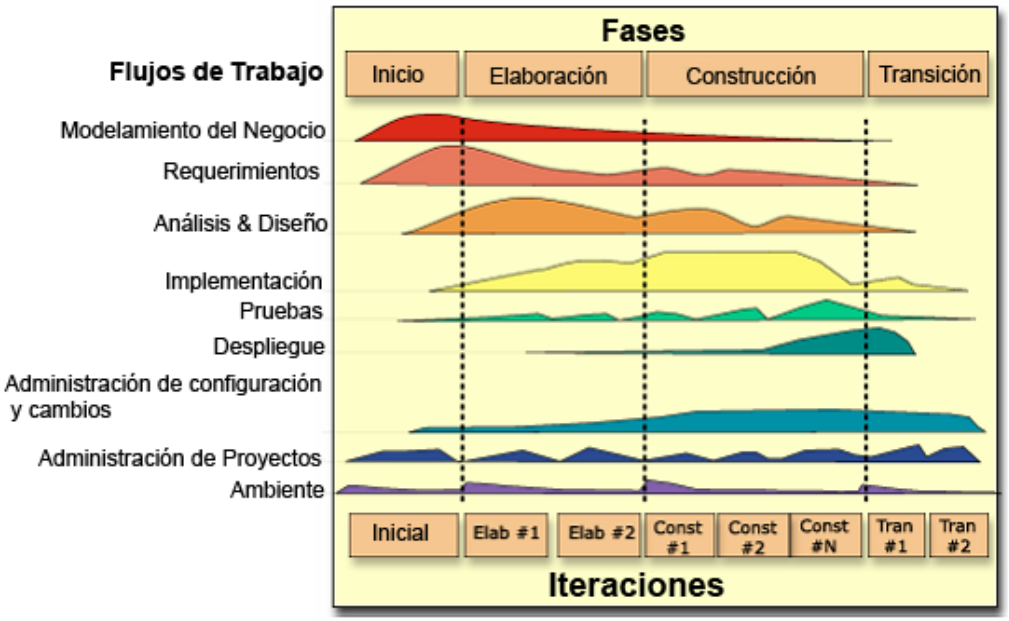

Figura 4: Fases e iteraciones de RUP

RUP determina 6 flujos de trabajos que se pueden observar en la figura 4 y serán descritos a continuación:

- **Ingeniería o modelado del negocio**: Analizar y entender las necesidades del negocio para el cual se está desarrollando el software.
- **Requisitos:** Proveer una base para estimar los costos y tiempo de desarrollo del sistema.
- **Análisis y diseño:** Trasladar los requisitos analizados anteriormente a un sistema automatizado y desarrollar una arquitectura para el sistema.

- **Implementación:** Crear software que se ajuste a la arquitectura diseñada y que tenga el comportamiento deseado.
- **Pruebas:** Asegurarse de que el comportamiento requerido es correcto y que todo lo solicitado está presente.
- **Despliegue:** Producir distribuciones del producto y distribuirlo a los usuarios.

Entre sus características se encuentra que: utiliza el Lenguaje Unificado de Modelado (UML), para la confección de todos los esquemas de los sistemas; es una metodología dirigida por casos de uso (CU), centrada en la arquitectura, iterativo e incremental; es usada comúnmente para proyectos de larga duración y alta complejidad; los elementos que la componen son: las *actividades*, los *trabajadores* y los *artefactos;* en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo de software; se representa mediante un número diferente de vistas arquitectónicas llamadas el "modelo de vista 4+1" que se muestra en la figura 5 y se describe a continuación (Kruchten, et al., 2000).

Figura 5: Modelo de vista 4+1

- La v**ista de casos de uso** representa la selección y descripción de los CU arquitectónicamente significativos brindando una noción general del sistema.
- La **vista lógica** representa los elementos de diseño más importantes para la arquitectura del sistema. En esta vista se describe las clases más importantes, su organización en paquetes y subsistemas y estos a su vez en capas.
- La **vista de implementación** describe la organización estática del software en su ambiente de desarrollo.
- La **vista despliegue** suministra una base para la comprensión de la distribución física de un sistema a través de nodos. Suele utilizarse cuando el sistema está distribuido.

- La **vista de procesos** suministra una base para la comprensión de la organización de los procesos de un sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Solo suele usarse cuando el sistema presenta procesos concurrentes o hilos.

1.10.2 Agile Unified Process (AUP)

Agile Unified Process es una versión simplificada de RUP que fue desarrollado por Scott Amber. Éste describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software usando técnicas ágiles y conceptos que permanecen fieles al RUP. Se ha tratado de mantener AUP tan simple como sea posible, tanto en su enfoque como en su descripción. El AUP aplica técnicas ágiles incluyendo Desarrollo Dirigido por Pruebas (Test Driven Development - TDD), Modelado Ágil, Gestión de Cambios Ágil y Refactorización de Base de Datos para mejorar la productividad.

La figura 6 representa el ciclo de vida de AUP, en esta imagen se puede observar que las disciplinas han cambiado. La disciplina de Modelado abarca las disciplinas de Modelado del Negocio, de Requerimientos y de Análisis y Diseño de RUP. El modelado es una parte importante en AUP pero no domina el proceso. Las disciplinas de la Administración de la Configuración y Cambios ahora es la disciplina de Administración de la Configuración. A continuación se describen las fases y las disciplinas.

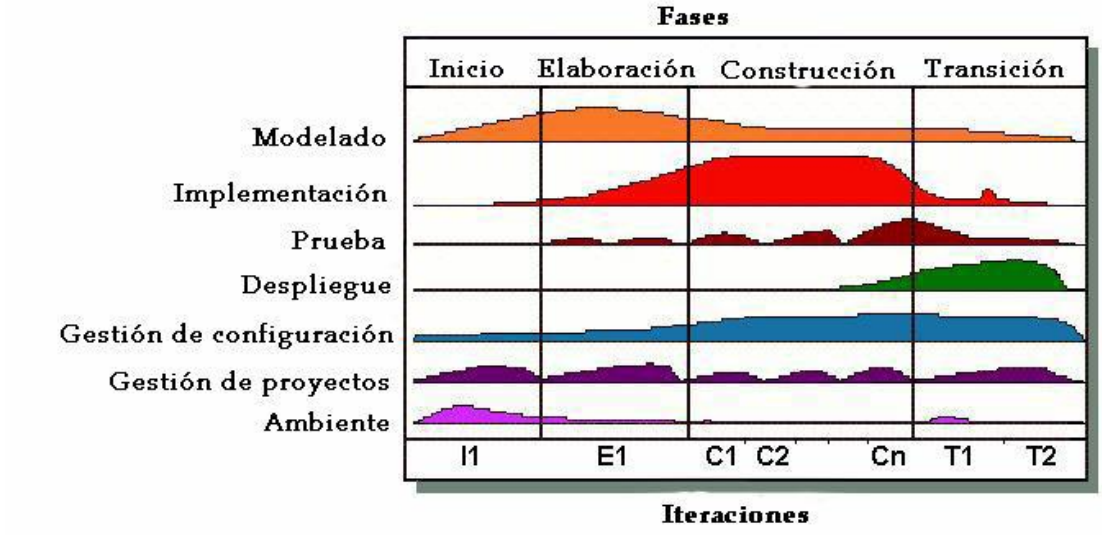

Figura 6: Disciplina, fases, iteraciones del AUP

El ciclo de desarrollo está dividido en 4 fases:

- **Inicio:** Identificación del alcance y dimensión del proyecto, propuesta de la arquitectura y de presupuesto del cliente.
- **Elaboración:** Confirmación de la idoneidad de la arquitectura.
- **Construcción:** Desarrollo incremental del sistema, siguiendo las prioridades funcionales de los implicados.
- **Transición:** Validación y despliegue del sistema en su ambiente de la producción.

AUP define 7 disciplinas:

- **Modelado:** El objetivo de esta disciplina es entender el negocio de la organización, el problema de dominio que se aborda en el proyecto, e identificar las soluciones viables para manejar el dominio del problema.
- **Implementación:** El objetivo de esta disciplina es transformar su modelo (s) en código ejecutable y llevar a cabo un nivel básico de las pruebas, en particular, las pruebas unitarias.
- **Pruebas:** El objetivo de esta disciplina es ejecutar una objetiva evaluación para asegurar la calidad. Esto incluye la detección de defectos, validaciones para verificar que el sistema funciona como fue diseñado y verificar que se cumplan los requerimientos.
- **Despliegue:** El objetivo de esta disciplina es planificar la entrega del proyecto de desarrollo y ejecutar el plan para dejar disponible el sistema al usuario final.
- **Administración de Configuración:** La meta de esta disciplina es manejar el acceso a los productos del trabajo del proyecto. Ésta no sólo incluye el rastreo de versiones del trabajo del producto en el tiempo, sino también el control y administración de los cambios de estos productos.
- **Gestión del Proyecto:** El objetivo de esta disciplina es dirigir las actividades a lo largo del proyecto. Esto incluye la administración del riesgo, dirección del personal (asignación de tareas, rastreo del progreso, etc.) y coordinación con personas y sistemas fuera del alcance del proyecto para asegurar su liberación a tiempo y dentro del presupuesto.
- **Ambiente:** El objetivo de esta disciplina es soportar el resto del esfuerzo asegurando que el proceso apropiado, las guías (normas y directrices) y herramientas (hardware y software) estén disponibles para cuando el equipo las necesite.

AUP basa su filosofía en que el personal necesita saber lo que está haciendo, la simplicidad, la agilidad, se centra en las actividades importantes, en la independencia de las herramientas y en la flexibilidad de la metodología. Estos son beneficios que se pueden tener en cuenta para utilizar esta metodología.

1.10.3 Programación Extrema (XP)

La Programación Extrema, es una metodología ligera de desarrollo de software que se basa en la simplicidad, la comunicación y la reutilización del código desarrollado. La metodología consiste en una programación rápida o extrema, utilizada para proyectos de corto plazo. El desarrollo bajo XP tiene características que lo distinguen claramente de otras metodologías: los diseñadores y programadores se comunican efectivamente con el cliente y entre ellos mismos; los diseños del software se mantienen sencillos y libres de complejidad o pretensiones excesivas; se obtiene retroalimentación de usuarios y clientes desde el primer día gracias a las baterías de pruebas; el software es liberado en entregas frecuentes tan pronto como sea posible; los cambios se implementan rápidamente tal y como fueron sugeridos; las metas en características, tiempos y costos son reajustadas, permanentemente en función del avance real obtenido.

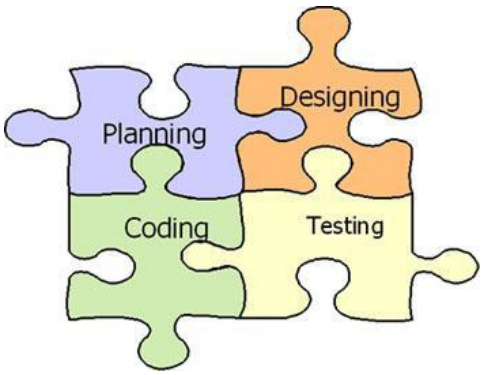

Figura 7: Metodología Extreme Programing

Esta metodología está basada en:

- **Pruebas Unitarias:** se basa en las pruebas realizadas a los principales procesos, de manera tal que se pueda prevenir posibles errores que en el futuro pudieran ocurrir.
- **Refabricación:** se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

- **Programación en pares:** una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

1.11 Herramientas CASE

Una herramienta CASE es una tecnología para automatizar el desarrollo y mantenimiento del software, combinando herramientas de software y metodologías. Estas herramientas deben constituir un conjunto integrado que automatice todas las partes del ciclo de vida de desarrollo de un software y por tanto ahorren trabajo.

1.11.1 Rational Rose Enterprise Edition

Rational Rose Enterprise Edition proporciona un lenguaje de modelado visual que permite crear de prisa arquitecturas, componentes y datos usando UML. Agiliza el desarrollo de aplicaciones C++, CORBA, Java, J2EE, Visual C++, Visual Basic, entre otros, con código generado a partir de modelos visuales. Es compatible con el lenguaje UML (Unified Modeling Language) y es uno de los productos más completos de la familia Rational Rose. Es capaz de analizar la calidad del código y de generar código gracias a las capacidades de sincronización configurable entre el modelo y el código, además de una gestión más detallada y el uso de modelos con la función de componentes de modelos controlables por separado. Incluye un complemento de modelado Web, que proporciona la capacidad de visualización y modelado para desarrollar aplicaciones Web. Permite el modelado UML para diseñar BD, con la posibilidad de representar la integración de los requisitos de datos y aplicaciones mediante diseños lógicos y físicos. Funciona sobre el sistema operativo Windows y es de licencia propietaria (IBM , 2011).

1.11.2 Visual Paradigm

Visual Paradigm-UML (VP-UML) es una herramienta diseñada para usuarios como Ingenieros de Software, Analistas de Sistemas, Arquitectos de Sistemas y otros que estén interesados en el diseño de software. Con VP-UML se puede crear los diferentes diagramas de UML con solo realizar la operación de arrastrar y soltar. Presenta un diseño centrado en CU y enfocado al negocio que permite generar un software de mayor calidad. Brinda un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación. Tiene capacidades de ingeniería directa (versión profesional) e inversa. El modelo y

código permanece sincronizado en todo el ciclo de desarrollo, está disponible en múltiples versiones, para cada necesidad y puede integrarse a los principales IDEs como Eclipse, JBuilder, Oracle JDeveloper entre otros y es multiplataforma. Soporta aplicaciones Web y puede generar código C#, VB.NET, Action Script, Delphi, C, entre otros. Es fácil de instalar y actualizar (Vizcaíno, et al., 2005).

Conclusiones del capítulo

En este capítulo quedaron expuestos los principales conceptos con los que se va a trabajar en el transcurso del desarrollo de la arquitectura, entre ellos el de AS, DSSA y RIA. Se realiza además en este capítulo un estudio de los diferentes estilos y patrones arquitectónicos que pueden ser utilizados en el desarrollo de la arquitectura. El análisis de diferentes herramientas informáticas que pueden ser usados en el desarrollo también está entre los aspectos que recoge este capítulo, entre éstas se estudian varios IDE, lenguajes de programación, herramientas de modelado, metodologías de desarrollo, SGBD, framework de comunicación y Servidores Web. Después del análisis de estos conceptos y herramientas se tiene los conocimientos teóricos necesarios para el entendimiento del presente trabajo.

2 Capítulo 2: Propuesta de Solución

Introducción

En este capítulo se realiza la propuesta de herramientas necesarias para construir una *DAL*. Se expone la arquitectura propuesta de la *DAL* así como la descripción de sus componentes y cómo interactúan entre sí. Se proponen además los estilos y patrones arquitectónicos que se pueden usar para lograr un desarrollo eficiente de esta capa. Finalmente, se describe la interacción de la *DAL* propuesta con las demás capas de la arquitectura.

2.1 Línea base de la arquitectura

La Línea Base de la Arquitectura contiene elementos de gran importancia para lograr la máxima abstracción en el diseño arquitectónico de la aplicación. En la misma se exponen los patrones de diseño utilizados, las tecnologías y herramientas de software que se utilizarán para desarrollar la capa de Acceso a Datos de una RIA, se describe además como queda estructurada la propuesta a través del modelo 4+1 vista propuesto por AUP.

2.1.1 Propósito

El propósito de la línea base es proporcionar la información para estructurar el sistema desde el más alto nivel de abstracción. Se define la estructura del sistema en cuanto a los elementos, la configuración y sus restricciones.

Los usuarios de la línea base de la arquitectura son:

- Equipo de arquitectos del proyecto, estos guían la toma de decisiones que tengan que ver con la arquitectura, son los encargados del mantenimiento y refinamiento de la línea base.
- Los miembros del equipo de desarrollo encargados de realizar la aplicación, la utilizan para guiar la implementación del sistema.
- Los clientes tienen en ella una garantía de la calidad y el conocimiento sobre la tecnología en que está desarrollada la solución.

2.2 Selección de las tecnologías a utilizar

En este epígrafe se proponen los patrones de diseño, los IDE, el framework de comunicación, el servidor Web, el SGBD, la metodología de desarrollo y la herramienta de modelado que pueden ser utilizadas para crear una *DAL*.

2.2.1 Patrones de diseño

Para el desarrollo de la *DAL* se propone la utilización de los patrones Alta Cohesión, Bajo Acoplamiento, Experto, Creador, Controlador y Singleton. A través de estos patrones el sistema será más fácil de mantener, modificar y adicionar nuevos módulos o clases que brinden nuevas funcionalidades. El uso específico de estos patrones será explicado en próximos epígrafes detalladamente.

2.2.2 Entorno de desarrollo

Se propone como IDE para el desarrollo de RIA el Adobe Flash Builder 4 por las capacidades de codificación y prueba que presenta, así como por las posibilidades de integración con una variedad de servidores. A través de éste se puede conectar rápidamente los datos y los servicios así como enlazar los datos a los componentes de interfaz de usuario, esto beneficia a los desarrolladores pues brinda nuevos métodos para la construcción de aplicaciones avanzadas orientadas a datos. Para la implementación en PHP se propone el Zend Studio, esta herramienta brinda grandes facilidades para los desarrolladores y agiliza el proceso de implementación.

2.2.3 Lenguaje de desarrollo.

Los lenguajes de desarrollo propuesto son: Action Script 3 para el desarrollo de aplicaciones RIA utilizando el IDE Adobe Flash Builder 4; el lenguaje PHP será usado para acceder a los datos almacenados en la BD. Puede utilizarse además el XML para guardar información de una manera estructurada si es necesario para la aplicación que se desee desarrollar.

2.2.4 Framework de comunicación

El framework de comunicación seleccionado es el AMFPHP, pues es uno de los más utilizados en el mundo existiendo gran cantidad de bibliografía que se puede consultar. Presenta además una vista en la cual se puede probar los métodos creados en las clases PHP y visualiza conjuntamente los resultados que éstos devuelven así como información del tiempo de respuesta.

2.2.5 Servidor Web

El servidor Web propuesto para alojar las aplicaciones RIA es el Apache, ya que es el servidor Web más usado en el mundo, es multiplataforma, libre y de código abierto, altamente configurable y su robustez, rapidez y seguridad están ampliamente probadas. Soporta además una gran variedad de lenguajes que lo hacen una elección indiscutible para albergar cualquier tipo de aplicación Web. En el anexo 1 se muestra una comparación con otro servidor que puede demostrar porqué se ha seleccionado el servidor Apache.

2.2.6 Metodología de desarrollo

Se propone como metodología de desarrollo AUP ya que es una metodología liviana y fácil de entender para desarrollar software a corto plazo y con poca documentación técnica. Está basada en RUP, por lo que es muy familiar a las personas que hayan trabajado con esta metodología anteriormente, ya que se utilizan las técnicas y conceptos que RUP. Otra ventaja de esta metodología es que permite centrarse en las actividades de valor y es flexible a cambios que podamos realizarle para que se ajuste más a nuestro proyecto, además es dirigida por casos de usos y centrada en la arquitectura.

2.2.7 Herramienta de modelado

La herramienta de modelado escogida es Visual Paradigm pues su licencia es libre. Entre sus ventajas está que soporta el ciclo de vida completo del desarrollo de software, permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas para varios lenguajes y generar documentación. Es muy sencilla de usar, fácil de instalar y actualizar.

2.2.8 Gestor de Base de Datos

Se recomienda la utilización de PostgreSQL pues es una alternativa de software libre multiplataforma que permite ser modificado según las necesidades de los usuarios. Este gestor brinda seguridad e integridad en los datos almacenados, además es compatible con varios lenguajes y tienen un buen rendimiento ante múltiples conexiones de usuarios.

2.3 Requerimientos del sistema

Un requisito de software es una condición que debe cumplir un sistema para satisfacer un contrato, estándar, especificación u otra documentación formalmente impuesta.

2.3.1 Requerimientos Funcionales

Los requerimientos funcionales (RF) no son más que las condiciones o capacidades que el sistema debe cumplir. A continuación se describen los RF que debe cumplir la propuesta de solución:

RF 1. El diseño de arquitectura propuesto debe permitirle a una RIA que incluya escenarios tridimensionales interactivos visualizar interfaces gráficas de usuarios (GUI).

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 RIA que incluya escenarios tridimensionales interactivos administrar modelos 3D.

RF 2.1. El diseño de arquitectura propuesto debe permitir cargar modelos 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 RIA que incluya escenarios tridimensionales interactivos administrar los eventos generados por un dispositivo de entrada.

2.3.2 Requisitos No Funcionales

Los requerimientos no funcionales (RNF) son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. Los mismos están vinculados con los requisitos funcionales, es decir, una vez que se conozca lo que el sistema debe hacer podemos determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser. A continuación se detallan los RNF que debe cumplir el diseño propuesto.

RNF 1. Requisitos de Mantenibilidad

- La propuesta de arquitectura debe ser de fácil mantenimiento.
- La propuesta de arquitectura debe brindar la facilidad de que los programadores y diseñadores realicen el mínimo esfuerzo para corregir un error del programa o hacerle una actualización al mismo.
- La propuesta debe tener un diseñado modular, así todas las funcionalidades deben estar encapsuladas en estos para que sea fácil el proceso de agregar nuevas funcionalidades.

RNF 2. Requisitos de Seguridad

- Garantizar que la información sea vista únicamente por quien tiene derecho a verla según sus privilegios.
- Protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.
- Se aplicarán las reglas de la “programación segura”, mediante el tratamiento de excepciones.

RNF 3. Requisitos de Rendimiento

- La BD deberá resistir un gran número de operaciones simultáneas sobre la misma.
- El diseño de la propuesta debe permitir respuestas a las peticiones del usuario en un tiempo relativamente rápido.

RNF 4. Requisitos Software

- Se requiere un Servidor Web Apache en su versión 2. X con PHP versión 5.X.
- SGDB PostgreSQL.
- Navegador Web Opera, Internet Explorer 6 o superior, Safari, Netscape, Chrome o Mozilla Firefox.

- Se requiere complemento Adobe Flash Player instalado en el navegador.

RNF 5. Requisitos de Diseño e Implementación.

- El diseño de la arquitectura debe ser flexible.
- El diseño debe permitir la fácil integración o desintegración de componentes.
- Se debe hacer uso de los patrones arquitectónicos.
- La arquitectura debe soportar migrar a distintas fuentes de almacenamiento de datos de forma rápida.

RNF 6. Requisitos de Portabilidad

- Las herramientas y el diseño de la arquitectura propuesto debe permitir que el software que se desarrolle usando esta arquitectura pueda ejecutarse sobre cualquier plataforma.

2.4 Descripción de la Arquitectura de la Capa de Acceso a Datos

Los componentes de la *DAL* proporcionan la funcionalidad de acceder a diferentes fuentes de datos, éstas pueden estar dentro de las fronteras del sistema que se desee construir ya sea almacenados en una BD o en un fichero, pero también pueden encontrarse fuera de las fronteras de éste, como por ejemplo a un Servicios Web o RSS que ofrezcan sistemas externos al dominio de la aplicación.

La estructura de la *DAL* será descrita a través del modelo 4+1 vista que propone RUP (figura 5), este modelo describe la AS usando cinco vistas concurrentes, donde cada vista se refiere a un conjunto de intereses de los diferentes *stakeholders* del sistema permitiendo así una visión más detallada de la arquitectura del sistema.

Dado que la capa de Acceso a Datos forma parte de una arquitectura en 3 Capas se hace necesario definir como interactúa ésta con las capas superiores, específicamente con la capa Lógica de Negocio, además se debe describir la estructura lógica de la propuesta de solución. En la figura 8 se puede observar que la interacción con la capa superior se realiza a través de las clases o componentes que se encuentran dentro del paquete *Controladores Acceso a Datos.* Para darle respuesta a las peticiones de la capa superior se hace uso de las clases o componentes que se encuentran dentro de los paquetes *Controladores BD, Controladores Ficheros* y *Controladores Servicios Web.* Los paquetes antes mencionados son los encargados de acceder a fuentes de datos específicas para responder las peticiones de la capa superior.

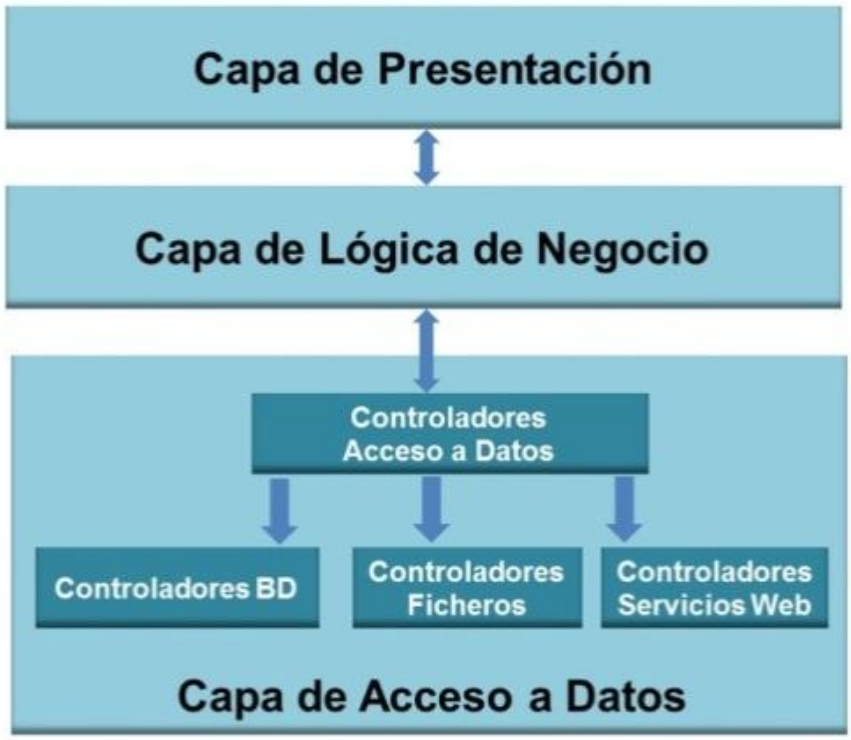

Figura 8: Interacción entre las capas

2.4.1 Vista de Casos de Uso

En esta vista se representa los actores y los casos de usos arquitectónicamente significativos para una RIA que incluya escenarios tridimensionales interactivos. A continuación se representa esta vista:

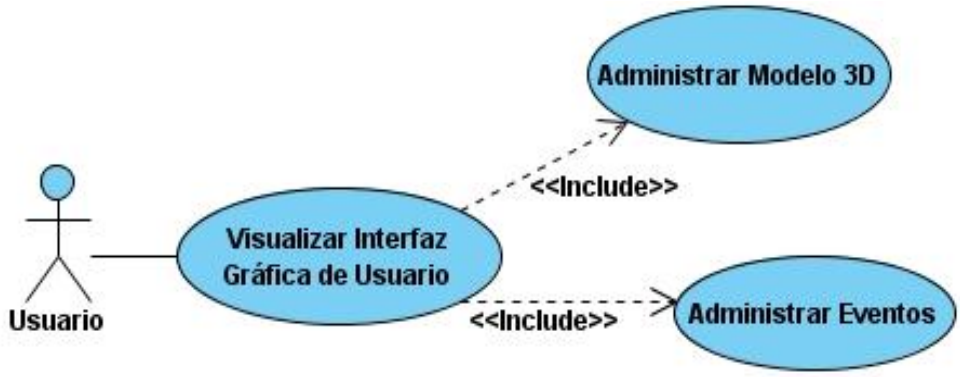

Figura 9: Vista de Casos de Uso.

A continuación se describen los casos de uso:

Tabla 1: Descripción CU Visualizar Interfaz Gráfica de Usuario

<table>
<tr><td>Caso de Uso:</td><td colspan="3">Visualizar Interfaz Gráfica de Usuario (GUI)</td></tr>
<tr><td>Actores:</td><td colspan="3">Usuario</td></tr>
<tr><td>Resumen:</td><td colspan="3">El caso de uso se inicia cuando el usuario solicita acceder a la aplicación o cuando realiza alguna acción que modifica la GUI de la aplicación.</td></tr>
<tr><td>Precondiciones:</td><td colspan="3"></td></tr>
<tr><td>Referencias</td><td colspan="3">RF 1, RF 1.1, RF 1.2, RF 1.3</td></tr>
<tr><td>Prioridad</td><td colspan="3">Crítico</td></tr>
<tr><td colspan="4">Flujo Normal de Eventos</td></tr>
<tr><td colspan="2">Acción del Actor</td><td colspan="2">Respuesta del Sistema</td></tr>
<tr><td colspan="2">1. El usuario solicita acceder a una aplicación web.</td><td colspan="2">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.</td></tr>
<tr><td colspan="4">Flujos Alternos</td></tr>
<tr><td colspan="2">Acción del Actor</td><td colspan="2">Respuesta del Sistema</td></tr>
<tr><td colspan="2">2. Solicita modificar la GUI de la aplicación web.</td><td colspan="2">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.</td></tr>
<tr><td rowspan="2">Relaciones</td><td>CU Incluidos</td><td colspan="2">a) Administrar Modelos 3D
b) Administrar Eventos</td></tr>
<tr><td>CU Extendidos</td><td colspan="2"></td></tr>
<tr><td>Poscondiciones</td><td colspan="3">Se visualiza la GUI de la aplicación Web</td></tr>
</table>

Tabla 2: Descripción del CU Administrar Modelo 3D

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 GUI.	
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 la GUI de la aplicación Web.	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 carga un modelo 3D con las texturas, los materiales y las colisiones definidas.	

Tabla 3: Descripción CU Administrar Eventos

Caso de Uso:	Administrar Eventos
Actores:	Usuario

Resumen:	El caso de uso se inicia cuando el usuario realiza algún evento a través de un dispositivo de entrada.	
Precondiciones:		
Referencias	**RF 3**	
Prioridad	Crítico	
Flujo Normal de Eventos		
Acción del Actor	**Respuesta del Sistema**	
	1. El sistema espera un evento realizado por el usuario a través de dispositivo de entrada.	
2. El usuario al interactuar con la escena realiza un evento.		
	3. Dependiendo del evento realizado el sistema realiza una acción.	
Flujos Alternos		
Acción del Actor	**Respuesta del Sistema**	
Relaciones	CU Incluidos	
	CU Extendidos	
Poscondiciones	El sistema después de captar el evento realiza una acción.	

2.4.2 Vista lógica

En esta vista se representan los elementos de diseño más importantes para la arquitectura del sistema. Se describen las clases más importantes, su organización en paquetes y/o subsistemas, así como la relación que existen entre ellos y la organización de estos a su vez en capas.

A continuación se representa la organización en capas de la arquitectura:

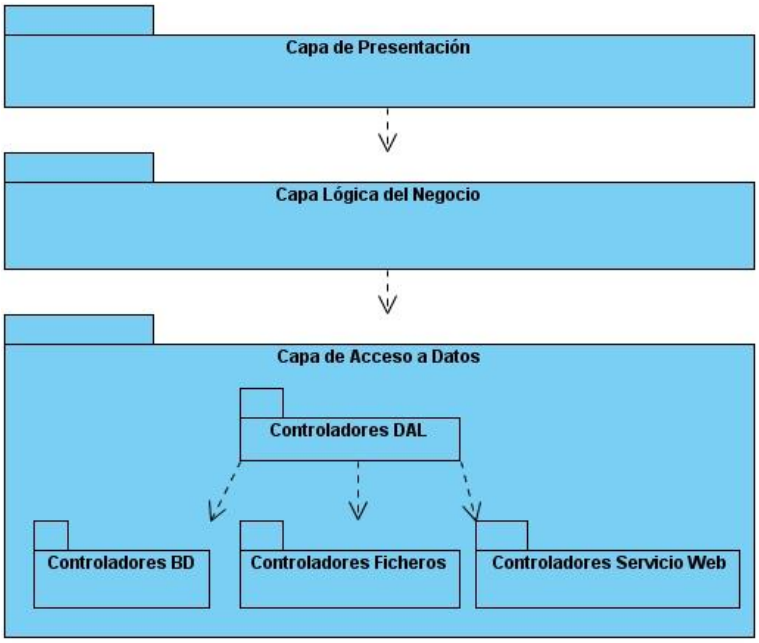

Figura 10: Vista Lógica

La capa de Acceso a Datos está compuesta por 4 paquetes:

Controladores DAL

En este paquete se encapsulan las clases encargadas de recibir las peticiones de la capa Lógica de Negocio, es además la encargada de controlar toda la lógica de la *DAL* y solicitar a los demás paquetes la información que se necesita enviar para dar respuesta a las solicitudes de la capa superior. Dependiendo de lo complicado que sea la lógica de esta capa en el software que se desee desarrollar se puede tener más de una clase para dividir responsabilidades y el diseño sea más desacoplado.

Controladores BD

En este paquete se encapsulan las clases requeridas para acceder y gestionar todos los datos almacenados en la BD, debe permitir adicionar, eliminar, modificar y leer datos de ésta. Estas clases deben estar diseñadas de tal forma que si cambia el SGBD se deban afectar la menor cantidad de clases posibles, esto permitiría un fácil mantenimiento y reutilización de código para otros proyectos. Dado que el lenguaje Action Scritp no puede conectarse directamente a la DB y realizar consultas a ésta, se utiliza para realizar esta conexión el lenguaje PHP. Los datos obtenidos por este lenguaje son serializados por el framework AMFPHP y enviados a las Action Script de forma que este lenguaje pueda entenderlo.

Controladores fichero

En este paquete se encuentra las clases encargadas de crear, modificar, eliminar, leer o escribir sobre archivos de tipos XML. Dependiendo de la cantidad de archivos o información con que trabaje el sistema que se desee construir es recomendable que se utilicen varias clases que se repartan las responsabilidades, esto permite lograr un mejor desacoplamiento entre clases y mantenimiento del sistema.

Controladores Servicios Web

En este paquete se encuentra las clases que tiene como funcionalidad acceder a datos proporcionados por un servicio distribuido externo por ejemplo un Servicio-Web o RSS brindado por otra aplicación fuera del dominio. Se puede hacer uso de varias clases si es necesario, dependiendo de la cantidad de información que se necesite o el manejo que se debe hacer con la información adquirida.

2.4.3 Vista de implementación

La vista de implementación ilustra la descomposición del software en componentes, subsistemas de implementación y las dependencias que se establecen entre ellos. Un componente de software es una parte física de un sistema como puede ser un módulo, una BD, un programa ejecutable, varias clases en un mismo archivo, entre otros. A continuación se presenta el diagrama de componentes propuesto.

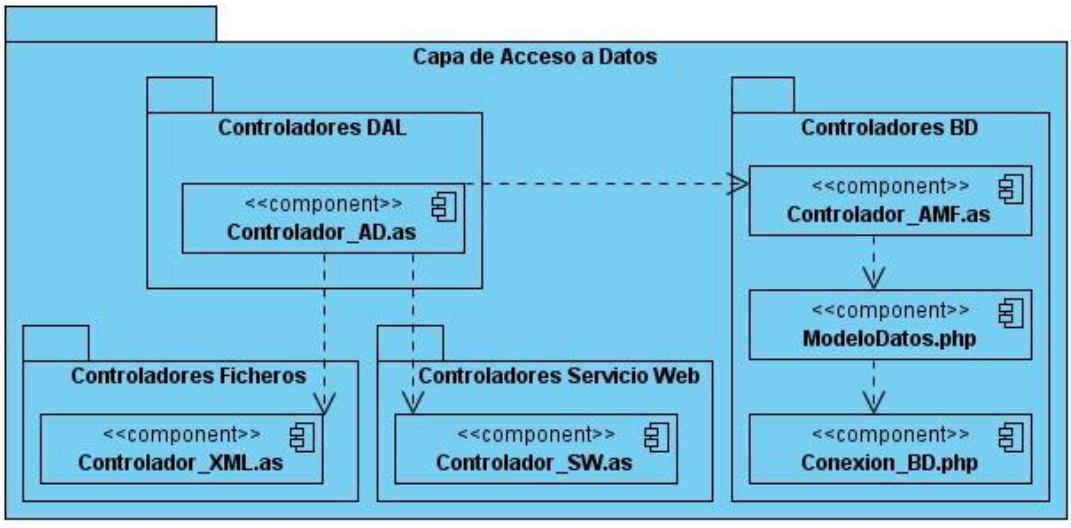

Figura 11: Vista de Implementación

Controlador_AD.as: en este componente se encuentra la declaración e implementación de una o varias clases en lenguaje Action Script, por lo tanto, es un archivo de tipo .AS. Estas clases son las encargadas de atender las solicitudes de la capa Lógica de Negocio, a su vez éstas le solicitan al componente que maneja los datos solicitados por la capa superior que realice la acción requerida y le devuelva un resultado que se envía de regreso a la capa Lógica de Negocio para responder su petición.

Controlador_XML.as: en este componente se encuentra la o las clases escrita en lenguaje Action Script que atienden las peticiones que le solicite el componente Controlador_AD.as, estas peticiones pueden ser crear, eliminar, modificar, leer y escribir sobre un fichero de tipo .XML.

Controlar_SW.as: en este componente se declaran la clase que tiene como responsabilidad conectarse a un Servicio Web proporcionado por otra aplicación, debe por lo tanto declarar los métodos y atributos para dar cumplimiento a dicha funcionalidad. Las solicitudes a este componente son realizas por el componente Controlador_AD.as y a éste se le envían las respuestas de la consulta. Se recomienda el uso de varias clases si las peticiones que se le soliciten al componente son muy complejas, esto permite tener un mayor desacoplamiento entre clases.

Controlador_AMF.as: este componente es el encargado de solicitarle información almacenada en la BD al componente ModeloDato.php que se encuentra dentro del framework AMFPHP, la respuesta que el componente escrito en PHP le devuelve es almacenada en variables Action Script, para que el componente Controlador_AD.as pueda hacer uso de esos datos. Por tanto, el componente Controlador_AMF.as debe implementar los métodos que se conecten a través de procedimientos remotos a la clase PHP que están dentro del AMFPHP y también debe implementar métodos que en caso de que la conexión no se realice devuelvan un mensaje de error, además de otros métodos si son necesarios por la complejidad de la aplicación a desarrollar.

ModeloDatos.php: este componente se encuentra dentro del framework AMFPHP, en él se encapsulan clases escritas en lenguaje PHP. Estas clases implementan métodos que incluyen consultas SQL para obtener información de la BD, estos datos son retornados al componente Controlador_AMF.as a través del AMFPHP. El componente ModeloDatos.php utiliza Conexion_BD.php para acceder a la BD ya que éste es el que tiene los atributos y métodos que permiten conectarse a la BD, si en algún momento se tiene que

cambiar el SGBD solo se requiere modificar este último componente mencionado y hacerle unas modificaciones a las consultas SQL que están en el componente ModeloDatos.php.

Conexion_BD.php: este componente tiene una clase escrita en PHP que es la encargada de abrir la conexión con la BD, debe tener métodos que permitan guardar en variables o arreglos información proveniente de una consulta a la BD y también debe permitir cerrar la conexión con la BD cuando se termine de adquirir la información que se necesite. La información proveniente de la BD es enviada al componente ModeloDatos.php para que sea procesada y enviada al componente controlador de la capa.

Después de haber descrito brevemente los componentes ilustrados en la figura 10 se presenta un diagrama de clases en la figura 12 que cumple con las funcionalidades básicas de acceso a datos que puede tener una aplicación RIA que incluya escenarios tridimensionales interactivos. Este diagrama de clase fue diseñado haciendo uso de los patrones de diseños seleccionados en el epígrafe 2.2.1.

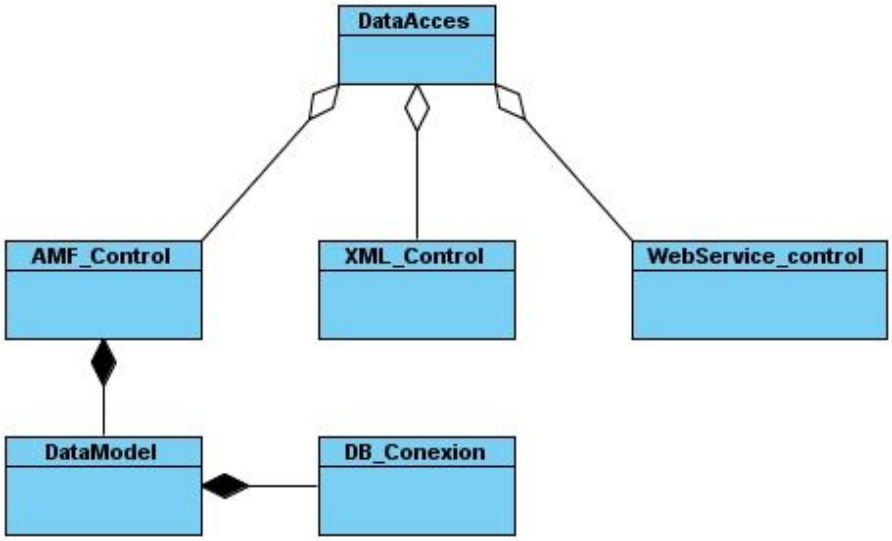

Figura 12: Diagrama de clases

Las clases propuestas en el diagrama solo cumplen una función, por ejemplo la clase DataModel solo implementa las consultas SQL mientras que la conexión a la BD la realiza BD_Conexion, posibilitando así

una Alta Cohesión entre clases, este diseño además permite un bajo Acoplamiento entre éstas, pues si cambia el SGBD solo hay que modificar la clase DB_Conexion.

El patrón Experto se puede ver en las clases XML_Control y WebService_Control pues estas son las expertas en un tipo de fuente de almacenamiento específica, como son los XML y los Servicios Web respectivamente, por tanto, cuando se desee realizar una operación que sea necesario trabajar con estas fuentes de datos ellas son las encargadas de implementarlas.

La clase DataAcces por su parte utiliza el patrón Singleton, Creador y Controlador para crear instancias únicas de las clases AMF_Control, XML_Control, WebService_Control y además es la que controla toda la lógica de la capa de Acceso a Datos. Las respuestas a las peticiones que se le realicen a esta clase son manejadas a través de la instancia única definida para responder a funciones determina. La utilización de estos patrones posibilita que si se necesita agregar nuevas funcionalidades o utilizar este diseño para otra aplicación se pueda realizar de la forma más rápida y entendible posible.

2.4.4 Vista de despliegue

La vista de despliegue muestra un conjunto de nodos unidos por conexiones de comunicación. Esta vista suministra además una base para comprender cómo queda la distribución física del sistema y cómo estarán distribuidos los componentes de la aplicación en ella. Para realizar el despliegue de la aplicación se requiere de un servidor Web Apache. Se deberá contar también con un servidor de BD, donde se estará ejecutando el sistema SGBD PostgresSQL y se requiere además que las PC Clientes tengan un navegador Web y el complemento Flash Player instalado. El diseño propuesto de *DAL* se debe encontrar en el nodo Servidor Web y mediante el uso del protocolo TCP\IP se conecta al nodo Servidor BD para obtener la información que necesite.

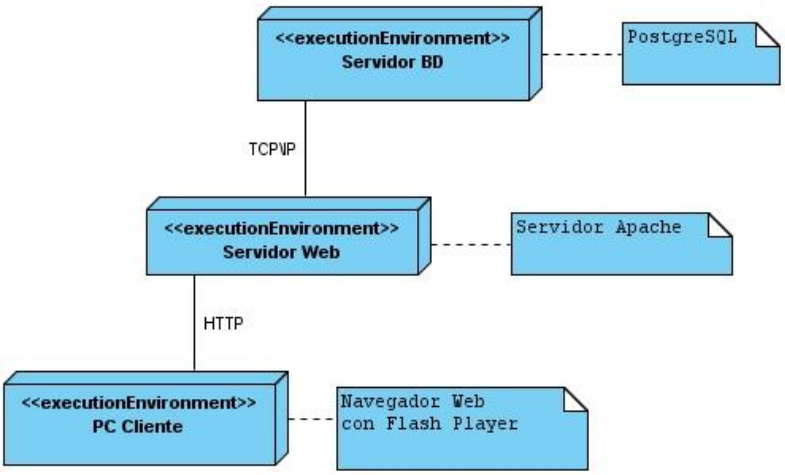

Figura 13: Vista de despliegue

2.4.5 Vista de Procesos

Esta vista suministra una base para la comprensión de la organización de los procesos de un sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Solo suele usarse cuando el sistema presenta procesos concurrentes o hilos. En este caso la solución propuesta no presenta procesos concurrentes por tanto no se hace representación de esta vista.

Conclusiones del capítulo

En este capítulo se realizó la propuesta de herramientas de desarrollo y de modelado que pueden ser utilizadas para desarrollar la DAL de una RIA. Para realizar la selección de las herramientas se tuvo como criterio que éstas debían ser libres, fáciles de utilizar y compatible en varias plataformas. Se recomienda también la utilización de varios patrones de diseño que pueden ser útiles en el desarrollo de la capa. Se realizó en este capítulo además la especificación de los requerimientos funcionales y no funcionales. La DAL fue descrita a través de las 4+1 que propone RUP, de esta forma, se tiene una representación visual de cómo quedarán los componentes que integran la solución y las relaciones que existen entre ellos.

3 Capítulo 3: Validación de la propuesta de solución.

Introducción

En el presente capítulo se aborda sobre la evaluación de la arquitectura, enfatizando en por qué se hace necesario su realización y los beneficios que brinda. Se estudian además las técnicas, métodos y atributos de calidad con el objetivo de seleccionar los que serán usados para realizar la evaluación de la solución propuesta.

3.1 Evaluación de la Arquitectura de Software

La evaluación es un estudio de factibilidad que pretende detectar posibles riesgos, así como buscar recomendaciones para contenerlos. El objetivo de evaluar una arquitectura es saber si puede habilitar los requerimientos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los *stakeholders* (Brey, et al., 2005).

Debido a lo importante que es para el desarrollo de software tener una arquitectura que logre satisfacer los requerimientos no funcionales y de calidad es conveniente realizar actividades de verificación de la misma, con el fin de identificar posibles problemas que podría resultar muy costoso de eliminar posteriormente. Según (Kazman, et al., 2001) el primer paso para la evaluación de una arquitectura es conocer qué es lo que se quiere evaluar. De esta forma, es posible establecer la base para la evaluación, puesto que la intención es saber qué se puede evaluar y qué no. Esta evaluación no define si una arquitectura es buena o mala, solo identifica donde están los riesgos, las fortalezas y debilidades.

La evaluación de una AS es una tarea no trivial, puesto que se pretende medir propiedades del sistema en base a especificaciones abstractas, como por ejemplo los diseños arquitectónicos. Por ello, la intención es más bien la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos. Las mediciones que se realizan sobre una AS pueden tener distintos objetivos, dependiendo de la situación en la que se encuentre el arquitecto y la aplicabilidad de las técnicas que emplea (Bosch, 2000).

La arquitectura puede ser evaluada en dos momentos, (Kazman, et al., 2001) define las siguientes variantes: la *evaluación temprana,* para su realización no es necesario que se haya especificado

completamente la arquitectura, ya que esta evaluación abarca desde las fases tempranas de diseño y a lo largo del desarrollo; la *evaluación tardía* se realiza antes del comienzo de la implementación cuando ya fue especificada completamente la arquitectura.

3.2 Atributos de calidad

Los atributos de calidad hacen referencia a requerimientos adicionales que brinda el sistema para lograr una mayor satisfacción del cliente, estos definen las propiedades de los servicios brindados. La calidad del software está dada por el grado de combinación que puedan tener sus atributos. Según un estudio plasmando en el libro “Arquitecturas de Software” por (Camacho, et al., 2004) los atributos están divididos en dos grupos:

Observables vía ejecución: aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución, estos son descritos en el anexo 2.

No observables vía ejecución: aquellos atributos que se establecen durante el desarrollo del sistema, estos son descritos en el anexo 3.

En su mayoría, los atributos de calidad se pueden organizar y descomponer de maneras diferentes, en lo que se conoce como modelos de calidad. Los modelos de calidad de software facilitan el entendimiento del proceso de la ingeniería de software. Pressman indica que los factores que afectan a la calidad del software no cambian, por lo que resulta útil el estudio de los modelos de calidad que han sido propuestos en este sentido desde los años 70. Entre los modelos de calidad más importante se encuentra el establecido por el estándar ISO/IEC 9126 en el año 1991. El estándar ISO/IEC 9126 ha sido desarrollado en un intento de identificar los atributos clave de calidad para un producto de software. Este estándar es una simplificación del Modelo de McCall del año 1977, e identifica seis características básicas de calidad que pueden estar presentes en cualquier producto de software (Pressman, 2002), estas características y subcaracterísticas de calidad son mostradas en el anexo 4.

3.3 Técnicas de evaluación de la Arquitectura de Software

Existen varias técnicas para evaluar las arquitecturas, éstas se dividen en *cualitativas* y en *cuantitativas,* como se puede observar en la figura 14 dentro de estos dos grupos se encuentran varias técnicas, que

dependiendo del momento en que se realiza la evaluación y el tipo de resultado que se espera se debe escoger la más acorde a lo que se desea. 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.

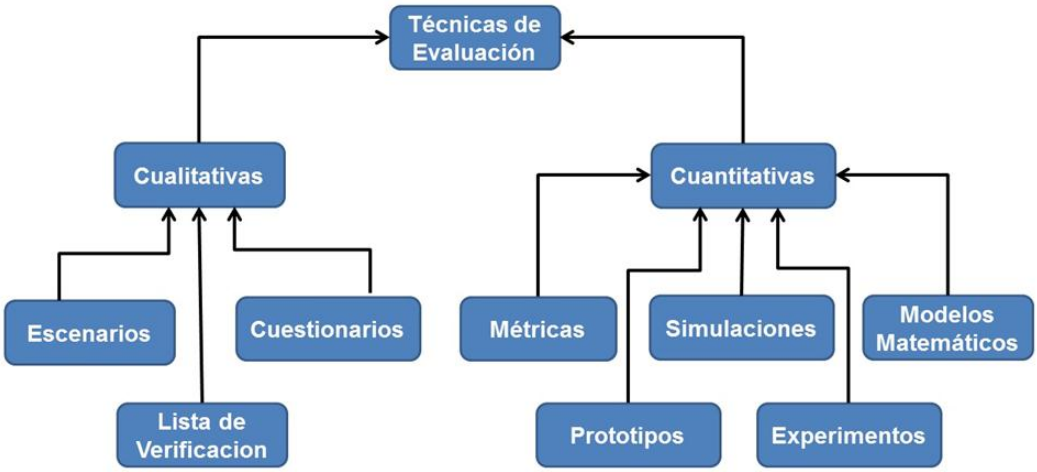

Figura 14: Técnicas de Evaluación de Arquitecturas.

3.3.1 Evaluación Basada en Escenarios

Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con el propio sistema. Un escenario consta de tres partes: el estímulo, el contexto y la respuesta. El estímulo es la parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. Puede incluir ejecución de tareas, cambios en el sistema, ejecución de pruebas, configuración, etc. El contexto describe qué sucede en el sistema al momento del estímulo. La respuesta describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado. Los escenarios proveen un vehículo que permite concretar y entender atributos de calidad (Kazman, et al., 2001).

Actualmente las técnicas basadas en escenarios cuentan con dos instrumentos de evaluación relevantes: el Utility Tree propuesto por (Kazman, et al., 2001) y los Profiles, propuestos por (Bosch, 2000).

Utility Tree

Es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle, el nivel de prioridad de cada uno. La intención del uso del Utility Tree es la identificación de los atributos de calidad más importantes para un proyecto particular. No existe un conjunto preestablecido de atributos, sino que son definidos por los involucrados en el desarrollo del sistema al momento de la construcción del árbol. El Utility Tree contiene como nodo raíz la utilidad general del sistema. Los atributos de calidad asociados al mismo conforman el segundo nivel del árbol los cuales se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado y otorgarle prioridad a cada atributo considerado. Cada atributo de calidad perteneciente al árbol contiene una serie de escenarios relacionados y una escala de importancia y dificultad asociada, que será útil para efectos de la evaluación de la arquitectura.

Profiles

Es un conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de ellos. El uso de Perfiles (Profiles en inglés) permite hacer especificaciones más precisas del requerimiento para un atributo de calidad. Para la definición de un perfil es necesario seguir tres pasos: definición de las categorías del escenario, selección y definición de los escenarios para la categoría y asignación del peso a los escenarios. Cada atributo de calidad tiene un perfil asociado. Algunos perfiles pueden ser usados para evaluar más de un atributo de calidad. Esta técnica de evaluación basada en escenarios es dependiente de manera directa del perfil definido para el atributo de calidad que va a ser evaluado. La efectividad de la técnica es altamente dependiente de la representatividad de los escenarios.

3.3.2 Evaluación basada en simulación

La evaluación basada en simulación utiliza una implementación de alto nivel de la AS. El enfoque básico consiste en la implementación de componentes de la arquitectura y la implementación a cierto nivel de abstracción del contexto del sistema donde se supone va a ejecutarse. La finalidad es evaluar el comportamiento de la arquitectura bajo diversas circunstancias. Una vez disponibles estas implementaciones, pueden usarse los perfiles respectivos para evaluar los atributos de calidad (Bosch, 2000).

3.3.3 Evaluación basada en modelos matemáticos

La evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. Permite una evaluación estática de los modelos de diseño arquitectónico y se presenta como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados utilizando los resultados de uno como entrada para el otro. Entre las desventajas que presenta esta técnica se encuentra la inexistencia de modelos matemáticos apropiados para los atributos de calidad relevantes y el hecho de que el desarrollo de un modelo de simulación completo puede requerir esfuerzos sustanciales (Bosch, 2000).

3.4 Métodos de evaluación de Arquitecturas de Software

De los planteamientos de evaluación establecidos por (Bosch, 2000) y (Kazman, et al., 2001), se tiene que la evaluación de las AS puede ser realizada mediante el uso de diversas técnicas y métodos. En este sentido, resulta interesante estudiar las distintas opciones que existen en la actualidad para llevar a cabo esta tarea. Un estudio realizado por Kazman y sus colegas presenta tres métodos de evaluación, estos son: *Architecture Trade-off Analysis Method (ATAM)*, *Software Architecture Analysis Method (SAAM)* y *Active Intermediate Designs Review (ARID)*.

3.4.1 Architecture Trade-off Analysis Method (ATAM)

Este método está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM. El nombre del método ATAM surge del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos de calidad y provee una visión de cómo los atributos de calidad interactúan con otros. Los análisis de atributos de calidad específicos deben realizarse teniendo en cuenta su dependencia con el resto de los atributos del sistema. Los atributos de calidad no son independientes, sino que interaccionan entre sí a través de las relaciones estructurales impuestas por la arquitectura (Camacho, et al., 2004).

El método de evaluación ATAM comprende nueve pasos, agrupados en cuatro fases:

Fase 1: Presentación.
Paso 1-Presentar el ATAM.

Paso 2-Controladores de negocios actual.

Paso 3-Presentar arquitectura.

Fase 2: Investigación y análisis.

Paso 4-Identificar los enfoques de arquitectura.

Paso 5-Generar árbol de utilidad de los atributos de calidad.

Paso 6-Analizar enfoques de arquitectura.

Fase 3: Prueba.

Paso 7-LLuvia de ideas y la prioridad de escenarios.

Paso 8-Analizar enfoques de arquitectura.

Fase 4: Reporte

Paso 9-Presentar los resultados.

3.4.2 Software Architecture Analysis Method (SAAM)

Este método fue el primero ampliamente promulgado y documentado. El método fue originalmente creado para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad. Se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro. Como entrada principal, es necesaria alguna forma de descripción de la arquitectura a ser evaluada (Camacho, et al., 2004).

Las salidas de la evaluación del método SAAM son las siguientes:

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

El método consta de seis pasos principales los cuales son:

Paso 1 - Desarrollar escenarios.

Paso 2 - Describir la arquitectura.

Paso 3 - Clasificar y priorizar escenarios.

Paso 4 - Evaluar individualmente escenarios indirectos.

Paso 5 - Evaluar la interacción de los escenarios.

Paso 6 - Crear una evaluación global.

3.4.3 Active Intermediate Designs Review (ARID)

ARID es un método de bajo costo y gran beneficio, el mismo es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. Hacer revisiones en las etapas intermedias provee una valiosa visión de la viabilidad de la arquitectura a construir, además de que permite descubrir errores e inconsistencias en la misma desde el punto de vista de otras partes de la arquitectura.

ARID es un híbrido entre Active Design Review (ADR) y ATAM; del primer método toma la participación activa de los entrevistados, ya que los involucrados reciben documentación detallada y completan cuestionarios, cada uno por separado; del segundo método se queda con la idea de la generación de los escenarios por parte de todos los interesados (*stakeholders*), los usuarios le dirán a los diseñadores que es lo que ellos necesitan o que es lo que ellos esperan para que los diseñadores puedan demostrar en las pruebas que el diseño cumple con los requerimientos. 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* (Clements, 2000).

Este método consta de 9 pasos agrupados en dos fases, a continuación se describen:

Tabla 4: Pasos del método de evaluación ARID

Fase 1: Actividades Previas	
Identificación de los encargados de la revisión	Ingenieros de software que van a usar el diseño.
Preparar el informe de diseño	El arquitecto prepara un informe que explica el diseño, el mismo deberá ser lo suficientemente detallado como para que una capacitada audiencia pueda usar el diseño.
Preparar los escenarios base	El arquitecto y el líder preparan los escenarios base que sirven para ilustrar los conceptos a los revisores.
Preparar los materiales	Copias de las presentaciones, escenarios.
Fase 2: Revisión	

Presentación del ARID	El líder utiliza 30 minutos para explicar los pasos de la evaluación a los participantes.
Presentación del diseño	El arquitecto realiza una presentación del diseño mostrando los ejemplos.
Lluvia de ideas y establecimiento de prioridad de escenarios	Se proponen escenarios relevantes para solucionar problemas.
Aplicación de los escenarios	El líder del grupo de revisión es el encargado de probar que el diseño provee los escenarios, comenzando por los de mayor prioridad, hasta que concluya el tiempo estimado de la evaluación, se hayan analizado los escenarios de mayor prioridad, o la conclusión de la revisión satisfaga a los implicados.
Resumen	Al final, el facilitador recuenta la lista de puntos tratados, pide opiniones de los participantes sobre la eficiencia del ejercicio de revisión y agradece por su participación.

3.5 Evaluación de la propuesta de solución.

Con el objetivo de evaluar la arquitectura propuesta y después de estudiar diferentes métodos y técnicas de evaluación se decide utilizar el método *ARID* junto a la técnica basada en Escenarios con el instrumento Perfiles, en el anexo 5 se puede observar una comparación entre los métodos presentado. Entre las ventajas de este método se nombran su facilidad de uso, su aplicación poco costosa y el gran beneficio que brinda para realizar una evaluación de la factibilidad de la arquitectura.

Para realizar la evaluación se seleccionaron los atributos de calidad más importantes en aquellos escenarios de acuerdo con un perfil específico. Los atributos que se analizarán para esta estrategia de evaluación temprana son algunos de los propuestos por el estándar ISO 9126, fueron seleccionados según el equipo de arquitectos los de mayor importancia para realizar la evaluación de la propuesta de solución, estos son: Modificabilidad, Portabilidad, Eficiencia y Funcionalidad. El atributo Confiabilidad no fue evaluado pues no se puede medir el nivel de madurez, la tolerancia a fallos y la recuperación ante estos porque no ha sido implementada la capa, por este mismo motivo no se puede evaluar el esfuerzo que deberá invertir el usuario para utilizar el sistema que describe el atributo Usabilidad.

Debido a 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.

A continuación se muestran diferentes tablas que resumen los pasos finales de la fase última del método ARID formado por: el establecimiento de prioridad de escenarios, la aplicación de estos y el resumen de la relación entre los atributos de calidad y dichos escenarios. Estos escenarios han sido escogidos por ser primordiales para evaluar algunos de los atributos escogidos del estándar ISO 9126.

Tabla 5: Migración del Sistema Gestor de Base de Datos

Escenario # 1	Migración del Sistema Gestor de Base de Datos
Atributos de Calidad	Mantenibilidad
Perfil	Modificabilidad
Relación Atributo-Escenario	
El diseño de clases propuesto está creado de tal manera, que si se hace necesario cambiar el SGBD solo habría que configurar algunos parámetros en la clase DB_Conexion y modificar algunas consultas SQL que son implementadas en la clase DataModel. El acceso a la BD le es transparente a las demás clases del diseño propuesto, facilitando así que se tenga que realizar un mínimo esfuerzo para adaptar el software a un ambiente diferente.	

Tabla 6: Migración de Sistema Operativo

Escenario # 2	Migración de Sistema Operativo
Atributos de Calidad	Portabilidad
Perfil	Adaptabilidad

Relación Atributo-Escenario
Para el desarrollo de la capa de Acceso a Datos se propuso como lenguajes de programación PHP, Action Script y XML, como servidor Web Apache y como SGBD PostgreSQL. Cada uno de estos elementos para el desarrollo de la capa son compatibles con la mayoría de sistemas operativos más usados en la actualidad. Por esto se puede concluir que la capa se puede adaptar a diferentes ambientes sin la necesidad de aplicarle modificaciones.

Tabla 7: Aumento del modelo de datos

Escenario # 3	Aumento del modelo de datos
Atributos de Calidad	Eficiencia
Perfil	Comportamiento con respecto a Recursos
Relación Atributo-Escenario	
El SGBD PostgreSQL brinda la posibilidad de crear tantos clústeres de bases de datos como sea necesario. Esto permite aumentar la capacidad de almacenamiento y disminuir la sobrecarga del proceso servidor en caso que la concurrencia o el espacio sean insuficientes. Por tanto, el aumento del modelo de dato no dificulta el funcionamiento normal del sistema.	

Tabla 8: Realizar una consulta a una fuente de almacenamiento de datos

Escenario # 4	Realizar una consulta a una fuente de almacenamiento de datos
Atributos de Calidad	Funcionalidad
Perfil	Adecuación
Relación Atributo-Escenario	

El diseño propuesto de la capa de Acceso a Datos posee una estructura de clases, donde el acceso a la información que está almacenada en alguna fuente es realizado por una clase en específico o varias que cumplan con esta función, en dependencia de lo complejo que pueda ser acceder a la información de una fuente determinada. La propuesta de clases fue prevista para tres tipos de fuentes de almacenamiento específicas, pero se puede acceder a tantas como se necesite, solo será necesario agregar una nueva clase que pueda manejar la nueva fuente de datos. Esto es posible porque el diseño propuesto es flexible.

Tabla 9: Interacción con otro sistema

Escenario # 5	Interacción con otro sistema
Atributos de Calidad	Funcionalidad
Perfil	Interoperabilidad.
Relación Atributo-Escenario	
La capa de Acceso a Dato propuesta permite intercambiar información con otro sistema a través de Servicios Web. Esta funcionalidad fue tenida en cuenta cuando fue diseñado el diagrama de clase, pues hay una clase que es la que se encarga de permitir este intercambio de datos.	

La evaluación cualitativa realizada refleja que el diseño propuesto cumple con los atributos de calidad de la norma ISO 9126 que fueron evaluados. Se demuestra además que la utilización de los patrones de diseño seleccionados para guiar el diseño de clases propuesto dan cumplimiento a los atributos de calidad de Mantenibilidad y Funcionalidad, mientras que las tecnologías propuestas satisfacen los atributos de Portabilidad y Eficiencia. A su vez se arrojan resultados que evidencian la resolución de la situación problemática que da vida a esta investigación. Por lo anteriormente planteado se puede afirmar que el diseño de la arquitectura propuesta, dentro de una etapa temprana de desarrollo, es adecuado para desarrollar la capa de Acceso a Datos de una RIA que incluya escenarios tridimensionales interactivos.

Conclusiones del capítulo

En el presente capítulo se han analizado los elementos fundamentales relacionados con la evaluación de la AS, entre estos, se estudiaron los principales métodos y técnicas que se pueden utilizar en la evaluación, los atributos de calidad por los que se puede evaluar y las etapas en que se puede realizar una evaluación. Este estudio posibilitó que se pudiera seleccionar los métodos, técnicas y atributos para evaluar la propuesta de solución. Se obtuvo un resultado satisfactorio en la evaluación pues la arquitectura cumple con cada uno de los escenarios que se definieron.

Conclusiones

Luego de culminar el diseño de la capa de Acceso a Datos de una Arquitectura de Software de Dominio Específico se puede arribar a las siguientes conclusiones:

- Se realizó un estudio de los conceptos de Arquitectura de Software, Arquitectura de Software de Dominio Específico y Aplicaciones Enriquecidas de Internet. Se describe y se seleccionan además diferentes estilos arquitectónicos y patrones de diseño. Se hizo también un análisis de las principales tecnologías y herramientas para el desarrollo de RIA con el fin de seleccionar las más convenientes para la realización del presenta trabajo de diploma.
- Se propone una arquitectura de referencia de la capa de Acceso a Datos para construir RIA dentro de un dominio específico.
- Se describió y se seleccionó las técnicas, métodos y atributos de calidad que fueron usados para realizar la evaluación de la arquitectura, arrojando que la propuesta de capa de Acceso a Datos cumple con los atributos de calidad de la norma ISO 9126 que fueron evaluados en diferentes escenarios.

Recomendaciones

- Utilizar la arquitectura propuesta de la capa de Acceso a Datos para desarrollar RIA que incluyan escenarios tridimensionales interactivos y necesiten trabajar con datos almacenados en alguna fuente.
- En etapas posteriores, cuando la arquitectura haya sido implantada, realizar la evaluación tardía de la capa de Acceso a Datos propuesta, garantizando así, la obtención de un resultado completo y detallado de los riesgos que puedan afectar la propuesta realizada.
- Integrar la capa de Acceso a Datos con las capas de Presentación y Lógica de Negocio para conformar la arquitectura.

Bibliografía

Adobe Corporation. 2011. Adobe. [En línea] 2011. http://www.adobe.com.

AMFPHP. 2011. AMFPHP. [En línea] 2011. http://amfphp.sourceforge.net/about.html.

Apache Foundation. 2011. The Apache Software Foundation. [En línea] 2011. http://www.apache.org/.

Bass, L., Clements, P. y Kazman, R. 1998. *Software Architecture in Practice.* 1998.

Boehm, B. 1976. *Software Engineering.* s.l. : IEEE Transactions on Computers, 1976.

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

Bosch, J. 1998. *Design Patterns as Language Constructs: Journal of Object Oriented Programming(JOOP).* 1998.

Brey, Gustavo Andrés, y otros. 2005. *Evaluación de Arquitecturas.* Universidad Tecnológica Nacional. Facultad Regional de Buenos Aires : s.n., 2005.

Camacho, Erika, Cardeso, Fabio y Nuñez, Gabriel. 2004. *Arquitectura de Software: Guía de Estudio.* 2004.

Clements, Paul C. 2000. *Active Reviews forIntermediate Designs.* 2000.

Fain, Yakov, Rasputnis, Victor y Tartakovsky, Anatole. 2007. *Aplicaciones Ricas de Internet con Adobe Flex y Java.* 2007.

Fundación Eclipse . 2011. Eclipse. [En línea] 2011. http://plataformaeclipse.com/.

Gamma, Erich, y otros. 1998. *Design Patterns - Elements of Reusable Object-Oriented Software.* 1998.

Garlan, David y Shaw, Mary. 1994. *An Introduction to Software Architecture. .* 1994.

Gerrikagoitia, Dr. Jon Kepa. 2009. *Aplicaciones de Internet.* 2009.

Giacomo, Mariella Di. 2005. *MySQL: Lessons Learned on a Digital Library.* s.l. : IEEE Computer Society, 2005.

Gilmore, W. Jason y Treat, Robert H. 2006. *Beginning PHP and PostgreSQL 8: From Novice to Professional.* 2006.

Hayes-Roth, Barbara, y otros. 1995. *A Domain-Specific Software Architecture for Adaptive Intelligent Systems.* s.l. : Computer Science Department, Stanford University, 1995.

Henry, Hector A. 2010. Blinkyit. [En línea] 2010. http://blinkyit.com/ya-estn-disponibles-flash-builder-4-coldfusion-builder-el-servicio-flash-platform-social.

IBM . 2011. IBM. [En línea] 2011. http://www-01.ibm.com/software/rational/.

IEEE. 1993. *IEEE: Standards Collection: Somare Engineering.* 1993. 610.12-1990..

—. 2000. *Recommended Practice for Architecture Description of Software-Intensive Systems.* s.l. : ANSI/IEEE Std 1471-2000, 2000.

Jacobs, Sas y Weggheleire, Koen De. 2008. *Flex for Developers: Data-Driven Applications with PHP, ASP.NET, ColdFusion, and LCDS.* 2008.

KAISER, S. H. 2005. *Software Paradigms.* 2005.

Kazman, Rick, Clements, Paul y Klain, Mark. 2001. *Evaluating Software Architectures. Methods and case studies.* s.l. : Adison-Wesley Professional, 2001. ISSN: 978-0201704822.

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

Kruchten, Philippe y Wesley, Addison. 2000. *The Rational Unified Process: An Introduction. .* 2000. 0-201-70710-1.

López, Xavier Farré. 2005. *Rich Internet Applications.* 2005.

Maceda, Humberto Cervantes. 2008. *Ingeniería de Software.* 2008.

Microsoft Corporation. 2011. Internet Information Services. [En línea] 2011. http://www.iis.net/.

Netcraft. Netcraft. [En línea] http://news.netcraft.com.

Pressman, R. 2002. *Ingeniería del software: un enfoque práctico.* 2002.

Reynoso, Carlos Billy. 2004. *Introducción a la Arquitectura de Software.* BUENOS AIRES : s.n., 2004.

Rivero, José Matías y Buzzo, Hernán Marcos. 2008. *Definición de Rich Internet Applications a través de Modelos de Dominio Específico.* 2008.

Rondón, Yoandri Quintana. 2009. Desarrollo de la arquitectura del proyecto Captura y. 2009.

Sanchez, María A. Mendoza. 2004. Informatizate. [En línea] 7 de junio de 2004. http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html.

SEI. 2011. Software Engineering Institute. [En línea] 2011. http://www.sei.cmu.edu/architecture/.

Sommerville, Ian. 2004. *Ingeniería de Software.* 2004.

Southwell, Michael y Snyder, Chris. 2005. *Pro PHP Security. .* New York : s.n., 2005.

Tracz, Will. 1995. *Domain-Specific Software Architecture.* s.l. : Software Engineering, 1995. vol 20 no 3.

Vizcaíno, Aurora, García, Felix Óscar y Caballero, Ismael. 2005. *Trabajando con Visual Paradigm for UML.* 2005.

Zend Technologies Ltd. 2011. Zend Framework. [En línea] 2011. http://framework.zend.com/.

Zend. 2011. Zend. [En línea] 2011. http://www.zend.com/en.

Anexos

Anexo 1: Comparación entre Servidores Web.

	Apache	Internet Information Server
Sistemas Operativos	Multiplataforma	Windows
Lenguaje que soportan	PHP, Perl, Python, Ruby, Coldfusión, mysql, jsp, SQL, MSsql, asp, asp.net, xml, ajax	ASP, ASP.net, PHP, Perl, vbscript, Ajax, msSql, mysql, xml.
Licencia	Licencia Apache	Software privativo
Por ciento de uso en el mundo	61.13% (abril 2011)	18.83% (abril 2011)
Última versión	2.2.17 (19 de octubre de 2010)	7.5 (2008)
Facilidad de administración	Menor	Mayor

Anexo 2: Descripción de atributos de calidad observables vía ejecución.

Atributos de calidad	Descripción
Disponibilidad	Es la medida de disponibilidad del sistema para el uso.
Confidencialidad	Es la ausencia de acceso no autorizado a la información.
Funcionalidad	Habilidad del sistema para realizar el trabajo para el cual fue concebido.
Desempeño	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones como velocidad, exactitud o uso de memoria.

Confiabilidad	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo.
Seguridad Externa	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información.
Seguridad Interna	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos.

Anexo 3: Descripción de atributos de calidad no observables vía ejecución.

Atributos de calidad	Descripción.
Configurabilidad	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema.
Integrabilidad	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.
Integridad	Es la ausencia de alteraciones inapropiadas de la información.
Interoperabilidad	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de integrabilidad.
Modificabilidad	Es la habilidad de realizar cambios futuros al sistema.
Mantenibilidad	Es la capacidad de someter a un sistema a reparaciones y evolución.
Portabilidad	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.

Reusabilidad	Es la capacidad de diseñar un sistema de forma tal que su estructura o partes de sus componentes puedan ser reutilizados en futuras aplicaciones.
Escalabilidad	Es el grado con el que se puede ampliar el diseño arquitectónico, de datos o procedimental.
Capacidad de prueba	Es la medida 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.

Anexo 4: Características y subcaracterísticas de calidad – Modelo ISO/IEC 9126

Características	Subcaracterísticas
Funcionalidad	✓ Adecuación ✓ Exactitud ✓ Interoperabilidad ✓ Seguridad
Confiabilidad	✓ Madurez ✓ Tolerancia a fallas ✓ Recuperabilidad
Usabilidad	✓ Entendibilidad ✓ Capacidad de aprendizaje ✓ Operabilidad
Eficiencia	✓ Comportamiento en tiempo ✓ Comportamiento de recursos
Mantenibilidad	✓ Analizabilidad ✓ Modificabilidad ✓ Estabilidad ✓ Capacidad de pruebas
Portabilidad	✓ Adaptabilidad ✓ Instalabilidad ✓ Reemplazabilidad

Anexo 5: Comparación entre 3 métodos de evaluación (Pressman, 2002).

	ATAM	SAAM	ARID
Atributos de Calidad contemplados	✓ Modificabilidad. ✓ Seguridad. ✓ Confiabilidad. ✓ Desempeño.	✓ Modificabilidad. ✓ Funcionalidad.	✓ Conveniencia del diseño evaluado.
Objetos analizados	✓ Estilos arquitectónicos. ✓ Documentación. ✓ Flujo de datos Vistas. ✓ Arquitectónicas.	✓ Documentación. ✓ Vistas arquitectónicas.	✓ Especificación de los componentes.
Etapas del proyecto en las que se aplica	✓ Después de que el diseño de la arquitectura ha sido establecido.	✓ Después de que la arquitectura cuenta con funcionalidad ubicada en módulos.	✓ A lo largo del diseño de la arquitectura.
Enfoques utilizados	✓ Utility Tree y lluvia de ideas para articular los requerimientos de calidad. ✓ Análisis arquitectónico que detecta puntos sensibles, puntos de balance y riesgos	✓ Lluvia de ideas para escenarios y articular los requerimientos de calidad. ✓ Análisis de los escenarios para verificar funcionalidad o estimar el costo de los cambios.	✓ Revisiones de diseños, lluvia de ideas para obtener ✓ Escenarios.

Glosario de Términos y Siglas

Términos

Adobe Flash Player: Aplicación en forma de reproductor multimedia creado inicialmente por Macromedia y actualmente distribuido por Adobe System. Permite reproducir archivos SWF.

Aplicación: Tipo de programa informático diseñado como herramienta para permitir a un usuario realizar diversos trabajos.

Aplicaciones Web: Aplicaciones que los usuarios pueden utilizar accediendo a un servidor Web a través de Internet o de una Intranet mediante un navegador.

Artefacto: Es un término general, para cualquier tipo de información creada, producida, cambiada o utilizada por los trabajadores en el desarrollo del sistema.

Calidad: Conjunto de propiedades y características de un producto o servicio que le confieren su aptitud para satisfacer unas necesidades explícitas o implícitas.

Calidad de software: Grado con que el que un sistema, componente o proceso cumple con los requerimientos especificados y las necesidades o expectativas del cliente o usuario.

Clase(s): Abstracciones de objetos. Una clase es una definición de un objeto. Un objeto es una instancia de una clase.

Código Abierto: Término con el que se conoce al software distribuido y desarrollado libremente. Fue utilizado por primera vez en 1998 por algunos usuarios de la comunidad del software libre, tratando de usarlo como reemplazo al ambiguo nombre original en inglés del software libre (*free software*).

Componentes de código: es una parte física y reemplazable del sistema, que cumple y proporciona la realización de un conjunto de interfaces.

Dominio de aplicación: Es un área de interés, usualmente representando un espacio del problema que es un subconjunto de una o más disciplinas.

Escenario: Es el espacio destinado para la representación de un dominio de aplicación.

Escenarios tridimensionales interactivos: Es el espacio destinado para la representación de un dominio de aplicación en las tres dimensiones ancho, largo y profundidad que permite la interacción con el usuario.

Fichero: Conjunto de información que se almacena en algún medio de escritura que permita ser leído o accedido por una computadora. Es identificado por un nombre y la descripción de la carpeta o directorio que lo contiene y facilita una manera de organizar los recursos usados para almacenar permanentemente información dentro de un computador.

Framework: Es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, librerías y un lenguaje de *scripting* entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Herramientas: Utensilios o provisiones necesarias para poder emprender un proyecto de software. Soportan los procesos de desarrollo de software modernos.

Implementación: Disciplina del proceso de desarrollo de cuya finalidad es implementar los componentes y productos para satisfacer las necesidades funcionales y la calidad de un sistema.

Internet: Conjunto descentralizado de redes de comunicación interconectadas, que utilizan la familia de protocolos TCP/IP, garantizando que las redes físicas heterogéneas que la componen funcionen como una red lógica única, de alcance mundial.

Iteración: Técnica de desarrollar y entregar componentes incrementales de funcionalidades de un negocio.

Licencia BSD: Licencia de software otorgada principalmente para los sistemas BSD (*Berkeley Software Distribution*). Pertenece al grupo de licencias de software Libre. Esta licencia tiene menos restricciones en

comparación con otras como la GPL estando muy cercana al dominio público. La licencia BSD al contrario que la GPL permite el uso del código fuente en software no libre. Es muy similar en efectos a la licencia MIT.

Navegador Web: Es un programa que permite visualizar la información que contiene una página Web.

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

Protocolo: Conjunto de normas que rigen un determinado proceso de comunicación.

Reutilización: Es la acción de volver a utilizar los bienes o productos ya elaborados y probados. Puede venir propiciada por una mejora o restauración o sin modificarse, usarlo en la creación de un nuevo producto.

Servicio Web: (en inglés *Web Services*) es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios Web para intercambiar datos en redes de ordenadores.

Sitio Web: Conjunto de páginas Web, típicamente comunes a un dominio o subdominio de Internet.

Software Privativo: Software en el que los usuarios tienen limitadas las posibilidades de usarlo, modificarlo o redistribuirlo (con o sin modificaciones), o cuyo código fuente no está disponible o el acceso a éste se encuentra restringido.

Stakeholders: Quienes pueden afectar o son afectados por las actividades de una empresa o negocio. Partes interesadas.

Siglas

3D: Tridimensional.

ASP (Active Server Pages): Es una tecnología del lado servidor de Microsoft para páginas web generadas dinámicamente, que ha sido comercializada como un anexo a Internet Information Server (IIS).

BD: Base de Datos

DAL: Capa de Acceso a Datos.

DARPA: Agencia de Investigación de Proyectos Avanzados de Defensa.

DSSA: Arquitectura de Software de Dominio Específico.

FR: Requerimientos Funcionales.

FTP: Protocolo de Transferencia de Archivos del inglés File Transfer Protocol.

GNU: Licencia Pública General de GNU (General Public License).

GPL: La Licencia Pública General de GNU (General Public License).

GUI: Interfaz Gráfica de Usuario (Graphic User Interface), artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático.

HTML: Lenguaje de Marcado de Hipertexto, es el lenguaje autoritario para crear documentos en la World Wide Web. Define la estructura de un documento web usando etiquetas y atributos.

HTTP: Protocolo de trasferencia de hipertextos. Protocolo que se utiliza para la transferencia/recepción de páginas Web.

IBM: (**I**nternational **B**usiness **M**achines) es una empresa que fabrica y comercializa hardware, software y servicios relacionados con la informática.

IEEE: (The **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers) el Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas.

ITU: Unión Internacional de Telecomunicaciones.

MVCC: Control de Concurrencia Multiversión.

PC: (en inglés Personal Computer) Computadora Personal.

POO: Programación Orientada a Objeto.

RIA: Rich Internet Applications.

SEI: Software Engineering Institute.

SGBD: Sistema Gestor de Base de Dato.

SQL: El Lenguaje de consulta estructurado o SQL (Structured Query Language), es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas.

TIC: Tecnologías de la Información y las Comunicaciones.

UCI: Universidad de Ciencias Informáticas.

UML: Lenguaje Unificado de Modelado