UNIVERSIDAD DE LAS CIENCIAS INFÓRMATICAS

Título:

Capa de lógica del negocio de una arquitectura para la construcción de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos.

TESIS EN OPCIÓN AL TÍTULO ACADÉMICO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autor: Liusba Cañete Núñez

Tutor: Ing. Gilberto Cao Tarrero

Cuidad de la Habana

Junio 2011

DECLARACIÓN DE AUTORÍA

Declaro ser autora de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Liusba Cañete Núñez

_____

Firma de la Autora

Ing. Gilberto Cao Tarrero

_____

Firma del Tutor

DATOS DE CONTACTO

Nombre y Apellidos: Gilberto Cao Tarrero.

Edad: 25 años.

Ciudadanía: Cubano.

Institución: Universidad de las Ciencias Informáticas.

Título: Ingeniero en Ciencias Informáticas.

Categoría Docente: Trabajador Docente.

e-mail: gcao@uci.cu

Graduado en la Universidad de las Ciencias Informáticas, 2 años de experiencia.

AGRADECIMIENTOS

A mis padres, que independientemente de que nunca estuvieron de acuerdo con la carrera que escogí, siempre han estado al tanto de todo, esforzándose para que nunca me falte nada.

A mis tíos, grandes ejemplos de lucha y perseverancia, demostrándome que nada es imposible.

A mi primita y toda mi familia que ha contribuido en mi formación profesional y personal.

A mis amigas y amigos que han compartido conmigo momentos de alegrías y dificultades, y en especial a Libeidy, Irenna, Yoana, Liudmila, Dayanis y Sandra que han estado presente desde el inicio y con las cuales he compartido momentos inolvidables. A mi amigo Alexis con quien compartí lindos momentos y una persona que admiro muchísimo. A mi gran amigo Adriel, creo que si el cariño se pudiera medir no me alcanzarían los números para expresarte cuanto te quiero, gracias por ser tan atento y preocuparte tanto por mí y sobre todo por dejarme conocerte y mostrarme la bella persona que eres.

A mis compañeros de aula y apartamento de todos estos años.

A todas las personas que me han ayudado en mi tesis, en específico a Julio, Ernesto y Yerandi quienes se brindaron a ayudarme y cuyo apoyo fue de vital importancia en el desarrollo de la tesis, a mis casi compañeros de tesis Libeidy y Andy con quienes compartí momentos de angustias y estrés pero también de esperanzas y apoyo mutuo.

Muchas gracias a todos.

DEDICATORIA

A mi mamá por ser la persona que siempre ha estado presente, iluminando mi camino y dándome fuerzas cuando siento que me quedo sin ellas. A mi papá que es un ejemplo a seguir y lo mejor que me ha pasado en la vida. Gracias por estar y ser mi padre. A mis tíos que han sido como otros padres para mí, siempre presentes y ayudándome en todo.

A mis primitas, que espero este trabajo les sirva de inspiración para su propia superación.

RESUMEN

La realización del presente trabajo tiene como objetivo principal proponer una capa de lógica del negocio para el desarrollo de Aplicaciones Enriquecidas de Internet (RIA) que incluyan escenarios tridimensionales interactivos, que facilite un apropiado entendimiento del sistema a sus desarrolladores, al tiempo que favorezca la construcción de RIA, proporcionando una mayor organización en la aplicación, fomentando la reutilización y haciendo evolucionar el sistema. Para la realización del mismo se hace uso de los métodos analítico-sintético y análisis histórico-lógico, para analizar y extraer la síntesis de los elementos más importantes relacionados con las Arquitecturas de *Software*, Arquitecturas de *Software* de Dominio Específico, estilos arquitectónicos y patrones de diseño utilizados para el diseño de este tipo de aplicaciones, además de analizar las distintas tecnologías, herramientas, lenguaje de modelado y metodologías de desarrollo de *software* que se utilizan en la actualidad para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos.

Luego del análisis realizado se pasó a establecer la línea base de la arquitectura, argumentando el uso cada uno de los patrones de diseño, estilo arquitectónico, tecnología, herramientas, lenguaje de modelado y metodologías utilizadas para el diseño de la propuesta realizada. Para lograr un mejor entendimiento, se realizó la descripción de la estructura de la capa de lógica del negocio propuesta, definiendo las metas y restricciones arquitectónicas, así como la representación arquitectónica de la misma a través del modelo 4+1 Vistas.

Por último se analizan aspectos importantes relacionados con la evaluación de la Arquitectura de *Software*, escogiendo el método Revisiones Activas para Diseños Intermedios (ARID) y la técnica basada en escenarios para la evaluación de la propuesta. Para la misma se seleccionaron diferentes atributos de calidad con el objetivo de detectar debilidades y fortalezas de la propuesta realizada.

Palabras Clave: Aplicaciones Enriquecidas de Internet, Arquitectura de *Software*, Arquitectura de *Software* de Dominio Especifico, Estilo arquitectónico.

ÍNDICE DE CONTENIDO

INTRODUCCIÓN 1

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 4

1.1 Introducción 4

1.2 Aplicaciones Enriquecidas de Internet, RIA 4

1.3 Arquitectura de Software 5

1.3.1 Arquitectura de Software de Dominio Específico 6

1.4 Estilos arquitectónicos 7

1.4.1 Tubería y filtros 7

1.4.2 Arquitectura en capas 8

1.4.3 Arquitecturas Orientadas a Servicios 11

1.4.4 Arquitectura basada en componentes 11

1.5 Patrones de diseño 12

1.5.1 Modelo-Vista-Controlador 13

1.5.2 Patrones Generales de Software de Asignación de Responsabilidades 14

1.5.3 Patrones Gang of Four 16

1.6 Metodologías de desarrollo de software 18

1.6.1 Proceso Unificado Ágil, AUP 19

1.6.2 Programación Extrema, XP 21

1.6.3 SCRUM 22

1.7 Bases tecnológicas 23

1.7.1 Ambiente Integrado de Desarrollo 23

1.7.2 Lenguaje de Programación 25

1.7.3 Bibliotecas de código 26

1.7.4 Herramientas de modelado 27

1.7.5 Servidores Web 28

1.8 Conclusiones 29

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN 30

2.1 Introducción 30

2.2 Línea base de la arquitectura. 30

2.3 Alcance 30

2.4 Estilo arquitectónico seleccionado 30

2.5 Patrones de diseño seleccionados 32

2.6 Metodología de desarrollo a utilizar 33

2.7 Lenguaje y herramienta de modelado a utilizar 33

2.8 Selección de las tecnologías a utilizar 33

2.9 Estructura de la capa de lógica del negocio 34

2.10 Metas y limitaciones de la arquitectura 35

2.10.1 Requisitos funcionales 36

2.10.2 Requisitos no funcionales 37

2.11 Descripción de la capa de lógica de negocio 38

2.11.1 Vista de casos de uso 39

2.11.2 Vista lógica 42

2.11.3 Vista de despliegue 44

2.11.4 Vista de implementación 45

2.11.5 Vista de procesos 46

2.12 Conclusiones 46

CAPÍTULO 3: EVALUACIÓN DE LA CAPA DE LÓGICA DEL NEGOCIO 48

3.1 Introducción 48

3.2 Evaluando la SA 48

3.2.1 ¿Por qué es necesario evaluar una SA? 48

3.2.2 Atributos de calidad 49

3.2.3 ¿Cuándo una arquitectura puede ser evaluada? 50

3.2.4 Resultado de la evaluación 50

3.3 Técnicas de evaluación 51

3.4 Métodos de evaluación de SA 51

3.4.1 Método de Análisis de Arquitectura de Software 52

3.4.2 Método de Análisis de Acuerdos de Arquitectura de Software 52

3.4.3 Método de Revisión Intermedio de Diseño 54

3.4.4 Selección del método de evaluación 55

3.5 Evaluación de la SA propuesta 55

3.6 Conclusiones 58

CONCLUSIONES GENERALES 59

RECOMENDACIONES 60

REFERENCIAS BIBLIOGRÁFICAS 61

BIBLIOGRAFÍA 65

GLOSARIO DE TÉRMINOS Y SIGLAS 67

ANEXOS 70

INTRODUCCIÓN

Las Tecnologías de la Información y las Comunicaciones (TIC) forman parte de la cultura tecnológica que nos rodea y con la que debemos convivir. Uno de los principales aportes de las TIC es el fácil acceso a todo tipo de información sobre cualquier tema y en cualquier formato, especialmente a través de internet.

Internet ha extendido su uso por todo el mundo gracias a la Web que ha tenido un desarrollo acelerado: pasando de páginas sencillas, con pocas imágenes y contenidos estáticos, a páginas complejas con contenidos dinámicos que provienen de bases de datos, lo que permite la creación de aplicaciones Web, y las mismas se pueden definir como "aplicaciones en las cuales los usuarios por medio de un navegador realizan peticiones a una aplicación remota accesible a través de Internet, o a través de una intranet, y reciben una respuesta que se muestra en el propio navegador". (Mora, 2002).

Con el continuo desarrollo y evolución de la Web han surgido nuevas aplicaciones que combinan las ventajas de las aplicaciones tradicionales de escritorio y las aplicaciones Web, ofreciendo mayores posibilidades y funcionalidades a los usuarios, las que se conocen como Aplicaciones Enriquecidas de Internet, RIA (*Rich Internet Applications*) por sus siglas en inglés. Las RIA han tomado un gran auge en el mundo empresarial, donde la fidelidad de los clientes, que cada vez son más exigentes, es muy difícil de lograr, provocando que los desarrolladores necesiten buscar nuevas soluciones, y una de las alternativas es la inserción de escenarios tridimensionales interactivos en la Web.

La Universidad de las Ciencias Informáticas (UCI) y en específico el Centro de Informática Industrial (CEDIN), han desarrollado RIA con escenarios tridimensionales interactivos, ejemplo de ello es el "Paseo virtual del salón de la industria informática cubana", realizado en el curso 2009-2010 por la línea de productos de *software* Visualización Web. Estas aplicaciones se han realizado de forma artesanal, ya que el centro y la universidad de forma general no cuentan con una arquitectura capaz de guiar el proceso de desarrollo de este tipo de aplicaciones.

La ausencia de una arquitectura para el desarrollo de aplicaciones puede provocar que al cometer errores en un estado avanzado del trabajo estos sean muy complejos de resolver, además de generar pérdida de tiempo y recursos. La capa de lógica del negocio junto con las capas de presentación y acceso a datos

conforman un estilo arquitectónico muy utilizado conocido como arquitectura en capas, este estilo permite llevar el desarrollo de las aplicaciones en varias capas, de esta forma si es necesario realizar cambios solo se afecta la capa requerida.

Para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos, el CEDIN no cuenta con los componentes necesarios responsables de comunicar las interfaces de usuario con el acceso a los datos, no se encuentra definido dónde establecer las reglas que deben cumplirse e implementar las funcionalidades principales del sistema. Esto lleva a la necesidad de estructurar una capa de lógica del negocio con altos niveles de reutilización y flexible que permita a los desarrolladores refinar y optimizar estas aplicaciones.

Teniendo en cuenta lo antes expuesto se plantea como **problema científico**: ¿Cómo establecer la comunicación entre la capa de presentación y la de acceso a datos en una Arquitectura de *Software* de Dominio Específico? Definiendo como **objeto de estudio:** Las Arquitecturas de *Software* de Dominio Específico (DSSA). Precisando como **campo de acción:** La capa de lógica del negocio en las DSSA. Y para darle solución al problema planteado se propone como **objetivo general**: Proponer una capa de lógica del negocio de una DSSA, que soporte el desarrollo de RIA que incluyan escenarios tridimensionales interactivos.

Para dar respuesta al objetivo general se han planteado las siguientes **tareas de investigación**:

- Elaboración del marco teórico de la investigación a partir del estado del arte existente sobre el tema.
- Identificación de las diferentes tecnologías empleadas para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos.
- Selección del lenguaje de programación que se definirá para el desarrollo de la capa de lógica del negocio.
- Descripción de los componentes que integran la propuesta de solución.
- Validación de la propuesta de solución con la evaluación de la arquitectura para identificar riesgos y fortalezas durante el diseño de la misma.

El resultado esperado con el desarrollo de la presente investigación es estructurar la capa de lógica del negocio lo cual permitirá contar con una capa personalizada que tribute a una DSSA robusta y reutilizable.

Para darle cumplimiento a las tareas de investigación propuestas se hará uso de los siguientes métodos científicos de investigación:

Analítico y Sintético: para analizar las teorías y documentos; permitiendo la extracción de los elementos más importantes que se relacionan con las DSSA.

Análisis histórico–lógico: este método nos permitirá realizar un análisis de la evolución histórica de las RIA y de las DSSA, en cuanto a características y funcionalidades.

El trabajo fue dividido en tres capítulos, a continuación se presentará el nombre de cada capítulo y su objetivo en un contexto general.

Capítulo 1: "Fundamentación teórica", se realiza un estudio de los aspectos relacionados con las RIA, profundizando en los conceptos precisos para su estudio: Arquitectura de *Software*, DSSA, metodologías, patrones de diseño, estilos arquitectónicos, herramientas y lenguajes de desarrollo. Además de analizar las diferentes características de la capa lógica del negocio. Se realiza un estudio y análisis del estado actual de las tecnologías a utilizar.

Capítulo 2: "Propuesta de solución", se describe el estilo arquitectónico que guiará la arquitectura y los patrones de diseño utilizados para llevar a cabo la estructuración de la capa de lógica de negocio y se especifican las herramientas y tecnologías de desarrollo, la metodología de desarrollo de *software*, así como el lenguaje y herramienta de modelado. Se describe la estructura de la propuesta de capa de lógica del negocio.

Capítulo 3: "Evaluación de la capa de lógica del negocio", se describe la importancia de la evaluación arquitectónica dentro del desarrollo de sistemas de *software*, sus costos y beneficios. Se realiza la evaluación de la arquitectura a través de una de las técnicas que permiten realizar evaluaciones a la arquitectura.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se realiza un estudio de los aspectos relacionados con la Arquitectura de *Software* para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos. Se brinda una visión general de los principales conceptos asociados al dominio del problema que son necesarios para la investigación: DSSA, estilos arquitectónicos, patrones de diseño, metodologías y herramientas de desarrollo.

1.2 Aplicaciones Enriquecidas de Internet, RIA

Las RIA consisten en un nuevo tipo de aplicación Web cuyo objetivo es incrementar y mejorar las opciones y capacidades de las aplicaciones Web tradicionales. Permitiendo a los usuarios realizar operaciones comunes en clientes pesados como arrastrar y soltar *(drag&drop)*, cambiar tamaño *(resize*) o animar objetos. (López, 2005). Según la definición de la compañía desarrolladora de *software* Macromedia, las RIA son participativas, interactivas, ligeras, ofrecen la flexibilidad y facilidad de uso inteligente de una aplicación de escritorio y añaden el amplio alcance de las aplicaciones Web tradicionales. (Fain, 2007).

Estas aplicaciones consisten en el aprovechamiento de la experiencia del usuario en herramientas y funciones de escritorio tan naturales como copiar, cortar, pegar, redimensionar columnas y ordenar, con el alcance y la flexibilidad de presentación y despliegue que ofrecen las aplicaciones o páginas Web junto con lo mejor de la multimedia (voz, vídeo, etc.). (Estudis, 2009). Se trata de aplicaciones que ofrecen mejoras en la conectividad y despliegue instantáneo de la aplicación, agilizando su acceso y permitiendo comunicación en tiempo real entre los usuarios, incentivando la participación y la colaboración de los internautas. (CXO Community Latam, 2009).

Algunas de las características de las RIA son: la creación de aplicaciones más atractivas mediante la utilización de audio, video y gráficos; disminuye el ancho de banda utilizado en el uso de la aplicación ya que permite almacenar más información en el cliente de manera que se reduce el uso de transacciones HTTP; con las RIA, el usuario interactúa con la interfaz de usuario y sólo se realizan comunicaciones con

el servidor cuando son necesarias; la mayoría de las RIA pueden ejecutar las aplicaciones *online/offline*. (López, 2005).

Se puede decir que las RIA insertan un nuevo modelo de programación de aplicaciones, que combina las ventajas de las aplicaciones Web y las aplicaciones tradicionales de escritorio. Son aplicaciones multiplataforma ya que pueden funcionar independientemente del sistema operativo que el usuario tenga instalado en su ordenador, además pueden ser accedidas desde cualquier equipo que tenga acceso a una red.

1.3 Arquitectura de Software

El término Arquitectura de *Software* (en adelante, SA) comienza a utilizarse a partir de 1968 cuando Edsger Dijkstra propone establecer una estructuración correcta de los sistemas de *software* antes de lanzarse a programar, escribiendo código de cualquier manera. (Dijkstra, 1968). Desde entonces son muchas las definiciones que se le han dado a la SA.

Según Pressman la SA efectiva, su representación y diseño explícitos se han vuelto temas dominantes en la Ingeniería de S*oftware (IS)*. Se trata de una representación que le posibilita al ingeniero de *software* analizar la efectividad del diseño para cumplir con los requisitos establecidos, tener en cuenta las opciones arquitectónicas en una etapa que aún resulte fácil hacer cambios al diseño y reducir los riesgos asociados con la construcción del *software*. (Pressman, 2005).

Otra de las definiciones conocidas es la que brindan Bass y sus colegas: La SA es una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones. (Len Bass, 2003).

El documento de IEEE Std 1471-2000*,* que nos brinda la definición oficial referente a este tema, plantea: La SA es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución. (IEEE, 2000).

Para concluir se puede decir que la SA se centra en las decisiones importantes, brinda conceptos y un lenguaje común que proporciona la comunicación entre los equipos que participan en un proyecto, es necesaria cuando se realizan proyectos de *software* ya que ayuda a los desarrolladores a comprender el sistema, organizar el desarrollo y permitir la reutilización. De una SA es importante su uso, funcionalidad, rendimiento, capacidad de recuperación y reutilización, además de permitir la flexibilidad de un sistema, brindando la posibilidad de realizar futuros cambios.

1.3.1 Arquitectura de Software de Dominio Específico

Los sistemas en el mismo dominio de aplicación a menudo tienen arquitecturas similares que reflejan los conceptos fundamentales del dominio. Cuando se diseña la arquitectura de un sistema, se deben determinar los elementos semejantes que puedan existir en aplicaciones del mismo dominio de trabajo y en qué medida el conocimiento de esas arquitecturas de aplicaciones se puede reutilizar. (Sommerville, 2005).

Existen modelos arquitectónicos que son específicos para un dominio particular de aplicación, en los cuales, aunque los sistemas difieran en detalles, la estructura arquitectónica común puede reutilizarse cuando se desarrollan nuevas aplicaciones. Estos modelos arquitectónicos se conocen como: arquitecturas de dominio específico. (Sommerville, 2005).

Existen dos tipos de modelos arquitectónicos de dominio específico: los modelos genéricos que son abstracciones obtenidas a partir de varios sistemas reales y encapsulan las características principales de los sistemas. Y los modelos de referencia los cuales son más abstractos, describen una clase más amplia de sistemas y constituyen un modo de informar a los diseñadores sobre la estructura general de esta clase de sistemas. (Sommerville, 2005).

La existencia de una DSSA dictará un planteamiento muy diferente al desarrollo de *software* de aplicación. El desarrollador no tiene que esperar hasta el diseño detallado y la aplicación de búsqueda de oportunidades de reutilización, en su lugar, es conducido por la arquitectura a lo largo del desarrollo de la aplicación.

1.4 Estilos arquitectónicos

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 como 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**.** (Pressman, 2002)**.**

Un estilo arquitectónico provee una colección de elementos edificadores del diseño en bloque, reglas y restricciones para componer los bloques constructivos, y las herramientas para analizar y manipular los diseños creados en el estilo. Los estilos generalmente proveen guía y análisis para crear una clase amplia de arquitecturas en un dominio específico donde los patrones se enfocan en solucionar los problemas más pequeños y más específicos dentro de un estilo dado. (Kiccillof, 2004).

Mary Shaw y Paul Clements definen los estilos arquitectónicos como un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer en sistema o módulo, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo. (Clements, 1997).

En la actualidad existe gran variedad de estilos arquitectónicos. A continuación se analizan y caracterizan algunos de los más difundidos, para luego determinar cuál usar en el desarrollo de la capa de lógica del negocio.

1.4.1 Tubería y filtros

El sistema tubería-filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes, los cuales son capaces de filtrar los datos, que interactúan con otros filtros, sólo a través de una interfaz de entrada y una de salida. Utilizando este estilo arquitectónico los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. Gracias a su simplicidad y su facilidad para captar una funcionalidad, es una arquitectura mascota cada vez que se trata de demostrar ideas sobre la formalización del espacio de diseño arquitectónico. (Doberkat, 2002).

1.4.2 Arquitectura en capas

Al utilizar un estilo de arquitectura en capas, cada capa se maneja de forma independiente, sin necesidad de conocer los detalles de las demás. Dividir una aplicación en capas separadas que desempeñan diferentes roles y funcionalidades, ayuda a mejorar el mantenimiento del código; permite diferentes tipos de despliegue y proporciona una clara delimitación de dónde debe estar cada tipo de componente funcional e incluso cada tipo de tecnología. (Javier Calvarro Nelson, 2010).

La clave de una aplicación en N capas está en la gestión de dependencias: los componentes de una capa pueden interactuar solo con componentes de la misma capa o bien con otros componentes de capas inferiores, lo cual ayuda a reducir las dependencias entre componentes de diferentes niveles. (Javier Calvarro Nelson, 2010). Esto permite que, en caso que sea necesario realizar cambios, solo se afecta la capa en cuestión, sin tener que influir en el resto del sistema.

Entre las ventajas de utilizar este estilo de arquitectura se encuentran las siguientes: admite optimizaciones y refinamientos; gracias al encapsulamiento, las aplicaciones son más robustas; soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez facilita a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. (Kiccillof, 2004). Permite un desarrollo estructurado, homogéneo y similar de las diferentes aplicaciones de una organización. (Javier Calvarro Nelson, 2010).

Además proporciona amplia reutilización, al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes, conduciendo a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se puedan construir extensiones o prestaciones específicas. (Kiccillof, 2004).

En una arquitectura en N capas, cada capa usa los servicios brindados por la capa inferior y expone servicios a la capa superior, además dicho estilo favorece la abstracción entre las capas, minimiza las dependencias y permite que se pueden reemplazar fácilmente. Para la utilización del estilo arquitectura en capas es importante decidir que capas tener y la responsabilidad de cada una.

1.4.2.1 Arquitectura en tres capas.

La arquitectura en tres capas es una de las formas que puede tomar el estilo de arquitectura en capas, y el objetivo fundamental de este estilo es mantener un bajo acoplamiento entre las capas y fomentar la reutilización. El mismo está compuesto por las capas de presentación, lógica del negocio y acceso a datos, como se muestra en la Fig. 1.

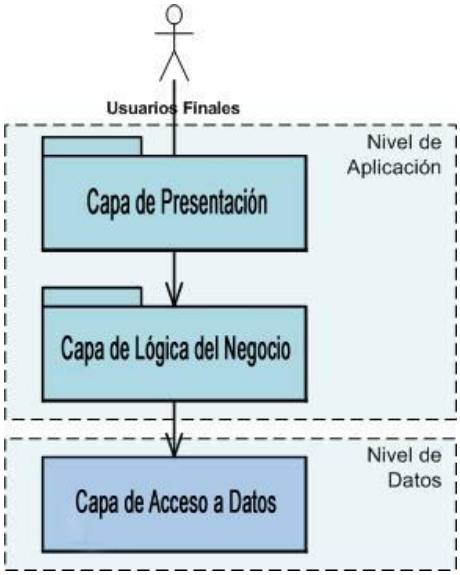

Fig. 1: Representación del estilo Arquitectura en tres capas.

La capa de presentación o también conocida como “capa de usuario”, es la responsable de comunicar y capturar información al usuario e interpretar sus acciones. Tiene como principales responsabilidades las de presentar al usuario los conceptos de negocio mediante una interfaz de usuario, facilitar la explotación de dichos procesos, informar sobre el estado de los mismos e implementar las reglas de validación de dicha interfaz. (Javier Calvarro Nelson, 2010).

Desde el punto de vista del usuario final, esta capa es la “aplicación” en sí, ya que es la que se encarga de interactuar con el mismo, comunicándole y recogiendo la información suministrada por él. Esta capa se comunica solamente con la capa de lógica del negocio, al ser la que interactúa directamente con el usuario debe ser entendible, amigable y fácil de usar.

La separación entre el código de la lógica de presentación y la interfaz reduce los gastos que pudieran ocasionar cambios futuros. Además favorece la colaboración entre diseñadores, desarrolladores y jefes de proyectos, lo que conlleva a evitar pérdidas de tiempo por mala comunicación. (Javier Calvarro Nelson, 2010).

La capa de lógica del negocio o también conocida como lógica del dominio es la responsable de representar conceptos de negocio, información sobre la situación de los procesos de negocio e implementación de las reglas del dominio. Definida por Zorrilla y sus colegas como el "corazón del *software*", esta capa debe contener los estados que reflejan la situación de los procesos de negocio, aun cuando los detalles técnicos de almacenamiento se delegan a las capas inferiores de infraestructura. (Javier Calvarro Nelson, 2010).

La principal razón de implementar capas de lógica del negocio radica en diferenciar y separar muy claramente entre el comportamiento de las reglas del negocio de los detalles de implementación de infraestructura (acceso a datos y repositorios concretos ligados a una tecnología específica). (Javier Calvarro Nelson, 2010). Resumiendo, esta capa obtiene las solicitudes enviadas por la capa de presentación y envía los resultados de la solicitud y se comunica con la capa de acceso a datos para pedirle la información necesaria que se almacena en las bases de datos. Además, es aquí donde se establecen las reglas que deben cumplirse.

La capa de acceso a datos es la encargada de persistir las entidades que se manejan en el negocio, el acceso a los datos almacenados y reúne todos los aspectos del *software* que tienen que ver con el manejo de los datos persistentes. Es donde se gestiona el almacenamiento físico y recuperación de datos. Ayuda a separar el acceso a los datos de la lógica del negocio, por lo que no contiene ninguna lógica de negocio, así como tampoco contiene elementos de la interfaz de usuario.

Esta capa expone el acceso a datos a la capa superior. Además debe cumplir los requerimientos de la aplicación a nivel de rendimiento, seguridad, mantenibilidad y soportar cambios de requerimientos de negocio. (Javier Calvarro Nelson, 2010).

1.4.3 Arquitecturas Orientadas a Servicios

La Arquitectura Orientada a Servicios (SOA) establece un marco de diseño para la integración de aplicaciones independientes de manera que desde la red pueda accederse a sus funcionalidades, las cuales se ofrecen como servicios. La forma más habitual de implementarla es mediante servicios Web, una tecnología basada en estándares e independiente de la plataforma, con la que SOA puede descomponer aplicaciones monolíticas en un conjunto de servicios e implementar esta funcionalidad en forma modular. (Microsoft Corporation, 2006). Permite simplificar las relaciones entre distintos sistemas, optimizar su funcionamiento, facilitar la incorporación de nuevos elementos en la arquitectura e incluso cambiar los existentes de una manera sencilla. (Alba, 2008).

Los componentes del estilo SOA son los servicios. Un servicio es una entidad de *software* que encapsula funcionalidad de negocios y proporciona dicha funcionalidad a otras entidades a través de interfaces públicas bien definidas. (Kicillof, 2004). Los servicios en SOA están débilmente acoplados, lo que permite que los componentes de *software* que se desarrollen sean altamente reusables.

Los servicios para comunicarse entre sí se basan en una definición formal independiente de la plataforma y del lenguaje de programación. La definición de interfaces permite encapsular la implementación, lo que la hace independiente del desarrollador, del lenguaje de programación o de la tecnología de desarrollo. (Puebla, 2008). SOA permite mejorar la toma de decisiones y facilita una mayor capacidad de respuesta a los clientes. Con este estilo las aplicaciones son más seguras y flexibles.

1.4.4 Arquitectura basada en componentes

El estilo arquitectura basada en componentes se centra en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas. “Un componente es una implementación opaca de funcionalidad, sujeta a composición por terceros y que cumple con un modelo de componentes. (Bachmann, 2000).

Algunas de las características de la arquitectura basada en componentes son: *Permiten la reutilización* de los componentes. *Identificables,* cada componente debe tener una identificación clara y consistente que facilite su catalogación y búsqueda en repositorios de componentes. *Accesible sólo a través de su interfaz*, los componentes deben exponer al usuario únicamente el conjunto de operaciones que lo caracteriza

(interfaz) y ocultar sus detalles de implementación, esta característica permite que un componente sea reemplazado por otro que implemente la misma interfaz. *Servicios invariantes*, las operaciones que ofrecen los componentes a través de sus interfaces no deben variar, la implementación de estos servicios puede ser modificada, pero no deben afectar la interfaz. *Documentados*, los componentes deben tener una documentación adecuada que facilite su búsqueda en repositorios de componentes, evaluación, adaptación a nuevos entornos, integración con otros componentes y acceso a información de soporte. (Puebla, 2008).

Una importante ventaja de este estilo es que facilita la realización de pruebas a cada componente antes de hacer las pruebas para el sistema completo. También permite un fácil mantenimiento cuando existe un débil acoplamiento entre los componentes, ya que se pueden gestionar libremente sin afectar otras partes del sistema u otros componentes.

1.5 Patrones de diseño

Los patrones de diseño ayudan a mejorar la calidad del diseño de las aplicaciones. Con la utilización de los mismos se obtiene un mayor dominio del análisis y diseño del sistema.

Los patrones de diseño no son fáciles de entender, pero una vez entendido su funcionamiento, los diseños serán mucho más flexibles, modulares y reutilizables. Han revolucionado el diseño orientado a objetos y todo buen arquitecto de *software* debería conocerlos. (García, 2005).

La aplicación de un patrón de diseño no tiene ningún efecto sobre la estructura fundamental de un sistema de *software*, pero puede tener una fuerte influencia en la arquitectura de un subsistema. Una propiedad importante de los patrones de diseño es que son independientes de un dominio de aplicación y un paradigma de programación particular. Por lo general, se puede implementar fácilmente en programación orientada a objetos, pero todos los patrones de diseño son suficientemente generales como para ser adaptados a las prácticas más tradicionales de programación. (Frank Buschmann, 2001).

Pressman plantea que un patrón de diseño describe una estructura de diseño que resuelve un problema de diseño particular dentro de un contexto específico y cuya finalidad es proporcionar una descripción que le permita al diseñador determinar si el patrón es aplicable al trabajo actual, si se puede reutilizar y si

puede servir como guía para desarrollar un patrón similar, pero diferente en cuanto funcionalidad o estructura. (Pressman, 2005).

En definitivas, los patrones de diseño brindan soluciones simples a problemas que ocurren y se repiten una y otra vez en el desarrollo de un *software*. Se pueden reutilizar de manera reiterada sin tener que hacerlo de la misma forma.

1.5.1 Modelo-Vista-Controlador

El patrón clásico del diseño Web conocido como arquitectura Modelo-Vista-Controlador (en adelante, MVC) se encarga de separar la lógica de negocio de la interfaz de usuario de una aplicación. Como la presentación está separada de la programación, la aplicación es fácil de modificar puesto que se pueden realizar cambios o reemplazar uno de los componentes sin que los demás se vean afectados.

El MVC es muy utilizado en la creación de aplicaciones Web por la forma de organizar el código debido a que separa los datos de la aplicación, la interfaz de usuario, y la lógica del negocio en tres entidades diferentes:

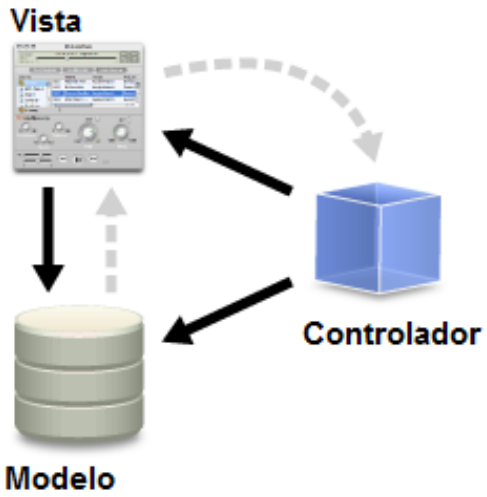

Fig. 2: Modelo-Vista-Controlador.

Modelo: El modelo es un conjunto de clases que representan la información del mundo real que el sistema debe procesar. Encapsula los datos y las funcionalidades y administra el comportamiento y los datos del dominio de aplicación.

Vista: Conjunto de clases responsables de mostrar al usuario la información contenida en el modelo. (Pantoja, 2004). Es la representación visual de los datos y se encarga de la interacción con el usuario.

Controlador: Es donde se guarda la lógica de la aplicación y se realizan todas las acciones indispensables para generarla. (Álvarez, 2009).

De este patrón se puede decir que proporciona, al equipo de desarrollo, un sistema flexible y potente. Brinda la posibilidad de programas mejor organizados, que permitan un fácil mantenimiento e incremento de la reutilización; reduciendo el esfuerzo de programación en la implementación de los datos.

Algunas de las ventajas que proporciona la utilización del patrón arquitectónico MVC son: Las vistas siempre muestran información actualizada. El programador no debe preocuparse de solicitar que las vistas se actualicen; este proceso lo realiza automáticamente el modelo de la aplicación. MVC es muy utilizado en la actualidad en marcos de aplicación orientados a objeto desarrollados para construir aplicaciones de gran tamaño. MVC está demostrando ser un patrón de diseño bien elaborado pues las aplicaciones que lo implementan presentan una extensibilidad única comparada con otras aplicaciones basadas en otros patrones. (Pantoja, 2004).

1.5.2 Patrones Generales de Software de Asignación de Responsabilidades

Los Patrones Generales de *Software* de Asignación de Responsabilidades (GRASP) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Dentro de los patrones GRASP se encuentran: bajo acoplamiento, alta cohesión, experto, creador y controlador que son los patrones principales, aunque también cuenta con cuatro de apoyo: polimorfismo, fabricación pura, indirección y no hables con extraños. En el presente acápite se analizarán algunos de los patrones antes mencionados.

1.5.2.1 Experto

El patrón experto se usa para asignar responsabilidades. Ofrece una analogía con el mundo real, ya que permite asignar responsabilidad a individuos que disponen de la información necesaria para llevar a cabo una tarea.

Con la utilización del patrón experto se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide, soportando un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento. El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases sencillas y cohesivas que son más fáciles de comprender y de mantener. (Larman, 1999).

1.5.2.2 Bajo acoplamiento

El bajo acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que incrementan la oportunidad de una mayor productividad. No puede considerarse en forma independiente de otros patrones, sino más bien ha de incluirse como uno de los principios del diseño que influyen en la decisión de asignar responsabilidades. (Larman, 1999).

No es conveniente que exista muy poco o nulo acoplamiento entre las clases, ya que esto daría origen a un diseño deficiente por producir objetos incoherentes y complejos que hacen todo el trabajo, con muchos otros objetos muy pasivos y de acoplamiento cero que funcionan como meros depósitos de datos. Un grado moderado de acoplamiento entre las clases es normal y necesario si quiere crearse un sistema orientado a objetos, donde las tareas se realicen por colaboración entre objetos conectados. (Larman, 1999).

Sin embargo, una clase con alto acoplamiento trae como consecuencia que sea más difícil de reutilizar porque se requiere la presencia de otras clases de las que depende. Además los cambios de las clases afines ocasionan cambios locales. El bajo acoplamiento se encarga de asignar responsabilidades de modo que no incremente el acoplamiento entre componentes. Favorece la reutilización y que un componente no se afecte por cambios en otros componentes.

1.5.2.3 Alta cohesión

La cohesión, en la perspectiva del diseño orientado a objetos, es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. (Larman, 1999).

Una clase con baja cohesión es difícil de comprender, de reutilizar y de conservar, además se ven afectadas constantemente por los cambios y presentan un alto grado de abstracción. Sin embargo, la utilización del patrón alta cohesión mejora la claridad y la facilidad con que se entiende el diseño, simplifica el mantenimiento y mejoras en funcionalidad, ayuda a mantener un bajo acoplamiento y soporta una mayor capacidad de reutilización, debido a que una clase muy cohesiva puede destinarse a un propósito muy específico. (Larman, 1999).

La utilización de este patrón permite que cada clase tenga las responsabilidades necesarias y que colabore con las demás clases para compartir el esfuerzo si la tarea a realizar es muy grande. Es importante mencionar que una clase con mucha cohesión es útil ya que posibilita un fácil mantenimiento, entendimiento y reutilización.

1.5.2.4 Controlador

Un controlador es un objeto de interfaz no destinado al usuario que se encarga de manejar un evento del sistema. Este patrón ofrece una guía para tomar decisiones apropiadas que generalmente se aceptan. Un defecto frecuente al diseñar controladores consiste en asignarles demasiada responsabilidad. Normalmente un controlador debería delegar a otros objetos el trabajo que ha de realizarse mientras coordina la actividad. (Larman, 1999).

1.5.3 Patrones Gang of Four

Los patrones Gang of Four (GoF) se denominaron así gracias a la Banda de los Cuatro, nombre con el que se conoce comúnmente a los autores del libro Design Patterns, Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. En este libro se recopilaron y documentaron 23 patrones de diseño clasificados según su propósito en creacionales, estructurales y de composición. A continuación se analizan algunos de estos patrones.

1.5.3.1 Observador

El patrón observador define una dependencia uno-a-muchos entre objetos, para que, cuando uno de ellos cambie su estado, todos los que dependan de él sean avisados y puedan actualizarse convenientemente. Este patrón permite de forma dinámica implementar dependencias entre objetos, de forma que los objetos dependientes sean notificados de los cambios que se producen en los objetos de los que dependen.

1.5.3.2 Fachada

EL patrón fachada o *facade* proporciona una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. Es decir, provee una interfaz sencilla que hace de intermediaria entre un usuario y una interfaz o grupo de interfaces más complejas (Fig. 3). El uso de este patrón permite que al separar al cliente de los componentes del subsistema, se reduzca el número de objetos con los que el cliente trata, facilitando así el uso del subsistema. Promueve un acoplamiento débil entre el subsistema y sus clientes, eliminando o reduciendo las dependencias.

Se utiliza para simplificar la dependencia entre clases, ofreciendo un punto de acceso al resto de las clases. Si estas cambian o se sustituyen por otras solo hay que actualizar la clase Fachada sin que el cambio afecte al resto del sistema.

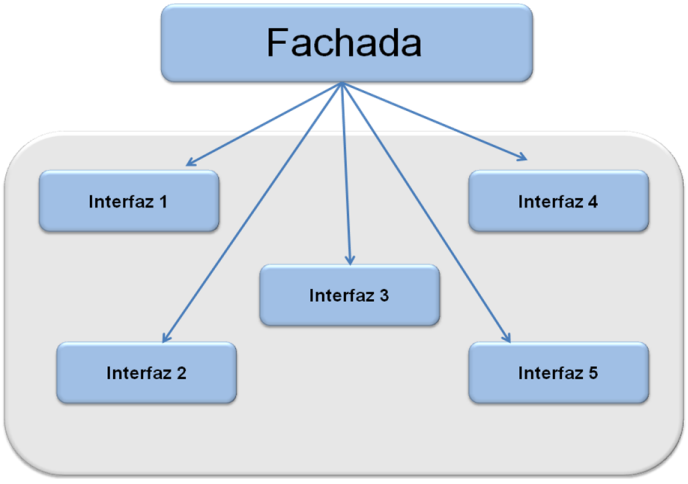

Fig. 3: Patrón Fachada

La fachada sabe qué clase es responsable de cada petición, pero es completamente transparente para ellas. Por lo que, aunque las funciones las desempeñan otras clases, es a la Fachada a quien le corresponde traducirlas.

1.5.3.3 Singleton

El patrón *singleton* o también conocido como solitario garantiza que una clase sólo tenga una única instancia, proporcionando un punto de acceso global a la misma. Es un patrón sencillo, muy conocido y utilizado en el desarrollo de aplicaciones.

Es un patrón que permite un acceso controlado a la única instancia. Puede tener un control estricto sobre cómo y cuándo acceden los clientes a la instancia. Establece un espacio de nombres reducido al presentar una mejora sobre las variables globales. Permite el refinamiento de operaciones y la representación y un número variable de instancias, haciendo que sea fácil cambiar de opinión y permitir más de una instancia de la clase *Singleton*.

Se puede usar el patrón *singleton* cuando deba existir exactamente una instancia de una clase y ésta deba ser accesible a los clientes desde un punto de acceso conocido, o cuando la única instancia debería ser extensible mediante herencia y los clientes deberían ser capaces de utilizar una instancia extendida sin modificar su código.

1.6 Metodologías de desarrollo de software

Las metodologías de desarrollo de *software* son un conjunto de pasos, procedimientos, técnicas y ayudas a la documentación que se deben seguir para el desarrollo de productos de *software*, indicando las actividades que se realizan para lograr el producto informático deseado. Plantea qué personas participan en el desarrollo de las actividades y qué papel juegan, además detallan tanto la información necesaria para comenzar una actividad, así como la que se debe producir como resultado de esta. (Barzanallana, 2006).

Las metodologías persiguen tres necesidades principales: mejores aplicaciones, tendientes a una mejor calidad, aunque a veces no es suficiente; un proceso de desarrollo controlado, que asegure el uso de recursos apropiados y costo adecuado; y un proceso estándar en la organización, que no sienta los cambios del personal. (Cataldi, 2000). Las metodologías para el desarrollo de *software* se agrupan en dos

grandes grupos: metodologías tradicionales y metodologías ágiles. Cada una de ellas con las características que las distinguen y hacen convenientes de usar o no según las especificaciones del proyecto que se desee realizar.

Las **metodologías tradicionales** se centran en la documentación, planificación y procesos del proyecto que se lleve a cabo. Una característica importante dentro de este enfoque, es el alto costo al implementar un cambio. Además, estas metodologías exigen un sobreesfuerzo por parte del equipo de desarrollo en fases poco productivas, como es la documentación, y debido a su estructuración, son poco flexibles a los cambios de requisitos y de nuevas funcionalidades. Como se puede observar, este tipo de metodología presenta varios inconvenientes.

Por su parte, las **metodologías ágiles** surgen como una alternativa, una respuesta a los problemas de las metodologías tradicionales antes mencionados. Se basan en cuatro principios definidos por Kent Beck en el documento “Manifiesto Ágil”. Los **principios** son: Los individuos y las interacciones entre ellos son más importantes que las herramientas y los procesos empleados; es más importante crear un producto *software* que funcione que escribir documentación exhaustiva; la colaboración con el cliente debe prevalecer sobre la negociación de contratos y la capacidad de respuesta ante un cambio es más importante que el seguimiento estricto de un plan. (Kent Beck, 2001).

La flexibilidad que ofrecen las metodologías ágiles significa una ventaja competitiva a la hora de elegir una. Estar preparados para el cambio constituye otra ventaja ya que implica reducir su coste.

Por las razones antes expuestas y basándose en la comparacion efectuada en el “Taller Metodologías Ágiles en el Desarrollo de *Software*” realizado en el marco de las VIII Jornadas de IS y Bases de Datos, JISBD 2003, (Anexo 1), se eligen las metodologias ágiles para guiar el desarrollo de la propuesta de capa de lógica del negocio. Dentro de las metodologías ágiles se pueden encontrar *eXtreme Programming* (XP), SCRUM, Proceso Unificado Ágil, entre otras.

1.6.1 Proceso Unificado Ágil, AUP

El Proceso Unificado Ágil (AUP, por sus siglas en inglés) es una versión simplificada del *Rational Unified Process* (RUP), desarrollada por Scott Amber. (Prabhudas, 2008). Describe de una manera simple y fácil

de entender la forma de desarrollar aplicaciones de *software* usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP.

El proceso AUP establece un modelo más simple que el que aparece en RUP, reúne en una sola las disciplinas de Modelado de Negocio, Requisitos, Análisis y Diseño. El resto de las disciplinas (Implementación, Pruebas, Despliegue, Gestión de Configuración, Gestión y Entorno) coinciden con las restantes de RUP. Aplicando esta metodología se puede eliminar gran cantidad de artefactos que no son necesarios y entorpecen la fluidez en el desarrollo. (Hirsch, 2002).

Igual que RUP, AUP establece cuatro fases que transcurren de manera consecutiva. (Prabhudas, 2008). **Incepción**: el objetivo es obtener una comprensión común cliente-equipo de desarrollo del alcance del nuevo sistema y definir una o varias arquitecturas candidatas para el mismo. **Elaboración**: el objetivo es que el equipo de desarrollo profundice en la comprensión de los requisitos del sistema y en validar la arquitectura. **Construcción**: durante la fase de construcción el sistema es desarrollado y probado al completo en el ambiente de desarrollo. **Transición**: el sistema se lleva a los entornos de preproducción donde se somete a pruebas de validación y aceptación y finalmente se despliega en los sistemas de producción. (Ver Fig. 4).

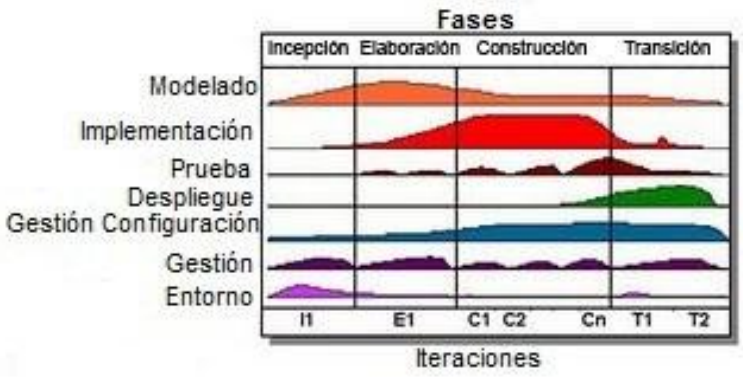

Fig. 4: Ciclo de vida de AUP.

Como proceso unificado al fin, cuenta con tres características fundamentales: guiado por casos de uso, iterativo e incremental y centrado en la arquitectura. El AUP es un acercamiento al desarrollo *software* orientado a las personas, permitiéndoles responder con eficacia al cambio y que resultan en la creación de sistemas de trabajo que satisfaga las necesidades de sus grupos de interés. (Prabhudas, 2008).

La utilización del AUP facilita la reutilización, además permite la realización de cambios en los requerimientos. Con dicha metodología se obtienen productos más robustos ya que, al ser iterativa, los errores se van corrigiendo en cada iteración.

1.6.2 Programación Extrema, XP

La programación extrema, XP (*eXtreme Programing*), es una metodología de desarrollo ágil atribuida a Kent Beck, autor de los libros más influyentes sobre el tema. Tiene como objetivos principales, en primer lugar, la satisfacción del cliente, ya que intenta brindarle al mismo, el *software* que él necesita, cuando lo necesita y, como segundo objetivo, potenciar lo más posible el trabajo en equipo, donde, tanto los jefes de proyecto, los clientes y desarrolladores, son parte de este y están involucrados en el desarrollo del *software*.

Es una metodología orientada por pruebas y refactorización, las pruebas se diseñan e implementan antes de programar la funcionalidad. Su ciclo de vida se caracteriza por ser iterativo e incremental. Está diseñado para grupos de trabajo pequeños, con pocos programadores, más de 10 ya sería muy complicado. Es un modelo de desarrollo sencillo, común y que se adapta fácilmente a los cambios y exigencias de los clientes. (Villegas, 2007).

Engloba las mejores características de las distintas metodologías, por lo que se puede decir que XP no está basada en principios nuevos, la mayoría de sus características ya se conocen dentro de la IS. Está basado en cuatro valores fundamentales para el desarrollo del trabajo: **Comunicación,** XP ayuda a fomentar la comunicación entre los integrantes del grupo de trabajo: jefes de proyecto, clientes y desarrolladores. **Sencillez,** los programas deben ser los más sencillos posibles y tener la funcionalidades necesarias que se indican en los requisitos. **Retroalimentación,** por medio de las pruebas funcionales que se le realizan al *software* este nos mantiene informados del grado de fiabilidad del sistema. **Valentía,** asumir retos, ser valientes ante los problemas y afrontarlos, intentar mejorar algo que ya funciona.

XP consta de cuatro actividades básicas para el desarrollo de un buen *software*: **Codificar**, es una actividad de la cual no se puede prescindir, sin código fuente no hay programa, en una programación utilizando la metodología XP, el código expresa la interpretación del problema. **Hacer pruebas**, las pruebas indican si el trabajo funciona o no, muestran si lo implementado es realmente lo que se tenía en mente. Es necesario hacer pruebas porque sin estas no se sabe si se ha terminado de codificar.

Escuchar, se debe escuchar de los clientes cuales son los problemas de su negocio. La realimentación entre clientes y desarrolladores ayuda a todos a entender los problemas y resolverlos. Si no se escucha, no se sabe que codificar ni probar. **Diseñar**, el diseño crea una estructura que organiza la lógica del sistema. Los diseños deben de ser sencillos, si alguna parte del sistema es de desarrollo complejo, lo apropiado es dividirla en varias. Para poder codificar, probar y escuchar indefinidamente se tiene que diseñar.

A pesar de que XP es una metodología que resalta por contar con una gran cantidad de información disponible, no se centra en la arquitectura. Además, en caso de fallar, puede presentar altas comisiones.

1.6.3 SCRUM

SCRUM es una metodología desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle que define un marco para la gestión de proyectos, indicada para proyectos con un rápido cambio de requisitos. El desarrollo de *software* se realiza mediante iteraciones, denominadas *sprints*, con una duración aproximada de 30 días. El resultado de cada *sprint* es un incremento ejecutable que se muestra al cliente.

En SCRUM se pueden identificar tres fases: **planificación del s*print***, donde se define el *Product Backlog* que consiste en una lista priorizada de requisitos del sistema. En cada iteración el *Product Backlog* es revisado por el equipo, también se lleva a cabo la planificación del primer *sprint*, se determinan cuáles son los objetivos y el trabajo que se deben cubrir en la iteración, además se obtiene una lista de tareas que denominamos *sprint backlog*, y el lema u objetivo principal del sprint. Otra fase es el **seguimiento del *sprint*,** a lo largo de esta fase se llevan a cabo breves reuniones diarias, para ver el avance de las tareas y el trabajo que está previsto para la jornada. Y la última fase es la **revisión del *sprint*,** donde se realiza un análisis y revisión del incremento generado, se presentan los resultados finales y se recomienda tener preparado un demo, entre otras razones, para mejorar la retroalimentación con los interesados. (Ver Fig. 5).

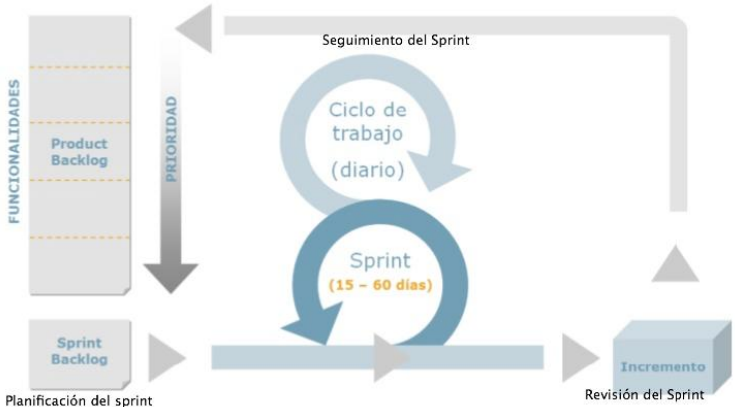

Fig. 5: Fases de SCRUM.

SCRUM es un proceso ágil y liviano cuyo desarrollo se realiza en forma iterativa e incremental. Es una metodología de desarrollo muy simple, que requiere trabajo duro porque no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto: Es un modo de desarrollo de carácter adaptable más que predictivo. Orientado a las personas más que a los procesos. Emplea la estructura de desarrollo ágil: incremental basada en iteraciones y revisiones. (Palacio, 2006).

A pesar de ser una metodología simple y que muchos expertos en SCRUM coincidan en que es mejor el desarrollo sin preocuparse por la arquitectura, hay que tener, al menos, una arquitectura mínima prediseñada y definida. No es recomendable hacer las cosas sin una guía, ya que no se prevén riesgos, algo fundamental en todo proyecto.

1.7 Bases tecnológicas

La construcción de la capa de lógica del negocio para la creación de RIA que incluyan escenarios tridimensionales interactivos conlleva al conocimiento de sus bases tecnológicas. A continuación se realiza un estudio de las mismas.

1.7.1 Ambiente Integrado de Desarrollo

Un Ambiente Integrado de Desarrollo (IDE por sus siglas en inglés) es un entorno de programación que consta de un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de usuario. Un IDE puede ser una aplicación independiente o puede formar parte de una o más aplicaciones existentes y compatibles. Es un programa compuesto por un conjunto de herramientas para un

programador que proporciona un marco de trabajo amigable para la mayoría de los lenguajes de programación.

Eclipse es un IDE de código abierto e independiente de la plataforma, diseñado desde cero para ser un sistema basado en funciones. (Linux Journal, 2004). Este IDE permite la refactorización, instalación y actualización de código y se puede utilizar como base para el desarrollo de aplicaciones. (Díaz, 2009). Eclipse proporciona un ambiente altamente personalizable que puede ser adaptado a las necesidades de los desarrolladores de *software*.

La construcción de RIA, a través de Eclipse, se puede realizar incluyendo el SDK (Software *Development Kit*) de código abierto de Flex al Eclipse, obteniendo de esta forma un entorno de desarrollo libre. También se pueden desarrollar RIA a través de *Adobe Flash Builder* que es un potente IDE construido en la parte superior de Eclipse.

Flex SDK proporciona soporte para *Flash Player 10.1* y tiempos de ejecución de *Adobe AIR* 2 y correcciones de errores críticos. Permite conexiones *HTTPService* y *WebService* y cuenta con el componente *Data Visualization* el cual proporciona el *Grid* de datos avanzados y las librerías para gráficos (*charts*). (Rovira, 2006).

Adobe Flash Builder por su parte es un IDE multiplataforma que incluye el *framework* de código abierto Flex, compiladores, biblioteca de componentes y depuradores. Es una plataforma multimedia que utiliza *plug-ins* para ofrecer contenido enriquecido a través de la Web, posibilitando a los desarrolladores de *software* crear RIA. Este IDE genera dos tipos de archivos: archivos de tipo *flash* (.SWF) para aplicaciones Web que pueden ser distribuidas a través de un servidor Web y se ejecutan dentro de un navegador cliente, y archivos tipo *Adobe AIR* (.air) para aplicaciones de escritorio que puede ser descargados e instalados como aplicaciones locales. (Daugherty, 2010).

Incluye compatibilidad con la codificación inteligente, la depuración y el diseño visual, presenta potentes herramientas de prueba que agilizan el desarrollo y hacen que las aplicaciones tengan un rendimiento más elevado. (Adobe Corporation, 2009). Además, es un IDE muy cómodo que permite diseñar las interfaces arrastrando y soltando los componentes y ver sus propiedades y funciones, cuenta con un excelente completamiento de código y la descripción de propiedades y métodos es muy detallada, también soporta la integración con otros lenguajes y servidores. (Santiago, 2010).

Adobe Flash Builder permite la creación de RIA utilizando Flex, logrando acelerar el desarrollo mediante funciones como códigos inteligentes, depuración interactiva estratificada y diseño visual del aspecto de la interfaz de usuario. El *framework* **Flex** permitir a los desarrolladores de aplicaciones Web construir RIA de forma fácil y rápida. Facilita la incorporación de funciones interactivas y dinámicas, servicios de aplicación, componentes para el diseño visual, redes de datos, servidor de comunicaciones, gráficos, efectos visuales y muchas funcionalidades más. Además cuenta con varios componentes y características que aportan funcionalidades tales como objetos remotos, arrastrar y soltar, columnas ordenables, gráficas, efectos de animación y otras interacciones simples.

Al realizar una aplicación Flex es necesario compilar en un único archivo SWF, el archivo MXML y el código *ActionScritp*. MXML es el lenguaje que utilizan los desarrolladores para definir el diseño, la apariencia y comportamiento de una aplicación Flex y *ActionScript* es un lenguaje orientado a objeto que define la lógica de la aplicación del cliente. (Stan, 2009).

1.7.2 Lenguaje de Programación

Un lenguaje de programación es independiente de la computadora en la que se utiliza. Cada lenguaje se representa en forma simbólica y está diseñado para describir el conjunto de acciones consecutivas que un programa debe ejecutar.

ActionScript 3.0 es un lenguaje de programación orientado a objetos basado en ECMAScript y administrado por el AVM (*ActionScript Virtual Machine*) en *Flash Player 9*, que proporciona un enfoque más robusto y apoyado en estándares. (Weggheleire, 2008). Como la mayoría de los lenguajes de programación, *ActionScript* tiene su propia gramática y reglas de puntuación, que determinan los caracteres y las palabras usadas para crear significado y el orden en que se pueden escribir. Se ha diseñado para facilitar la creación de aplicaciones complejas con conjuntos de datos voluminosos y bases de código reutilizables.

Este lenguaje está compuesto por el núcleo del lenguaje el cual se encarga de describir los elementos básicos del lenguaje, incluyendo tipos de datos, expresiones, bucles y condiciones; y por la API (del inglés *Application Programming Interface*) de *Flash Player* la cual describe las clases de *ActionScript* que son ejecutadas por el *Flash Player*. (Weggheleire, 2008). Las aplicaciones Flex con *ActionScript 3.0* se pueden desarrollar adicionando las sentencias de *ActionScript* a las etiquetas MXML, añadiendo bloques de

código con la etiqueta <mx:Script> en un fichero MXML o trabajando con archivos externos de *ActionScript*, tal vez como la fuente para un elemento <mx:Script> o como una clase personalizada. (Weggheleire, 2008).

ActionScript 3.0 no requiere declarar todos los tipos de datos, pero hacerlo ayudará a localizar los errores en la aplicación con mayor facilidad. En un entorno flexible, se pueden lograr los resultados deseados utilizando sólo MXML, usando sólo el código *ActionScript* o usando una combinación de código *ActionScript* y MXML, siendo esta última variante la más conveniente ya que el poder real de Flex no se desencadena hasta que combina MXML y su propio código *ActionScript 3.0*. (Brown, 2008). Por último, este lenguaje proporciona no sólo una mejora significativa en el rendimiento, sino también un modelo de programación más sólido que se presta al complejo desarrollo de RIA.

1.7.3 Bibliotecas de código

Las bibliotecas de código brindan una colección de subprogramas para el desarrollo de *software*. Contienen código y datos auxiliares, que brindan servicios a programas independientes, lo que permite la distribución y modificación del código además de los datos de forma modular. Existen bibliotecas de código para crear espacios en tres dimensiones (3D) en *ActionScript*, dentro de las que se encuentran principalmente: *Papervision3D, Away3D y Sandy 3D*.

Away3d es un motor 3D en tiempo real para *Flash* en *ActionScript 3.0,* iniciado por Alexander Zadorozhny y Rob Bateman. (Away3d, 2007). Publicado bajo una licencia Apache 2.0 y de *software* libre. Para la inicialización de objetos, *Away3D* utiliza constructores con un mínimo de parámetros, a menudo con un único parámetro *Object* que contiene todas las propiedades del objeto. (Titos, 2010). Esta biblioteca define las clases como secciones de código con formas y funciones específicas, agrupadas en paquetes que sostienen los diferentes tipos de clases. (Olsson, 2010).

Papervision3D es de *software* libre y publicado bajo una licencia MIT, que está considerada una licencia *open source*. Como la mayoría de los sistemas de modelado y visualización 3D, *Papervision* ofrece objetos simples que se pueden usar como bloques con los cuales construir escenas más complejas. Facilita la creación de vistas simples mediante una vista ya predefinida, de esta forma, bastaría con instanciar la clase "*BasicView*" con unos pocos parámetros. *Papervision* emplea constructores que generalmente admiten un gran número de parámetros, a la hora de inicializar objetos. (Titos, 2010).

Sandy 3d Engine es de *software* libre y un aspecto interesante, y que puede resultar útil a la hora de reutilizar en *Sandy* objetos 3D modelados con otro *software* (como *3ds Max*, *SketchUp* o *Maya*), es la habilidad que tiene *Sandy* 3d para importar archivos de "*collada*"**,** un formato XML abierto para el intercambio de información de objetos entre programas 3D. *Sandy 3D* no incluye ningún mecanismo para obtener información interna sobre el motor de 3D (cuadros por segundo, número total de polígonos en la escena, etc.) que podría ser útil durante el desarrollo, o para hacer pruebas de rendimiento. (Titos, 2010).

1.7.4 Herramientas de modelado

La mayoría de los sistemas de *software* son complejos, para entenderlos y administrarlos es necesario dividir el sistema en partes, representadas como modelos que describan sus elementos fundamentales. Por lo que un paso de avance en la realización de un *software* es la creación de modelos que presenten diversas perspectivas de un sistema. El Lenguaje Unificado de Modelado (UML) es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de *software*. (G. Booch, 2000). UML está compuesto por diferentes elementos gráficos que se combinan para conformar diagramas, además describe lo que se supone que hará el sistema, lo cual no significa que dice cómo implementarlo. Entre las herramientas que se utilizan para el modelado se encuentran el *Visual Paradigm* y el *Rational Rose Enterprise Edition*.

1.7.4.1 Visual Paradigm

Visual Paradigm es un modelador UML con soporte multiplataforma con una licencia legalmente pagada por la UCI que permite el diseño de sistemas con todo tipo de diagrama UML. Es una herramienta de IS asistida por computadora (CASE) que ofrece a los equipos de desarrollo de *software* un conjunto de herramientas necesarias para la captura de requisitos, la planificación de programas, la planificación de controles, las clases de modelado, modelado de datos, etc. (Visual Paradigm, 2010).

Visual Paradigm soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Esta herramienta de modelado UML ayuda a una construcción de aplicaciones de calidad, más rápida, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, generar código desde diagramas y generar documentación. También proporciona abundantes tutoriales, demostraciones interactivas y proyectos UML. (Free Donwload Manager, 2007).

1.7.4.2 Rational Rose Enterprise Edition

Rational Rose Enterprise Edition es una herramienta desarrollada por *Rational Corporation* basada en UML que permite construir y desarrollar los casos de uso en diferentes diagramas, modelando los flujos de trabajo por los que transita el desarrollo de un *software*. Permite la realización de los diferentes diagramas y la posterior generación del código, todo orientado a objetos. Posee librerías que facilitan la obtención de una ingeniería inversa sobre diferentes lenguajes como Java, C++, Corba, XML_DTD, ADA, Visual Basic. Es una herramienta que posibilita gestionar la evolución del ciclo de vida de un proyecto de *software*. Aligera la implementación al automatizar los modelos arquitectónicos, además, permite visualizar, entender, y refinar los requerimientos y arquitectura antes de introducirse en el código. (Zapata, 2005).

1.7.5 Servidores Web

Un servidor Web es un programa que se ejecuta continuamente y que permite alojar sitios y/o aplicaciones, que pueden ser accedidas por los clientes utilizando un navegador que se comunica con el servidor utilizando el protocolo HTTP (*hypertext markup language*). El servidor Web se encarga de responder a las peticiones realizadas por clientes o usuarios, entregando como resultado una página Web o información de todo tipo, de acuerdo a los comandos solicitados.

1.7.5.1 Servicios Informativos de Internet

Servicios Informativos de Internet o IIS (*Internet Information Services*) es un servidor propietario y privativo que sólo funciona para el sistema operativo *Microsoft Windows*. Permite una autenticación robusta y segura de los usuarios, los servicios que ofrece incluyen el protocolo de transferencia de noticias a través de la red (NNTP), el protocolo de transferencia de archivos (FTP) y el protocolo simple de transferencia de correo (SMTP). Proporciona las herramientas y funciones necesarias para administrar de forma sencilla un servidor web seguro

1.7.5.2 Apache

Apache es un poderoso servidor HTTP de código abierto, que permite la creación de dominios virtuales lo cual tiene una gran importancia cuando se trata de tener varios sitios Web dentro del mismo servidor. Entre sus características más destacadas están que es seguro, flexible, multiplataforma y presenta buen rendimiento y escalabilidad. Tiene una arquitectura modular que posibilita adicionar o quitar

funcionalidades, permitiendo aumentar los servicios del servidor para varios usos. Gracias a que es modular se han desarrollado diversas extensiones, entre las que se destaca PHP.

Ese considerado uno de los servidores más utilizado y potente. Es un servidor Web configurable y de diseño modular, capaz de extender su funcionalidad y la calidad de sus servicios. Trabaja en conjunto con gran cantidad de lenguajes de programación interpretados como PHP, Perl, Java, JSP (Java Server Pages) y otros lenguajes de script, que son el complemento ideal para los sitios web dinámicos. (Morales, 2009). Una ventaja importante de este servidor es que al ser tan popular y utilizado, es posible encontrar gran cantidad de documentos, ejemplos y ayuda en internet en todos los idiomas.

1.8 Conclusiones

En el capítulo que concluye se han abordado temas fundamentales relacionados con la propuesta de capa de lógica del negocio para la construcción de RIA que incluyan escenarios tridimensionales interactivos, entre ellos los de SA, DSSA. Se realizó un estudio detallado de los principales estilos arquitectónicos y patrones de diseño, caracterizando los más utilizados para tomar posición de los más idóneos para dar solución al problema planteado. Se detallaron elementos importantes a tener en cuenta sobre algunas de las metodologías de desarrollo de *software* existentes, herramientas y lenguaje de modelado y tecnología a utilizar en el desarrollo de la propuesta. A partir del estudio realizado se ganó en conocimiento, aspecto que es de gran importancia para realizar una mejor propuesta arquitectónica.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

2.1 Introducción

En el presente capítulo se establece la línea base de la arquitectura y se identifican el estilo arquitectónico y patrones de diseño utilizados. Se realiza una descripción de la propuesta de solución al problema planteado en el diseño teórico de la investigación, para ello se presentan las características principales de la capa de lógica del negocio y se describen los componentes de la misma, detallando cómo se utilizará la propuesta de bases tecnológicas. Además se establecen los requerimientos no funcionales de la aplicación.

2.2 Línea base de la arquitectura.

La línea base de la arquitectura tiene como propósito lograr un mejor entendimiento del sistema. Contiene elementos que ayudan a maximizar la abstracción en el diseño arquitectónico de la aplicación y es aquí donde se agrupan elementos de configuración que comparten un mismo estado, establece los principales patrones de diseño y estilos arquitectónicos utilizados, elementos de la arquitectura, así como las tecnologías y herramientas de *software* usadas para el desarrollo de la aplicación. La definición de la arquitectura base es un paso esencial en el desarrollo de una aplicación. Para el desarrollo de la capa de lógica del negocio se especificarán los patrones de diseño, el estilo arquitectónico y las tecnologías y herramientas propuestas para el desarrollo de la misma.

2.3 Alcance

La línea base de la arquitectura describe la estructura del sistema en un alto nivel de abstracción. El presente documento contiene los detalles del diseño de la capa de lógica del negocio. Se explica la utilización del estilo arquitectónico y los patrones de diseño que resuelven problemas que se pudieran presentar en el desarrollo de una aplicación. Se analizan las principales herramientas de desarrollo y como se adaptan a la solución propuesta.

2.4 Estilo arquitectónico seleccionado

La SA abarca decisiones importantes sobre la organización del *software*, los elementos estructurales que compondrán la aplicación y el estilo arquitectónico que guiará el desarrollo del sistema. La selección del

estilo arquitectónico debe estar dada por el tipo de aplicación que se desee desarrollar. Teniendo en cuenta esto, y que lo que se va a desarrollar es la capa de lógica del negocio, además, que es un estilo arquitectónico clásico para el desarrollo de RIA, se selecciona el estilo arquitectura en tres capas. Este estilo soporta un diseño basado en niveles de abstracción crecientes y ayuda a mantener un bajo acoplamiento en la aplicación, proporcionando una amplia reutilización. En la figura que se muestra a continuación, se presenta la estructura lógica del diseño de una arquitectura tres capas.

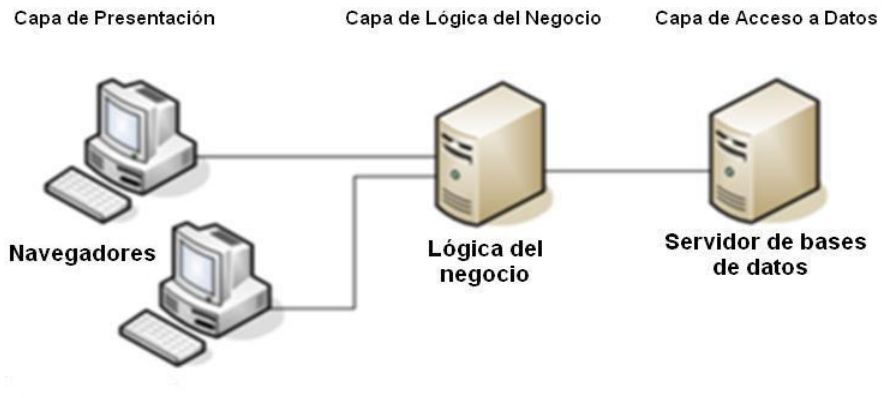

Fig. 6: Arquitectura en tres capas.

Esta arquitectura funciona de la siguiente forma: El usuario interactúa con la aplicación y realiza peticiones a través del navegador, en la capa de presentación. Esta última se encarga de realizar las peticiones a la capa de lógica del negocio, donde se aloja toda la lógica del negocio y que por lo general realiza peticiones a la capa de acceso a datos la cual se encarga de gestionar toda la información almacenada en las bases de datos, la capa de lógica del negocio procesa la petición realizada y envía la respuesta a la capa de presentación y esta a su vez la muestra al usuario.

Con la utilización de dicho estilo arquitectónico cada capa tendrá responsabilidades y funcionalidades independientes de las del resto de las capas. La capa de lógica de negocio, que es la que abarca este trabajo, se considera el corazón del *software*, como se mencionó en el capítulo anterior, ya que la misma se encarga de implementar las funcionalidades principales del sistema, es aquí donde se establecen las reglas del negocio que deben cumplirse y donde se encapsula toda la lógica del negocio.

2.5 Patrones de diseño seleccionados

Luego del estudio realizado en el capítulo anterior sobre algunos patrones de diseño, en el presente acápite se seleccionan y se especifica la utilización de cada uno de ellos durante el diseño de la capa de lógica del negocio.

Patrón Fachada. Se utilizará para proveer una interfaz sencilla entre la capa de presentación y una interfaz o grupo de interfaces más complejas en la capa de lógica del negocio. Este patrón se aplicará de manera opcional y en dependencia de la complejidad del negocio. Si se necesitan desarrollar varios componentes del negocio, dentro de la lógica de la aplicación a realizar, es necesario utilizarlo, para lograr ofrecer un fácil acceso y desacoplar al máximo la capa de lógica del negocio de las demás capas.

Patrón *Singleton*: Como el patrón *Singleton* garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella, se utilizará junto con el patrón fachada, para garantizar que exista una única instancia de la fachada.

Alta Cohesión: Se utiliza para delimitar las responsabilidades de cada uno de los componentes de la capa. También se pone de manifiesto en la estructura de la capa de forma general, ya que al ser independiente los cambios que se realicen sobre la misma no influyen en las demás capas y al mismo tiempo permita su reutilización.

Bajo Acoplamiento: El bajo acoplamiento consiste en que un objeto conozca lo menos posible sobre los detalles internos de otro y permita asignarle el mínimo de responsabilidad posible a los objetos del sistema. Por lo tanto ayudará a mantener débilmente acoplado los componentes de la capa, es decir, que las relaciones entre los componentes sea mínima, posibilitando que al realizar cambios en uno de ellos los demás se afecten lo menos posible y ayudando a mejorar el diseño del sistema.

Experto: En la capa de lógica del negocio, cuando la capa de presentación realice alguna petición, los componentes del negocio, que contienen la información necesaria para realizar las operaciones de la lógica del negocio, serían los encargados de dar respuesta a dicha petición. Utilizando este patrón la estructura de la capa puede llegar a ser más fácil de entender, mantener y ampliar, y brindar la posibilidad de reutilizar los componentes en futuras aplicaciones.

2.6 Metodología de desarrollo a utilizar

No existe una metodología universal para realizar con éxito cualquier proyecto de desarrollo de *software*. Compararlas entre sí no es algo fácil ya que una metodología puede no ser mejor ni peor que otra. La clave para optar por una u otra metodología es la visión de cómo plantear y desarrollar el proyecto.

Las metodologías analizadas tienen en común que evalúan el producto en desarrollo de forma periódica, potencian la interacción entre el equipo de desarrollo y el cliente y permiten la realización de cambios. Sin embargo la única que se centra en la arquitectura es AUP, lo cual es una característica fundamental para el desarrollo propuesto en la presente investigación. Además presenta una infraestructura flexible que permita hacer cambios y se puede adaptar fácilmente a cualquier proyecto, además, al ser una metodología basada en RUP, tiene bien definido los roles, artefactos, flujos de trabajo y actividades. Por las razones antes expuestas se define AUP como metodología para guiar el proceso de desarrollo de la propuesta de capa de lógica del negocio.

2.7 Lenguaje y herramienta de modelado a utilizar

Se define como lenguaje de modelado el UML y como herramienta el *Visual Paradigm* ya que es una potente herramienta CASE, fácil de instalar, usar y actualizar. Ofrece un conjunto de herramientas necesarias para la captura de requisitos, la planificación de programas y de controles, las clases de modelado y el modelado de datos. Soporta el ciclo de vida completo del desarrollo de *software*, es multiplataforma y cuenta con una licencia legalmente pagada por la UCI.

2.8 Selección de las tecnologías a utilizar

Para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos se propone como IDE el *Adobe Flash Builder 4* ya que ayuda a que las aplicaciones tengan un mayor rendimiento, cuenta con potentes herramientas de prueba e incluye compatibilidad con la codificación inteligente, la depuración y el diseño visual permitiendo agilizar el desarrollo de las RIA. Como lenguaje de desarrollo se propone *ActionScript 3.0* para facilitar la creación de aplicaciones complejas como lo son las RIA. Como biblioteca de animación 3d *Away3d*, basándose para su elección el Anexo 2 y teniendo en cuenta que la misma contiene una estructura sólida, donde cada paquete de clases brinda información sobre las funciones que se pueden realizar. Por último se propone el Apache como servidor Web, un potente servidor de código

abierto y multiplataforma, muy utilizado en el mundo actualmente. Para su selección se realizó una comparacion que se muestra en el Anexo 3 que ayudó a elegir el servidor seleccionado.

2.9 Estructura de la capa de lógica del negocio

La razón fundamental de proponer la capa de lógica del negocio radica en diferenciar y separar el comportamiento de las reglas del negocio de los detalles de implementación de infraestructura de acceso a datos. Logrando de esta forma incrementar la mantenibilidad del sistema, facilitando la actualización y posibilitando que se pueda sustituir la capa de acceso a datos provocando el menor impacto posible sobre el resto de la aplicación. Es importante destacar que en las aplicaciones complejas, el comportamiento de las reglas de negocio puede cambiar de manera frecuente y es muy importante poder modificar, construir y realizar pruebas sobre dichas capas de lógica del negocio de una forma fácil e independiente. Por lo que un aspecto principal a tener en cuenta es el mínimo acoplamiento entre esta capa y el resto de capas del sistema.

En una arquitectura en capas, cada capa contiene una serie de componentes que implementan la funcionalidad de la misma, los cuales deben ser coherente y débilmente acoplados. Los componentes de la capa de lógica del negocio implementan la funcionalidad principal del sistema y encapsulan toda la lógica de negocio relevante. La siguiente imagen muestra cómo queda estructurada la capa de lógica del negocio que se propone:

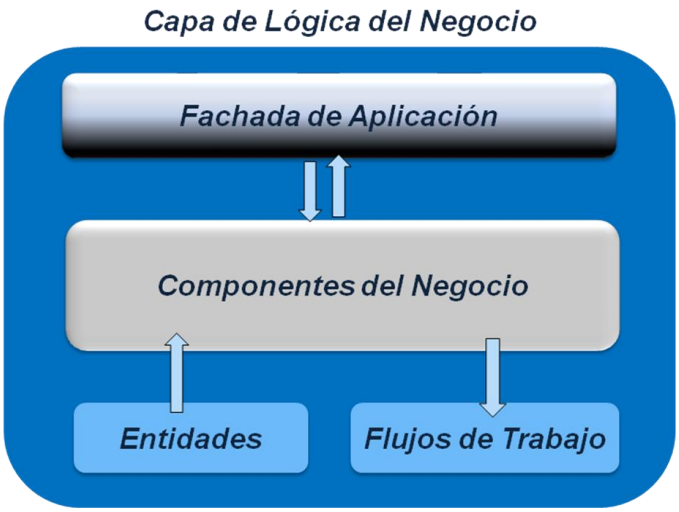

Fig. 7: Estructuración de la capa de lógica del negocio.

Autora: Liusba Cañete Núñez

Fachada de aplicación: Es utilizado para combinar múltiples operaciones de negocio en las operaciones basadas en el envío de un solo mensaje. Fachada de aplicación se considera un componente opcional, ya que depende de la complejidad del negocio que se desee desarrollar. Es decir, si la aplicación a desarrollar cuenta con una lógica del negocio compleja, que necesite de varios componentes de negocio, es necesario implementar una fachada de aplicación que se encargue del intercambio entre los diferentes componentes de negocio y la capa de presentación.

Componentes del negocio: Implementan la lógica y las reglas del negocio de la aplicación y realizan las tareas del negocio sin importar si un proceso de negocio consiste en un solo paso o si implica un flujo de trabajo. Es decir, después que los datos requeridos por el usuario pasan a la capa de lógica del negocio, pueden ser manipulados y transformados usando las reglas del negocio, según lo dictado por el negocio que se esté desarrollando. Los componentes del negocio utilizan flujos de trabajo y entidades del negocio para manejar e intercambiar datos con la capa de acceso a datos.

Con los **Flujos de trabajo del negocio** después que los componentes de interfaz de usuario recogen los datos requeridos por el usuario y pasan a la capa de lógica del negocio, la aplicación puede utilizar estos datos para llevar a cabo un proceso de negocio. Varios procesos de negocio implican varios pasos que se deben realizar en el orden correcto, y pueden interactuar entre sí a través de una orquestación. Estos procesos de negocio son, básicamente, operaciones que se realizan sobre la capa de acceso a datos, enviando los datos recogidos en la interfaz de usuario, o realizando peticiones de datos a dicha capa.

Por su parte las **entidades del negocio** se utilizan para obtener y transferir datos de entidades entre las capas de lógica del negocio y acceso a datos. Estos datos representan entidades de negocio del mundo real, tales como productos o pedidos. Las entidades del negocio que son utilizadas internamente por la aplicación son por lo general estructuras de datos, tales como conjuntos de datos (DataSets, DataReader) o lenguaje de marcado extensible (XML) corrientes, pero pueden ser también implementadas mediante clases personalizadas orientadas a objetos que representan las entidades del mundo real de las que la aplicación se encargará.

2.10 Metas y limitaciones de la arquitectura

En el presente acápite se exponen los requisitos no funcionales con los que debe cumplir la aplicación, así como los requisitos funcionales arquitectónicamente significativos y básicos que siempre se deben realizar

cuando se desarrollan RIA que incluyan escenarios tridimensionales interactivos. Estos requisitos muestran las metas y restricciones que se deben cumplir para que el diseño arquitectónico propuesto funcione de forma correcta.

2.10.1 Requisitos funcionales

Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. Describen lo que el sistema debe hacer y dependen del tipo de *software* que se desarrolle, de los posibles usuarios del *software* y del enfoque general tomado por la organización al redactar los mismos. (Sommerville, 2005). A continuación se presentan los requisitos funcionales presentes en el diseño arquitectónico propuesto para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos.

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

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

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

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

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

RF 2.1. El diseño de arquitectura propuesto debe permitir cargar y aplicar texturas a un modelo 3D.

RF 2.2. El diseño de arquitectura propuesto debe permitir definir los materiales a aplicar en un modelo 3D.

RF 2.3. 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.

2.10.2 Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener, las cuales hacen al producto atractivo, usable, rápido o confiable. Los requisitos no funcionales son importantes para que clientes y usuarios puedan valorar las características no funcionales del producto, además de marcar la diferencia entre un producto bien aceptado y uno con poca aceptación. A continuación se analizan los requisitos no funcionales presentes en la capa de lógica del negocio.

Requisitos de *software*:

- Se requiere como sistema operativo (SO): *Windows xp* o superior y Linux.

Requisitos de seguridad:

- La información manejada por el sistema está protegida de acceso no autorizado y divulgación.
- Se les garantiza a los usuarios autorizados el acceso a la información.
- Los mecanismos utilizados para lograr la seguridad no retrasarán a los usuarios para obtener los datos deseados en un momento dado.

Requisitos de portabilidad:

- La arquitectura es multiplataforma, por lo que podrá ser usada en cualquier sistema operativo (*Windows XP* o superior y Linux).

Requisitos de rendimiento:

- La lógica del negocio debe soportar varias peticiones al mismo tiempo y las respuestas a las mismas deben ser relativamente rápidas.

Requisitos de escalabilidad:

- Se desarrollará cada pieza de la capa en forma de componentes, utilizando patrones de diseño con el fin de facilitar su integración futura con el resto de las capas.

Restricciones en el diseño y la implementación:

- La capa de lógica de negocio permitirá una fácil integración o desintegración con las capas de presentación y acceso a datos.
- Se requiere como IDE de desarrollo el Adobe Flash Builder.
- Lenguaje de programación Actionscript 3.0.
- Servidor Web Apache.
- Se debe emplear en el desarrollo de la arquitectura el estilo arquitectura en capas.

Requisitos de mantenibilidad:

- La capa debe proporcionar un mínimo esfuerzo a la hora de localizar y arreglar un error o realizar una actualización a la misma.
- El proceso de agregar nuevas funcionalidades se realiza de forma sencilla y el sistema permitirá un fácil mantenimiento.
- La capa permitirá realizar modificaciones asegurando su extensibilidad y logrando mejores prestaciones.

2.11 Descripción de la capa de lógica de negocio

Con el fin de lograr una mejor comprensión del diseño, es necesario ver el sistema desde diferentes perspectivas, es por eso que la arquitectura se representa mediante varias vistas que se centran en aspectos concretos del sistema, abstrayéndose de los demás. Para la descripción de la misma se utilizará el modelo 4+1 Vistas, (Fig. 8), definido por la metodología propuesta (AUP). Este modelo define cuatro vistas principales: Vista lógica, vista de implementación, vista de despliegue y vista de proceso, y una vista más, la “+1”, que es la vista de casos de uso, la cual es considerada la vista rectora que describe las funcionalidades del sistema que más inciden sobre la arquitectura.

Fig. 8: Modelo 4+1 Vistas.

2.11.1 Vista de casos de uso

La vista de casos de uso representa un subconjunto del artefacto Modelo de casos de uso y describen los escenarios o casos de uso más significativos con las funcionalidades centrales del sistema. Los casos de uso arquitectónicamente significativos describen funcionalidades indispensables para el sistema y a través de los mismos se valida la arquitectura propuesta. En la vista de casos de uso se representa el diagrama de casos de uso con los casos de uso arquitectónicamente significativos para el sistema y una descripción de cada uno.

A continuación se presenta la vista de casos de uso, donde se muestran las funcionalidades básicas y arquitectónicamente significativas que siempre se realizarán al desarrollar RIA que incluyan escenarios tridimensionales interactivos.

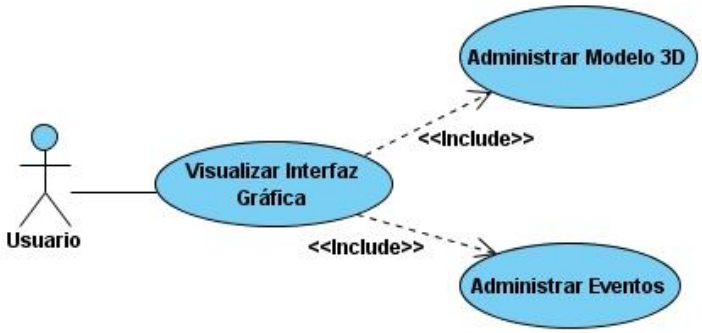

Fig. 9: Diagrama de Casos de Uso.

Autora: Liusba Cañete Núñez

A continuación se describen los casos de uso arquitectónicamente significativos, mostrados en el diagrama anterior:

Caso de Uso:	Visualizar Interfaz Gráfica	
Actores:	Usuario	
Resumen:	El caso de uso se inicia cuando el usuario solicita acceder a la aplicación o cuando realiza alguna acción que modifica la interfaz gráfica de la aplicación.	
Precondiciones:	El usuario debe haber solicitado acceder a la aplicación.	
Referencias	**RF 1, RF 1.1, RF 1.2, RF 1.3**	
Prioridad	Crítico	
Flujo Normal de Eventos		
Acción del Actor	**Respuesta del Sistema**	
1. El usuario solicita acceder a una aplicación enriquecida de internet que incluya escenarios tridimensionales interactivos.	1.1. El sistema inicializa una escena para mostrar en la aplicación. 1.2. El sistema define, dada la escena, la vista que será mostrada. 1.3. El sistema define la cámara con que se visualizará la vista.	
Flujos Alternos		
Acción del Actor	**Respuesta del Sistema**	
2. El usuario solicita modificar la interfaz gráfica de la aplicación.	2.1 El sistema inicializa una escena para mostrar en la aplicación. 2.2 El sistema define, dada la escena, la vista que será mostrada. 2.3 El sistema define la cámara con que se visualizará la vista.	
Relaciones	CU Incluidos	a) Administrar Modelos 3D b) Administrar Eventos
	CU Extendidos	
Poscondiciones	Se visualiza la interfaz gráfica de la aplicación.	

Tabla 1: Descripción textual del Caso de Uso Visualizar Interfaz Gráfica.

Caso de Uso:	Administrar Modelo 3D
Actores:	Usuario

Autora: Liusba Cañete Núñez

Resumen:	El caso de uso se inicia cuando se necesita cargar algún modelo 3D para ser mostrado en la interfaz gráfica.
Precondiciones:	El usuario debe haber solicitado mostrar una escena que contenga un modelo 3D.
Referencias	**RF 2, RF 2.1, RF 2.2, RF 2.3, RF 2.4**
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El usuario solicita visualizar una escena de la aplicación.	1.1. El sistema carga el modelo 3D. 1.2. El sistema carga las texturas a aplicar al modelo 3D. 1.3. El sistema define los materiales a aplicar al modelo 3D. 1.4. El sistema carga el mapa de colisiones para lograr una interacción en el modelo 3D.

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**

Relaciones	CU Incluidos	
	CU Extendidos	
Poscondiciones	Se tiene un modelo 3D con las texturas, los materiales y las colisiones definidas.	

Tabla 2: Descripción textual del Caso de Uso Administrar Modelo 3D.

Caso de Uso:	Administrar Eventos
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario realiza algún evento a través del mouse o el teclado.
Precondiciones:	
Referencias	**RF 3**
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El usuario al interactuar con la escena realiza un evento.	1.1 El sistema espera un evento realizado por el usuario. 1.2 Dependiendo del evento realizado el sistema realiza una acción.

Flujos Alternos

Autora: Liusba Cañete Núñez

Acción del Actor		Respuesta del Sistema
Relaciones	CU Incluidos	
	CU Extendidos	
Poscondiciones	El sistema después de captar el evento realiza una acción.	

Tabla 3: Descripción textual del Caso de Uso Administrar Eventos.

2.11.2 Vista lógica

La vista lógica representa un subconjunto del artefacto Modelo de diseño. Describe las clases más importantes, su organización en paquetes y subsistemas, y la organización de dichos paquetes y subsistemas en capas. En esta vista se representan los elementos de diseño más significativos para la arquitectura del sistema.

Es necesario tener en cuenta que el objetivo del presente trabajo no se enfoca en describir los elementos del diseño de una aplicación en específico, sino en brindar una vista lo más abstracta posible de cómo quedaría distribuida lógicamente una RIA que incluya escenarios tridimensionales interactivos, y en específico la vista de la capa de lógica de negocio, de tal manera que sea posible utilizarla en cualquier aplicación de este tipo.

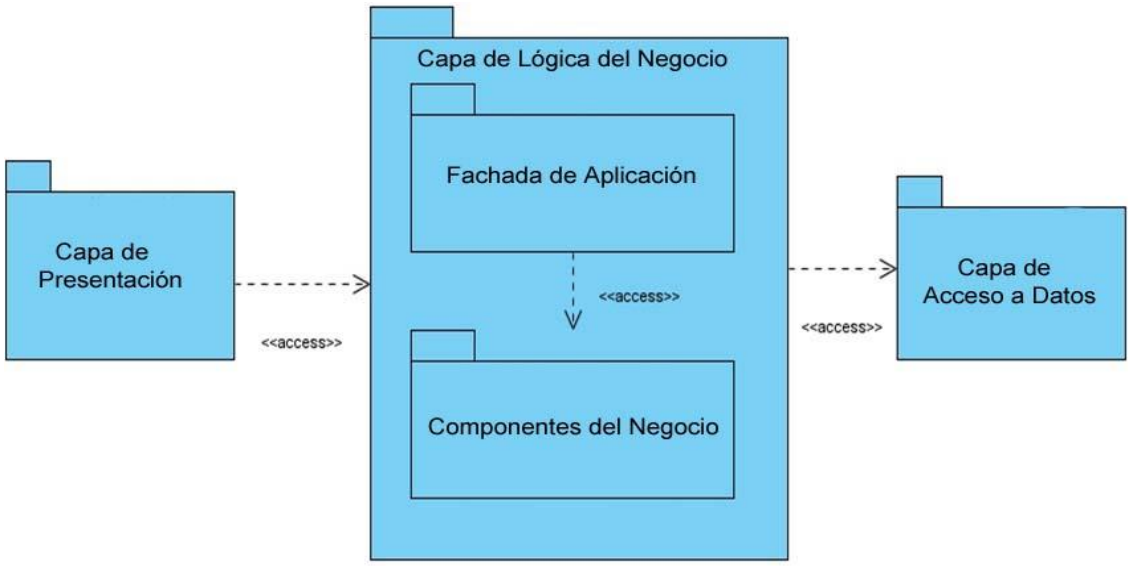

Fig. 10: Estructura en capas.

Autora: Liusba Cañete Núñez

Nombre	Estereotipo	Descripción
Capa de Presentación	Paquete	En el paquete Capa de Presentación se localizarán las clases que conforman la interfaz del sistema con la cual interactuará el usuario.
Capa de Lógica del Negocio	Paquete	En el paquete Capa de Lógica del Negocio se encontrarán el conjunto de clases encargadas de implementar la funcionalidad principal del sistema y encapsular toda la lógica de negocio relevante.
Capa de Acceso a Datos	Paquete	En el paquete Capa de Acceso a Datos comprenderá el conjunto de clases que controlan el acceso a los datos.

Tabla 4: Descripción de los paquetes.

Estructura de la capa de lógica del negocio.

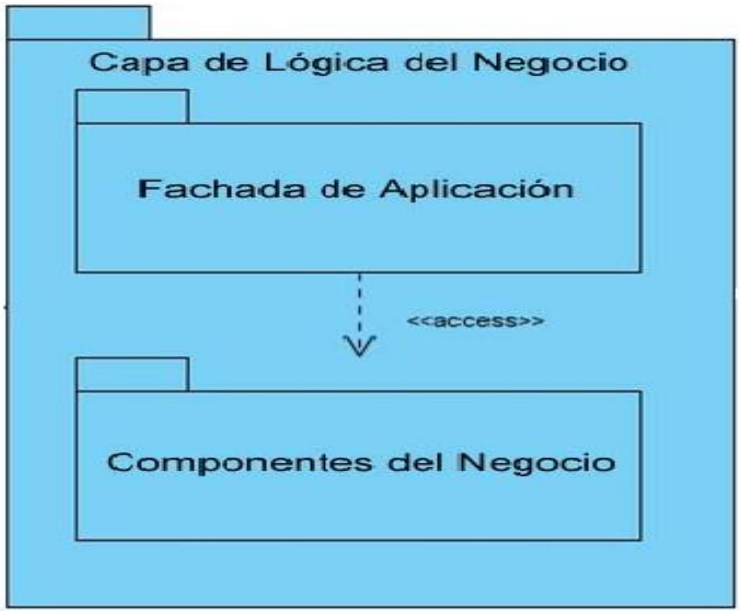

Fig. 11: Estructura lógica de la capa de lógica del negocio.

Como se muestra en la Fig. 11, la capa de lógica del negocio estará conformada por el paquete **Fachada de Aplicación** en el cual se encontrará la clase encargada de la comunicación entre los paquetes “Capa de Presentación” y “Capa de Lógica del Negocio”. En el mismo se reciben los datos recogidos en la “Capa de Presentación” y los agrupa de acuerdo a las necesidades de los diferentes componentes de negocio existentes, en caso de que la aplicación cuente con más de un componente de negocio. También pasa la

información procesada por el paquete “Capa de Lógica del Negocio” a la “Capa de Presentación” para que los usuarios puedan acceder a ella. **Componentes del Negocio** es el otro paquete que está contenido dentro de la capa de lógica del negocio, en dicho paquete se encontrarán las clases encargadas de implementar las reglas y la lógica del negocio de la aplicación a realizar. Cuenta además con una clase encargada de administrar los eventos generados por un dispositivo de entrada. El paquete de componentes del negocio cuenta con entidades y flujos de trabajos para el manejo e intercambio de datos con el paquete “Capa de Acceso a Datos”. Las entidades permiten que se pueda manejar en estructuras de datos definidas los datos que provienen de la capa de acceso a datos. Por su parte los flujos de trabajo permiten que se puedan realizar operaciones sobre la capa de acceso a datos, enviando los datos recogidos en la capa de presentación.

2.11.3 Vista de despliegue

La vista de despliegue proporciona una base para la comprensión de la distribución física de un sistema a través de nodos y propone como estarán distribuidos los componentes de la aplicación en ellos. Existe una traza directa del modelo de implementación, debido a que cada componente físico debe almacenarse en un nodo, lo cual implica además la asignación de tareas provenientes de la vista de procesos en los nodos.

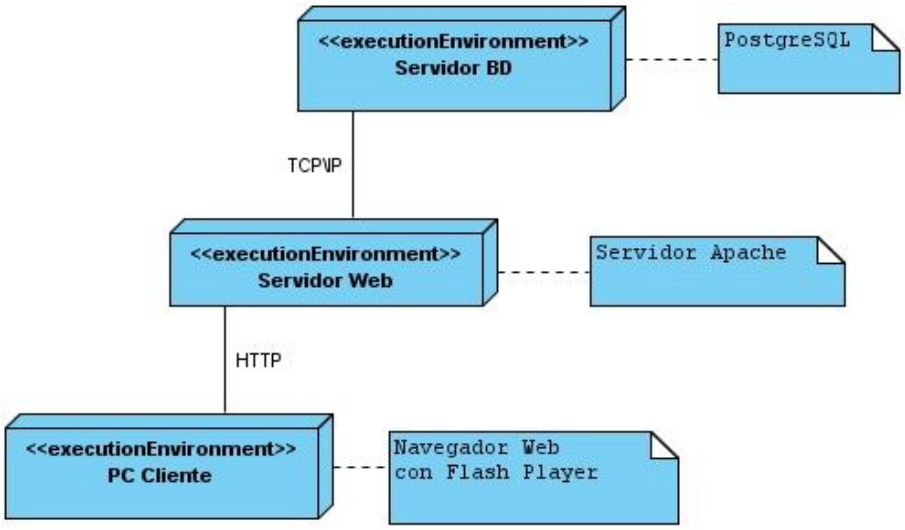

Fig. 12: Vista de Despliegue.

A continuación se describen los nodos físicos representados en el diagrama de despliegue anterior:

PC Cliente: Representa el nodo desde el cual los usuarios podrán acceder e interactuar con la aplicación utilizando un navegador Web con *Flash Player*.

Servidor Web: Representa el nodo donde se estarán ejecutando todas las funcionalidades del servidor Web Apache.

Servidor BD: En este nodo se estará ejecutando el servidor de base de datos PostgreSQL. La lógica del tratamiento de los datos no se implementará aquí sino en la misma aplicación en la capa de acceso a datos.

2.11.4 Vista de implementación

La vista de implementación provee una descripción de las principales capas y subsistemas de componentes de la aplicación. Proporciona una vista de la trazabilidad de los elementos de diseño de la vista lógica para la implementación.

En esta vista ocurre algo similar a la vista lógica, ya que, como se mencionó anteriormente, con este trabajo no se persigue explicar la descomposición de un *software* en específico, sino que se establece donde aparecerá cada uno de los componentes generados en la realización de una RIA que incluya escenarios tridimensionales interactivos, de la forma más abstracta posible para que pueda utilizarse en cualquier aplicación con las características antes mencionadas.

A continuación se muestra la descomposición de la vista de implementación de la capa de lógica del negocio.

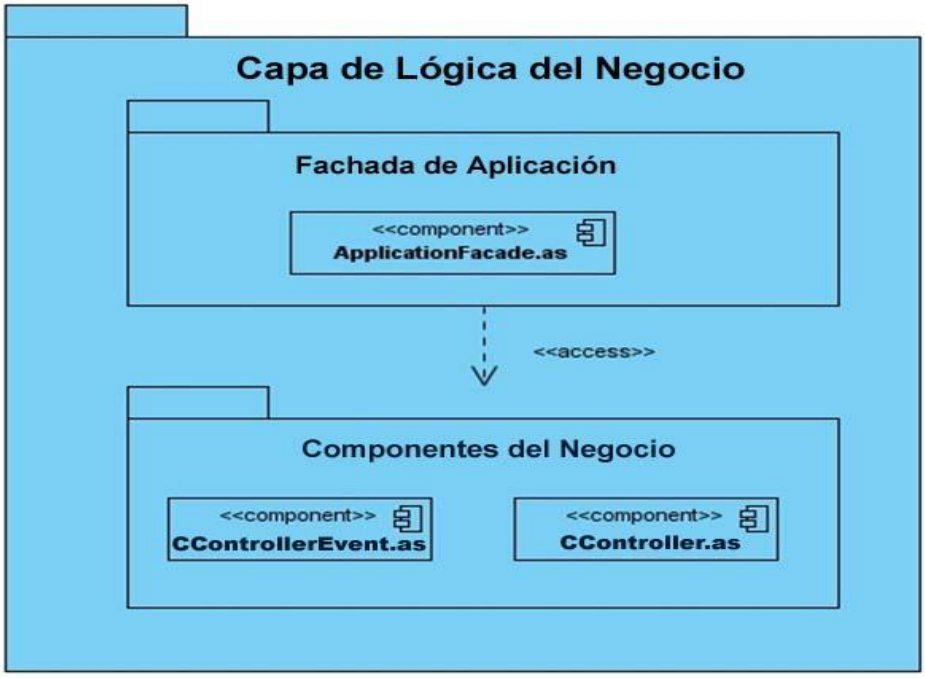

Fig. 13: Vista de implementación.

2.11.5 Vista de procesos

La vista de procesos describe los aspectos de concurrencia y sincronización del diseño. Provee una base para el entendimiento de la organización de los procesos de un sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos.

Esta vista se representa cuando el sistema cuenta con procesos concurrentes o hilos. En el caso del presente trabajo, la solución que se propone no presenta procesos concurrentes por lo que no se realiza la representación de dicha vista.

2.12 Conclusiones

En el capítulo que concluye se definió la línea base de la capa de lógica del negocio para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos, para ello se tuvieron en cuenta aspectos arquitectónicamente significativos que tienen que ver con las tecnologías y herramientas definidas para el desarrollo de la propuesta. Se seleccionaron los patrones de diseño, metodología y estilo arquitectónico a utilizar, además de realizarse la descripción de la estructura de la capa de lógica de negocio propuesta,

mostrándose las metas y restricciones que se deben cumplir para que la arquitectura propuesta funcione de forma correcta, además se realizó la representación arquitectónica a través del modelo 4+1 Vistas. A partir de los resultados obtenidos ya está todo disponible para la utilización de la capa de lógica del negocio propuesta.

CAPÍTULO 3: EVALUACIÓN DE LA CAPA DE LÓGICA DEL NEGOCIO

3.1 Introducción

En este capítulo se describe, a manera de síntesis, la necesidad de la evaluación arquitectónica dentro del desarrollo de sistemas de *software*, sus costos y beneficios. Se expondrán las etapas en las que se evalúa la SA, así como las técnicas existentes utilizadas para realizar dichas evaluaciones. Se presentan los principales atributos de calidad por los que puede ser evaluada una arquitectura y se describen algunos de los métodos existentes para determinar en qué medida se cumplen los atributos de calidad establecidos para la evaluación. Luego se pasará a evaluar el diseño arquitectónico propuesto con el fin de conocer si es adecuado o no para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos.

3.2 Evaluando la SA

La arquitectura es un aspecto importante dentro del desarrollo de un *software*, por lo que es conveniente realizar actividades de verificación de la misma de forma temprana, para identificar problemas que podrían resultar muy costosos de eliminar en etapas posteriores. No cuidar de los aspectos relacionados con el desarrollo de la arquitectura puede resultar en sistemas que no cubren las expectativas de los clientes y de la organización de desarrollo; por lo tanto, la evaluación del diseño de la arquitectura es una actividad fundamental dentro del proceso de desarrollo de un *software* y aumenta las probabilidades de tener un sistema de calidad.

3.2.1 ¿Por qué es necesario evaluar una SA?

Realizar una evaluación a una SA es la forma más económica de evitar desastres, mientras más temprano se encuentre un problema en un proyecto de *software*, más rápido podrá solucionarse. “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*”. (Gustavo Andrés Brey, 2005).

El propósito de realizar evaluaciones a la arquitectura es para analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de *software* resultante, verificar que los requisitos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad. Cabe señalar que los requisitos no funcionales también son llamados atributos de calidad. (Gómez, 2007).

La evaluación de una SA 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 lo que 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 SA pueden tener distintos objetivos, dependiendo de la situación en la que se encuentre el arquitecto y la aplicabilidad de las técnicas que emplea. Algunos de estos objetivos son: cualitativos, cuantitativos y máximos y mínimos teóricos. (Erika Camacho, 2004).

Luego de evaluar una SA, se pueden tomar algunas decisiones importantes como: si se puede seguir el proyecto con las áreas de debilidad dadas en la evaluación, si hay que reforzar la SA o si hay que comenzar de nuevo toda la arquitectura. (Peláez, 2009).

3.2.2 Atributos de calidad

Los atributos de calidad son los aspectos del sistema, que en general no afectan directamente a la funcionalidad necesitada, sino que definen la calidad y las características que el sistema debe soportar. (Nicolás Passerini, 2009). Estos pueden definirse como aquellas características deseadas en determinado sistema, que definen las propiedades de los servicios que se brindan. (Gregory Abowd, 2007).

Los atributos de calidad, según Bass y sus colegas, se clasifican en dos categorías (Gregory Abowd, 2007):

Observables vía ejecución los cuales determinan del comportamiento del sistema en tiempo de ejecución. (Carrascoso Puebla, 2009).

No observables vía ejecución los cuales se establecen durante el desarrollo del sistema. (Carrascoso Puebla, 2009).

3.2.3 ¿Cuándo una arquitectura puede ser evaluada?

La evaluación de una arquitectura puede realizarse en cualquier momento. Para la realización de la misma existen dos variantes que se agrupan en dos etapas distintas: la evaluación temprana y la evaluación tardía.

La evaluación temprana se puede realizar cuando no es necesario que la arquitectura esté completamente especificada para efectuar la evaluación, y abarca desde las fases tempranas de diseño y a lo largo del desarrollo.

La evaluación tardía se puede realizar cuando la arquitectura del sistema se encuentra establecida y la implementación se ha completado. Este es el caso general que se presenta al momento de la adquisición de un sistema ya desarrollado. (Carrascoso Puebla, 2009).

Existen dos reglas de oro para determinar el momento de realizar evaluaciones:

1. Realizar una evaluación cuando el equipo de desarrollo inicia a tomar decisiones que afectan directamente a la arquitectura.

2. Cuando el costo de no tomar estas decisiones podría tomar más, que el costo de realizar una evaluación. (Gómez, 2007).

3.2.4 Resultado de la evaluación

La evaluación de una arquitectura no produce resultados cuantitativos, en su lugar, ayuda a encontrar debilidades. Luego de efectuada la misma se debe elaborar un reporte que debe presentarse como un documento preliminar, con la finalidad de que se corrija por las personas que participaron en la evaluación. El contenido del reporte responde a dos tipos de preguntas: ¿Se ha diseñado la arquitectura más apropiada para el sistema? ¿Cuál de las arquitecturas propuestas es la más apropiada para el sistema a construir? El reporte no solo responde las preguntas antes mencionadas, también indica el grado en que se cumplieron los atributos de calidad. (Gómez, 2007).

Una arquitectura es adecuada cuando cumple dos criterios:

- El sistema resultante cumple con los atributos de calidad definidos.

- El sistema puede ser construido con los recursos disponibles para el mismo, es decir, es construible.

La razón fundamental de realizar la evaluación de una arquitectura es conocer como los atributos de calidad son afectados por las decisiones del diseño arquitectónico.

3.3 Técnicas de evaluación

Existe un grupo de técnicas que posibilitan evaluar arquitecturas, las cuales se clasifican en cualitativas y cuantitativas. (Gustavo Andrés Brey, 2005). Las técnicas cualitativas o de cuestionamiento utilizan preguntas cualitativas para hacerle a la arquitectura y son utilizadas cuando la arquitectura está en construcción. Para realizar evaluaciones con estas técnicas se pueden utilizar cuestionarios, listas de verificación o escenarios. Por otra parte, las técnicas cuantitativas se usan cuando la arquitectura ya ha sido implantada, las mismas proponen realizarle medidas cuantitativas a la arquitectura, para ello utilizan métricas, simulaciones, prototipos, experimentos o modelos matemáticos.

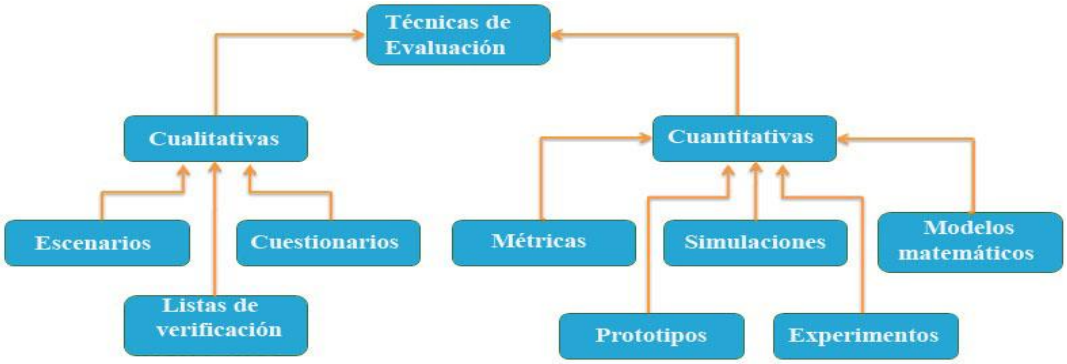

Fig. 14: Clasificación de las Técnicas de Evaluación.

3.4 Métodos de evaluación de SA

Para la evaluación de una SA existen gran cantidad de métodos, dentro de los que podemos encontrar: El Método de Análisis de Arquitectura de *Software* (SAAM), el Método de Análisis de Acuerdos de Arquitectura de *Software* (ATAM) y el Método de Revisión Intermedio de Diseño (ARID). Cada uno de ellos cuenta con características específicas, estos métodos consisten en una serie de actividades o pasos que se deben desarrollar para determinar en qué medida se cumplen los atributos de calidad establecidos

para evaluar la arquitectura. A continuación se explican los métodos antes mencionados para luego seleccionar el que será utilizado en la evaluación de la capa de lógica del negocio propuesta.

3.4.1 Método de Análisis de Arquitectura de Software

El Método de Análisis de Arquitectura de *Software* o SAAM (del inglés Software *Architecture Analysis Method*) es un método para el análisis de una SA basado en escenarios que permite analizar arquitecturas con respecto al logro de los atributos de calidad. En SAAM se analiza la arquitectura con respecto a qué tan bien o con qué facilidad satisface las restricciones impuestas por cada escenario. (Mojica, 2008).

Algunos de los beneficios de utilizar el método SAAM son: Los interesados comprenden con facilidad las arquitecturas evaluadas, la documentación es mejorada y el esfuerzo y el costo de los cambios pueden ser estimados con anticipación. Por otra parte, SAAM presenta algunas debilidades: La generación de escenarios está basada en la visión de los interesados, no provee una métrica clara sobre la calidad de la arquitectura evaluada y el equipo de evaluación confía en la experiencia de los arquitectos para proponer arquitecturas candidatas. (Medina, 2008).

El método SAAM consta de seis pasos, los cuales se muestran en la siguiente tabla. (Medina, 2008).

Pasos del método SAAM	
Paso 1	Desarrollo de escenarios
Paso 2	Descripción de la arquitectura
Paso 3	Clasificación de escenarios
Paso 4	Evaluación de escenarios
Paso 5	Interacción de escenarios
Paso 6	Evaluación general

Tabla 5: Pasos del método SAAM.

3.4.2 Método de Análisis de Acuerdos de Arquitectura de Software

Método de Análisis de Acuerdos de Arquitectura de *Software*, ATAM por sus siglas en inglés, (*Architecture Tradeoff Analysis Method*) es un método de evaluación de arquitecturas basado en escenarios y

cuestionarios, que permite evaluar qué tan bien un atributo de calidad es soportado por la arquitectura, y provee un entendimiento de cómo los atributos de calidad interactúan. (Gustavo Andrés Brey, 2005). El propósito de ATAM es evaluar las consecuencias de decisiones arquitectónicas a la luz de los requerimientos de calidad y las metas del negocio. Además este método no tiene por intención proporcionar análisis precisos, sino que pretende descubrir los riesgos originados por decisiones arquitectónicas, como también identificar y documentar desventajas. (Mojica, 2008).

Este método de evaluación es más profundo para evaluar aspectos más relacionados con la arquitectura, como el rendimiento o la confiabilidad. Además indica cuán bien una arquitectura particular satisface las metas de calidad.

ATAM consta de nueve pasos, divididos en cuatro grupos, los cuales consisten en un primer grupo de **presentación**, donde se intercambia información del sistema, un segundo grupo de **investigación y análisis**, donde se valoran los atributos de calidad claves uno a uno con las propuestas arquitectónicas, un tercer grupo de **pruebas** donde se revisan los resultados obtenidos contra las necesidades relevantes de los *stakeholders*, y un cuarto y último grupo de **reporte**, donde se presentan los resultados del ATAM. A continuación se presentan cada uno de estos grupos con los pasos que contiene cada uno. (Medina, 2008).

Fases	Pasos
Fase 1: Presentación	1. Presentación del ATAM.
	2. Presentación de las metas del negocio.
	3. Presentación de la arquitectura.
Fase 2: Investigación y análisis	4. Identificación de los enfoques arquitectónicos
	5. Generación del UtilityTree
	6. Análisis de los enfoques arquitectónicos
Fase 3: Pruebas	7. Lluvia de ideas y establecimiento de prioridad de escenarios.
	8. Análisis de los enfoques arquitectónicos
Fase 4: Reporte	9. Presentación de los resultados

Tabla 6: Fases del método ATAM.

3.4.3 Método de Revisión Intermedio de Diseño

Método de Revisión Intermedio de Diseño, ARID por sus siglas en inglés, (*Active Reviews for Intermediate Design*) 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. (Erika Camacho, 2004). Este es un método de análisis de arquitecturas resultado de combinar los métodos ADR y ATAM.

ARID resulta más ligero que ATAM y es útil para ser aplicado en etapas más tempranas del diseño arquitectónico e incluso sobre partes del sistema en lugar del sistema completo. (Clements, 2000). La evaluación se basa en detallar el funcionamiento del sistema en una serie de escenarios predefinidos, a través de lo cual se pueden identificar posibles puntos problemáticos de la arquitectura. Su utilidad no se encuentra en la sustitución de ATAM, sino que más bien prepara el camino en sistemas en los que, por su complejidad, puedan ser necesarias las revisiones de la arquitectura en etapas intermedias del diseño. (Méndez, 2008).

Este método se basa en ensamblar el diseño de los *stakeholders* para articular los escenarios de usos importantes o significativos, y probar el diseño para ver si satisface los escenarios. 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*. (Gómez, 2007). ARID consta de 9 pasos agrupados en dos fases: actividades previas y evaluación.

Fases	Pasos
Fase 1: Actividades previas	Paso 1: Identificación de los encargados de la revisión.
	Paso 2: Preparar el informe de diseño.
	Paso 3: Preparar los escenarios base.
	Paso 4: Preparar los materiales.
Fase 2: Evaluación	Paso 5: Presentación del ARID.
	Paso 6: Presentación del diseño.
	Paso 7: Lluvia de ideas y establecimiento de prioridad de escenarios.
	Paso 8: Aplicación de los escenarios.
	Paso 9: Resumen.

Tabla 7: Fases del método ARID

Autora: Liusba Cañete Núñez

3.4.4 Selección del método de evaluación

En función de evaluar la propuesta arquitectónica realizada, luego de analizar cada uno de los métodos de evaluación antes expuestos y con la ayuda de la tabla comparativa (Kazman, 2001) presente en el Anexo 4, se decide utilizar el método ARID para la evaluación de la propuesta de capa de lógica del negocio realizada. Se escoge este método ya que el mismo posibilita utilizar los atributos de calidad a conveniencia del diseño a evaluar, permite realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo, es un método de bajo costo, gran beneficio y fácil de usar, además, como resultado de la aplicación del mismo se consigue un diseño de alta fidelidad.

3.5 Evaluación de la SA propuesta

La evaluación de la capa de lógica del negocio propuesta se realizará de forma temprana empleando el método ARID. Este método utiliza la técnica de evaluación basada en escenarios con el instrumento perfiles, donde se hace un análisis de los principales escenarios arquitectónicos y se evalúa el grado en que los atributos de calidad se satisfacen en cada uno de los escenarios definidos. La técnica de evaluación basada en escenarios consiste en crear escenarios donde exista un contexto determinado y una respuesta. Un escenario es una breve descripción de la interacción de uno de los involucrados en el desarrollo del sistema con este.

Teniendo en cuenta que el arquitecto de software fue el encargado de la definición de la arquitectura y entiende los procesos descritos en la misma, siendo parte del equipo de desarrollo de la propuesta de diseño arquitectónico, se consideró omitir los pasos de la primera fase del método, pasando directamente a aplicarse los tres últimos pasos de la Fase 2. Definiendo para la evaluación los siguientes atributos de calidad: Portabilidad, Integrabilidad, Reusabilidad, Interoperabilidad y Mantenibilidad. Para la selección de los mismos se tuvieron en cuenta los atributos propuestos por el Modelo ISO/IEC 9126-1:2001.

Atributo de calidad	Perfil	Escenario
Portabilidad	Portabilidad	Migración de SO.
Relación atributo de calidad-escenario.		
Teniendo en cuenta que la **Portabilidad** es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación, pudiendo ser estos ambientes hardware, *software* o una combinación de ambos, el diseño arquitectónico propuesto podrá ser ejecutado en cualquier SO ya que el lenguaje de programación definido para el desarrollo de la capa de lógica de negocio es *ActionScript 3.0* soportado por el IDE *Adobe Flash Builder 4*, el cual es multiplataforma y el servidor Web definido es Apache, multiplataforma también. Esto permite que se pueda desarrollar en cualquier SO, logrando su uso sin tener restricciones que provoquen que no pueda ser utilizado, tributando en gran medida, a que el misma sea escalable.		

Tabla 8: Evaluando el Atributo de Calidad Portabilidad.

Atributo de calidad	Perfil	Escenario
Integrabilidad	Integrabilidad	Integración de los componentes.
Relación atributo de calidad-escenario.		
La **Integrabilidad** es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al integrarlos. La integración de los componentes de la capa de lógica del negocio propuesta no será un problema debido a que todos los programadores utilizarán un mismo estándar de codificación.		

Tabla 9: Evaluando el Atributo de Calidad Integrabilidad.

Atributo de calidad	Perfil	Escenario
Reusabilidad	Reusabilidad	Capa de lógica del negocio.
Relación atributo de calidad-escenario.		
La **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. Los componentes de la capa de lógica del negocio pueden ser reutilizados en cualquier RIA que incluya escenarios tridimensionales interactivos, por lo que la capa de lógica del negocio en su totalidad podrá ser reutilizada para el desarrollo de cualquier aplicación de este tipo.		

Tabla 10: Evaluando el Atributo de Calidad Reusabilidad.

Atributo de calidad	Perfil	Escenario
Funcionalidad	Interoperabilidad	Interoperabilidad con RIA que incluyan escenarios tridimensionales interactivos.
Relación atributo de calidad-escenario.		
Partiendo de que la **Interoperabilidad** es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema, y teniendo en cuenta que la reutilización de componentes es un factor fundamental en el diseño arquitectónico propuesto, se podrá utilizar la capa de lógica del negocio propuesta en la realización de cualquier RIA que incluya escenarios tridimensionales interactivos, por lo que todos los cambios que se realicen en la capa estarán en función de esto.		

Tabla 11: Evaluando el Atributo de Calidad Funcionalidad.

Atributo de calidad	Perfil	Escenario
Mantenibilidad	Modificabilidad	Realizar modificaciones en la capa de lógica del negocio.
Relación atributo de calidad-escenario.		
Teniendo en cuenta que la **Modificabilidad** es la habilidad de realizar cambios futuros al sistema, a la capa de lógica del negocio propuesta se le podrán realizar los cambios que sean necesarios en un futuro, sin que esto signifique un gran impacto en el resto de la aplicación. La misma debe ser modificable, permitir agregar nuevas funcionalidades y reutilizar los métodos definidos en las clases controladoras. El desarrollo de la capa es flexible adecuándose a cambios para extender, cambiar o eliminar funcionalidades al sistema a desarrollar, sin necesidad de volver a escribir los programas y provocando la menor alteración al sistema en su totalidad. Además la capa de lógica del negocio propuesta proporciona un mínimo esfuerzo para localizar y corregir un error, la misma permitirá realizar modificaciones provocando el menor costo posible en el resto del sistema. El uso de los patrones de diseño posibilita mantener un diseño claro. Además, la capa de lógica del negocio forma parte de una arquitectura en tres capas, lo que trae consigo que realizar modificaciones sobre una capa tenga las menores consecuencias posibles sobre el funcionamiento de otra.		

Tabla 12: Evaluando el Atributo de Calidad Mantenibilidad.

La evaluación realizada refleja que el diseño arquitectónico propuesto cumple con los atributos de calidad requeridos, donde se puede concluir que la capa de lógica del negocio propuesta se puede desarrollar en cualquier SO. Los componentes de la misma podrán integrarse sin dificultades, presentan altos niveles de

reutilización y permiten agregar nuevas funcionalidades, siendo una capa de lógica del negocio flexible que permite ampliar, cambiar o eliminar funcionalidades a la aplicación a desarrollar. Se puede agregar además que sobre la capa propuesta se pueden realizar modificaciones sin que esto provoque un gran impacto en el resto de la aplicación. Por lo anteriormente planteado se puede afirmar que el diseño arquitectónico propuesto, dentro de una etapa temprana de desarrollo, es adecuado para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos.

3.6 Conclusiones

Al concluir este capítulo se han analizado elementos importantes relacionados con la evaluación de la SA, para esto se realizó un estudio de las técnicas y métodos de evaluación, analizando porqué atributos puede ser evaluada una SA. La evaluación de la misma permite identificar el impacto que tiene la arquitectura sobre los atributos de calidad definidos, posibilitando detectar problemas, debilidades y puntos de interés para corregirlos a tiempo. Finalmente se evaluó la capa de lógica de negocio de manera parcial, a través del método ARID, concluyendo que la misma es adecuada para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos.

CONCLUSIONES GENERALES

AL finalizar la realización del presente trabajo y luego de analizar todo el contenido expuesto, se arribó a las siguientes conclusiones:

- Se obtuvo la propuesta de la capa de lógica del negocio de una arquitectura para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos.
- Se definió la línea base de la arquitectura que tiene la importancia de concretar las funcionalidades que va a tener el sistema y por la cual se debe regir el desarrollo de RIA que incluyan escenarios tridimensionales interactivos.
- Se analizaron elementos fundamentales relacionados con la evaluación de la SA. Se observaron una serie de atributos de calidad por los que puede ser evaluada la arquitectura, de los que se seleccionaron para la evaluación de la capa de lógica del negocio los atributos: Portabilidad, Integrabilidad, Reusabilidad, Interoperabilidad y Mantenibilidad.
- La evaluación del diseño arquitectónico propuesto mediante el método ARID, utilizando la técnica basada en escenarios, condujo a que se considerara adecuada la capa de lógica del negocio propuesta para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos.

RECOMENDACIONES

A partir del trabajo realizado se proponen las siguientes recomendaciones:

- Profundizar y refinar la propuesta de capa de lógica del negocio durante todo el ciclo de desarrollo del *software*, con el objetivo de adaptarse a posibles cambios y obtener mejoras.
- Aplicar los resultados de la investigación, es decir, que la capa de lógica del negocio propuesta sea utilizada en el desarrollo de alguna RIA que incluya escenarios tridimensionales interactivos.
- En etapas posteriores realizar la evaluación tardía de la capa de lógica del negocio propuesta, garantizando así, la obtención de un resultado completo y detallado de los riesgos que puedan afectar la propuesta realizada.
- Integrar la capa de lógica del negocio con las capas de presentación y acceso a datos para conformar la arquitectura.

REFERENCIAS BIBLIOGRÁFICAS

Adobe Corporation. 2009. Adobe. [Online] 07 14, 2009. [Cited: 01 28, 2011.] http://www.adobe.com/es/products/flashbuilder.

Alba, Julio. 2008. SOA: ARQUITECTURA ORIENTADA AL SERVICIO . 2008. 0210-3923.

Álvarez, Miguel Angel. 2009. desarrolloweb.com. [En línea] 23 de Diciembre de 2009. [Citado el: 25 de noviembre de 2010.] http://www.desarrolloweb.com/articulos/modelo-vista-controlador-codeigniter.html.

Away3d. 2007. Away3d. [En línea] 2007. [Citado el: 16 de 2 de 2011.] http://code.google.com/p/away3d/.

Bachmann. 2000. Technical concepts of component-based software engineering., 2000.

Barzanallana, Rafael. 2006. Universidad de Murcia. [Online] 06 17, 2006. [Cited: 11 30, 2010.] http://www.um.es/docencia/barzana/IAGP/Iagp2.html.

Brown, Charles E. 2008. *The Essential Guide to Flex 3.* New York : friendsof, 2008. 978-1-59059-950-1.

Carrascoso Puebla, Enrique Chaviano Gómez y Anisleydi Céspedes Vega. 2009. GestioPolis.com. [En línea] 7 de Mayo de 2009. [Citado el: 5 de Abril de 2011.] http://www.gestiopolis.com/administracion-estrategia/procedimiento-para-la-evolucion-de-las-arquitecturas-de-software.htm.

Cataldi, Ing. Zulma. 2000. Metodología de diseño, desarrollo y evaluación. 2000. 960-34-0204-2.

Clements. 2000. Active Reviews for Intermediate Designs. Software Engineering Institute., 2000.

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

CXO Community Latam. 2009. CXO Community. [Online] 09 29, 2009. [Cited: 12 1, 2010.] http://cxo-community.com.

Daugherty, Eric. 2010. DZone. [Online] 03 22, 2010. [Cited: 12 9, 2010.] http://refcardz.dzone.com/refcardz/getting-started-adobe-flash?oid=hom19969.

Díaz, Dayrien Corrales. 2009. monografias.com. [Online] 04 20, 2009. [Cited: 01 20, 2011.] http://www.monografias.com/trabajos69/tecnicas-desarrollo-servicios-Web-linux/tecnicas-desarrollo-servicios-Web-linux.shtml?monosearch.

Dijkstra, Edsger. 1968. *"The Structure of the THE Multiprogramming system."*. 1968.

Doberkat, Ernst-Erich. 2002. Pipes and filters: Modelling a software architecture through relations. 2002.

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

Estudis, Josep Figueras. 2009. *RIA y Adobe Flex un nuevo concepto de aplicaciones Web.* 2009.

Fain, Yakov, Rasputnis y Anatole Tartakovsky. 2007. *Aplicaciones Ricas de Internet con ADOBE FLEX Y JAVA.* 2007.

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sornmerlad y Michael Stal. 2001. *Pattern-oriented software architecture. A system of patterns.* 2001. 0 471 95889 7.

Free Donwload Manager. 2007. Free Donwload Manager. [En línea] 5 de marzo de 2007. [Citado el: 3 nov. de 2010.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/

García, Joaquin. 2005. IngenieroSoftware. [En línea] 27 de mayo de 2005. [Citado el: 15 de diciembre de 2010.] http://www.ingenierosoftware.com.

G. Booch, J. Rumbaugh y I. Jacobson. 2000. *El Lenguaje Unificado de Modelado.* s.l. : Addison Wesley, 2000.

Gómez, Omar Salvador. 2007. Evaluando Arquitecturas de Software. Parte 1. Panorama General. México : s.n., 2007. 1870-0888.

Gregory Abowd, Len Bass, Rick Kazman y Mike Webb. 2007. SAAM: A Method for Analyzing the Properties of Software Architectures. 2007.

Gustavo Andrés Brey, Gastón Escobar, Nicolas Passerini y Juan Arias. 2005. Arquitectura de Proyectos de IT. Evaluación de Arquitecturas., 2005.

Hirsch, Michael. 2002. *Making RUP Agile.* 2002.

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

Javier Calvarro Nelson, César de la Torre Llorente, Unai Zorrilla Castro, Miguel A. Ramos Barroso. 2010. *Guía de arquitectura N-capas orientada al dominio con .Net 4.0. Primera edición.* España : s.n., 2010. 978-84-936696-3-8.

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

Kent Beck, Mike Beedle, Arie van Bennekum y otros. 2001. Manifiesto por el Desarrollo Ágil de Software. 2001.

Kiccillof, Carlos Reynoso y Nicolás. 2004. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft., 2004.

Larman, Craig. 1999. *UML Y PATRONES: Introducción al análisis y diseño orientado a objetos .* México : PRENTICE HALL, 1999. 970-17-0261-1.

Len Bass, Paul Clements y Rick Kazman. 2003. *Arquitectura de Software en la práctica. Segunda edición.* s.l. : Addison-Wesley Profesional, 2003. 0321154959.

Linux Journal. 2004. Linux Journal. [Online] 10 5, 2004. [Cited: 01 13, 2011.] www.linuxjournal.com/article/7807.

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

Medina, Cesar Julio Bustacara. 2008. Evaluación de Arquitecturas de Software. Pontificia Universidad Javeriana : s.n., 2008.

Méndez, Gonzalo. 2008. Una Arquitectura Software Basada en Agentes y. Madrid: Universidad Politécnica de Madrid : s.n., 2008.

Microsoft Corporation. 2006. *La Arquitectura Orientada a Servicios (SOA) de Microsoft.* 2006.

Mojica, Jamir Antonio Avila. 2008. Arquitectura de software. Análisis de la arquitectura. SAAM / ATAM. 2008.

Mora, Sergio Luján. 2002. *Programación de aplicaciones Web: historia, principios básicos y clientes Web.* s.l. : Editorial Club Universitario, 2002.

Morales, Perla Azucena Arredondo. 2009. monografias.com. [En línea] 22 de octubre de 2009. [Citado el: 5 de abril de 2011.] http://www.monografias.com/trabajos75/servidores-web/servidores-web.shtml.

Nicolás Passerini, Gustavo Andrés Brey y Gastón Coco. 2009. Arquitectura de Proyectos de IT.Atributos de Calidad / Tácticas., 2009.

Olsson, Rob Bateman y Richard. 2010. *The Essencial Guide To 3D in Flash.* 2010.

Palacio, Juan. 2006. Navegapolis. . [Online] 2006. [Cited: 12 10, 2010.] http://www.navegapolis.net/content/view/694.

Pantoja, Ernesto Bascón. 2004. El patrón de diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swing. 2004. Vol. II.

Peláez, Juan Carlos. 2009. Definiciones - Atributos de Calidad para Aplicaciones Distribuidas y de Alta Disponibilidad. 2009.

Prabhudas, Siddharth. 2008. *Agile Unified Process.* 2008.

Pressman, Roger S. 2005. *Ingeniería de software un enfoque práctico. Sexta edición.* s.l. : McGraw-Hill, 2005. 9701054733.

—. 2002. *Ingeniería del software un enfoque práctico. Quinta edición.* Madrid : McGRAW-HIL, 2002. 84-481-3214-9.

Puebla, Enrique Chaviano Gómez y Yoan Arlet Carrascoso. 2008. Propuesta de Arquitectura Orientada a Servicios para el ERP Cubano. Ciudad de la Habana : s.n., 2008.

Rovira, Carlos. 2006. carlosrovira.com. [En línea] 8 de 05 de 2006. [Citado el: 15 de 03 de 2011.] http://www.carlosrovira.com/blog/search/Flex+SDK+%3D+Java+SDK+.

Santiago. 2010. HardCyberSoft. [En línea] 23 de 08 de 2010. [Citado el: 15 de 03 de 2011.] http://www.hardcybersoft.cl/blog/flex-sdk-open-source.

Sommerville, Ian. 2005. *Ingeniería del software. Séptima edición.* Madrid : Pearson addison wesley, 2005. 84-7829-074-5.

Stan, Monica Macoveiciuc y Constantin. 2009. slideshare. [Online] 2009. [Cited: 01 23, 2011.] http://www.slideshare.net/stanconstan/flex-framework-presentation-pdf.

Titos, Antonio Olmo. 2010. No Smoke. [En línea] 2010. [Citado el: 5 de Febrero de 2010.] http://nosmoke.cycle-it.com/category/3d/.

Villegas, Adrian Anaya. 2007. monografias.com. [Online] 10 29, 2007. [Cited: 12 5, 2010.] http://www.monografias.com/trabajos51/programacion-extrema/programacion-extrema.shtml.

Visual Paradigm. 2010. Visual Paradigm. [Online] 12 20, 2010. [Cited: 02 3, 2011.] http://www.visual-paradigm.com/.

Weggheleire, Sas Jacobs y Koen De. 2008. *Foundation Flex for Developers Data-Driven Applications with PHP, ASP.NET, ColdFusion, and LCDS.* New York : friendsof, 2008. 978-1-59059-894-8.

Zapata, Luis Giraldo y Liliana. 2005. Herramientas de Desarrollo de Ingeniería de Software para Linux. 2005.

BIBLIOGRAFÍA

Alba, Julio. 2008. SOA: ARQUITECTURA ORIENTADA AL SERVICIO . 2008. 0210-3923.

Alvarez, Miguel Angel. 2009. desarrolloweb.com. [En línea] 23 de Diciembre de 2009. [Citado el: 25 de noviembre de 2010.] http://www.desarrolloweb.com/articulos/modelo-vista-controlador-codeigniter.html.

Bachmann. 2000. Technical concepts of component-based software engineering. Carnegie Mellon University : s.n., 2000.

Blasi, Enmanuel. *Resumen de Patrones de Diseño.*

Canales Mora, Roberto. *Patrones GRASP. Madrid: s.n., 2006. Disponible en:* http://www.adictosaltrabajo.com/tutoriales/pdfs/grasp.pdf

Doberkat, Ernst-Erich. 2002. Pipes and filters: Modelling a software architecture through relations. 2002.

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

Garcia, Joaquin. 2005. IngenieroSoftware. [En línea] 27 de mayo de 2005. [Citado el: 15 de diciembre de 2010.] http://www.ingenierosoftware.com..

Gómez, Omar Salvador. 2007. Evaluando Arquitecturas de Software. Parte 1. Panorama General. México : s.n., 2007. 1870-0888.

—. 2007. Evaluando Arquitecturas de Software. Parte 2. Métodos de Evaluación. México : s.n., 2007. 1870-0888.

Gregory Abowd, Len Bass, Rick Kazman y Mike Webb. 2007. SAAM: A Method for Analyzing the Properties of Software Architectures. 2007.

ISO/IEC 9126-1:2001. Software Engineering. Product Quality- Part 1: Quality Model. [En línea]. [Consultado el: 25 de Abril de 2010]. Disponible en: http://webstore.iec.ch/preview/info_isoiec9126-1%7Bed1.0%7Den.pdf

Javier Calvarro Nelson, César de la Torre Llorente, Unai Zorrilla Castro, Miguel A. Ramos Barroso. 2010. *Guía de arquitectura N-capas orientada al dominio con .Net 4.0. Primera edición.* España : s.n., 2010. 978-84-936696-3-8.

Kicillof, Carlos Reynoso y Nicolás. 2004. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. UNIVERSIDAD DE BUENOS AIRES : s.n., 2004.

Kruchten, P. (Noviembre de 1995). Architectural Blueprints—The “4+1” View Model of Software Architecture.

Larman, Craig. 1999. *UML Y PATRONES: Introducción al análisis y diseño orientado a objetos .* México : PRENTICE HALL, 1999. 970-17-0261-1.

López, Patricio Letelier Torres y Emilio A. Sánchez. 2003. Metodologías Ágiles en el Desarrollo de Software. 2003.

Méndez, Gonzalo. 2008. Una Arquitectura Software Basada en Agentes y. Madrid: Universidad Politécnica de Madrid : s.n., 2008.

Mojica, Jamir Antonio Avila. 2008. Arquitectura de software. Análisis de la arquitectura. SAAM / ATAM. 2008.

Morales, Perla Azucena Arredondo. 2009. monografias.com. [En línea] 22 de octubre de 2009. [Citado el: 5 de abril de 2011.] http://www.monografias.com/trabajos75/servidores-web/servidores-web.shtml.

Olsson, Rob Bateman y Richard. 2010. *The Essencial Guide To 3D in Flash.* 2010.

Pantoja, Ernesto Bascón. 2004. El patrón de diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swing. 2004. Vol. II.

Pressman, Roger S. 2005. *Ingeniería de software un enfoque práctico. Sexta edición.* s.l. : McGraw-Hill, 2005. 9701054733.

—. 2002. *Ingeniería del software un enfoque práctico. Quinta edición.* Madrid : McGRAW-HIL, 2002. 84-481-3214-9.

Puebla, Enrique Chaviano Gómez y Yoan Arlet Carrascoso. 2008. Propuesta de Arquitectura Orientada a Servicios para el ERP Cubano. Ciudad de la Habana : s.n., 2008.

Riola, Jose Carlos Carvajal. 2008. METODOLOGÍAS ÁGILES: HERRAMIENTAS Y MODELO DE DESARROLLO PARA APLICACIONES JAVA EE COMO METODOLOGÍA EMPRESARIAL. 2008.

Solís, Manuel Calero. 2003. Una explicación de la programación extrema (XP). Madrid : s.n., 2003.

Sommerville, Ian. 2005. *Ingeniería del software. Séptima edición.* Madrid : Pearson addison wesley, 2005. 84-7829-074-5.

Titos, Antonio Olmo. 2010. No Smoke. [En línea] 2010. [Citado el: 5 de Febrero de 2010.] http://nosmoke.cycle-it.com/category/3d/.

Valverde, David. 2007. davidvalverde.com. [Cited: 12 10, 2010.] http://www.davidvalverde.com/blog/introduccion-a-la-programacion-extrema-xp/.

Weggheleire, Sas Jacobs y Koen De. 2008. *Foundation Flex for Developers Data-Driven Applications with PHP, ASP.NET, ColdFusion, and LCDS.* New York : friendsof, 2008. 978-1-59059-894-8.

Zapata, Luis Giraldo y Liliana. 2005. Herramientas de Desarrollo de Ingeniería de Software para Linux. 2005.

GLOSARIO DE TÉRMINOS Y SIGLAS

Términos

Aplicaciones Web: Aplicaciones que los usuarios pueden utilizar accediendo a un servidor web a través de internet o de una intranet mediante un navegador.

Atributo de calidad: Es una característica medible que permite expresar la calidad de un sistema de un punto de vista del cliente y de la organización de desarrollo.

Componente: Es una parte de una arquitectura de *software* claramente identificable, independiente a la aplicación en la que se utiliza y de otros componentes, que describe y realiza funciones específicas y claras dentro del contexto de la arquitectura.

Eficiencia: Capacidad de alcanzar los objetivos y metas programadas con el mínimo de recursos disponibles y tiempo, logrando su optimización.

Flujo de trabajo: Es el estudio de los aspectos operacionales de una actividad de trabajo: cómo se estructuran las tareas, cómo se realizan, cuál es su orden correlativo, cómo se sincronizan, cómo fluye la información que soporta las tareas y cómo se le hace seguimiento al cumplimiento de las tareas.

***Framework*:** Representa una abstracción de diseño y tiene un comportamiento en sí. No es solamente una clase, sino que es un conjunto de objetos que se relacionan para servir a un dominio específico.

Funcionalidad: Conjunto de atributos que se relacionan con la existencia de un conjunto de funciones y sus propiedades específicas. Las funciones son aquellas que satisfacen lo indicado o implica necesidades.

Gestión: Comprende todas las actividades de una organización que implican el establecimiento de metas u objetivos, así como la evaluación de su desempeño y cumplimiento; además del desarrollo de una estrategia operativa que garantice la supervivencia de la misma, según al sistema social correspondiente.

Mantenibilidad: Conjunto de atributos relacionados con la facilidad de extender, modificar o corregir errores en un sistema *software*.

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

Autora: Liusba Cañete Núñez

Orquestar: Organizar o dirigir algo, coordinando sus distintos elementos: orquestar un proceso de negocio.

Portabilidad: Conjunto de atributos relacionados con la capacidad de un sistema *software* para ser transferido desde una plataforma a otra.

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.

Requerimiento: Capacidad o característica que debe tener un sistema o modelo de desarrollo para satisfacer la demanda y/o necesidad del cliente.

Stakeholders: Aquellas personas que están relacionadas, de cierta forma, con el sistema, ya sea un desarrollador, usuario o gerente, etcétera.

Técnicas: Sucesión ordenada de acciones que se dirigen a un fin concreto, conocido y que conduce a unos resultados precisos.

Web: Es un sistema de documentos de hipertexto y/o hipermedios enlazados y accesibles a través de internet. Con un navegador web, un usuario visualiza páginas web que pueden contener texto, imágenes, videos u otros contenidos multimedia, y navega a través de ellas usando hiperenlaces.

Siglas

API: Interfaz de programación de aplicaciones (*Application Programming Interface*).

ARID: Método de Revisión Intermedio de Diseño (*Active Reviews for Intermediate Design*).

ATAM: Método de Análisis de Acuerdos de Arquitectura de *Software* (*Architecture Tradeoff Analysis Method*).

AUP: Proceso Unificado Ágil, (*Agile Unified Process*).

CASE: Ingeniería de *Software* Asistida por Computadoras, (*Computer Aided Software Engineering*).

DSSA: Arquitectura de *Software* de Dominio Específico (*Domain-Specific Software Architecture*).

FTP: Protocolo de Transferencia de Archivos (*File Transfer Protocol*)

GRASP: Patrones Generales de *Software* de Asignación de Responsabilidades.

HTML: Lenguaje de Marcas de Hipertexto, (*HyperText Markup Language*).

IDE: Ambiente Integrado de Desarrollo (*Integrated Development Environment*).

IEEE: El Instituto de Ingenieros Eléctricos y Electrónicos, (*The Institute of Electrical and Electronics Engineers*), una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas.

IIS: Servicios Informativos de Internet (*Internet Information Services*).

IS: Ingeniería de *Software*.

MVC: Modelo-Vista-Controlador.

MXML: Lenguaje descriptivo desarrollado para la plataforma FLEX de Adobe, basado en XML (*Macromedia eXtensible Markup Language*).

NNTP: Protocolo para la transferencia de noticias en red (*Network News Transport Protocol*)

PHP: Lenguaje interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor (*Hypertext Preprocessor*).

RIA: Aplicaciones Enriquecidas de Internet (*Rich Internet Applications*).

SAAM: Método de Análisis de Arquitectura de *Software* (*Software Architecture Analysis Method*).

SMTP: Protocolo Simple de Transferencia de Correo (*Simple Mail Transfer Protocol*).

SOA: Arquitectura Orientada a Servicios (*Service Oriented Architecture*).

UCI: Universidad de Ciencias Informáticas.

UML: Lenguaje Unificado de Modelado (*Unified Modeling Language*).

XP: Programación Extrema, (*eXtreme Programming*).

ANEXOS

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del *software*	La arquitectura del *software* es esencial y se expresa mediante modelos

Anexo 1: Diferencias entre metodologías ágiles y tradicionales.

Biblioteca / Indicador	Away3d	Papervision3d	Sandy 3d
Licencia	Apache 2.0.	MIT	Mozilla Public License 1.1
Cuadros por segundo	Ciento once cuadros por segundos. Es la biblioteca más eficiente de las tres analizadas.	Cien cuadros por segundos.	Ochenta y siete cuadros por segundos.
Lenguaje de programación	*ActionScript 3.0*	*ActionScript 3.0*	*ActionScript 2.0, ActionScript 3.0* y *haXe*
Frecuencia de	Se mantiene actualizado,	Ha perdido fuerzas, ha	En los últimos once

actualización	evolucionando con mejoras, correcciones, nuevos ejemplos y documentación, y es la favorita de los programadores *Flash*.	disminuido.	meses del año 2010, apenas ha habido veinte contribuciones al repositorio.
Última versión empaquetada	3.5.0	2.1.932	3.1.2
Vistas básicas	Incluyen instancias completas y funcionales de escena y cámara.	Incluyen instancias completas y funcionales de escena y cámara.	Se necesita crear explícitamente una escena y una cámara y asignarlas como atributos a la vista.

Anexo 2: Comparación de bibliotecas de código.

Servidor / Indicador	Apache	IIS
Licencia	Licencia Apache	Propietario
Mantenimiento	Medio	Alto
Sistemas Operativos	Multiplataforma	Windows
Documentación	Mucha	Regular
Creador	Fundación del *software* de Apache	Microsoft
Flexibilidad	Alta	Alta
Seguridad	Alta	Media

Anexo 3: Comparación de servidores Web.

Autora: Liusba Cañete Núñez

Técnica / Indicador	SAAM	ATAM	ARID
Atributos de calidad contemplados	Modificabilidad Funcionabilidad	Modificabilidad Seguridad Confiabilidad Desempeño	Conveniencia del diseño evaluado.
Objetos analizados	Documentación, y Vistas Arquitectónicas.	Estilos Arquitectónicos, Documentación, Flujo de Datos y Vistas Arquitectónicas.	Especificación de los componentes.
Etapas del proyecto en las que se aplica	Luego que la arquitectura cuenta con funcionalidad ubicada en módulos.	Luego que el diseño de la arquitectura ha sido establecido.	A lo largo del diseño de la arquitectura.
Enfoques utilizados	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.	Árbol de Utilidad y lluvia de ideas para articular los requerimientos de calidad. Análisis arquitectónico que detecta puntos sensibles, puntos de balance y riesgos.	Revisiones de diseño, lluvia de ideas para obtener escenarios.

Anexo 4: Comparación de los métodos de evaluación.

Autora: Liusba Cañete Núñez