Universidad de las Ciencias Informáticas
Facultad 5

INTEGRACIÓN DE LOS COMPONENTES ARQUITECTÓNICOS Y TECNOLOGÍAS PARA EL PROYECTO SISTEMA INTEGRAL DE CONFIABILIDAD

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Miguel Angel Socorro Borges

Tutor: Ing. David Vargas Hernández

Ciudad de La Habana

"Año 53 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro ser autor 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 _____.

Miguel Angel Socorro Borges	Ing. David Vargas Hernández
Autor	Tutor

DATOS DE CONTACTO

Tutor: Ing. David Vargas Hernández.

Edad: 26

Ciudadanía: Cubano.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias de la Informática.

Categoría Docente: Instructor.

E-mail: dvhernandez@uci.cu

DEDICATORIA

A mi madre y mi abuela

Las personas que me observaron dar los primeros pasos, pronunciar las primeras palabras y levantarme de las primeras caídas. Reconocieron mis triunfos, lloraron conmigo en los fracasos y me cuidaron en todo momento. A las que cambiarían todo por ofrecerme un minuto de felicidad.

AGRADECIMIENTOS

Quisiera agradecer a todas las personas que de una forma u otra han contribuido a mi formación personal y profesional, en muchos casos, las palabras no son suficientes para expresar las dimensiones de mi agradecimiento.

A mi madre y mi abuela

Por estar siempre a mi lado y brindarme el apoyo necesario en cada momento. Soy dichoso de tenerlas a ambas. Para ustedes todo mi agradecimiento por tanto amor y dedicación. Gracias

A mis tíos

Las personas que fueron mis segundos padres, a nono por su ejemplo, a rufo por su legado y a Randy por ser la persona que siempre me ha representado en todo momento.

A mi tutor

Por haber confiado mí para la realización de este trabajo y guiarme en el desarrollo del mismo.

A mis compañeros

Ustedes han sido durante estos cinco años mis familias más cercanas en esta maravillosa escuela, gran parte de lo que soy se lo debo a ustedes.

RESUMEN

En el presente trabajo investigativo se define la arquitectura, tecnologías, estilos y patrones arquitectónicos a utilizar en el proyecto de software Sistema Integral de Confiabilidad (SIC), desarrollado en el Centro de Informática Industrial CEDIN de la Universidad de las Ciencias Informáticas (UCI). Este sistema está orientado a la Web, debido a lo cual la arquitectura definida está basada en un estilo arquitectónico en capa, aplicando el patrón Modelo-Vista-Presentador (MVP).

Como tecnología a utilizar se definen los framework de desarrollo Google Web Toolkit (GWT) y Grails, los cuales a pesar de ser altamente conocidos y usados a nivel mundial presentan ciertas incompatibilidades entre ellos, al no ser creados para operar juntos en una misma aplicación; por lo que se describe todo el proceso necesario, a realizar, para integrar correctamente estas tecnologías aprovechando la ventajas ofrecidas por ambas.

Finalmente la arquitectura y el proceso de integración definidos se prueban y evalúan a través del método Architecture Trade-off Analysis Method (ATAM), utilizando un árbol de utilidad basado en el modelo de calidad ISO-9126, instanciado para evaluar Arquitectura de Software. Unido a esto también se le realizaron pruebas a uno de los módulos del sistema ya terminado arrojando buenos resultados.

Palabras Claves: arquitectura, componentes, integración, tecnologías.

Índice

Índice de Figuras

Índice de Tablas

INTRODUCCIÓN

La modernización de los equipos y el continuo desarrollo tecnológico ha generado un cambio en el contexto operacional a nivel mundial, y unido a la gran cantidad de variables presentes en el mismo es difícil determinar una relación directa entre el tiempo de vida útil y la probabilidad de falla de los equipos. En este sentido se hace necesario adoptar nuevas filosofías de mantenimiento que permitan mantener la calidad de los procesos y garantizar la integridad de los equipos.

Ante esta panorámica, una vía efectiva que le permite a las organizaciones enfrentar de forma eficiente el nuevo reto, son los conceptos y principios de Confiabilidad Operacional; siendo esta la capacidad de una instalación o sistema integrado por procesos, tecnología y personas, para cumplir su función dentro de sus límites de diseño y bajo un contexto operacional específico (Arata, 2009).

A lo largo de las últimas décadas, se han ido desarrollando varios modelos, métodos y técnicas que permiten cuantificar y/o hacer valoraciones del impacto que sobre la confiabilidad operacional tienen las diversas acciones que se realizan sobre un sistema técnico complejo, esto ha dado lugar a un desarrollo a nivel mundial de distintos sistemas de confiabilidad operacional enmarcados en tecnologías específicas de una empresa, pero a medida que el mercado se hace más competitivo, la disponibilidad de estos modelos, métodos y técnicas se hacen insuficiente, debido al continuo desarrollo y en muchos casos cambio total de esta tecnología. Unido a esto la experiencia de muchas organizaciones indica que estas acciones aisladas que atienden algún elemento e ignoran los otros, pueden proporcionar beneficios pero normalmente son limitados y no duraderos.

Debido a esto en la Universidad de las Ciencias Informáticas (UCI), específicamente en la Facultad 5 en el Centro de Informática Industrial CDIN, se desarrolla el proyecto Sistema Integral de Confiabilidad (SIC) con el objetivo de generar herramientas e instrumentos, que se ajusten a toda clase de tecnología y permitan mejorar la estimación del impacto de las acciones realizadas sobre los sistemas técnicos complejos, durante las fases del ciclo de vida, relacionadas con la operación y el mantenimiento, a través de la vinculación de las diferentes ramas de estudio de esta ciencia.

Dichas herramientas están integradas en una aplicación que de acuerdo a sus requisitos debe contar con una alta disponibilidad, y ofrecer el servicio para el cual será desarrollado desde múltiples localizaciones en una determinada organización,

razón por la cual será orientada a la web; lo cual cumple con los aspectos antes dichos implicando un significativo ahorro de tiempo, evitando los problemas de incompatibilidad con el sistema operativo instalado en el cliente, la vulnerabilidad a los virus y el alto consumo de recursos.

Para el desarrollo de esta aplicación web se definió el uso de diversas tecnologías entre ellas los framewoks de desarrollo Google Web Toolkit (GWT) y Grails, los cuales a pesar de estar establecidos y probados a nivel mundial presentan incompatibilidades entre ellos al no ser concebidos para operar juntos en una misma aplicación.
En la actualidad son muchos los desarrolladores que utilizan los frameworks de desarrollo Grails y GWT, beneficiándose de las ventajas de cada uno de ellos. Estas ventajas pueden ser mejoradas potencialmente si se integran ambas tecnologías. Con este propósito algunas comunidades de desarrollo han creado plugins que permiten incorporar código GWT en las aplicaciones Grails, pero estos presentan algunas deficiencias en el envío y recibo de datos entre ambas tecnologías.

Este problema de incompatibilidad implica un mayor esfuerzo en el diseño arquitectónico de la aplicación, con el objetivo de integrar correctamente estas tecnologías, a fin de garantizar satisfacer las exigencias de las mismas y las funcionalidades y requerimientos de desempeño del sistema; así como los criterios de calidad indicados como seguridad, disponibilidad, eficiencia y usabilidad, requisitos indispensables para la correcta arquitectura de un sistema (Pressman, 2010).

Teniendo en cuenta estos aspectos, y con el propósito de dar cumplimiento a los mismos surge el siguiente **problema a resolver** ¿Cómo integrar los componentes arquitectónicos y tecnologías del proyecto SIC? Dicho problema está enmarcado específicamente en la integración entre los componentes arquitectónicos y tecnologías seleccionados para la creación del proyecto SIC como **campo de acción,** contenido en los fundamentos correspondientes al estudio de la arquitectura de sistemas web, lo cual constituye el **objeto de estudio.**

Por tanto el **objetivo general** de este trabajo es: integrar los componentes arquitectónicos y tecnologías para el proyecto SIC.

Para dar cumplimiento al objetivo planteado es necesario realizar un grupo de **tareas investigativas** tales como:

- Fundamentación de los distintos componentes arquitectónicos y tecnológicos en la arquitectura de sistemas web.
- Descripción de la arquitectura e integración así como herramientas y diagramas a utilizar en la misma.
- Descripción de los requisitos arquitectónicos del sistema.
- Desarrollo de pruebas a la arquitectura e integración.

Para la realización de la investigación y elaboración del presente trabajo se emplearon varios **métodos científicos de investigación**, entre los cuales se pueden mencionar los siguientes.

Métodos teóricos:

Análisis Histórico – Lógico: Mediante este método se analiza la evolución y desarrollo del objeto de investigación y sus elementos más importantes y transcendentales.
Analítico – Sintético: Se usa para analizar la información de las técnicas y tecnologías existentes para la integración de componentes arquitectónicos.
Modelación Analógica: Se emplea para realizar una representación simplificada de la realidad a través de diagramas de clases, de flujo y de componentes.

Métodos empíricos:

Pruebas al sistema: Se realizan diferentes pruebas al sistema para determinar si se comporta según los resultados esperados.
Búsqueda bibliográfica: Se usa para la búsqueda y localización de referencias bibliográficas relacionadas con el objeto de investigación.

A continuación se muestra la estructura del presente trabajo, incluyendo una síntesis de los capítulos y secciones fundamentales:

Capítulo 1. Fundamentación Teórica.
En este capítulo se definen los principales conceptos que serán empleados durante todo el trabajo y se presentan las bases teóricas fundamentales relacionadas con los sistemas web y sus arquitecturas. También se profundiza en la integración de componentes de software y tecnologías web.

Capítulo 2. Integración de los componentes arquitectónicos y tecnologías.
En este capítulo se describe y concluye la solución de la arquitectura propuesta, definiéndose los estilos arquitectónicos y tecnologías a usar. Unido a esto se relacionan los patrones, métodos y un grupo ordenado de pasos para lograr con éxito la integración de los componentes arquitectónicos y tecnologías del proyecto SIC.

Capitulo 3. En este capítulo se elige y describe un método para evaluar la arquitectura y integración de tecnologías y componentes definida; para ello se hace uso del árbol de utilidad basado en el modelo de calidad ISO-9126, instanciado para Arquitectura de Software, el cual propone un conjunto de escenarios que ponen a prueba las propuestas arquitectónicas definidas. Como resultado final se evalúa la arquitectura e integración propuesta.

Conclusiones: Se dan a conocer las conclusiones a las que se llegó, después del trabajo realizado.

Recomendaciones: Se muestran las recomendaciones hechas al trabajo para futuros análisis.

Capítulo 1 Fundamentación Teórica.

En este capítulo se define el estado del arte sobre la arquitectura y la integración de tecnologías, así como los principales conceptos que serán empleados a lo largo del trabajo, se presentan las bases teóricas fundamentales relacionadas con sistemas web y sus arquitecturas. También se profundiza en la integración de componentes de software y tecnologías web.

1.1 Sistemas Web

La web ha influido enormemente tanto en el mundo de la informática como en la sociedad en general. En poco menos de 10 años ha transformado los sistemas informáticos: ha roto las barreras físicas (debido a la distancia), económicas y lógicas (debido al empleo de distintos sistemas operativos, protocolos, etc.) y ha abierto todo un abanico de nuevas posibilidades (Arquitectura y Diseño de Sistemas Web Modernos, 2004). Una de las áreas que más expansión está teniendo en la web en los últimos años, son las aplicaciones web; estas aplicaciones permiten usar la infraestructura de la web para desarrollar aplicaciones globales. Reportes recientes indican que las aplicaciones web representan más de la mitad del total de todas las aplicaciones de la industria de software. Esta cifra indica que las aplicaciones web siguen creciendo y ganando popularidad en un mercado muy competitivo, y que se perfilan a ser las aplicaciones del futuro.

Las aplicaciones web son aquellas aplicaciones que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador (González Romano, y otros, 2001). En otras palabras, es una aplicación de software que se codifica en un lenguaje soportado por los navegadores web en la que se confía la ejecución al navegador. Estas aplicaciones permiten la generación automática de contenido y la creación de páginas personalizadas según el perfil del usuario. Además, una aplicación web permite interactuar con los sistemas informáticos de gestión de una empresa, como puede ser gestión de clientes, contabilidad o inventario, a través de un browser.

Una ventaja significativa de estos sistemas como se dijo antes es su funcionamiento independiente del sistema operativo instalado en el cliente, puesto que en vez de crear clientes para Windows, Mac OS X, GNU/Linux u otros sistemas operativos, la aplicación web se escribe una vez y se ejecuta igual en todas partes. Sin embargo,

hay aplicaciones inconsistentes escritas con HTML, CSS, DOM y otras especificaciones estándar para navegadores web que pueden causar problemas en el desarrollo y soporte de estas aplicaciones, principalmente debido a la falta de adicción de los navegadores a dichos estándares web. Dicho inconveniente puede ser solucionado mediante el uso de tecnologías que ignoren las configuraciones de los navegadores, permitiendo más control sobre las interfaces de usuarios, puesto que las mayorías de los navegadores incluyen soporte para este tipo tecnología.

De forma general no es difícil darse cuenta que las aplicaciones web se han convertido en pocos años en complejos sistemas con interfaces de usuario cada vez más parecidas a las aplicaciones de escritorio, dando servicio a procesos de negocio de considerable envergadura y estableciéndose sobre ellas requisitos estrictos de accesibilidad, respuesta, seguridad, disponibilidad entre otros aspectos arquitectónicos; pero a su vez esto ha exigido reflexiones sobre la arquitectura y las técnicas de diseño más adecuadas a utilizar en estos sistemas, capaces de satisfacer estos requerimientos.

1.2 Arquitectura de software

El diseño de la arquitectura de software es el proceso por el cual se define una solución para los requisitos técnicos y operacionales del mismo. Este proceso define que componentes forman el sistema, como se relacionan entre ellos, y como mediante su interacción llevan a cabo la funcionalidad especificada, cumpliendo con los criterios de calidad indicados como seguridad, disponibilidad, eficiencia o usabilidad (Pressman, 2010). En el diseño de la arquitectura lo primero que se decide es el tipo de sistema o aplicación que se va a construir, como pueden ser aplicaciones móviles, de escritorio, o web. El tipo de aplicación viene determinado por la topología de despliegue y los requisitos y restricciones indicadas en los requisitos; y su selección determina en cierta medida el o los estilos arquitectónicos que se van a usar, puesto que una aplicación puede responder a más de un estilo arquitectural. Este estilo arquitectónico es una transformación que se impone en el diseño del sistema completo, con la intención de establecer una estructura para todos los componentes del sistema (Pressman, 2010). En esencia es un conjunto de principios que definen a alto nivel un aspecto de la aplicación. Viene definido por una serie de componentes, las conexiones entre dichos componentes y un conjunto de restricciones sobre cómo se comunican dos componentes cualesquiera conectados. A manera general se puede

ver como la partición más básica del sistema en bloques y la forma en que se relacionan estos bloques.

Una vez seleccionado el tipo de aplicación y determinado el o los estilos arquitecturales que más se ajustan al tipo de sistema que se va a construir, se decide cómo construir los bloques que forman el sistema. Por ello el siguiente paso es seleccionar las distintas tecnologías a usar. Estas tecnologías estarán limitadas por las restricciones de despliegue y las impuestas por el cliente; y una vez analizado el sistema y fragmentado en partes más manejables, se procede a implementar todos los requisitos que debe satisfacer el sistema.

1.3 Estilos arquitectónicos

Como se vio anteriormente un estilo arquitectónico no es más que un conjunto de principios que definen a alto nivel un aspecto de la aplicación. A continuación se muestran los estilos arquitectónicos más utilizados en la arquitectura de software, organizados por Clases de Estilos, que engloban una serie de estilos arquitectónicos específicos:

Clase de Estilos	Estilos arquitectónicos
Centrado en Datos	Arquitecturas de Pizarras o Repositorios.
Flujo de Datos	Tuberías y Filtros.
Llamada y Retorno	Arquitectura en Capas, Orientada a Objetos, Basada en Componentes.
Código Móvil	Arquitectura de Máquinas Virtuales.
Peer To Peer	Arquitecturas Basadas en Eventos, Orientadas a Servicios (SOA en inglés), Basadas en Recursos.

Tabla. 1 Clases de estilos arquitectónicos *(Pressman, 2010)**.***

La arquitectura de un sistema de software casi nunca se limita a un estilo arquitectónico único, a menudo es una combinación de estilos arquitectónicos que forman el sistema completo aprovechando las ventajas de cada uno de ellos, los cuales mantienen una relación muy estrecha con un conjunto de patrones que ayudan a derivar el diseño arquitectónico del sistema.

1.4 Patrones

Se entiende que un patrón es una solución probada que da respuesta a un problema dado, que se presenta en un instante de tiempo determinado en un campo o área de acción determinada. Un patrón describe un problema que se repite una o varias veces en el ambiente de trabajo y a la vez describe el núcleo de la solución a ese problema, de tal manera que la solución que brinda el patrón puede utilizarse cuantas veces ocurra el problema, sin tener que hacerlo de la misma manera dos veces (Fowler, 2002).

Los patrones cubren diferentes rangos de escalas y niveles de abstracción, algunos ayudan a estructurar un sistema en subsistemas base. Mientras otros ayudan al afinamiento de subsistemas y componentes, o las relaciones entre estos, y otros ayudan en la implementación de aspectos de diseño particulares en un lenguaje de programación específico.

De acuerdo a estas características se pueden dividir en tres categorías principales:

- Patrones de Arquitectura.
- Patrones de Diseño.
- Patrones de Implementación (o Idiomas).

1.4.1 Patrones de Arquitectura

Expresan un esquema fundamental de organización estructural para sistema de software. Donde provee una serie de subsistemas predefinidos, especificando sus responsabilidades, e incluye reglas y guías para organizar las relaciones entre ellos (Marquina, 2008).

Básicamente, un patrón de arquitectura es una plantilla para una arquitectura de aplicaciones, el cual especifica las propiedades generales a la estructura del sistema, y repercute en la arquitectura de sus subsistemas.

Uno de los ejemplos más conocidos de patrones de arquitectura es el patrón Modelo-Vista-Controlador (MVC) que provee la infraestructura para sistemas interactivos.

1.4.2 Patrones de Diseño

Los subsistemas y componentes dentro de una arquitectura de software, al igual que sus interrelaciones, consisten normalmente en unidades arquitectónicas más pequeñas. Estas unidades son descritas a través de Patrones de Diseño.

Un Patrón de Diseño define un esquema de refinamiento de los subsistemas o componentes dentro de un sistema, o las relaciones entre estos. Este

describe una estructura común y recurrente de componentes interrelacionados, que resuelve un problema general de diseño dentro de un contexto particular (Marquina, 2008).

De manera general los patrones de diseño trabajan a una escala intermedia. Son menores en escala que un patrón de arquitectura, pero logran ser independientes del lenguaje de programación. La aplicación de un patrón de diseño no tiene efectos sobre la estructura fundamental del sistema (arquitectura), pero puede tener una fuerte influencia sobre la arquitectura de un subsistema. Un ejemplo concreto de patrones de diseño es el de Singleton.

1.4.3 Patrones de Implementación

Un patrón de implementación (también llamado Idioma) es un patrón de bajo nivel específico a un lenguaje de programación. Este describe como implementar aspectos particulares de componentes o sus relaciones utilizando las características de un lenguaje dado (Marquina, 2008).

De manera general estos patrones representan el nivel más bajo de patrones, y manejan aspectos tanto de diseño como implementación, y son mayormente utilizados durante la fase de desarrollo de un sistema.

1.5 Componentes de software

El concepto de componentes, en el desarrollo de software, para muchos autores es simplemente la evolución del la metodología orientada a objetos. De hecho, muchas de las características de los componentes para el desarrollo, parten de la idea del diseño orientado a objetos. Básicamente estos se pueden ver como partes del software que se pueden combinar para confeccionar un conjunto mayor como un subsistema, un sistema o incluso otro componente. Estos juegan el papel de una unidad de software reutilizable que puede interoperar con otros módulos del software mediante sus interfaces, desde las cuales se puede tener acceso a los servicios que este ofrece.

Una definición más formal puede ser la que se ofrece en la segunda edición del libro “Rational Unified Process “por Philippe Krutchen:

“Un componente es una parte no trivial, casi independiente, y reemplazable de un sistema que llena claramente una funcionalidad dentro de un contexto en una arquitectura bien definida. Un componente se conforma y provee la realización física por medio de un conjunto de interfaces”.

Para clasificarlos se pueden agrupar en dependencia de sus características y funcionalidades, dividiéndolos en 4 grupos fundamentales (Szypersky, 2002):

- **Componentes de Interfaces Gráficas de usuarios (GUI):** Proveen interfaces externas y el conocimiento de la interacción con el usuario.
- **Componentes de Flujo de Trabajo y Procesos de Control:** Manejan procesos complejos, automatizados del negocio, que interactúan con servicios del negocio.
- **Componentes de Servicios de Negocio y Adaptadores de Sistemas Delegados:** Proveen la implementación de reglas del negocio y la actividad operacional.
- **Componentes de Servicios de Sistema Operativo y Datos:** Proveen la funcionalidad que interactúa con el ambiente de almacenamiento persistente, incluyendo sistemas de administración de base de datos y sistemas de archivos.

Estos 4 tipos de componentes se pueden encontrar con mayor facilidad en aplicaciones que definan arquitectura en capas.

1.6 Proceso de integración en sistemas de software

La integración constituye la habilidad de recoger elementos o aspectos de una entidad o de varias e incorporarlos al ente o conjunto de organismos. En la industria del software permitan dar solución a problemas comunes de desarrollo, donde es necesario conectar sistemas aislados y heterogéneos dentro de una organización o entre varias y posibilitar su funcionamiento como un único sistema. Específicamente en la arquitectura de software busca la relación entre el espacio interior con el espacio exterior. Una dualidad que se complementa mutuamente con las características propias de cada ambiente o de cada plataforma operacional en el desarrollo de software. Este proceso persigue una manera eficiente y flexible de combinar recursos como tecnologías y componentes y reutilizar soluciones con el objetivo de mejorar operaciones.

Todo este proceso resulta una tarea difícil de realizar, en la que intervienen innumerables factores como plataformas, lenguajes de desarrollo, tiempo y costo, entre otros indicadores y unido a esto el contaste cambio de las tecnologías y componentes implícitos en un sistema implica la existencia de una arquitectura de múltiples capas que garantice la independencia de estas tecnologías y componentes al ser distribuidos en cada capa, considerando para cada capa el uso de soluciones

prácticas efectivas en el desarrollo del software; con el objetivo de mejorar la calidad del software, la portabilidad, la interoperabilidad y la reusabilidad del mismo.

Dependiendo del ambiente para realizar la integración, los componentes y tecnologías a integrar, la preparación que tienen estos para integrarse, la complejidad de la integración y los procedimientos y criterios para la integración se pueden identificar cuatro estilos fundamentales de integración (White, 2005):

- Replicación de Datos.
- Extracción, Transformación y Carga de Datos o ETL.
- Integración de Información Empresarial o EII.
- Integración de Aplicaciones Empresariales EAI.

1.6.1 Recopilación de Datos

La Replicación de Bases de Datos es un estilo de integración que se basa en la creación y mantenimiento de múltiples copias de una misma base de datos. En la mayoría de las implementaciones de Replicación, un servidor mantiene la copia primaria de la base de datos y servidores adicionales mantienen las copias esclavas de la misma (White, 2005).

En este estilo de integración la transferencia de datos se realiza de Base de Datos a Base de Datos siendo el estilo más antiguo de integración. Debido a estas características, se concluyó que esta técnica no es la adecuada para la solución que se necesita.

1.6.2 Extracción, Transformación y Carga de Datos o ETL

El estilo de integración Extracción, Transformación y Carga de Datos (Extract, Transform and Load), como su nombre lo indica extrae información de un sistema fuente, transforma esos datos para satisfacer los requerimientos del negocio y carga el resultado en un sistema destino. Tanto la fuente como el destino son generalmente Bases de Datos y archivos.

La transformación puede implicar la reestructuración y reconciliación del registro de datos, limpieza del contenido de datos (es decir, revisados por si existen discrepancias y eliminación de datos obviamente falsos), y/o agregación del contenido de datos. Esto sucede mediante una serie de procedimientos especiales que permiten obtener un formato unificado común y mejorado. Sólo después de la revisión y unificación de los datos estos son cargados. Esta técnica se encarga de la integración de datos, no de aplicaciones, y obtiene los datos

directamente de las Bases de Datos. De acuerdo a las características de esta técnica se concluyó que no es la más adecuada para la solución que se necesita.

1.6.3 Integración de Información Empresarial o EII

La Integración de Información Empresarial o EII es otro de los estilos de integración. Este consiste en un mecanismo de transformación y acceso a datos transparente y optimizado para suministrar una única interfaz a lo largo de los datos de las organizaciones. Dicha interfaz permite acceder a los datos y el resultado de este método es un Sistema de Información Heterogéneo Distribuido, Virtualmente Integrado. Este tipo de solución consiste en crear un intermediario que contenga los directorios de la Base de Datos y que a su vez sirva de canal de consulta y representación de la información recuperada. La información es capturada en tiempo real lo que implica que las fuentes de datos tengan una estructura tecnológica sólida y bien establecida.

EII protege a las aplicaciones de la complejidad de recuperar datos de múltiples localizaciones, donde los datos pueden diferir en semántica y formato, y emplear diferentes interfaces de datos.

Teniendo en cuenta estos aspectos, para la integración de Datos a Tiempo Real la técnica EII constituye una buena alternativa, sin embargo no es factible para la integración de componentes.

1.6.4 Integración de Aplicaciones Empresariales EAI

EAI es el proceso de integrar múltiples aplicaciones y componentes desarrollados independientemente, que utilizan tecnología incompatible y que son gestionados de forma independiente, permitiendo que se comuniquen e intercambien transacciones de negocio, mensajes, y datos entre sí. Uno de los principales objetivos de EAI es proporcionar acceso transparente a la amplia gama de aplicaciones que existen en una organización. Las características más importantes de esta técnica es que se utiliza para la integración de aplicación a aplicación y proporciona un enfoque de integración orientado a proceso basado en mensajes XML. Es generalmente utilizada para el procesamiento de transacciones de negocio operacional en tiempo real.

Entre sus beneficios encontramos los siguientes:

- Integración no invasiva.
- El contenido de los datos (el mensaje) es irrelevante.
- Puede generalmente acceder a muchos puntos finales.

- El contenido del mensaje podría ser XML, Servicios Web, datos planos, etc.
- Los puntos finales pueden ser colas, protocolos, Servicios Web, Bases de Datos, adaptadores de aplicaciones, entre otros.

De acuerdo a las características antes mencionadas de este estilo de integración, se puede concluir que constituye una solución a la problemática planteada de integrar los componentes y arquitectónicos y tecnologías del proyecto SIC.

1.7 Framework de Desarrollo

La palabra inglesa "**framework**" define, en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular, que sirve como referencia para enfrentar y resolver nuevos problemas de índole similar.

En el desarrollo de software, un ***framework*** **de desarrollo** o **infraestructura digital de desarrollo**, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, con base a la cual otro proyecto de software puede ser más fácilmente organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

En el caso de los **frameworks para aplicaciones web** son frameworks diseñados para apoyar el desarrollo de sitios web dinámicos, aplicaciones web y servicios web. Este tipo de frameworks intenta aliviar el exceso de carga asociado con actividades comunes usadas en desarrollos web. Ejemplo de lo antes dicho es que muchos frameworks proporcionan bibliotecas para acceder a las bases de datos, estructuras para plantillas y gestión de sesiones, y con frecuencia facilitan la reutilización de código.

1.8 Arquitectura de Software en Sistemas Web

Aunque existen muchas variantes, una aplicación web suele responder a más de un estilo arquitectónico. Normalmente están estructurada como una aplicación cliente-servidor, este estilo arquitectónico define una relación entre dos aplicaciones como se muestra en la figura 1, en las cuales una de ellas (cliente) envía peticiones a la otra (servidor fuente de datos) que ofrece servicios valiéndose de consultas y actualizaciones a la base de datos y a su vez proporciona una interfaz de usuario. Este

estilo puede usar un amplio rango de protocolos y formatos de datos para comunicar con la información.

Fig. 1 Estilo arquitectónico Cliente-Servidor

Estas características aportan mayor seguridad en el sistema, puesto que los datos se almacenan en un servidor que generalmente ofrecen más control sobre la seguridad. Unido a esto se facilita el acceso y actualización de los datos, y el mantenimiento del sistema se hace más sencillo, puesto que los roles y las responsabilidades se distribuyen entre los distintos servidores a través de la red, lo que permite que un cliente no se vea afectado por un error en un en un servidor particular.

Es apropiado usar este estilo arquitectónico en los siguientes casos:

- La aplicación se basa en un servidor y soportara múltiples clientes.
- La aplicación está normalmente limitada a un uso local y área LAN controlada.
- Se están implementando procesos de negocios que se usaran a lo largo de toda la organización.
- Se quiere centralizar el almacenamiento, Backus, y mantenimiento de la aplicación.
- La aplicación debe soportar distintos tipos de clientes y dispositivos.

En este estilo arquitectónico el cliente y el servidor siguen un estilo lógico En Capas (N-Layer) como se muestra en la figura 2. Este estilo en Capas se basa en una distribución jerárquica de los roles y las responsabilidades, para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades, la funcionalidad que implementan. Este estilo descompone los servicios de forma tal que la mayoría de las iteraciones ocurren solo entre capas vecinas y estas a su vez pueden residir en la misma

máquina, o ser distribuidas entre varios equipos. Los componentes de cada una de estas capas se comunican con las componentes de las otras capas, a través de interfaces bien conocidas y cada nivel agrega las responsabilidades y abstracciones del nivel inferior.

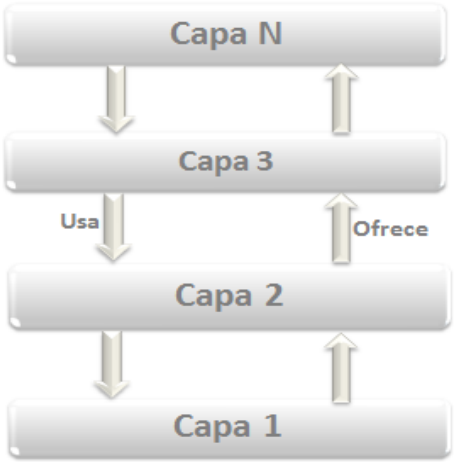

Fig. 2 Estilo arquitectónico En Capas

Los beneficios de esta distribución jerárquica de los roles y las responsabilidades en capas, se pueden enumerar como sigue:

- Abstracción: Los cambios se realizan a alto nivel y se puede incrementar o reducir el nivel de abstracción que se usa en cada capa del modelo.
- Aislamiento: Se pueden realizar actualizaciones en el interior de las capas, sin que esto afecte el resto del sistema.
- Rendimiento: Distribuyendo las capas en distintos niveles físicos se puede mejorar la escalabilidad, la tolerancia a fallos y el rendimiento.
- Testeabilidad: Cada capa tiene una interfaz bien definida, sobre las cual se pueden realizar pruebas y cambiar entre diferentes implementaciones de una capa.
- Independencia: Se elimina la necesidad de considerar el Hardware y el despliegue así como las dependencias con interfaces externas.

El uso de este estilo arquitectónico es apropiado cuando:

- Se tienen construidas capas de una aplicación anterior, que pueden reutilizare o integrarse.

- Se tienen aplicaciones que exponen su lógica de negocio a través de interfaces de servicios.

- La aplicación es compleja y tiene un alto nivel de diseño que requiere la separación para que los distintos equipos puedan concentrarse en distintas áreas de funcionalidad.

- La aplicación debe soportar distintos tipos de clientes y dispositivos.

- Se quieren implementar reglas o procesos de negocios complejos o configurables.

Dentro de este estilo arquitectónico en capas podemos encontrar, las arquitecturas de dos capas, las de tres capas que son las más tradicionales y las de N-Capas.

En este tipo de aplicaciones también se hace uso de algunos patrones arquitectónicos como MVC y MVP, para separar los conceptos entre la interfaz de usuario y la lógica de presentación.

1.9 Metodología de Desarrollo

Una metodología es un conjunto de procedimientos que permiten producir y mantener un producto software, definiendo una serie de pasos a seguir para obtener un software de calidad. Un proceso que define quién está haciendo qué, cuándo, y cómo alcanzar un determinado objetivo (Jacobson, 1999). Las metodologías se desarrollan con el objetivo de dar solución a los problemas existentes en la producción de software, que cada vez son más complejos. Estas abarcan procedimientos, técnicas, documentación y herramientas que se utilizan en la creación de un producto de software.
Ejemplos de metodologías Orientadas a Objetos son las siguientes:

- Proceso Unificado de Desarrollo de Software (RUP)
- Programación Extrema (XP)
- Marcos de Soluciones Microsoft (MSF)

1.9.1 Proceso Unificado de Desarrollo de Software (RUP)

RUP es la metodología de desarrollo que se decidió utilizar el proyecto, puesto que este es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo y trabajo de muchas metodologías. Es un proceso de software genérico que puede ser utilizado para una gran cantidad

de tipos de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones y diferentes tamaños de proyectos. Proporciona un enfoque disciplinado en la asignación de tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de muy alta calidad que satisfaga las necesidades de los usuarios finales, dentro de un calendario y presupuesto predecible.

RUP está basado en componentes, lo cual quiere decir que el sistema software en construcción está formado por componentes software interconectados a través de interfaces bien definidas. RUP utiliza el Lenguaje Unificado de Modelado (Unified Modeling Language, UML) para preparar todos los esquemas de un sistema software. De hecho, UML es una parte esencial de RUP.

No obstante, los verdaderos aspectos definitorios de RUP se resumen en tres frases claves, dirigido por casos de uso, centrado en la arquitectura, e iterativo e incremental. Esto es lo que hace único al Proceso Unificado.

RUP está compuesto por un conjunto de flujos de trabajo separados en dos grupos, los flujos de trabajo de ingeniería que son los siguientes, Modelamiento del negocio, Requerimientos, Análisis y diseño, Implementación, Prueba, y Despliegue, en el otro grupo están los flujos de trabajo de apoyo que son los siguientes, Administración del proyecto, Configuración y control de cambios, y Entorno, y cada uno de estos flujos está presente en las cuatro fases, Inicio, Elaboración, Construcción y Transición como se muestra en la siguiente figura.

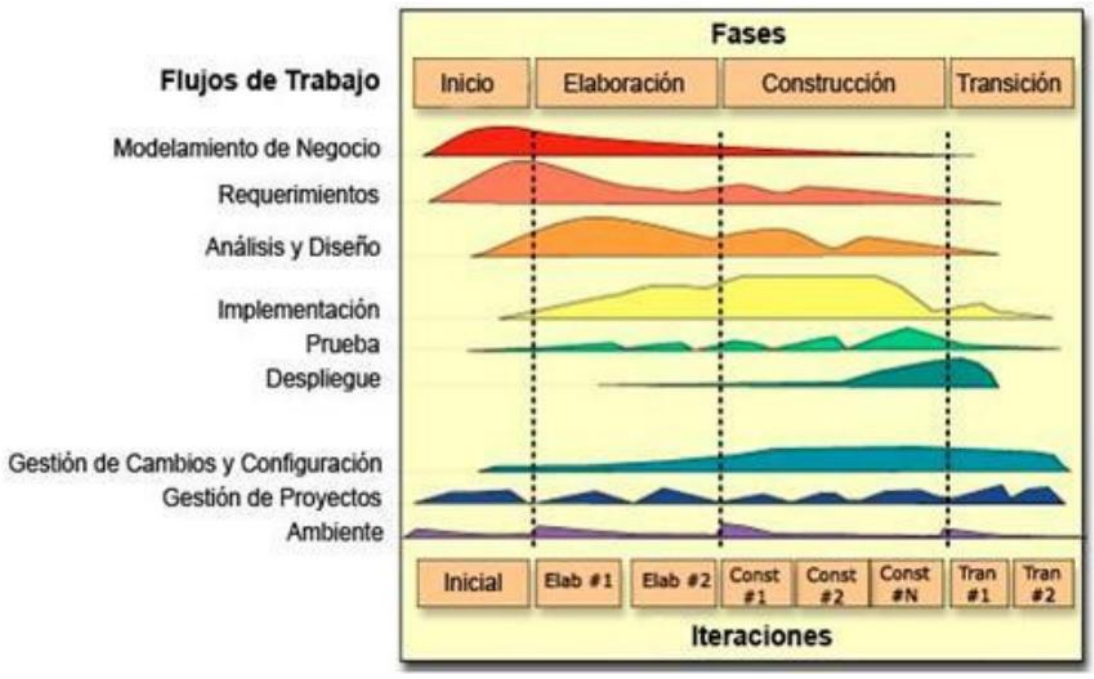

Fig. 3 Proceso de desarrollo de software según RUP

1.10 Caracterización de las Herramientas

1.10.1 Entorno Integrado de Desarrollo (IDE)

Spring Source Tool Suite (STS) es el entorno de desarrollo utilizado en el proyecto, este es un IDE basado en la versión Java EE de Eclipse, pero personalizado de manera especial para trabajar con el Framework Spring. STS proporciona herramientas para el trabajo con varias tecnologías enfocadas al desarrollo sobre la plataforma JavaEE como Grails y el lenguaje dinámico Groovy. Con su consola Spring Insight proporciona una interfaz gráfica en tiempo real de las métricas de rendimiento en las aplicaciones, que permite a los desarrolladores identificar y diagnosticar problemas desde sus escritorios.

Unido a esto STS apoya las aplicaciones dirigidas a los servidores locales, virtuales y basadas en nube. Es de libre acceso para el desarrollo y uso interno de las operaciones de negocios sin límites de tiempo.

Entre sus características más destacadas encontramos:

- Asistentes en la creación de proyectos Spring.
- Herramientas para la gestión de beans.
- Editores gráficos de archivos de configuración de Spring.
- Herramientas de desarrollo para Spring Web Flow y Spring Batch.

1.10.2 Herramienta Asistida por computadora para el modelado Visual Paradigm

Visual Paradigm es una herramienta para un diseño profesional, que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Utiliza el Lenguaje de Modelado Unificado (UML) ayudando a una rápida construcción de aplicaciones de calidad, con un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

Dentro de las características que ofrece se encuentran las siguientes:

- Entorno de creación de diagramas para UML 2.0.
- Diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa (versión profesional) e inversa.

- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Permite importar un área de trabajo.
- Disponibilidad de múltiples versiones para cada necesidad.
- Disponibilidad de integrarse en los principales IDEs.
- Disponibilidad en múltiples plataformas.

1.10.3 Postgres SQL

Gestor de Base de datos que realiza todo el almacenamiento de datos, y recibe solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

PostgreSQL es un sistema de gestión de base de datos relacional multihilo y multiusuario con más de seis millones de instalaciones además de ser orientada a objetos y ser un software libre. Se caracteriza por su amplia variedad de tipos nativos ya que provee nativamente soporte para números de precisión arbitraria, para texto de largo ilimitado, figuras geométricas y direcciones de IP. Presenta también una alta concurrencia pues permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos.

1.11 Proceso de evaluación de la arquitectura

Una evaluación es un estudio de factibilidad que pretende detectar posibles riesgos, así como buscar recomendaciones para contenerlos; en el caso de la arquitectura de software tiene como objetivo saber si pueden habilitar los requerimientos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los stakeholders (probadores) (Gustavo Andrés Brey, 2005).

Estas evaluaciones pretenden analizar e identificar riesgos potenciales en la estructura de la arquitectura, que puedan afectar al sistema de software resultante, verificar que los requerimientos no funcionales o atributos de calidad estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad.

Esta evaluación se puede realizar en cualquier momento según Kazman, pero propone dos variantes que agrupan dos etapas distintas: temprano y tarde (Erika Camacho, 2004).

- **Temprana**: No es necesario que la arquitectura esté completamente especificada para efectuar la evaluación, y esto abarca desde las fases tempranas de diseño y a lo largo del desarrollo.
- **Tarde**: Cuando ésta 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.

1.11.1 Partes de la arquitectura

Para poder analizar la arquitectura de software es necesario entender cuáles son sus partes y como pueden ser identificadas, de ahí se pueden saber con mayor precisión sus principales problemas.

Todo componente (elemento), relación entre componentes, o una propiedad que necesita ser externamente visible, con el objetivo de razonar sobre la habilidad del sistema de alcanzar sus requerimientos de calidad, o de soportar la descomposición del sistema en partes independientemente forma parte de la arquitectura.

1.11.2 Atributos de calidad

Un atributo de calidad es una característica de calidad que afecta a un elemento. Donde el término "característica" se refiere a aspectos no funcionales y el término "elemento" a un componente (Gómez, 2007).

Estos atributos constituyen las cualidades por las cuales puede ser evaluada una arquitectura y se pueden clasificar en dos categorías (Erika Camacho, 2004).

- **Observables vía ejecución**: Aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución. La descripción de algunos de estos atributos se presenta en la tabla 2.
- **No observables vía ejecución:** Aquellos atributos que se establecen durante el desarrollo del sistema. La descripción de algunos de estos atributos se presenta en la tabla 3.

Atributo de Calidad	Descripción
Disponibilidad	Es la medida de disponibilidad del sistema para el uso.
Confidencialidad	Es la ausencia de acceso no autorizado a la información.
Funcionalidad	Habilidad del sistema para realizar el trabajo para el cual fue concebido.
Desempeño	Es el grado en el cual un sistema o componente cumple con

	sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria.
Confiabilidad	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo.
Seguridad Externa	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información.
Seguridad Interna	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos.

Tabla. 2 Descripción de atributos de calidad observables vía ejecución *(Erika Camacho, 2004)*

Atributo de Calidad	Descripción
Configurabilidad	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema.
Integrabilidad	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.
Integridad	Es la ausencia de alteraciones inapropiadas de la información.
Interoperabilidad	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de integrabilidad.
Modificabilidad	Es la habilidad de realizar cambios futuros al sistema.
Mantenibilidad	Es la capacidad de someter a un sistema a reparaciones y evolución. Capacidad de modificar el sistema de manera rápida y a bajo costo.
Portabilidad	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de comsistemación. Estos ambientes pueden ser hardware, software o una combinación de los dos.
Reusabilidad	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.

Escalabilidad	Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental.
Capacidad de Prueba	Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba.

Tabla. 3 Descripción de atributos de calidad no observables vía ejecución *(Erika Camacho, 2004)*

1.11.3 Técnicas de evaluación de arquitectura

Existen un grupo de técnicas para evaluar la arquitectura que se clasifican en cualitativas y cuantitativas (Erika Camacho, 2004).

Técnicas de cuestionamiento o cualitativas: Utilizan preguntas cualitativas para preguntarle a la arquitectura.

- Cuestionarios
- Lista de verificación
- Escenarios

Cuantitativas: Sugiere hacerle medidas cuantitativas a la arquitectura.

- Utiliza métricas arquitectónicas, como acoplamiento, cohesividad en los módulos, profundidad en herencias, modificabilidad.
- Simulaciones, Prototipos, y Experimentos.

En la siguiente figura son mostradas las técnicas antes mencionadas.

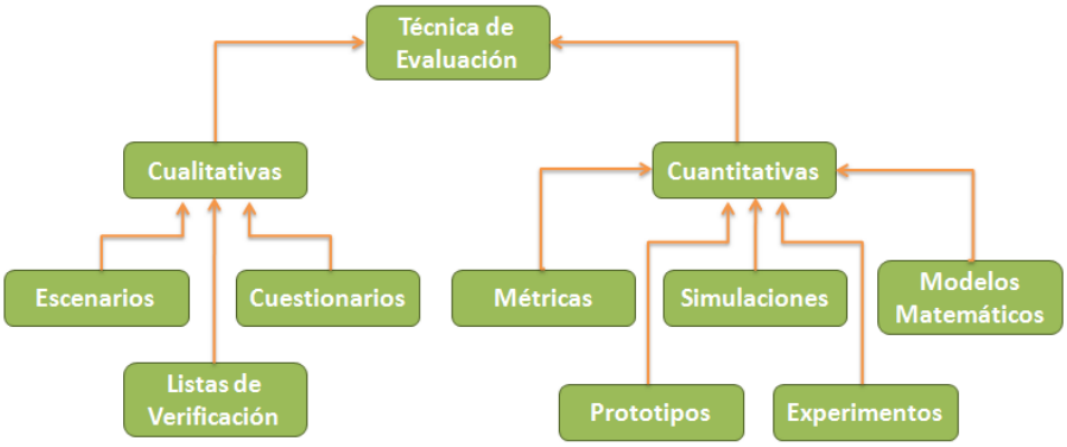

Fig. 4 Clasificación de las técnicas de evaluación *(Gómez, 2007)*

1.11.4 Métodos de evaluación de arquitectura

Existen varios métodos para evaluar una arquitectura, cada uno con enfoques diferentes pero con un mismo propósito detectar anomalías en las arquitecturas de software.

1.11.4.1 Método de Análisis de Arquitectura de software (SAAM)

El método SAAM es aplicable en los proyectos donde el atributo de calidad modificabilidad es el de mayor interés, ya que este método evalúa varios atributos de calidad como la flexibilidad, mantenimiento, portabilidad, extensibilidad, integridad y funcionalidad pero es la modificabilidad su objetivo principal y es el mejor para evaluar este atributo ya que provee escenarios de cambios para realizar la evaluación (Erika Camacho, 2004).

1.11.4.2 Architecture Trade-off Analysis Method (ATAM)

El método ATAM está inspirado en tres áreas distintas:

Los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM. El nombre del método ATAM surge del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros.

El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. Estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas, entre otros (Erika Camacho, 2004).

El método consta de nueve pasos principales agrupados en cuatro fases (Presentación, Investigación y Análisis, Pruebas y Reporte).

1. Presentar el ATAM.
2. Controlador de negocio actual.
3. Presentar arquitectura.
4. Identificar los enfoques de arquitectura.
5. Generar árbol de utilidad de los atributos de calidad.
6. Analizar enfoques de arquitectura.
7. Lluvia de ideas y la prioridad de escenarios.
8. Analizar enfoques de arquitectura.
9. Presentar los resultados.

1.11.4.3 Bosch (2000)

Plantea que: El proceso de evaluación debe ser visto como una actividad iterativa, que forma parte del proceso de diseño, también iterativo. Una vez que la arquitectura es evaluada, pasa a una fase de transformación, asumiendo que no satisface todos los requerimientos. Luego, la arquitectura transformada es evaluada de nuevo .Este método consta de 5 pasos divididos en dos etapas (Erika Camacho, 2004).

1.11.4.4 Losavio (2003)

Es un método para evaluar y comparar arquitecturas de software candidatas, que hace uso del modelo de especificación de atributos e calidad adaptado del modelo ISO/IEC 9126. La especificación de los atributos de calidad haciendo uso de un modelo basado en estándares internacionales ofrece una vista amplia y global de los atributos de calidad, tanto a usuarios como arquitectos del sistema, para efectos de la evaluación.

Conclusiones Parciales

En el presente capítulo se realizó un estudio de las diferentes arquitecturas de software en sistemas web, evidenciándose las ventajas y desventajas de cada una de ellas.

Se concretaron los principales etilos de integración en sistemas de software, comprobándose que el más adecuado a utilizar es el estilo EAI según sus características.

Se definieron los principales métodos de evaluación de arquitecturas software existentes, evidenciándose las ventajas del método ATAM sobre los demás.

Capítulo 2 Integración de los componentes arquitectónicos y tecnologías.

En este capítulo se describe la solución arquitectónica del sistema así como las vistas arquitectónicas definidas por la metodología RUP, se concluye la solución de la arquitectura propuesta y se relacionan los patrones, métodos y un grupo ordenado de pasos para lograr con éxito la integración de los componentes arquitectónicos y tecnologías del proyecto SIC.

2.1 Estilo Arquitectónico Propuesto

Hoy en día el desarrollo de aplicaciones web se hace cada vez más difícil y el problema se ha ido agravado en el entorno actual, donde las aplicaciones empiezan a entrar en la categoría Web 2.0 lo cual implica un montón de tecnologías, tales como Hipertexto, Lenguaje de Marcado (HTML), Hojas de Estilo (CSS), Java Script asíncrono y XML (Ajax), XML, servicios Web, Java y bases de datos. Para empeorar las cosas, mientras que la complejidad de la creación de aplicaciones sigue creciendo, los tiempos de respuesta siguen disminuyendo. Es por esto que se hace necesario el uso nuevas tecnologías y conceptos que posean un gran potencial y supongan una revolución a la hora de desarrollar aplicaciones web.

En busca de estos conceptos el sistema a construir estará desarrollado bajo tecnología JEE (en inglés: Java Enterprise Edition). Esta plataforma define estándares para desarrollar y ejecutar aplicaciones en el lenguaje de programación Java, empleando arquitecturas que definen un modelo multicapa y que se apoyan en componentes de software modulares. JEE incluye varias tecnologías de servicios web. Las aplicaciones desarrolladas en esta plataforma tienden a ser portables, escalables, robustas y seguras a la vez que son integrables con tecnologías anteriores. De acuerdo a estas características el sistema será basado en el estilo arquitectónico Cliente-Servidor y este a su vez seguirá un estilo lógico en 3 capas, aprovechando las ventajas que este nos brinda, como se muestra a continuación:

- **Capa de Presentación**: Es la encargada de generar la interfaz de usuario, en función de las acciones llevadas a cabo por el mismo.
- **Capa de Negocio**: Contiene todo la lógica que modela los procesos de negocio y es donde se realiza todo el procesamiento necesario para atender las peticiones del usuario.

- **Capa de acceso a Datos**: Es la encargada de hacer persistente toda la información. Suministra y almacena información para el nivel de negocio.

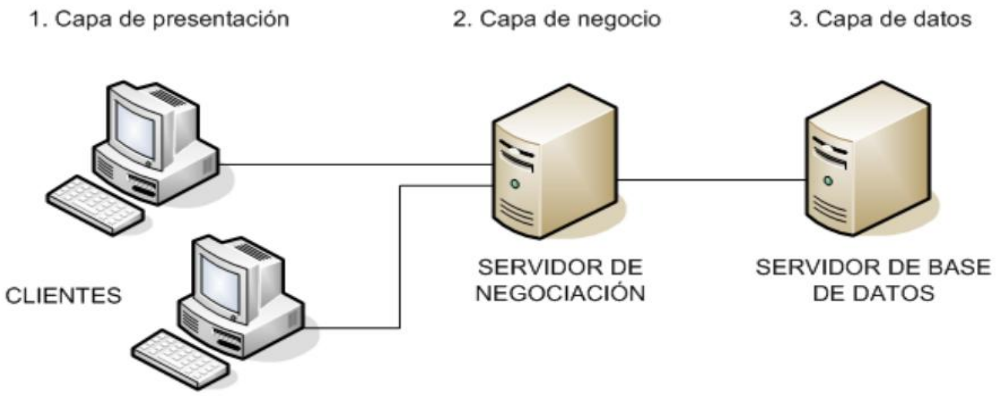

Fig. 5 Arquitectura 3-Capas

En la anterior figura se pude evidenciar más claramente la distribución de cada una de las capas de la arquitectura.

2.1.1 Capa de Presentación

Esta capa tiene la responsabilidad de presentar al usuario los conceptos de negocio mediante una interfaz de usuario, interpretar sus acciones, facilitar la explotación de dichos procesos e informar sobre la situación de los procesos de negocios e implementación de las reglas de validación de dicha interfaz. A manera general desde el punto de vista del usuario final, esta capa es la "Aplicación" en sí, y de nada sirve haber planteado la mejor arquitectura del mundo si no podemos facilitar la explotación de ella de la manera más satisfactoria posible para el usuario.

Está estructurada en componentes que implementan la funcionalidad requerida para que los usuarios interactúen con la aplicación; y dichos componentes están divididos en varias sub-capas aplicando el patrón Modelo Vista Presentador (MVP)**.**

- **Subcapas de Procesos de Interfaz de Usuario (Controladores)**: Estos controladores son típicos de patrones MVC, MVP y derivados. Ayudan a sincronizar las interacciones del usuario y son útiles para conducir el proceso utilizando componentes separados de los componentes propiamente gráficos.
- **Subcapa de componentes visuales (Vistas)**: Estos componentes proporcionan el mecanismo base para que el usuario utilice la aplicación.

Por lo que son componentes que formatean datos en cuanto a controles visuales, y también reciben datos proporcionados por el usuario.

Este patrón separa el modelo del dominio, la presentación y las acciones basadas en la interacción con el usuario en tres clases separadas. La vista le delega a su Presentador toda la responsabilidad del manejo de los eventos del usuario. El Presentador se encarga de actualizar el modelo cuando ocurre un evento en la vista, pero también es responsable de actualizar a la vista cuando el modelo le indica que ha cambiado. El modelo no conoce la existencia del Presentador; así que si el modelo cambia por acción de algún otro componente que no sea el Presentador, se dispara un evento para que el Presentador se entere.

A la hora de implementar este patrón, se identifican los siguientes componentes (De La Torre Llorente, y otros, 2010)**:**

- **IView**: Interfaz mediante la cual el Presentar se comunica con la vista.
- **View**: Vista que implementa la interfaz IView y se encarga de manejar los aspectos visuales. Mantiene una referencia a su Presenter al cual le delega la responsabilidad del manejo de los eventos.
- **Presenter**: Contiene la lógica para responder a los eventos y manipula el estado de la vista mediante una referencia a la interfaz IView. El Presenter utiliza el modelo para saber cómo responder a los eventos. El presentador es responsable de establecer y administrar el estado de una vista.
- **Model**: Está compuesto por los objetos que conocen y manejan los datos dentro de la aplicación. Ejemplo de esto son las clases que conforman el modelo de negocio.

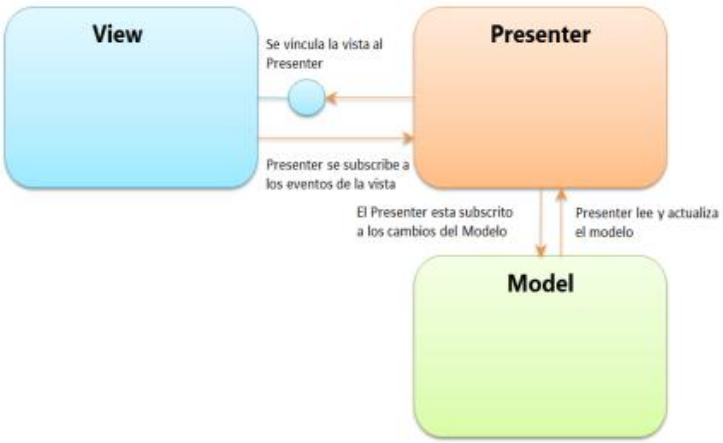

Fig. 6 Patrón MVP (Modelo Vista Presentador)

En la anterior figura se puede evidenciar los componentes antes mencionados y las relaciones existentes entre ellos.

2.1.1.1 Tecnología a Usar

Una de las peculiaridades de esta capa es que necesita cumplir con los requisitos de diseño estéticos de la aplicación, es aquí donde no se puede olvidar la inconsistencia de las aplicaciones web con los navegadores, que pueden causar problemas en el desarrollo y soporte de las mismas, principalmente debido a la falta de adicción de los mismos a algunos estándares web que suelen necesitar un código específico, para lograr un front-end correcto de la aplicación.

Esta desventaja puede ser solucionada mediante el uso de tecnologías que ignoren las configuraciones de los navegadores o conviertan el código fuente de las interfaces de usuarios a un código compatible con cualquier navegador web, permitiendo así más control sobre estas interfaces. En este caso se optara por la segunda opción para evitar estos posibles problemas de incompatibilidad; es decir traducir el código fuente de las interfaces de usuario a un código compatible con cualquier navegador web, mediante el uso del framework de desarrollo en java **Google Web Toolkit (GWT).** Este framework de código abierto permite escapar de la matriz de tecnologías usadas actualmente para escribir aplicaciones AJAX, las cuales son difíciles de manejar y propensas a errores. Con GWT se puede desarrollar y depurar aplicaciones AJAX usando el lenguaje de programación java en el entorno de desarrollo preferido lo cual facilita la aplicación de metodologías modernas y una vez acabada la aplicación, GWT compila y traduce dicho programa a JavaScript y HTML compatible con cualquier navegador web.

Internamente GWT funciona como un conversor de Java *bytecode* a estructuras de Javascript optimizadas para la manipulación de páginas, por lo que no se requieren componentes especiales en los clientes para poder visualizar páginas generadas con esta tecnología. La conversión a Javascript, al no realizarse por medio de un intérprete sino al momento de compilar el proyecto, reduce el *overhead* de ejecución de las páginas a sólo el procesamiento necesario para vincular elementos del servidor con llamadas desde la página misma.

GWT se encarga también de las comunicaciones entre el cliente y el servidor, realizándolas mediante Ajax en forma asincrónica. El servidor no necesariamente tiene que estar programado en java, pudiéndose utilizar cualquier otro lenguaje.

2.1.2 Capa de Negocio

Esta capa es el corazón de software, tiene la responsabilidad de representar conceptos del negocio, información sobre la situación de los procesos de negocio e implementación de las reglas del dominio. Básicamente se puede ver como una capa intermedia que implementa las funcionalidades principales del sistema y encapsula toda la lógica del negocio relevante. En ella residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él.

2.1.2.1 Tecnología a Usar

En los últimos años, las comunidades de desarrollo han tratado de proveer a los programadores nuevas plataformas de trabajo, las cuales han demostrado ser escalables y robustas pero no permiten el desarrollo ágil y rápido del software. Unido a esto el ciclo de desarrollo de código, compilación, embalaje, implementación, prueba y depuración toma demasiado tiempo para cualquier proyecto de relevante envergadura. Entonces surge la necesidad de adoptar nuevas filosofías, como el uso de lenguajes dinámicos para crear aplicaciones con menos código de fontanería, reduciendo los tiempos de desarrollo y aumentando la flexibilidad dinámica de las aplicaciones. Esta idea alcanza su máximo potencial cuando se habla de lenguajes dinámicos orientados a objetos. Estos lenguajes permiten definir clases en tiempo de ejecución, permitiendo que los objetos se ajusten a esquemas que no pueden conocerse de antemano, o añadir propiedades y métodos a un objeto para que haga más cosas de aquellas para las que fue inicialmente concebido. De acuerdo a estas características y siguiendo la idea del desarrollo sobre la plataforma java, se llega a un lenguaje dinámico desarrollado desde y para java, **Groovy**.

Groovy es un lenguaje de programación con una sintaxis de Java, se compila en bytecode igual que Java y se ejecuta en la JVM. Groovy se integra perfectamente con Java y permite la mezcla entre ambos; al punto de ser posible instanciar objetos Java desde Groovy y viceversa, es mucho más flexible y potente que esta y es totalmente

compatible con sus librerías. De forma general Groovy nació con la misión de llevarse bien con Java, vivir en la Máquina Virtual y soportar los tipos de datos estándar, pero añadiendo características dinámicas y sintácticas presentes en otros lenguajes como Python, Smalltalk o Ruby. Lo que Groovy aporta es una sintaxis que aumenta enormemente la productividad y un entorno de ejecución que nos permite manejar los objetos de formas que en Java serían extremadamente complicadas. La diferencia entre ambos lenguajes está en su propia naturaleza, que provoca que el código necesario para una determinada tarea en Groovy sea en promedio unas cinco veces más breve que el equivalente en Java. Las ventajas de Groovy no terminan aquí, unido a lo dicho anteriormente Groovy nos brinda un marco excelente y de gran alcance para el desarrollo de aplicaciones web; a través de un framework web dinámico y ágil construido sobre él y que constituye su proyecto más emblemático **Grails**.

Grails es el entorno para desarrollo de aplicaciones web sobre la plataforma Java Enterprise Edition que se utilizará en el desarrollo de la lógica del negocio; el cual nos brinda un marco de trabajo destinado a disminuir el tiempo de desarrollo y reutilizar los mejores componentes arquitectónicos de esta plataforma. Está diseñado sobre las bases de los principios que constituyen paradigmas del desarrollo ágil y hace alusión a que cada aspecto de la aplicación existe en un sólo lugar, no es necesario modificar múltiples capas de aplicación para realizar cambios. Debido a esto en su diseño Grails sigue el patrón Modelo-Vista-Controlador (MVC) el cual tiene como idea principal la Presentación Separada, que consiste en hacer una división clara entre objetos del dominio y la presentación. Los objetos del dominio deben ser completamente auto contenidos y trabajar sin referirse a la presentación, también deben ser capaces de soportar presentaciones múltiples que pudieran llegar a ser simultaneas (De La Torre Llorente, y otros, 2010).

Este patrón arquitectónico establece que los componentes de un sistema de software deben organizarse en tres capas distintas según su misión:

- **Modelo, o capa de datos**: Contiene los componentes que representan y gestionan los datos manejados por la aplicación.
- **Vista, o capa de presentación**: Los componentes de esa capa son responsables de mostrar al usuario el estado actual del modelo de datos, y presentarle las distintas acciones disponibles.

- **Capa de control**: Contendrá los componentes que reciben las ordenes del usuario, gestionan la aplicación de la lógica de negocio sobre el modelo de datos, y determinan que vista deben mostrarse a continuación.

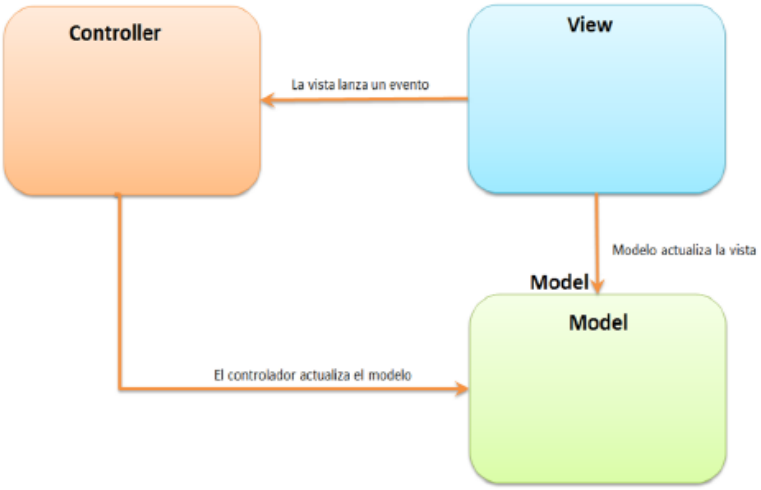

Fig. 7 Patrón MVC (Modelo Vista Controlador)

En la anterior figura se puede evidenciar los componentes antes mencionados y las relaciones existentes entre ellos.

Este Grails hace uso además de la estrategia Conventions Over Configurations, mediante la cual el programador tiene la opción de seguir un conjunto de pautas predefinidas, con el objetivo de no perder un tiempo vital configurando la arquitectura del sistema. Finalmente Grails es ligero y es posible observar los cambios realizados en tiempo real, sin necesidad de recompilación o redespliegue.

Está basado en tecnologías maduras y establecidas, tales como Hibernate (para mapeo objeto-relacional), Spring (para inyección de dependencias), SiteMesh(para las plantillas), Quartz (para planificación de tareas), y JavaServer Pages (para el nivel de visión), como es mostrado en la siguiente figura la cual representa la arquitectura interna de grails.

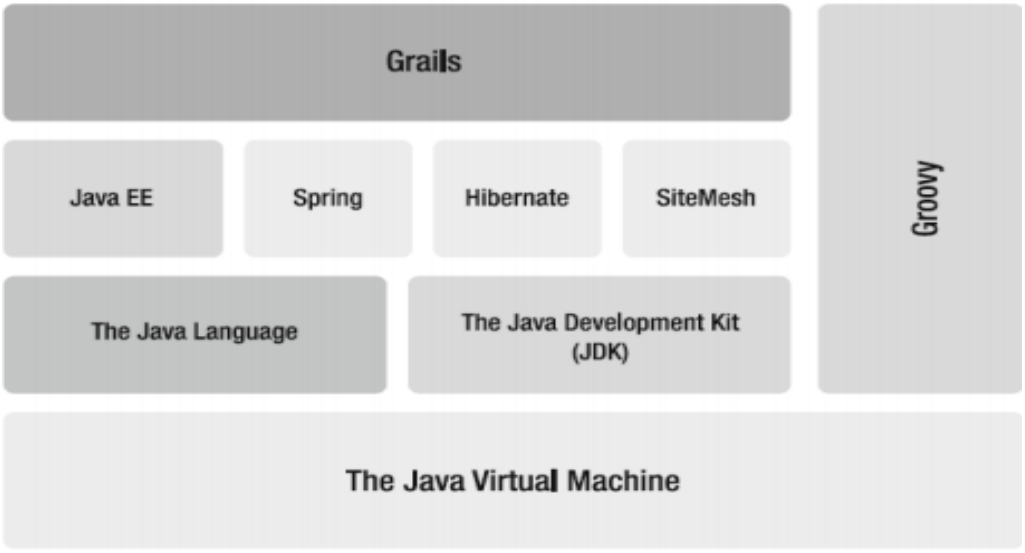

Fig. 8 Arquitectura de Grails

Las aplicaciones Grails se empaquetan como los tradicionales archivos WAR de Java y se puede implementar en cualquier Servidor de aplicaciones Java, como Tomcat, Jetty, JBoss, o WebLogic.

Al utilizar como base el lenguaje dinámico Groovy, Grails facilita tareas que consisten en complicados procesos para otros frameworks de Java, como el procesamiento de envío de formularios, implementación de tag libraries o escribir casos de prueba. Groovy provee al framework con una herramienta concisa y expresiva que facilita estas operaciones y una gran capacidad de mantenimiento.

A modo general Grails permite a los desarrolladores trabajar sobre un ambiente familiar, robusto y confiable, sin necesidad de extensos archivos de configuración, pudiendo integrar sistemas existentes, logrando así un proceso de desarrollo ágil y productivo.

Según cuenta Graeme Rocher en “The definitive guide to grails”:

“El objetivo de Grails era ir más allá de lo que otros lenguajes y sus frameworks asociados podían ofrecer en el espacio de las aplicaciones web. Grails se proponía hacer lo siguiente

-Integrarse estrechamente con la plataforma java.

-Ser sencillo en la superficie, pero mantener la flexibilidad para acceder a los potentes frameworks java sobre los que se construía.

-Aprender de los errores cometidos en la ya madura plataforma java.

El uso de Groovy como punto de partida para el framework le proporciono una enorme ventaja inicial. El objetivo de Groovy era crear un lenguaje que permitiese a programadores java una fácil transición al mundo de los lenguajes de script con tipado dinámico, proporcionando funcionalidades imposibles de implementar con lenguajes de tipado estático."

2.1.3 Capa de Acceso a Datos

Esta capa contiene la lógica de comunicación con otros sistemas que llevan a cabo tareas por la aplicación. Es la encargada de hacer persistente y suministrar toda la información para el nivel de negocio. Está dividida en tres subcapas las cuales facilitan el acceso de forma efectiva a la base de datos desde un contexto orientado a objetos.

- **Subcapa de ORM** (*Object-Relational-Mapping*) o *"mapeo de objetos a bases de datos":* Interfaz que traduce la lógica de los objetos a la lógica relacional y está formada por objetos que permiten acceder a los datos y que contienen en sí mismos el código necesario para hacerlo. En nuestro caso esta interfaz está dada por **GORM** (Grails O*bject Relational Mapping*), un gestor de persistencia escrito en Groovy, para controlar el ciclo de vida de las entidades y proporcionar una serie de métodos dinámicos, los cuales se crean en tiempo de ejecución para cada entidad y facilitan enormemente las búsquedas.

  GORM está construido sobre Hibernate, una herramienta de mapeo objeto-relacional, que se encarga de relacionar las entidades de una clase con las tablas de una base de datos, y las propiedades de las entidades con los campos en las tablas. Entonces cada una de las operaciones que se realicen sobre los objetos del modelo de datos, serán traducidas por Hibernate en las sentencias SQL necesarias para quedar reflejadas en la base de datos.
- **Subcapa física de la Base de Datos:** Contiene los componentes físicos de la base de datos, es decir los archivos de datos que residen en el disco.
- **Subcapa lógica de la Base de Datos:** Contiene las estructuras que mapean los datos hacia los componentes físicos.

2.2 Comunicación de una aplicación GWT con el servidor

Todas las aplicaciones GWT se ejecutan como código JavaScript y HTML compatible con cualquier navegador web del usuario final, esto les permite diferenciarse de las tradicionales aplicaciones web, puesto que las páginas construidas con Google Web Toolkit corren como aplicaciones sobre el navegador, lo cual es permite realizar actualizaciones en la interfaz de usuario sin hacer nuevas peticiones al servidor; sin embargo, como todas las aplicaciones cliente/servidor, los programas en Google Web Toolkit necesitan pedir ciertos datos al servidor para realizar determinadas tareas, como enviar solicitudes y recibir actualizaciones.

Este tipo de tareas GWT las realiza mediante llamadas asíncronas al servidor para el intercambio de información. Este tipo de llamadas se realizan en un hilo de ejecución diferente, por lo que inmediatamente después de realizar la llamada, el programa continúa con su ejecución en paralelo con la llamada anterior, lo cual en dependencia del hardware utilizado puede contribuir a una mayor agilidad en la aplicación, reduciendo los requisitos de una solicitud. Utilizando este tipo de llamadas GWT permite diferentes maneras de comunicarse con un servidor, estas maneras están relacionadas y dependen del servidor con el cual se va a interactuar a través de la red.

- Si se puede ejecutar Java en el servidor, las llamadas de procedimiento remoto o RPC que brinda GWT es la mejor opción. El RPC en Google Web Toolkit es un mecanismo para enviar objetos Java desde y hacia un servidor a través de HTTP (Protocolo de Transferencia de Hipertexto) estándar (Google, 2011). RPC da la oportunidad de mover toda la lógica de la interfaz de usuario al cliente, lo que mejora el funcionamiento de la aplicación, reduce el ancho de banda usado, reduce la carga al servidor y le presenta al usuario final una experiencia más agradable navegando por la página. El código del lado del servidor que es invocado desde el cliente es frecuentemente llamado “servicio”, por lo que el “llamar procedimientos remotos” es comúnmente llamado como invocación de servicios.

- Si la aplicación se comunica con un servidor que no puede albergar servlets Java, o uno en que ya se utiliza otro formato de datos como JSON (JavaScript Object Notation) o XML (lenguaje de marcas extensible), se pueden hacer peticiones HTTP para recuperar los datos. GWT ofrece clases genéricas HTTP

que se pueden utilizar para construir la petición, y clases de JSON y XML en el cliente, que se pueden utilizar para procesar la respuesta.

2.3 Captura y procesamiento de los datos en el servidor

Como se había explicado antes, Grails es el entorno para desarrollo de aplicaciones web sobre la plataforma Java Enterprise Edition que se utilizará en el desarrollo de la lógica del negocio, el cual en su diseño sigue el patrón MVC.

En las aplicaciones web con este diseño los controladores son los componentes responsables de interceptar las peticiones HTTP del navegador y generar la respuesta correspondiente (que puede ser XML, HTML, JSON, o cualquier otro formato), ya sea en el propio controlador o delegando el trabajo en una vista.

La convención utilizada en Grails es que para cada petición HTTP, Grails determinará el controlador que debe invocar en función de las reglas fijadas y creará una instancia de la clase elegida e invocará la acción correspondiente. Estos controladores presentan numerosas funcionalidades para procesar los datos de entrada haciendo uso del estándar de servicios de grails, los cuales permiten manejar las peticiones de los clientes utilizando las capacidades del contenedor Spring para procesarlos y asociarlos automáticamente a los objetos del modelo.

2.4 Integración entre la capa de presentación y capa de negocio

El sistema en construcción esta implementado siguiendo el diseño del patrón Modelo-Vista-Controlador definido por Grails.

Grails incluye la tecnología GSP (Groovy Server Pages), inspirada en las páginas JSP, debido a lo cual las vistas son archivos con extensión GSP, básicamente estas páginas son archivos de textos que contienen algún tipo de lenguaje de marcado (normalmente HTML) junto con bloques dinámicos que pueden definirse de varias maneras utilizando el lenguaje dinámico Groovy.

Las páginas construidas con GWT que es el framework de desarrollo utilizado en la capa de presentación pueden ser GSP y mediante el uso del plugin (GWT-Grails) que se encuentra en la página oficial de grails http://grails.org/ , se puede hacer fácil incorporar código de GWT en las vistas GSP generadas por grails, y debido a que grails utiliza como lenguaje base Groovy, el cual hereda su sintaxis de Java y el código GWT es código java, es posible la integración entre ambos lenguajes, al punto de ser posible instanciar objetos Java desde Groovy y viceversa, por lo que es recomendable manejar las peticiones de los clientes al servidor mediante las llamadas de

procedimientos remotos (RPC) que brinda GWT, enviando así un objetos java desde y hacia el servidor a través de HTTP estándar.

2.4.1.1 Configuración del servicio RPC de GWT

Como se había explicado antes el servicio RPC de GWT hace que sea fácil para los componentes del cliente y el servidor de la aplicación web el intercambio de objetos Java a través de HTTP. El código del lado del servidor que se invoca desde el cliente no es más que los servicios proporcionados por el servidor, los que en este caso son suministrados por Grails, el cual hace posible definir servicios para cada una de las clases del dominio. Estos servicios para Grails son clases cuyo nombre siguen el patrón XService, donde X es el nombre de la clase del dominio a la cual están asociados.

Basado en lo anterior al configurar el servicio de RPC, este se centrará en tres elementos principales, involucrados en las llamadas a los procedimientos que se ejecutan en los servidores remotos.

- El servicio que se ejecuta en el servidor (método que es invocado).
- El código de cliente que invoca el servicio.
- Los datos de los objetos Java que pasan entre el cliente y el servidor.

Con el fin de definir un mecanismo RPC, que se encargue de manejar los tres elementos antes mencionados, del lado del cliente es necesario definir dos interfaces para cada servicio. Debido a que normalmente se definen varias clases del dominio, y Grails define un servicio para cada clase del dominio, existirían varios servicios y para cada uno de ellos entonces habría dos interfaces para la comunicación entre el cliente y el servidor. Con el fin de evitarse el trabajo con tantas interfaces de comunicación se puede definir un servicio general que tenga acceso a los demás servicios mediante una relación de composición. Este servicio general extendería de una clase que implemente el patrón de diseño Abstract Factory, este patrón solucionaría el problema de crear diferentes familias de objetos servicios.

De esta manera se tendría un método encargado de crear el objeto necesario en cada llamada, el cual tendrá acceso a cada funcionalidad del servicio solicitado como se muestra a continuación:

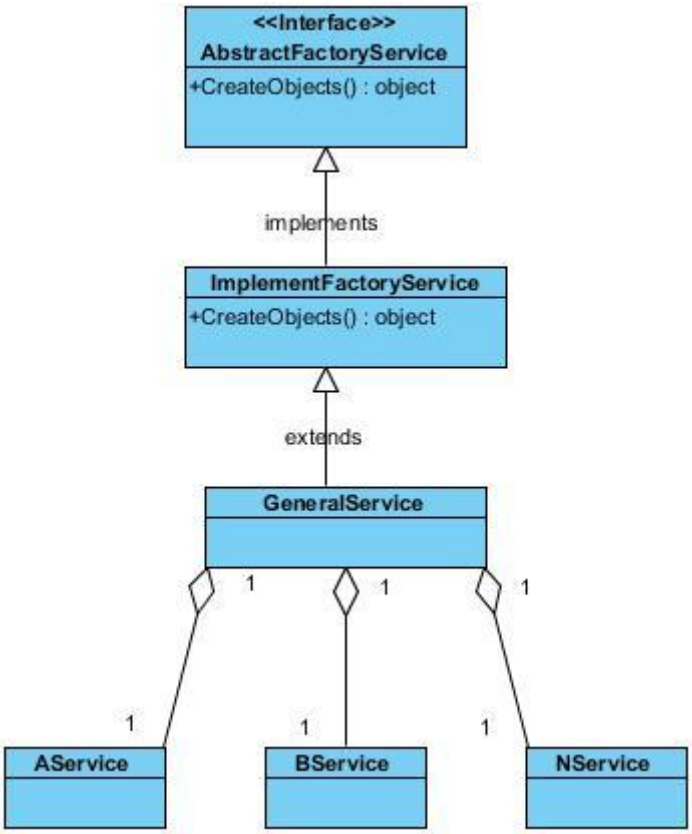

Fig. 9 Organización de las Clases del Diseño

Una vez implementado este patrón solo será necesario tener dos interfaces una síncrona que extienda de la interfaz RemoteService importada por el framework y contenga una lista de todos los métodos RPC que serán invocados. Esta interfaz debe tener el mismo nombre que el servicio al cual está asociada para poder localizarlo y es la versión definitiva del mismo al ser implementada por él.

```
import com.google.gwt.user.client.rpc.RemoteService;

public interface GeneralService extends RemoteService
{
    public String BuscarUsuario(int id);
}
```

Fig. 10 Ejemplo de Interfaz para un servicio

Una vez definida la interfaz síncrona para el servicio, es necesario invocar el servicio desde el cliente, pero aun la interfaz síncrona no es suficiente. Para lograr esta comunicación es necesario pasar un objeto de devolución de llamada AsyncCallback a todos los métodos de servicio y para agregar un parámetro AsyncCallback a todos los métodos del servicio, se debe definir una nueva interfaz de la siguiente manera:

- El nombre de esta interfaz debe ser el mismo que el de la interfaz de servicio, seguido del sufijo *Async.*

- Debe estar ubicado en el mismo paquete que el interfaz de servicio.

- Cada método de esta interfaz tiene que tener el mismo nombre que tenía en la interfaz de servicio con una diferencia importante: el método no tiene ningún tipo de retorno y el último parámetro es un objeto de tipo AsyncCallback.

```
import com.google.gwt.user.client.rpc.AsyncCallback;

public interface GeneralServiceAsync
{
    public void BuscarUsuario(int id,AsyncCallback<String> callback);
}
```

Fig. 11 Ejemplo de Interfaz Asíncrona

La relación entre una interfaz de servicio y su contraparte asíncrona debe seguir las normas de denominación antes mencionadas puesto que el compilador de GWT depende de estos estándares de nomenclatura con el fin de generar el código apropiado para implementar el RPC.

2.4.1.2 Realización de la llamada de procedimiento remoto

Para hacer una llamada a un procedimiento remoto, se debe especificar una forma de devolución de llamada que se ejecute cuando finalice la llamada. Esta forma de devolución se especifica pasando un objeto AsyncCallback a la clase proxy del servicio RPC. El objeto AsyncCallback contiene dos métodos, uno de los cuales se denomina en función de si la llamada falla **onFailure (Throwable)** y el otro si tiene éxito **onSuccess (T)**.

```
        //Creando el proxy del cliente
        GeneralServiceAsync rpcService=(GeneralServiceAsync)GWT.create(GeneralService.class);

        //Pasando el objeto AsyncCallback a la clase proxy
        rpcService.BuscarUsuario(id, new AsyncCallback<String>()
        {
            public void onSuccess(String result)
            {
              //Codigo para el exito de la llamada
            }

            public void onFailure(Throwable caught)
            {
              //Codigo para el fallo de la llamada
            }
         });
    }
}
```

Fig. 12 Realización de la llamada de procedimiento remoto

2.5 Componentes del sistema

Para el correcto funcionamiento del sistema se desarrollaron un conjunto de componentes de servicios de negocio, que permiten dar cumplimiento a los requisitos funcionales identificados para el mismo. Dichos componentes se identifican como sigue:

- **Herramienta de Simulación de Montecarlo:** Este componente permitirá la caracterización probabilística de variables aleatorias manejadas por el sistema, la propagación de la incertidumbre asociada a cada variable de los modelos matemáticos manejados en cada una de las metodologías, cuantificar el nivel de incertidumbre asociado a la salida de estas variables y realizar pruebas de bondad de ajuste con el objetivo de realizar predicciones a un cierto nivel de certeza.
- **Componente de auditoría:** Este componente es el encargado de registrar a través de trazas todas las acciones realizadas sobre el sistema, que son activadas por el usuario o por el mismo sistema; con el objetivo de realizar el análisis y evaluar la planificación, el control, la eficacia y la seguridad del proyecto, identificando riesgos y protegiendo la Información.
- **Componente de seguridad:** Este componente es el encargado de limitar el acceso al sistema sólo a los usuarios del mismo, controlar el acceso de estos a los recursos según su perfil y generar los Logs de seguridad correspondientes; gestionando así la seguridad del sistema.
- **Módulo de Reporte:** Este componente es el encargado de facilitar la obtención y presentación de la información procedente de los análisis de confiabilidad al

usuario, con el objetivo que este realice el análisis de la misma y tome las decisiones pertinentes a partir de los datos obtenidos.

2.6 Integración de componentes al sistema

Estos componentes serán integrados al sistema utilizando las funcionalidades de Grails, el cual para estos casos implementa el patrón Inversión de Control (IoC) heredado de Spring, según el cual las dependencias de un componente no deben gestionarse desde el propio componente asegurando que este solo contenga la lógica necesaria para hacer su trabajo.

Así cuando se crea un componente en la aplicación, Grails configura Spring para que gestione su ciclo de vida es decir, cuando se crea, cuantas instancias se mantiene vivas a la vez, como se destruyen y sus dependencias, que otros componentes necesita para realizar su trabajo y como conseguirlo. El objetivo de esta técnica es mantener los componentes de la forma más sencilla posible.

2.7 Metas y restricciones arquitectónicas

2.7.1 Requisitos de Hardware

- Estaciones de trabajo o PC Clientes
  - Periféricos: Mouse, Teclado
  - Tarjeta de red
  - 512 MB de Ram o Superior

- Servidor de Aplicaciones
  - Periféricos: Mouse, teclado
  - Tarjeta de Red
  - 2GB de RAM o más
  - 80 GB de espacio en disco o más

- Servidor de Base Datos
  - Periféricos: Mouse, teclado
  - Tarjeta de Red
  - 1GB de RAM o superior
  - 500 GB de espacio en disco o más

2.7.2 Requisitos de Software

- El sistema debe ser multiplataforma, especialmente para Windows XP y Linux

- El sistema debe visualizarse correctamente en Internet Explorer, Mozilla, Opera y Chrome.
- El sistema debe tener como gestor de Base de datos Postgres SQL.

2.7.3 Restricciones de diseño e implementación

- El diseño de la aplicación, se hará aplicando el paradigma de programación orientada a objetos.
- Se utilizara como herramienta de modelación Rational Rose.
- Se utilizarán las tecnologías que brindan los frameworks definidos, así como sus arquitecturas correspondientes.
- El sistema será implementado en el lenguaje Groovy.
- Se debe utilizar como IDE de desarrollo Spring Source Tool Suite (STS).

2.7.4 Requisitos de apariencia o interfaz externa

El sistema debe tener una interfaz intuitiva, organizada y de fácil entendimiento para el usuario que combine correctamente los colores, tipo de letra y tamaño y que los íconos estén en correspondencia con lo que representan.

2.7.5 Requisitos de Seguridad

- El sistema deberá ser capaz de permitir que cada usuario se autentique.
- El sistema deberá garantizar que las funcionalidades se muestren de acuerdo al nivel de usuario que este activo.
- El sistema deberá estar protegido contra acciones que no estén autorizadas o que puedan afectar la integridad de los datos.
- El sistema deberá verificar acciones irreversibles (eliminaciones).
- El sistema deberá verificar que los datos no viajen de forma transparente por la red, los mismos deberán ser encriptados.

2.7.6 Requisitos de rendimiento

El sistema deberá tener un tiempo de respuesta rápida a cualquier solicitud, para ello se utilizará al máximo los recursos del servidor donde esté instalado el sistema.

2.7.7 Requisitos de Usabilidad

El sistema deberá ser utilizado por usuarios que tengan conocimiento acerca del funcionamiento y procesamiento de la información con que se trabaja en el sistema.

2.7.8 Requisitos de soporte

El sistema deberá ser revisado y actualizado cada 6 meses.

2.8 Descripción de la arquitectura

Racional Unified Process metodología seleccionada para el desarrollo del sistema de software define la arquitectura siguiendo 4+1 vistas.

2.8.1 Vista de Caso de Uso

A partir de la vista de Casos de Uso, se puede definir los escenarios o los casos de uso que serán de interés para cada iteración del ciclo de desarrollo. Esta describe los escenarios o casos de uso que tienen significación y que encapsulan la funcionalidad central del sistema.

El sistema cuenta con 14 Casos de Usos, de los cuales 2 son arquitectónicamente significativos y serán desarrollados en el segundo ciclo de desarrollo en conjunto con otros que no someten a restricciones adicionales el diseño de la arquitectura.

Estos Casos de Uso son mostrados a continuación:

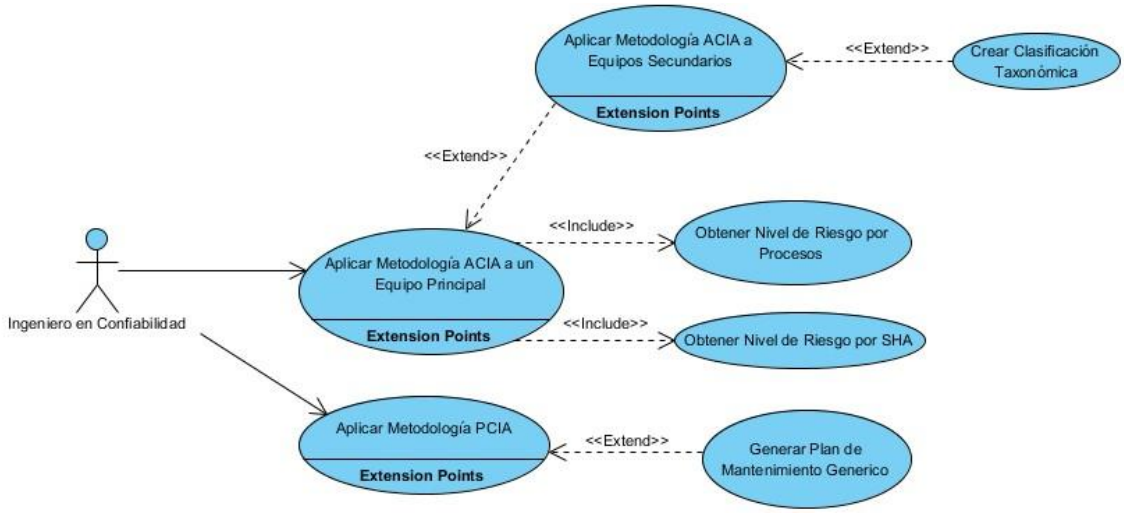

Fig. 13 DCU para el Ingeniero en Confiabilidad

CUS Arquitectónicamente significativos del DCUS Ingeniero en Confiabilidad

- **CUS Aplicar Metodología ACIA a un Equipo Principal:** Este caso de uso le permite al Ingeniero de Confiabilidad aplicar la Metodología de Análisis de Criticidad Integral de Activos (ACIA) a un Equipo Principal de un Grupo de Equipos con el objetivo de establecer una jerarquía o prioridades de sistemas, grupos de equipos y equipos, mediante la cuantificación del impacto global de su comportamiento en el

negocio, denominada riesgo, creando una estructura que facilita la toma de decisiones y el direccionamiento del esfuerzo y los recursos.

- **CUS Aplicar Metodología PCIA:** Este caso de uso le permite al Ingeniero de Confiabilidad aplicar la Metodología de Políticas de Cuidado Integral de Activos (PCIA) como herramienta de control para generar y adaptar planes de mantenimiento genérico de los diferentes Grupos de Equipos y Equipos que forman las unidades de negocio.

2.8.2 Vista Lógica

La vista lógica tiene gran importancia en el documento de descripción de la arquitectura pues en ella se hace referencia a la distribución lógica de la aplicación en partes, así como la dependencia que existen entre las mismas, por otro lado se encuentra la distribución en subsistemas y el análisis de las interfaces que proporcionan para la comunicación entre ellos.

2.8.2.1 Distribución en partes del sistema

La aplicación cuenta con 3 paquetes fundamentales como quedan descritos a continuación:

- **Control:** Este paquete cuenta con un subsistema que agrupan las clases encargadas de la selección de los eventos para cada Caso de Uso, darle interpretación a estos eventos y darle a las otras dos partes que conforman el sistema los mensajes de la acción a realizar, previamente procesada por la clase correspondiente dentro de este paquete. Estas acciones a realizar van desde informar a las interfaces que deben mostrar y en qué momento hasta informarle a las partes del modelo las acciones a realizar.
- **View:** Este paquete cuenta con la agrupación de tres subsistemas que responden a la aplicación del patrón MVP en la capa de presentación. Estos subsistemas van a agrupar las clases que se utilizan para mostrar la información que desea ver el usuario y los recursos que necesitan para la actualización de las vistas. Estos tres subsistemas que se agrupan en este paquete son los encargados de manejar la interacción del usuario con la aplicación.
- **Model:** Este paquete cuenta con un subsistema que es el encargado de representar detalladamente la información con la que el sistema debe operar.

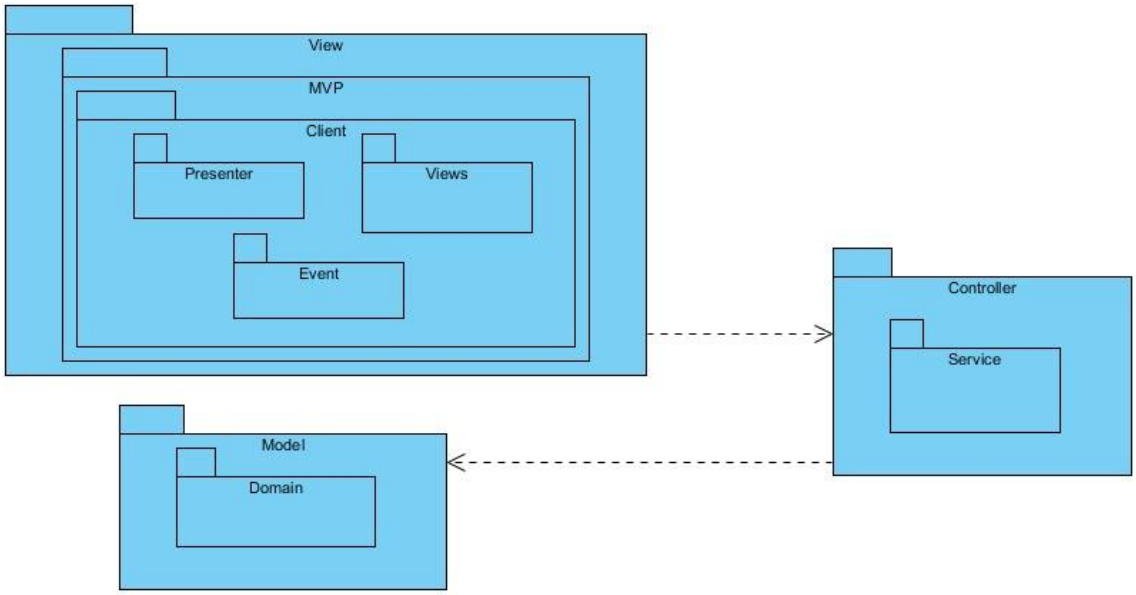

Fig. 14 Diagrama de Paquetes para los CUS Arquitectónicamente Significativos

En el anterior diagrama se puede evidenciar con más claridad los paquetes antes mencionados y las relaciones existentes entre ellos.

2.8.2.2 Subsistemas y Dependencias

Existe una relación lógica entre los diferentes subsistemas que conforman la aplicación la cual es mostrada a continuación:

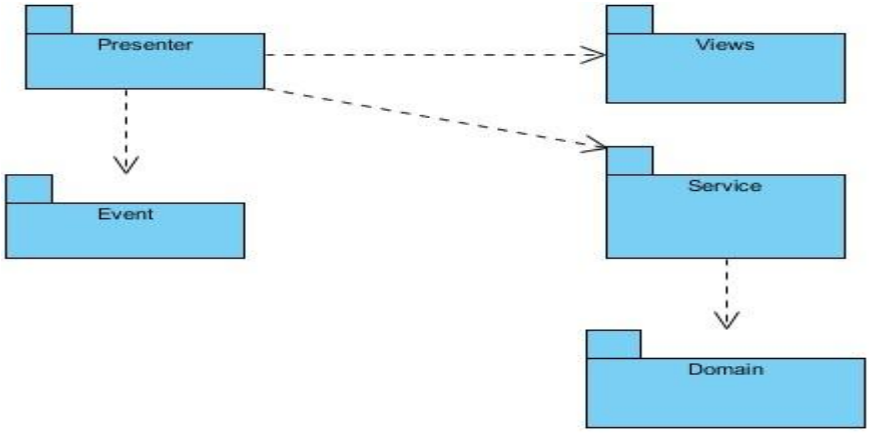

Fig. 15 Relaciones de los subsistemas de diseño

Service: Contiene un grupo de clases que garantizan para cada uno de los Casos de Uso Críticos el tratamiento y selección de los eventos, interpretan las acciones del usuario, accediendo a las operaciones de negocio de la aplicación y modificando a partir de sus resultados el estado del modelo y la navegación entre vistas.

Views: Contiene la agrupación de las clases encargadas de la visualización de los datos así como brindar las interfaces de los formularios con las cuales interactuarán los clientes de la aplicación.

Domain: El subsistema agrupa dentro del, todas las entidades correspondientes a los casos de uso arquitectónicamente significativos con que cuenta la aplicación.

Presenter: Contiene la agrupación de las clases encargadas de actualizar las vistas y gestionar los eventos de las mismas.

Event: Contiene la agrupación de las clases encargadas de registrar los eventos de cada una de las vistas.

2.8.2.3 Diagramas de Clases del Diseño para los CU Arquitectónicamente Significativos

En este subepígrafe se mostrarán los diagramas de clases del diseño para cada uno de los casos de usos arquitectónicamente significativos, definidos anteriormente en la vista de casos de uso, aplicando el patrón de diseño Abstract Factory.

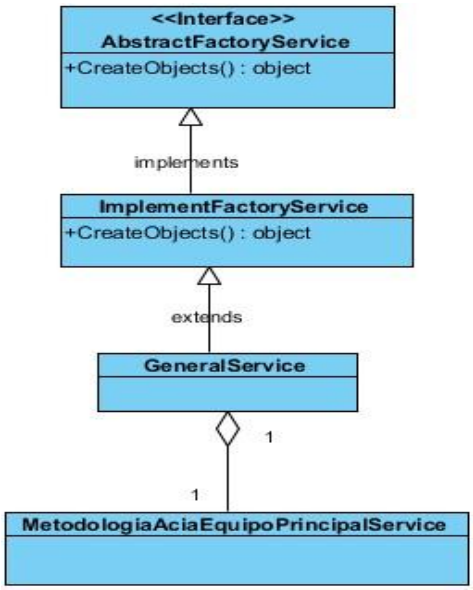

Fig. 16 Diagrama de clases del diseño para el CU Aplicar Metodología ACIA a un Equipo Principal

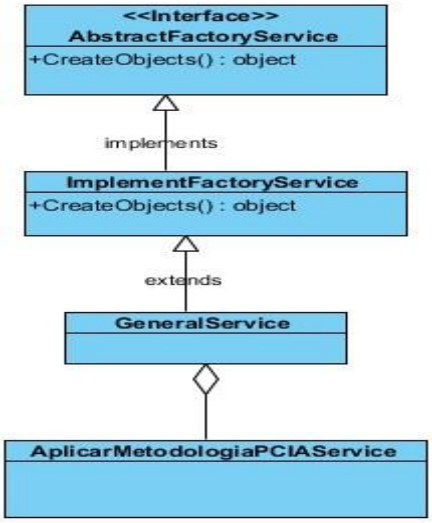

Fig. 17 Diagrama de clases del diseño para el CU Aplicar Metodología PCIA

2.8.3 Vista de Implementación

La vista de implementación describe como a través del resultado del diseño, se implementará el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares.

El flujo de trabajo de implementación describe cómo los elementos del modelo del diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el modelo de despliegue.

Los diagramas de despliegue y componentes conforman lo que se conoce como un modelo de implementación al describir los componentes a construir y su organización y dependencia entre nodos físicos en los que funcionará la aplicación.

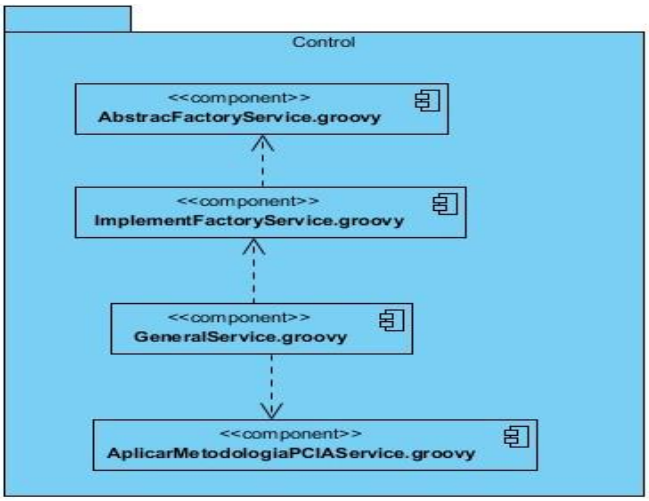

Fig. 18 Diagrama de componentes para el CU Aplicar Metodología PCIA

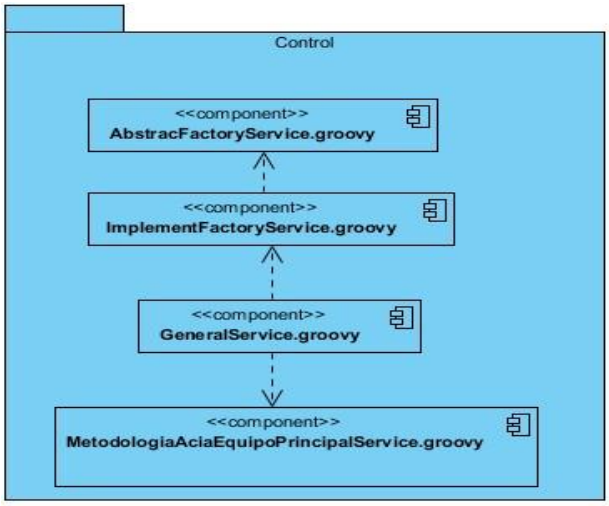

Fig. 19 Diagrama de componentes para el CU Aplicar Metodología ACIA a un Equipo Principal

En los diagramas anteriores se pude evidenciar la distribución de lo componentes del sistema por paquetes y sus relaciones.

2.8.4 Vista de Despliegue

La arquitectura física o Vista de despliegue toma en cuenta primeramente los requisitos no funcionales del sistema tales como la disponibilidad, confiabilidad, rendimiento y escalabilidad. Incluye además una representación de los principales componentes del sistema, la ubicación que tendrán esos componentes dentro de cada nodo y sus relaciones físicas a través del diagrama de despliegue.

2.8.4.1 Diagrama de Despliegue

La aplicación se encontrará distribuida en tres nodos principales. El nodo PC Cliente encargado del funcionamiento del navegador Web donde será visualizada la aplicación, el nodo Servidor de Aplicación con la funcionalidad de soportar sobre él la aplicación Web, en este es donde se montará el sistema y el nodo Servidor de Base Dato que es donde estará montada la base de dato de la aplicación. A continuación se muestra como quedan distribuidos estos nodos y sus relaciones de comunicación.

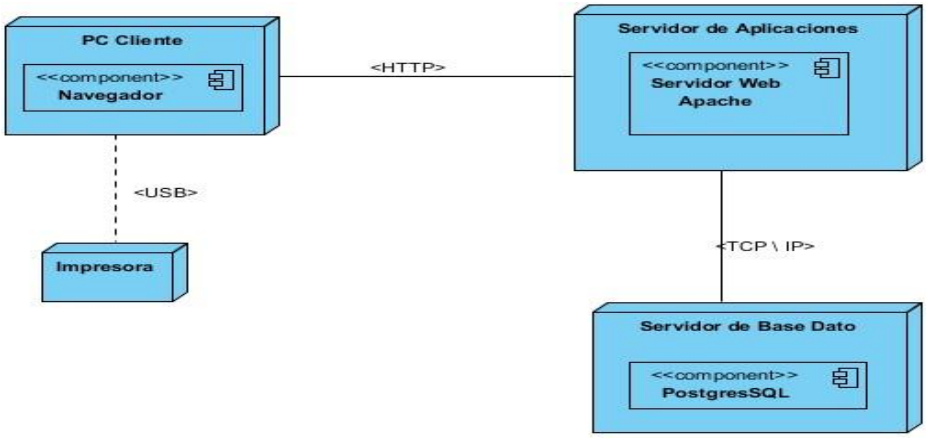

Fig. 20 Diagrama de Despliegue para la Aplicación

En la anterior figura se pude evidenciar más claramente la distribución de los nodos del sistema y sus dependencias.

Conclusiones Parciales

En el capítulo se abordaron todo los temas referentes a la alternativa de solución propuesta, definiéndose el diseño arquitectónico a utilizar en el sistema.

Este diseño permitió dividir el sistema en capas independientes, propiciando el uso de diferentes tecnologías en cada una de ellas.

También se definió el proceso de integración necesario para cumplir los requisitos del sistema.

Capítulo 3 Evaluación de la arquitectura e integración propuesta

El objetivo de este capítulo es evaluar la arquitectura e integración propuesta, para lo cual se realiza un estudio de los métodos y técnicas para evaluar arquitectura de software visto en el capítulo 1, eligiéndose finalmente un método que permita evaluar no solo la arquitectura, sino también la integración de componentes y tecnologías en el sistema. Como resultado final se evalúa la arquitectura e integración propuesta.

3.1 ¿Por qué evaluar la arquitectura de software?

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 requerimientos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad. Cabe señalar que los requerimientos no funcionales también son llamados atributos de calidad.

Realizar evaluaciones a las arquitecturas de software es la manera más económica de evitar desastres. La evaluación de una arquitectura no produce resultados cuantitativos, sino ayuda a encontrar debilidades. El tema de evaluar una arquitectura, estimar el comportamiento de los atributos ante cambios arquitectónicos, lograr hacer predicciones de diseño en etapas temprana, siempre han sido actividades de vital interés para la industria del software, debido a que todas estas predicciones aseguran la calidad del software, la disminución de los tiempos de desarrollo, y por lo tanto el esfuerzo y los costos de cada producto.

Método y técnica seleccionada

3.2 Método de evaluación seleccionado

Como resultado del análisis y comparación de los diferentes métodos de evaluación vistos en el capítulo 1, se observó que el Architecture Tradeoffs Analysis Method (ATAM) evalúa la arquitectura con mayor profundidad, tiene un conjunto de pasos, un equipo de evaluación, un conjunto de salidas mejor definidas y no presenta ninguna restricción con respecto a la características de calidad a evaluar. Unido a esto el método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados, los cuales representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas. Además se propone un

árbol de utilidad inicial basado en el modelo de calidad ISO-9126 instanciado para Arquitectura de Software propuesto por Losavio. La adopción de este modelo permite concentrarse en características que dependen exclusivamente de la arquitectura, y al ser un estándar facilita la correspondencia con características de calidad consideradas por otros métodos (Aleksander González, 2005).

3.3 Instrumentos y técnicas de evaluación

Para la especificación de la calidad se hace uso de un árbol de utilidad, el cual tiene como nodo raíz la "bondad" o "utilidad" del sistema. En el segundo nivel del árbol se encuentran los atributos de calidad. Las hojas del árbol de utilidad son escenarios, los cuales representan mecanismos mediante los cuales extensas (y ambiguas) declaraciones de cualidades son hechas específicas y posibles de evaluar. La generación del árbol de calidad incluye un paso que permite establecer prioridades. Por último para el análisis de la arquitectura se utiliza la técnica de escenarios.

3.4 Descripción de las fases del método

1. **Presentación**: Se da a conocer el método, el sistema y su organización, y finalmente se presenta cuál es la arquitectura que debe ser evaluada.
2. **Investigación y Análisis:** Se determina de qué manera se va estudiar la arquitectura, se decide que escenarios de calidad se desean, los elementos de diseño a evaluar y como estos responden a los escenarios de calidad seleccionados.
3. **Prueba**: Como resultado de la aplicación de las fases 1 y 2 se tienen un conjunto de decisiones arquitectónicas documentadas que fueron realizadas considerando el árbol de utilidad generado. En esta fase se examinan y comprueban las decisiones producidas hasta el momento.
4. **Reporte**: Se presentan los resultados.

3.5 Evaluación de la arquitectura

Una vez conocido y descrito el método de evaluación a utilizar se procederá a evaluar la arquitectura propuesta de acuerdo a las 4 fases del desarrollo del método ATAM antes descritas. Para llevar a cabo esta evaluación con mayor exactitud se cuenta con uno de los módulos del sistema conocido como Análisis de Criticidad Integral de Activos ACIA en el que ya se terminó todo el proceso de integración.

3.5.1 Fase de presentación

Esta fase ya ha sido desarrollada mediante la presentación del método ATAM en el capítulo 1, su explicación en este, y la descripción de la arquitectura en el capítulo 2.

3.5.2 Fase de investigación y análisis

Según lo explicado anteriormente el estudio de la arquitectura se realizará a partir de los atributos de calidad expuestos en el árbol de utilidad basado en el modelo de calidad ISO-9126, instanciado para Arquitectura de Software propuesto por Losavio.
Entonces una vez conocido el modo de estudio de la arquitectura a evaluar, esta fase se puede dividir 3 partes principales para su desarrollo:

- Identificación de los elementos de diseño
- Generación del árbol de utilidad
- Análisis de los elementos de diseño

3.5.2.1 Identificación de los elementos de diseño

En este paso se contemplan alternativas de diseño de la arquitectura, que posteriormente serán analizadas para ver cómo responden a los requerimientos de calidad citados. Durante esta fase la arquitectura debe ser evaluada completamente puesto que ATAM centraliza el análisis de una arquitectura en el entendimiento de sus propuestas arquitectónicas. Para lo cual se toman en cuenta todos los elementos que conforman la arquitectura del sistema, desde los componentes de mayor granularidad, hasta el mínimo nivel de granularidad que corresponde a los componentes, los estilos arquitectónicos y patrones definidos y los casos de usos arquitectónicamente significativos. Todos estos elementos fueron descritos en el capítulo 2 y listados a continuación.

- Estilo arquitectónico En Capas
- Patrón arquitectónico Modelo Vista Controlador (MVC)
- Patrón arquitectónico Modelo Vista Presentador (MVP)
- Componentes del sistema (Simulación de Montecarlo, Auditoria, Seguridad y Reporte)

3.5.2.2 Generación del árbol de utilidad

Para la evaluación de los elementos de diseño definidos en la arquitectura se hace uso de el árbol de utilidad basado en el modelo de calidad ISO-9126, instanciado para Arquitectura de Software propuesto por Losavio, el cual propone un conjunto de escenarios que ponen a prueba las propuestas arquitectónicas, para ver si estas

alcanzan los requerimientos de los atributos de calidad especificados. Cada uno de estos escenarios es ubicado en una hoja del árbol de utilidad, según el atributo de calidad que se está poniendo a prueba.

A continuación se muestra el árbol de utilidad utilizado:

No	Característica	Sub-Característica	Escenario
1	Funcionalidad	Precisión	La integración y comunicación entre los componentes no altera la exactitud de los datos.
2		Seguridad	El sistema detecta la actuación de un intruso e impide acceso a los componentes que manejen información sensible.
3			El sistema asegura que los componentes no pierdan datos ante un ataque (Interno o Externo).
4		Obediencia	El sistema respeta un estándar de fiabilidad.
5	Fiabilidad	Madurez	El sistema es capaz de manejar entradas de datos incorrectas.
6	Eficiencia	Tiempo de Comportamiento	El sistema debe recibir los servicios de sus componentes en el transcurso de un tiempo indicado.
7		Recursos utilizados	Los componentes pueden compartir recursos adecuadamente.
8			El sistema controla que ningún componente se quede sin recursos cuando los necesita.
9	Portabilidad	Adaptabilidad	El sistema debe continuar funcionando correctamente aun cuando los servicios de los componentes dotados por el ambiente varíen.
10		Capacidad de Instalación	El sistema puede instalarse fácilmente en todos los ambientes donde debe funcionar.
11	Usabilidad	Accesibilidad	Se debe poder acceder al sistema en cualquier momento.

Tabla. 4 Subconjunto del árbol de utilidad basado en el modelo de calidad ISO-9126

3.5.2.3 Análisis de los elementos de diseño

Una vez ubicados los escenarios de calidad se procede a establecer un orden de prioridad entre los mismos atendiendo a los requisitos funcionales y no funcionales del sistema, este orden está definido como un par (X; Y) donde (X) define la importancia de satisfacer el escenario y (Y) el riesgo que se corre al excluirlo del árbol de utilidad. Los posibles valores para X e Y son A (Alto), B (Bajo) y M (Medio).

Estos escenarios de calidad deben ser aplicados a la arquitectura definida y para ello el método propone un conjunto de preguntas de análisis que permiten determinar decisiones sobre la arquitectura. Se debe preguntar si las decisiones tomadas permiten alcanzar los escenarios de calidad planteados y en dependencia del grado en que se manifiesta el cumplimiento de estos escenarios en la arquitectura y el sistema se evaluaran como A (Alto), B (Bajo) y M (Medio).

Este árbol de utilidad generado unido a la evaluación y orden de prioridad de sus escenarios se toma como un plan para el resto de la evaluación que realiza el método. Indica además donde ocupar más tiempo en la evaluación y donde probar aproximaciones y riesgos arquitectónicos.

A continuación se muestran las preguntas de análisis propuestas por el método para determinar decisiones sobre la arquitectura y seguido de esto el orden de prioridad establecido a los escenarios y la evaluación del grado de cumplimiento de la arquitectura con los mismos, según el orden en que parecen estos escenarios en el árbol de utilidad de la tabla 4.

Característica	Sub-Característica	Preguntas de Análisis
Funcionalidad	Precisión	¿Puede la comunicación entre los componentes introducir imprecisiones en los servicios ofrecidos por los componentes?
Fiabilidad	Madurez	¿Existen decisiones para minimizar el manejo incorrecto de datos en la comunicación entre componentes?
Eficiencia	Tiempo de Comportamiento	¿Cómo es la relación entre el número de componentes de las diferentes partes de la arquitectura?
Portabilidad	Capacidad de Instalación	¿Existe un mecanismo para determinar si todos los componentes del sistema se encuentran correctamente instalados?

Tabla. 5 Algunas preguntas para analizar los elementos de diseño identificados *(Erika Camacho, 2004)*

Escenario	Prioridad	Cumplimiento en el Sistema		
		A (Alto)	M (Medio)	B (Bajo)
E1	(A;A)	*X*		
E2	(A;A)	*X*		
E3	(A;A)	X		
E4	(A;A)	*X*		
E5	(M;M)	*X*		
E6	(A;M)		X	
E7	(A;A)	*X*		
E8	(A;A)	*X*		
E9	(M;M)		*X*	
E10	(A;B)	*X*		
E11	(A;B)	*X*		

Tabla. 6 Prioridad y cumplimiento de los Escenarios

En el próximo apéndice se explica el cumplimento de cada uno de los escenarios por el sistema.

3.5.2.4 Análisis del cumplimiento de los escenarios

Para el análisis de los escenarios sobre como el sistema es capaz de manejar los datos su utilizó el framework JUnit creado por Erich Gamma y Kent Beck el cual viene integrado al IDE de desarrollo utilizado, Spring Source Tool Suite. Este framework permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento e integración de cada uno de los métodos de las clases se comporta como se espera (Spring). Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en

caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

JUnit es también un medio de controlar las pruebas de regresión, necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación.

En el caso de los escenarios que impliquen una evaluación sobre como el sistema responde bajo condiciones anormales se utilizó la herramienta JMeter la cual permite realizar pruebas de rendimiento y funcionales sobre aplicaciones web. Es una herramienta de carga para llevar a cabo simulaciones sobre cualquier recurso software, lo que permite poner en práctica sobre la arquitectura la técnica cuantitativa de evaluación basada en simulaciones.

Funcionalidad

- ***Precisión***: La integración y comunicación entre los componentes se realizó haciendo uso de las tecnologías de desarrollo y el estilo arquitectónico propuesto, para verificar que todo este proceso no altera la exactitud de los datos manejados por el sistema se hace uso del framework **JUnit** mediante el cual se evalúa el manejo de los datos por cada una de las funcionalidades de un módulo del sistema, verificando así que el resultado sea el esperado. A continuación se muestra el resultado de la prueba:

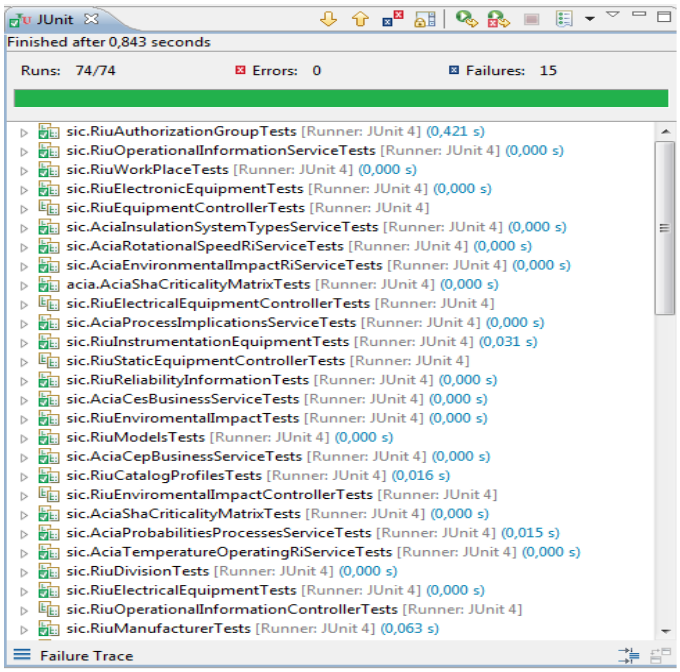

Fig. 21 Test de evaluación con JUnit a un módulo del sistema

Como se puede apreciar después de aplicado el Test de Junit al sistema, este respondió satisfactoriamente sin ningún tipo de error que comprometa la exactitud o integridad de los datos.

- ***Seguridad***: El sistema cuenta con un componente de seguridad, el cual se encarga de gestionar y proteger los perfiles de usuarios. De esta manera cada usuario del sistema solo tiene acceso a la información que le es permitida manejar de acuerdo a sus derechos**;** evitando así el acceso de cualquier intruso al sistema y a la vez protegiendo toda la información del mismo.

  Unido a esto el sistema implementa el patrón arquitectónico MVP en la capa de presentación el cual separa el modelo del dominio y la presentación, como alternativa de seguridad ante ataques internos o externos. También el framework de desarrollo Grails proporciona seguridad al sistema ante ataques de inyección SQL mediante escape de caracteres y ante ataques de denegación de servicios *(Brito Calahorro, 2009)*.

- ***Obediencia***: El componente de seguridad del sistema es capaz de identificar cada usuario del sistema con sus privilegios asociados, restringiendo la accesibilidad del los mismos a los datos y acciones que estén definidas solamente para ellos. Esta acción se realiza obligando a todos los usuarios a identificarse en el sistema, antes de acceder a cualquier funcionalidad del mismo.

Fiabilidad

- ***Madurez*:** El sistema no permite la entrada de datos incorrectos. Mostrando al usuario a través de recursos visuales, cual fue el error de entrada y no procesará la información hasta que esta sea entrada correctamente. Para verificar con más exactitud el comportamiento del sistema ante estas entradas incorrectas se utilizó nuevamente el framework JUnit, pero esta vez sobre la capa de presentación del sistema, que es la encargada del manejo de estos datos arrojando los siguientes resultados:

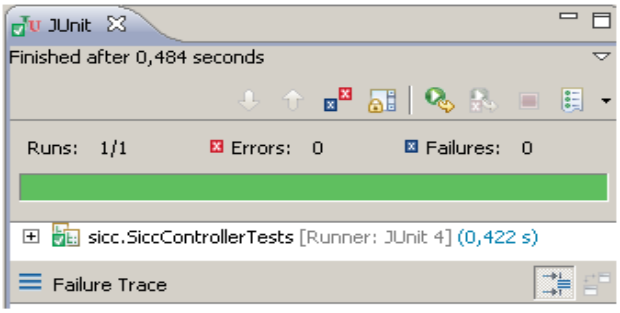

Fig. 22 Test de evaluación con JUnit a la capa de presentación de un módulo del sistema

Como se puede apreciar en el resultado del Test a la capa de presentación del sistema, esta respondió satisfactoriamente al mismo, verificándose el correcto funcionamiento de los métodos encargados de manejar los datos introducidos por el usuario.

Eficiencia

- ***Tiempo de Comportamiento:*** Para evaluar el tiempo de comportamiento de la aplicación se utilizó la herramienta JMeter, con la cual se realizo una prueba de stress y rendimiento a un módulo de aplicación, a través de la simulación de un grupo de conexiones al sistema.

  A continuación se muestran los resultados obtenidos para un total de 5 pruebas:

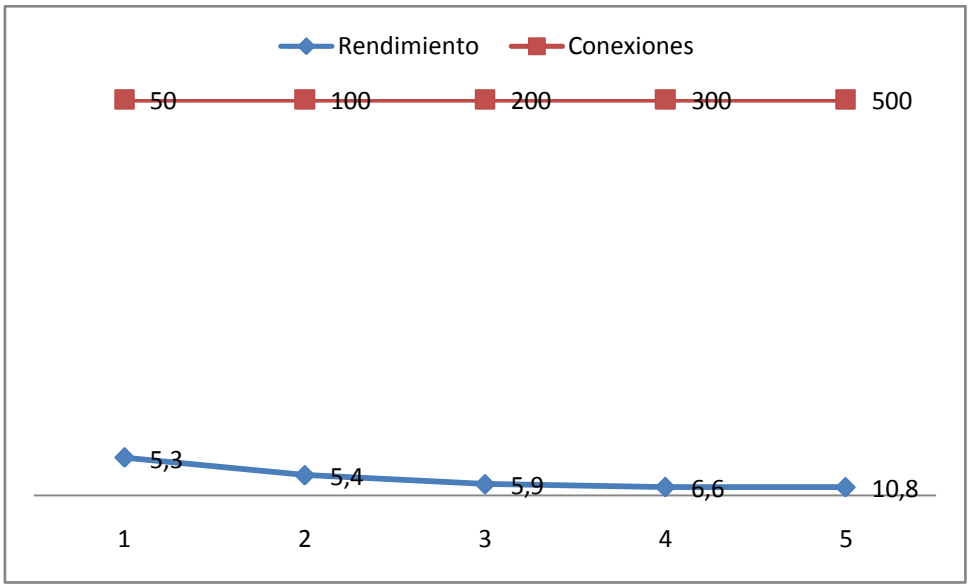

Fig. 23 Resultado del Test de rendimiento a un módulo del sistema con la herramienta JMeter

Según se puede observar en la grafica de 1 a 100 conexiones el rendimiento del sistema es bastante estable. Considerando que al sistema solo deben acceder los ingenieros en confiabilidad de la empresa donde este operativa la aplicación, el resultado de la prueba de rendimiento es bastante aceptable.

- ***Recursos Utilizados:*** Los componentes del sistema comparten recursos adecuadamente gracias a la integración de los mismos en el sistema, ejemplo de estos recursos compartidos son la Base de Datos y los contenedores de Grails. Unido a esto el sistema se encarga de verificar que ningún componente se quede sin recursos cuando lo necesita, gracias a las validaciones realizadas

en la lógica del negocio y en caso de no tenerse los recursos necesarios para el correcto funcionamiento del componente, el sistema lanzara un aviso informando de la situación. Prueba de todo esto es el Test de JUnit hecho anteriormente al sistema completo, que no solo sirvió para verificar la exactitud de los datos, sino también la correcta integración de los componentes y el funcionamiento de los mismos ante situaciones inesperadas.

Portabilidad

- ***Adaptabilidad:*** El sistema al ser multiplataforma puede funcionar correctamente en cualquier estación cliente con la ayuda de un navegador, lo cual lo hace independiente de cualquier cambio del sistema operativo en la estación. En el caso del servidor del sistema si se puede ver afectado por algún cambio en el ambiente principalmente los cambios que tienen que ver con la memoria RAM del servidor, debido al alto consumo de recursos del framework de desarrollo Grails y la Máquina Virtual de Java.
- ***Capacidad de Instalación:*** Al ser una aplicación web el sistema puede instalarse fácilmente en todos los ambientes donde debe funcionar, siempre y cuando el servidor de la aplicación cuente con los requisitos descritos en el capítulo 2 apéndice 2.7.1.

Usabilidad

- **Accesibilidad:** El sistema al ser una aplicación web hace posible acceder a él en cualquier momento por las personas autorizadas, siempre que el servidor este funcionando correctamente.

3.5.3 Fase de Prueba

El objetivo de esta fase es analizar los resultados arrojados por la aplicación de las fases 1 y 2, y comprobar que las decisiones tomadas fueron las más idóneas, para ello se analiza nuevamente el árbol de utilidad generado, verificándose si los escenarios han recibido la importancia adecuada y si es necesaria la inclusión de nuevos escenarios.

Esta fase se llevó a cabo con la ayuda de un grupo de integrantes del proyecto, comprobándose los aspectos antes dichos; llegándose a la conclusión de que las decisiones tomadas y los escenarios evaluados son los más adecuados para el proyecto. Unido a esto no hizo falta añadir más escenarios al árbol de utilidad puesto que el mismo es el propuesto por el modelo de calidad ISO-9126 para evaluar arquitecturas de software.

3.5.4 Fase de Reporte

Llegada esta fase ya se cuenta con un análisis de todas las decisiones y resultados obtenidos hasta el momento, los cuales son presentados como un resumen de la aplicación del método.

Atributo	A (Alto)	M (Medio)	B (Bajo)	Descripción
Funcionalidad	E1,E2 E3,E4			Según los escenarios expuestos en este atributo, los elementos que intervienen en él se ajustan adecuadamente a la arquitectura y el sistema.
Fiabilidad	E5			El escenario expuesto en este atributo, también se ajusta adecuadamente a la arquitectura y el sistema.
Eficiencia	E7,E8	E6		Para lograr que el sistema ante este escenario tenga un mejor rendimiento en caso de aumentar las estaciones clientes, sería necesario mejorar la capacidad de procesamiento del servidor utilizado.
Portabilidad	E10	E9		Para asegurar un completo cumplimiento del escenario E9 es necesario que el ambiente donde se desee instalar el sistema cumpla los requisitos expuestos en el capítulo 2 apéndice 2.7.1.
Usabilidad	E11			El escenario expuesto en este atributo, también se ajusta adecuadamente a la arquitectura y el sistema.

Tabla. 7 Resumen y Análisis de los resultados

Conclusiones Parciales

En este capítulo se realizó la evaluación de la arquitectura diseñada e integración propuesta.

La arquitectura definida fue evaluada utilizando un conjunto de atributos de calidad mediante el método ATAM, donde se obtuvieron resultados favorables sobre cada uno de los escenarios propuestos por cada atributo.

En cuanto a la integración propuesta se realizaron pruebas unitarias y test de rendimiento sobre uno de los módulos del sistema ya terminado, comprobándose la eficiencia de la misma.

Conclusiones

En el presente trabajo se definió la arquitectura a utilizar en el proyecto SIC. Este proceso de diseño arquitectónico permitió el uso de diversas tecnologías en el sistema que admiten su desarrollo sobre un ambiente ágil y robusto.

También se especificó y argumentó los pasos necesarios para integrar las tecnologías seleccionadas, cumpliendo con las exigencias de las mismas y las funcionalidades y requerimientos de desempeño del sistema. Todo esto permitió aprovechar y utilizar las ventajas de cada una de estas tecnologías.

Finalmente se realizó la evaluación de la arquitectura definida y el proceso de integración propuesto. Esta evaluación fue realizada mediante el método ATAM, comprobándose el desempeño del sistema sobre un grupo de escenarios de calidad previamente definidos que avalan el cumplimiento del objetivo general propuesto en esta investigación.

Recomendaciones

- Reevaluar los escenarios de calidad de la aplicación durante y luego de la fase de desarrollo de los demás módulos, para evaluar a fondo las restricciones que se le imponen a la arquitectura.
- Tener en cuenta la propuesta de arquitectura e integración desarrollada en esta investigación, para futuros desarrollos dentro del área temática.

Referencias Bibliográficas

Agüero, Martín. 2007. *Introducción a Spring Framework.* Buenos Aires, Argentina : s.n., 2007.

Aleksander González, Marizé Mijares, Luis E. Mendoza, Anna Grimán, María Pérez. 2005. *Método de Evaluación de Arquitecturas de Software Basadas en Componentes (MECABIC).* Mexico : s.n., 2005.

Arata, Adolfo. 2009. *Ingeniería y Gestión de la confiabilidad operacional en las plantas industriales.* Chile : Ril Editores, 2009.

Arquitectura y Diseño de Sistemas Web Modernos. **Castejón Garrido, Juan Salvador. 2004.** Murcia : InforMAS , 2004. ISSN: 1698-8841 .

Brito Calahorro, Nacho. 2009. *Manual de Desarrollo Web con Grails.* España : s.n., 2009. ISBN: 978-84-613-2651.

Comunity, Grails. 2010. observatoriodegrails. [En línea] 2010. [Citado el: 25 de Enero de 2012.] http://observatoriodegrails.com/.

Craic, Larman. 1999. *UML y Patrones.* United States : Prentice-Hall, 1999. ISBN 0-13-748880-7.

De La Torre Llorente, Cesar, y otros. 2010. *Guia de Arquitectura N-Capas orientada al Dominio com.NET 4.0.* España : Krasis Consulting, 2010.

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

Fowler, Martin. 2002. *Patterns of Enterprise Application Architecture.* Massachusetts : Addison Wesley, 2002. 0-321-12742-0.

Gómez, Omar Salvador. 2007. *Evaluando Arquitecturas de Software.* Mexico : Brainworx S.A, 2007. 1870-0888.

González Romano, Mariano y Cordero Valle, Manuel. 2001. *Diseño de Paginas Web.* Madrid : McGraw-Hill, 2001. ISBN:84-48 1-3135--5.

González, Rolando, Hernández Leon, Alfredo y Coello, Sayda. 2011. *El proceso de investigación científica.* La Habana : Editorial Universitaria, 2011.

Google. 2011. Google Code. [En línea] Google, 2011. [Citado el: 15 de marzo de 2012.] http://code.google.com.es.mk.gd/webtoolkit/.

Grails. 2009. Grails. [En línea] 2009. [Citado el: 20 de Enero de 2012.] http://grails.org/.

Gustavo Andrés Brey, Gastón Escobar, Nicolas Passerini y Juan Arias. 2005. *Arquitectura de Proyectos de IT. Evaluación de Arquitecturas.* Buenos Aires : s.n., 2005.

Integración de Tecnologías en una Plataforma JEE Dirigida por Modelos. **Colque, David y Valdivia, Ricardo. 2005.** 3, Chile : s.n., 2005, Vol. 14.

Jacobson, I., Booch, G., Rumbaugh, J. 1999. *El Proceso Unificado de Desarrollo del Sofware.* United States : Addison Wesley, 1999.

Jacobson, Ivan, Booch, Grady y Rumbaugh, James. 2000. *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Educacion, S.A., 2000. ISBN 84-7829-036-2.

Krutchen, Philippe. 2002. *Rational Unified Process . Segunda Edición.* United States : Addison Wesley, 2002.

Marquina, Ernesto. 2008. *Guía de Patrones, Prácticas y Arquitectura .NET.* United States : Microsoft Corporation, 2008.

Meier, J.D., y otros. 2008. *Application Architecture Guide 2.0.* United States : Microsoft Corporation, 2008.

Microsoft. 2006. La Arquitectura Orientada a Servicios (SOA) de Microsoft. [En línea] Microsoft Corporation , 2006. [Citado el: 20 de Abril de 2012.] http://www.microsoft.com/biztalk/en/us/soa.aspx.

Pressman, Roger. 2010. *Software Engineering, A Practitioner´s Approach.* New York : McGraw-Hill, 2010.
Rocher, Graeme. *The definitive guide to Grails.* s.l. : Apress. ISBN 1-59059-758-3.
Spring. http://www.springsource.com/. *SpringSource.* [En línea] [Citado el: 5 de mayo de 2012.] http://static.springsource.org/spring/docs/2.5.x/reference/testing.html.
Szypersky, Clemens. 2002. *Component Software Beyond Object Oriented Programming.* United States : Addison Wesley, 2002.
Universidad de Chile. 2008. *Como Funciona la Web.* Chile : s.n., 2008. ISBN: 978-9563192251.
Vargas Henández, David y Vargas Vento, Daisy Diana. 2012. *Propuesta para la Disminución del Tiempo de Desarrollo en Aplicaciones J2EE Utilizando GROOVY & GRAILS.* La Habana : s.n., 2012.
White, Colin. 2005. *Data Integration: Using ETL, EAI, and EII Tools to Create an Integrated Enterprise.* United States : s.n., 2005.
Zayas, Dr. Cs. Carlos Alvarez de. 1995. *Metodología de la Investigación Cientifica.* Santiago de Cuba : s.n., 1995.