Universidad de las Ciencias Informáticas

Facultad 6

Título: Diseño y evaluación de la Arquitectura de Software con la tecnología J2EE y el Framework Spring.

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

Autor(es): Beatriz Hernández Cervante
Dairon Freddy Calderio Hechevarria

Tutor(es): Lic. Ivan Alfonso Olamendy
Ing. Ana Lupe Delgado Montero
Ing. José Antonio Castaño Guevara

Cotutor(es): Lic. Liliannes Caridad Matamoros Benítez

Consultante(s): Ing. Reynaldo Álvarez Luna
Ing. Andrés Ballester Marsal

Junio, 2009

"Programar sin una arquitectura en mente es como explorar una gruta sólo con una linterna: no sabes dónde estás, dónde has estado, ni hacia dónde vas."

Danny Thorpe

Declaración de Autoría

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

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

Beatriz Hernández Cervantes

_____
Firma del Autor

Dairon Freddy Calderio Hechevarria

_____
Firma del Autor

Lic. Ivan Alfonso Olamendy

_____
Firma del Tutor

Ing. Ana Lupe Delgado Montero

_____
Firma del Tutor

Ing. José Antonio Castaño Guevara

_____
Firma del Tutor

DATOS DE CONTACTOS:

Tutores:
Lic. Ivan Alfonso Olamendy
Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba
e-mail: ivan@uci.cu

Ing. Ana Lupe Delgado Montero
Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba
e-mail: aldelgado@uci.cu

Ing. José Antonio Castaño Guevara
Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba
e-mail: joseantonio@uci.cu

Cotutor:
Lic. Liliannes Caridad Matamoros Benítez
Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba
e-mail: lmatamorosb@uci.cu

Consultantes:
Ing. Reynaldo Álvarez Luna
Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba
e-mail: rluna@uci.cu

Ing. Andrés Ballester Marsal
Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba
e-mail: aballester@uci.cu

Agradecimientos

En este papel es imposible plasmar todo los nombres que están ligados indisolublemente a nuestras vidas y no queremos ser injustos dejando de incluir a alguien por cuestiones de tiempo y espacio, por lo tanto agradecemos a todos los que nos han ayudado a pasar estos años. No obstante, hay nombres que no pueden dejar ser escritos porque son muy especiales y es el caso de:

Esta Revolución, porque de no ser por ella hoy no podríamos estar aquí.

Lupita, Ivancito, Lili, Rey y Castaño, que siempre estuvieron ahí para nosotros, gracias por todo.

Yolanda y Yanelis, que me acogieron en su familia como si hubiera nacido en ella, muchísimas gracias.

Todos los profesores que tanto empeño pusieron en formarnos y día a día por 5 largos años dieron lo mejor de ellos para que llegáramos a este feliz final.

Dedicatoria

A Fidel Castro Ruz, mi padre aunque no lleve su sangre, es mi ejemplo personal y el motivo de que viniera a estudiar en esta maravillosa universidad.

A Carmelina, mi abuelita del alma, y aunque no pueda estar a mi lado nunca más, siempre estará en mi corazón.

A mi mamá, que día a día, con tanta paciencia y amor, me fue formando hasta lo que hoy soy, eres la mejor mamita del mundo.

A mi hermanita, que a pesar de que hoy tenga que estar muy lejos cumpliendo una misión humanitaria, ha estado conmigo todo el tiempo, porque la llevo en el corazón.

A mi sobrinita, la niña más bella del mundo.

Al amor de mi vida, que me ha dado todo el amor del mundo que se puede dar, que sufrió y rió conmigo cada pena y alegría que viví todos estos años, te amo.

A mi tía Dannis, mis primas Maikecita y Yoly, y mi cuñado Iribin, gracias por todo el apoyo y el cariño que siempre me han dado.

A mis tías Chuta y Esperanza, mis primos Yuselí y César, y los tíos Dean y Yeyo, todos han sido tan buenos y comprensivos conmigo.

A todos mis amigos, especialmente a Massiel, Daryanis, Yumis y a César, que siempre confió en mi capacidad para enfrentarme a grandes proyectos.

Betty

A mi mamá por todo su cariño, esfuerzo, comprensión, por apoyarme y guiarme siempre por el buen camino, por ayudarme a ser todo lo que soy, por el amor y la educación que siempre me ha dado.

A mi papá por el amor, la sabiduría y su empeño en el desarrollo de este trabajo, sin el no se hubiera podido realizar. Por tenerlo ahí cuando lo he necesitado, por todo lo que me ha enseñado y educado.

A mi hermanita querida, que aunque la vida no le permitió estar aquí hoy, sé que estaría orgullosa de mí. Por ser mi fiel amiga, guiarme por el buen camino, ayudarme cuando lo necesité y estar ahí para mí en los buenos y malos mementos.

A mi sobrinita, que aunque es muy chiquitica me ha inspirado mucho a seguir adelante.

A mi novia por todo el amor y el apoyo que me ha brindado durante esta etapa final.

A mis abuelas y a mi abuelo, mis tías, mis primas y primos que siempre confiaron en mí. En general a toda mi familia.

A mis amigos Rachid, Raúl, Dioleysis, César que han compartido junto a mí todo el transcurso de estos 5 años de la carrera, dándome el apoyo y la confianza de poder llevar a cabo todas mis actividades docentes y productivas.

A mi compañera de tesis Betty que se esforzó muchísimo en el desarrollo de la tesis.

Dairon

Resumen

El presente Trabajo de Diploma describe una arquitectura de software basada en Java Enterprise Edition (J2EE)[1] como tecnología y el Framework Spring que podría ser utilizada como referencia en el desarrollo de aplicaciones Web que empleen estas tecnologías, por tal motivo uno de los objetivos fundamentales es la evaluación de la arquitectura de software por uno de los métodos propuestos en la literatura, el Architecture Trade-off Análisis Method, y reafirmarla con la implementación de la aplicación Web para la captura de la información requerida por el Centro Nacional de Balance Alimentario de la República Bolivariana de Venezuela, ejemplo de su aplicación exitosa. Para alcanzar los objetivos planteados se ha centrado la investigación en el estudio de los conceptos relacionados con la arquitectura de software considerando las características de la tecnología Java Enterprise Edition y el Framework Spring para el desarrollo de aplicaciones Web, con el fin de lograr una arquitectura que garantice buenas prácticas de programación, soluciones robustas que carezcan de complejidad innecesaria y con mejoras en propiedades como la escalabilidad y la reusabilidad.

Palabras Clave:

Arquitectura de software (AS), J2EE, Spring, aplicación Web, evaluación.

[1] Define el estándar para el desarrollo de aplicaciones empresariales multicapas agregando pleno apoyo a los componentes Enterprise JavaBeans, Servlets de Java API, Java Server Pages y tecnología XML, propiciando que se puedan desarrollar aplicaciones Web bajo la tecnología Java.[27]

Abstract

The present Work of Diploma describes a software architecture based on Java Enterprise Edition (J2EE) as technology and the Framework Spring that might be used as reference in the development of Web applications used by these technologies, that´s why one of the fundamental aims is the evaluation of the software architecture for one of the proposed methods in the literature, the Architecture Trade-off Analysis Method (ATAM), and to reaffirm with the implementation of the Web application to get the information needed by the National Center of Food Balance of the Bolivarian Republic of Venezuela example of its successful application. To reach the raised aims, the investigation has been focused on the study of the concepts related to the software architecture considering the characteristics of the Java Enterprise Edition technology and the Framework Spring for the development of Web applications, in order to achieve an architecture that guarantees good programming practices, robust solutions that lack unnecessary complexity and with improvements in properties as the scalability and the reusability.

Keywords:

Software architecture (SA), J2EE, Spring, Web application, evaluation.

Índice de contenido

Índice de figuras

Índice de tablas

Introducción

En la nueva era de "la sociedad del conocimiento", la información y las comunicaciones, son factores extremadamente claves en los procesos de producción y obtención de mejoras. Las Tecnologías de la Informática y las Comunicaciones (TIC) han demostrado ser instrumentos que pueden contribuir al logro de amplios objetivos nacionales, tanto sociales como económicos, en la medida en que los estados las incorporen a las principales políticas y programas de desarrollo de sus naciones.

Pero junto con la necesidad de llevar la informatización a cada esfera de la sociedad y la economía, las compañías de todo el mundo reconocen que la calidad del producto se traduce en ahorro de recursos, en un servicio superior y por tanto, en mejores resultados, por lo que en los últimos años se han realizado intensos esfuerzos por lograr que la calidad del software sea una premisa en todas las fases de desarrollo del mismo, ya que incluye todas las cualidades que lo caracterizan y que determinan su utilidad y existencia, sinónimo de eficiencia, flexibilidad, corrección, confiabilidad, portabilidad, usabilidad, seguridad e integridad.

Por lo que es de vital importancia considerar que estos proyectos informáticos estén desarrollados sobre una arquitectura sólida, ya que la misma no solo dicta como está construido el sistema, sino que determina si la aplicación tiene los atributos de calidad esenciales para un proyecto exitoso, estos incluyen usabilidad, facilidad de mantenimiento, si cumple con los requerimientos de rendimiento y estándares de seguridad, y si puede evolucionar fácilmente ante los cambios de los requerimientos, de ahí la necesidad de priorizar el diseño de la arquitectura tanto para el producto de software como para el proceso de desarrollo.

La arquitectura de software (AS) es considerada una disciplina por mérito propio, es la que establece los fundamentos para que analistas, diseñadores, programadores, entre tantos otros miembros del equipo de desarrollo del software trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades. Además, el arquitecto debe llevar a cabo cuidadosamente y con calidad, el diseño o selección, representación, evaluación, análisis y recuperación de la misma; tiene que garantizar un desarrollo y evolución basados estrictamente en ella.

Por estos motivos, la AS se convierte en un punto clave en el proceso de desarrollo del software, por lo que además de buscar un diseño sólido es preciso que se haga una evaluación de las decisiones tomadas durante el diseño, pues este paso brinda la posibilidad de saber el grado con que se han alcanzado los atributos de calidad que se persiguen, y pone al descubierto tanto los puntos débiles como las potencialidades de la arquitectura propuesta en una etapa donde se pueden hacer los cambios necesarios para mitigar estas debilidades sin un gran costo, maximizando los resultados del proceso de desarrollo del software.

Los países desarrollados convencidos de las ventajas irrefutables de la aplicación de la informática en todas las áreas, se han convertido, sin lugar a dudas, en los mayores productores y consumidores de software en el mundo. No obstante, existen algunos países en vías de desarrollo que han logrado introducirse significativamente en el mercado internacional de la industria del software, siendo los casos más evidentes el de la India, Irlanda e Israel, que han alcanzado un gran éxito en esta industria. Pero no solo los países en vías de desarrollo han adoptado una posición definitiva en dirección a la informatización, países como Cuba y la República Bolivariana de Venezuela se han propuesto como meta desarrollar la industria del software, con el fin de crear aplicaciones para la informatización de la sociedad, y lograr escalar en el mercado del software a nivel mundial.

Precisamente en el año 2006, el Ministerio del Poder Popular para la Alimentación (MINPPAL) de la República Bolivariana de Venezuela propone el surgimiento del Centro de Balance o Centro Nacional de Balance Alimentario (CNBA), un sistema dinámico que integra a todos los organismos competentes en la seguridad y soberanía alimentaria de la nación venezolana; con el objetivo de garantizar el consumo equilibrado y permanente de alimentos, así como la disponibilidad de las reservas [1].

El CNBA es una de las instituciones que ha dado el paso definitivo hacia la mejora de los servicios a través de la informatización de sus actividades, siendo necesario un sistema informático que le permita contribuir al perfeccionamiento continuo de sus funcionalidades, objetivo alcanzable solamente si esta aplicación informática es sustentada por una arquitectura robusta y flexible, que garantice el desarrollo y mantenimiento exitoso del software. Este centro con el propósito de darle seguimiento a la disponibilidad alimentaria requiere de los datos suministrados por los diferentes Entes distribuidos geográficamente por todo el país y con una marcada desigualdad de tecnología, puesto que algunos Entes manejan la información en formato digital mientras que la gran mayoría la tienen en copia dura, es decir, en los tradicionales archivos de papeles, esto provoca la demora en la entrega de los datos, puesto que esta actividad se realiza a través de llamadas telefónicas o correos electrónicos,

irregularidades y diferencias entre la información entregada por cada uno de ellos, y una significativa falta de seguridad en la información que se maneja y en las formas de suministro de la misma, elemento muy importante porque a partir del análisis de estos datos se generan los reportes que facilitan la toma de decisiones del MINPPAL, por lo que sería eficaz desarrollar un mecanismo para la recogida de los datos en todos los Entes independientemente de su ubicación, que erradique las diferencias, la demora de entrega de la información y la falta de seguridad.

Problema Científico

¿Cómo lograr el diseño de una arquitectura de software basada en la tecnología J2EE y el Framework Spring que garantice la implementación y evolución exitosa de una aplicación Web para la captura de la información requerida por el CNBA?

Objeto de Investigación

Arquitectura de Software.

Campo de acción

Arquitectura de Software con la tecnología J2EE y el Framework Spring.

Objetivos

Objetivo general:

Diseñar la arquitectura de software con J2EE como tecnología y el Framework Spring de la aplicación Web para la captura de la información requerida por el CNBA.

Objetivos específicos:

- Describir una arquitectura de software con la tecnología J2EE y el Framework Spring que pueda ser utilizada como referencia en el desarrollo de aplicaciones web que utilicen esta tecnología.
- Evaluar la arquitectura diseñada con uno de los métodos de evaluación de arquitectura propuestos en la literatura.
- Respaldar la capacidad de la arquitectura con la implementación de la aplicación Web para la captura de la información requerida por el CNBA.

Tareas de la investigación

1. Estudio de los temas relacionados con la arquitectura de software teniendo en cuenta la tecnología J2EE y el Framework Spring para el desarrollo de aplicaciones web.
2. Estudio de los requerimientos del sistema a implementar según las necesidades del CNBA como cliente de la aplicación Web.
3. Descripción de la arquitectura de software con la tecnología J2EE y el Framework Spring.
4. Evaluación de la arquitectura diseñada por uno de los métodos propuestos en la literatura.
5. Ratificación de la capacidad de la arquitectura propuesta con la implementación de la aplicación Web para la captura de la información requerida por el CNBA.

El trabajo consta de 3 capítulos que a su vez se dividen en epígrafes y estos en subepígrafes de acuerdo al nivel de detalle que requiere el contenido abordado en cada uno de ellos. A continuación se explica brevemente el contenido de cada capítulo:

Capítulo 1: Fundamentación Teórica. Este capítulo abarca todo el estudio de los temas relacionados con la AS como base teórica para fundamentar las decisiones tomadas, además se expone un resumen de los elementos esenciales del negocio y un estudio del estado del arte de las tecnologías.

Capítulo 2: Descripción de la arquitectura de software. Este capítulo comprende toda la descripción arquitectónica según los planteamientos de la metodología Rational Unified Process (RUP), abordando también temas relevantes como son los estilos de codificación y patrones de diseño.

Capítulo 3: Evaluación de la arquitectura propuesta. En este capítulo se muestra la evaluación de la arquitectura a través de uno de los métodos propuestos en la literatura, además se reafirma su capacidad mediante elementos de la implementación de la aplicación Web para la captura de la información requerida por el CNBA como el ejemplo práctico de su aplicación.

Capítulo 1: Fundamentación teórica

En el presente capítulo se hace un estudio profundo de los temas relacionados con la AS, temas tan relevantes como patrones y estilos arquitectónicos, patrones de diseño, lenguajes y herramientas de modelado y formas de evaluación de la AS, además se plantea la definición de AS que será seguida en el desarrollo ya que es un concepto muy controversial en la comunidad de la AS.

1.1 Descripción del entorno

El CNBA es el encargado de recopilar toda la información que le es suministrada por los diferentes Entes distribuidos geográficamente por el estado de Caracas con el objetivo de generar balances nacionales y alertas tempranas para garantizar el abastecimiento, conjugando producción, inventarios, importaciones, exportaciones y niveles de reserva. Como todos los Entes no cuentan con la tecnología adecuada, algunos envían la información en formato digital y los otros en papeles, lo que provoca demora en la generación de los reportes que el CNBA envía al MINPPAL pues la información no tiene un formato estándar y se les hace bastante complicado realizar los reportes, también es evidente una falta de seguridad, pues el suministro de la información se realiza a través de llamadas telefónicas, correo tradicional y electrónico, e incluso se lleva personalmente al CNBA. Todos los reportes que le envía el CNBA al MINPPAL son muy importantes ya que a partir de ellos se toman las decisiones adecuadas para lograr una correcta manipulación de los alimentos en los estados venezolanos.

Para contribuir al proceso de mejora de servicios del CNBA se necesita una aplicación que permita recopilar toda la información independientemente de la ubicación geográfica de los Entes, que establezca un formato estándar para erradicar las diferencias en la información, la falta seguridad y la demora en la entrega de los reportes.

1.2 Principales definiciones

1.2.1 Arquitectura de Software

Realmente en el mundo no existe una visión clara referente a la definición concreta de la AS que sea respaldada unánimemente por la totalidad de los arquitectos. El número de definiciones circulantes alcanza un orden de tres dígitos, amenazando llegar a cuatro. De hecho, existen grandes colecciones de definiciones alternativas o contrapuestas como la colección que se encuentra en el portal web del Software Engineering Institute (SEI) [2], a la que cada quien puede agregar la suya.

Una definición reconocida es la de Clements [3]: “La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se le percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.”

La definición más usada de AS es de la IEEE Std 1471-2000, plantea: “La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución”.

En el libro "An introduction to Software Architecture", David Garlan y Mary Shaw definen que la Arquitectura es un nivel de diseño que hace foco en aspectos "más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema" [4].

Antes el número y variedad de definiciones existentes de AS, Mary Shaw y David Garlan [5] proporcionaron una sistematización iluminadora, explicando las diferencias entre definiciones en función de distintas clases de modelos. Destilando las definiciones y los puntos de vista implícitos o explícitos, los autores clasifican los modelos de esta forma:

- **Modelos estructurales:** Sostienen que la AS está formada por componentes, conexiones entre ellos y (usualmente) otros aspectos tales como configuración, estilo, restricciones, semántica, análisis, propiedades, racionalizaciones, requerimientos, necesidades de los participantes. El trabajo en esta área está caracterizado por el desarrollo de Lenguajes de Descripción Arquitectónica (ADLs).
- **Modelos de Framework:** Son similares a la vista estructural, pero su énfasis primario radica en la (usualmente una sola) estructura coherente del sistema completo, en vez de concentrarse en su composición. Los modelos de Framework a menudo se refieren a dominios o clases de problemas específicos. El trabajo que ejemplifica esta variante incluye arquitecturas de software específicas de dominios.

- **Modelos dinámicos:** “Dinámico” puede referirse a los cambios en la configuración del sistema, o a la dinámica involucrada en el progreso de la computación, tales como valores cambiantes de datos.

- **Modelos de proceso**: Se concentran en la construcción de la arquitectura, y en los pasos o procesos involucrados en esa construcción. En esta perspectiva, la arquitectura es el resultado de seguir un argumento (script) de proceso.

- **Modelos funcionales:** Una minoría considera la arquitectura como un conjunto de componentes funcionales, organizados en capas que proporcionan servicios hacia arriba. Es tal vez útil pensar en esta visión como un Framework particular.

Ninguna de estas vistas excluye a las otras, ni representa un conflicto fundamental sobre lo que es o debe ser la AS. Por el contrario, representan un espectro en la comunidad de investigación sobre distintos énfasis que pueden aplicarse a la arquitectura: sobre sus partes constituyentes, su totalidad, la forma en que se comporta una vez construida, o el proceso de su construcción. Tomadas en su conjunto, destacan más bien un consenso.

Independientemente de las discrepancias entre las diversas definiciones, es común entre todos los autores el concepto de la arquitectura como un punto de vista que concierne a un alto nivel de abstracción.

En un sentido amplio y una vez analizada la definición de Bass et al. [6] se está de acuerdo en que la AS es el diseño de más alto nivel de la estructura de un sistema, programa o aplicación y tiene la responsabilidad de:

- Definir los módulos principales.
- Definir las responsabilidades que tendrá cada uno de estos módulos.
- Definir la interacción que existirá entre dichos módulos:
    - Control y flujo de datos.
    - Secuencia de la información.
    - Protocolos de interacción y comunicación.
    - Ubicación en el hardware.

La Arquitectura no es el software operacional, más bien es la representación que capacita al ingeniero de software para: analizar la efectividad del diseño para la consecución de los requisitos fijados, considerar las alternativas arquitectónicas en una etapa en la cual hacer cambios en el diseño es relativamente fácil y reducir los riesgos asociados a la construcción del software.

Dentro de la importancia de la AS el presente trabajo de diploma se basa en las tres razones claves definidas por Bass et al. [6] en su libro dedicado a la AS, las que consisten en:

- Las representaciones de la AS facilitan la comunicación entre todas las partes (partícipes) interesadas en el desarrollo de un sistema basado en computadora.
- La arquitectura destaca decisiones tempranas de diseño que tendrán un profundo impacto en todo el trabajo de ingeniería de software que sigue, y es tan importante en el éxito final del sistema como una entidad operacional.
- La arquitectura "constituye un modelo relativamente pequeño e intelectual comprensible de cómo está estructurado el sistema y de cómo trabajan juntos sus componentes" [6].

1.2.2 Estilos y Patrones

En algunas bibliografías referentes al estudio de los patrones arquitectónicos se suele llamar de la misma manera a los patrones y estilos, pero la mayoría de los autores los ven de manera separada. Ambos se refieren a formas de estructurar los sistemas y cómo relacionar los componentes de estos, la diferencia radica en el nivel de abstracción en que se aplican. Los estilos están en un plano de abstracción más alto, definiendo cómo configurar la arquitectura, mientras los patrones están más cercanos al diseño, incluso podría decirse que más cercano a algo físico, pues estos pueden representarse mediante código en un lenguaje de programación determinado. Los elementos por los que difieren son mostrados en la siguiente tabla.

Tabla 1. Diferencias entre Estilos y Patrones Arquitectónicos.

Elemento a comparar	Estilo Arquitectónico	Patrón Arquitectónico
Definición	Son una categorización de sistemas, además describen sistemas completos.	Son soluciones generales a problemas comunes, además describen parte del sistema.

Representación	Sólo describe el esqueleto estructural general para aplicaciones.	Existen en varios rangos de escala, comenzando con patrones que definen la estructura básica de una aplicación.
Dependencia de contexto	Son independientes del contexto al que puedan ser aplicados.	Partiendo de la definición de patrón, requieren de la especificación de un contexto del problema.
Solución	Expresan técnicas de diseño desde una perspectiva que es independiente de la situación actual de diseño.	Expresa un problema recurrente de diseño muy específico, y presenta una solución para él, desde el punto de vista del contexto en el que se presenta.

Una vez analizados los elementos esenciales de los estilos y patrones de arquitectura, se concluye que si bien existen convergencias entre ambos conceptos, como es el caso de algunos patrones que coinciden con estilos hasta en el nombre, por ejemplo el Modelo Vista Controlador; hay que tener en cuenta que los patrones se refieren más a prácticas de reutilización mientras que los estilos conciernen a teorías sobre la estructura de los sistemas, por tanto son términos diferentes en dependencia del nivel de abstracción que se tenga en cuenta.

1.2.3 Estilo arquitectónico

Shaw y Garlan [7] definen estilo arquitectónico como una familia de sistemas de software en términos de un patrón de organización estructural, que define un vocabulario de componentes y tipos de conectores y un conjunto de restricciones de cómo pueden ser combinadas. Para muchos estilos puede existir uno o más modelos semánticos que especifiquen cómo determinar las propiedades generales del sistema partiendo de las propiedades de sus partes.

Buschmann et al. [8] definen estilo arquitectónico como una familia de sistemas de software en términos de su organización estructural. Expresa componentes y las relaciones entre estos, con las restricciones de su aplicación y la composición asociada, así como también las reglas para su construcción. Así mismo, se considera como un tipo particular de estructura fundamental para un sistema de software, conjuntamente con un método asociado que especifica cómo construirlo. Éste incluye información acerca de cuándo usar la arquitectura que describe, sus invariantes y especializaciones, así como las consecuencias de su aplicación.

A simple vista, ambas definiciones parecen expresar la misma idea. La diferencia entre los planteamientos de Shaw y Garlan y Buschmann viene dada por la amplitud de la noción de componente en cada una de las definiciones. Buschmann asume como componentes a subsistemas conformados por otros componentes más sencillos, mientras que Shaw y Garlan utilizan la noción de componente como elementos simples, ya sean de dato o de proceso. En virtud de esto, la diferencia entre ambas definiciones gira en torno al nivel de abstracción, dado que Buschmann et al. plantean un grado mayor en su concepto de estilo arquitectónico, sugiriendo una estructura genérica para la organización de componentes de ciertas familias de sistemas, independientemente del contexto en que estas se desarrollen.

1.2.4 Patrón arquitectónico

Buschmann et al. [8] definen *patrón* como una regla que consta de tres partes, la cual expresa una relación entre un contexto, un problema y una solución. En líneas generales, un patrón sigue el siguiente esquema:

- *Contexto.* Es una situación de diseño en la que aparece un problema de diseño.
- *Problema*. Es un conjunto de fuerzas que aparecen repetidamente en el contexto.
- *Solución*. Es una configuración que equilibra estas fuerzas. Esta abarca:
    - Estructura con componentes y relaciones.
    - Comportamiento a tiempo de ejecución: aspectos dinámicos de la solución, como la colaboración entre componentes, la comunicación entre ellos, etc.

Partiendo de esta definición, propone los *patrones arquitectónicos* como descripción de un problema particular y recurrente de diseño, que aparece en contextos de diseño específico, y presenta un esquema genérico demostrado con éxito para su solución. El esquema de solución se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades y desarrollos, así como también la forma en que estos colaboran entre sí.

Así mismo, plantean que los patrones arquitectónicos expresan el esquema de organización estructural fundamental para sistemas de software. Provee un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos.

Propone que son plantillas para arquitecturas de software concretas, que especifican las propiedades estructurales de una aplicación con amplitud de todo el sistema y tienen un impacto en la arquitectura de subsistemas. La selección de un patrón arquitectónico es, por lo tanto, una decisión fundamental de diseño en el desarrollo de un sistema de software.

Visto de esta manera, el concepto de *patrón arquitectónico* propuesto por Buschmann et al. [8] equivale al establecido por Shaw y Garlan [7] para *estilo arquitectónico*, quienes tratan indistintamente estos dos términos. Barbacci et al. [20] hacen la analogía de la construcción de una arquitectura de un sistema complejo como la inclusión de instancias de más de un patrón arquitectónico, compuestos de maneras arbitrarias. La colección de patrones arquitectónicos debe ser estudiada en términos de factores de calidad e intereses, en anticipación a su uso. Esto quiere decir que un patrón puede ser analizado previamente, con la intención de seleccionar el que mejor se adapte a los requerimientos de calidad que debe cumplir el sistema. De manera similar, Barbacci et al. [20] proponen que debe estudiarse la composición de los patrones, pues esta puede dificultar aspectos como el análisis, o poner en conflicto otros atributos de calidad.

1.2.5 Patrones de diseño

Un *patrón de diseño* provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular [8].

Son menores en escala que los patrones arquitectónicos, y tienden a ser independientes de los lenguajes y paradigmas de programación. Su aplicación no tiene efectos en la estructura fundamental del sistema, pero sí sobre la de un subsistema [8], debido a que especifica a un mayor nivel de detalle, sin llegar a la implementación, el comportamiento de los componentes del subsistema.

Un patrón de diseño de manera general no es más que una solución estándar para un problema común de programación, una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios, un proyecto o estructura de implementación que logra una finalidad determinada, un lenguaje de programación de alto nivel, una manera más práctica de describir ciertos aspectos de la organización de un programa, conexiones entre componentes de programas y la forma de un diagrama de objeto o de un modelo de objeto. [10]

1.3 Estado del arte de la arquitectura de software

1.3.1 Estilos arquitectónicos y Patrones arquitectónicos

Los estilos y patrones arquitectónicos que habrán de describirse a continuación, así como los patrones de diseño no aspiran a ser todos los que se han propuesto en el tan amplio mundo de la AS, sino apenas los más representativos y vigentes que responden a las características que debe cumplir la aplicación.

Estilo Modelo Vista Controlador (MVC): El estilo conocido como Modelo Vista Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes: Modelo, Vista y Controlador.

Ventajas: Soporte de vistas múltiples, dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Adaptación al cambio.

Desventajas: Costo de actualizaciones frecuentes. Desacoplar el modelo de la vista no significa que los desarrolladores del modelo puedan ignorar la naturaleza de las vistas. Incrementa la complejidad de la solución.

Estilos Centrados en Datos: Esta familia de estilos enfatiza la integrabilidad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Sub-estilos característicos de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra.

Estilos de Llamada y Retorno: Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.

Estilo Arquitecturas en Capas: Garlan y Shaw [11] definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.

Patrón arquitectónico Modelo Vista Controlador: Separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes [12]:

- Modelo: Administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- Vista: Maneja la visualización de la información.
- Controlador: Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

Patrón arquitectónico Vista de Plantilla: La idea básica de la plantilla de vista es incrustar marcadores en una página HTML cuando esta es escrita. Cuando la página se utiliza para solicitar un servicio, los marcadores se sustituirán por los resultados obtenidos de acuerdo al servicio que brinda la página, como por ejemplo, el resultado de una consulta obtenida en una base de datos.

Patrón arquitectónico Página de Control: La idea básica detrás de una página Controlador es tener un módulo en el servidor web, dígase un controlador para cada página en el sitio web. En la práctica, no funciona exactamente a una por página, ya que a veces le puede realizar un enlace a cualquier página dinámica mediante alguna función de información dinámica.

Patrón de diseño Fachada: Sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Utilizado para reducir la dependencia entre clases, ya que ofrece un punto de acceso al resto de clases, si estas cambian o se sustituyen por otras solo hay que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones cliente. Fachada no oculta las clases sino que ofrece una forma más sencilla de acceder a ellas, en los casos en que se requiere se puede acceder directamente a ellas.

Patrón de diseño Bridge (Puente): es una técnica usada en programación para desacoplar una abstracción de su implementación, de manera que ambas puedan ser modificadas independientemente sin necesidad de alterar por ello la otra. Esto es, se desacopla una abstracción de su implementación para que puedan variar independientemente.

Patrón de diseño Inyección de Dependencia: (Dependency Inyection - DI, o también conocido como Inversion of Control - IoC). Este patrón radica en resolver las dependencias de cada clase (atributos) generando los objetos cuando se arranca la aplicación y luego inyectarlos en los demás objetos que los necesiten a través de los métodos set o del constructor. Es una forma para mitigar la proliferación de dependencias y así fomentar el bajo acoplamiento entre los componentes, y además tiene la

intención de reducir la cantidad de código de infraestructura que se debe escribir. Spring es uno de los Frameworks que provee un contenedor DI listo para usar.

Patrón de diseño DAO: Plantea como solución utilizar un Data Access Object (DAO) para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.

El DAO implementa el mecanismo de acceso requerido para trabajar con la fuente de datos. Los componentes de negocio que tratan con el DAO utilizan una interface simple expuesta por el DAO para sus clientes. El DAO oculta completamente los detalles de implementación de la fuente de datos a sus clientes. Como la interface expuesta por el DAO no cambia cuando cambia la implementación de la fuente de datos subyacente, este patrón permite al DAO adaptarse a diferentes esquemas de almacenamiento sin que esto afecte a sus clientes o componentes de negocio. Esencialmente, el DAO actúa como un adaptador entre el componente y la fuente de datos. [28]

Patrón de diseño Bajo Acoplamiento: Es uno de los patrones de asignación de responsabilidades o GRASP. El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, las conoce y recurre a ellas. Acoplamiento bajo significa que una clase no depende de muchas clases. Acoplamiento alto significa que una clase recurre a muchas otras clases. Esto presenta los siguientes problemas:

- Los cambios de las clases afines ocasionan cambios locales.
- Difíciles de entender cuando están aisladas.
- Difíciles de reutilizar puesto que dependen de otras clases.

Entonces la solución al problema: ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización?, es asignar una responsabilidad para mantener bajo acoplamiento. El grado de acoplamiento no puede considerarse aisladamente de otros principios como la Alta Cohesión, sin embargo, es un factor a considerar cuando se intente mejorar el diseño.

Patrón de diseño Alta Cohesión: Es una de los patrones de asignación de responsabilidades o GRASP. La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Una baja cohesión hace muchas cosas no afines o realiza trabajo excesivo. Esto presenta los siguientes problemas:

- Son difíciles de comprender
- Difíciles de reutilizar
- Difíciles de conservar
- Las afectan constantemente los cambios.

El patrón soluciona el problema ¿Cómo mantener la complejidad dentro de límites manejables?, con la asignación de una responsabilidad de modo que la cohesión siga siendo alta. Este diseño es conveniente ya que da soporte a una alta cohesión y a un bajo acoplamiento, ya que en la práctica, el nivel de cohesión no puede ser considerado independiente de los otros patrones y principios como el patrón Bajo Acoplamiento.

1.3.2 Relación entre los niveles de Abstracción

Con los estudios realizados de las diferentes definiciones de Estilos Arquitectónicos, Patrones Arquitectónicos y Patrones de Diseño, la figura que se muestra a continuación presenta la relación de abstracción existente entre los conceptos de estilo arquitectónico, patrón arquitectónico y patrón de diseño. En ella se representa el planteamiento de Buschmann et al. [8], que proponen el desarrollo de arquitecturas de software como un sistema de patrones, y distintos niveles de abstracción.

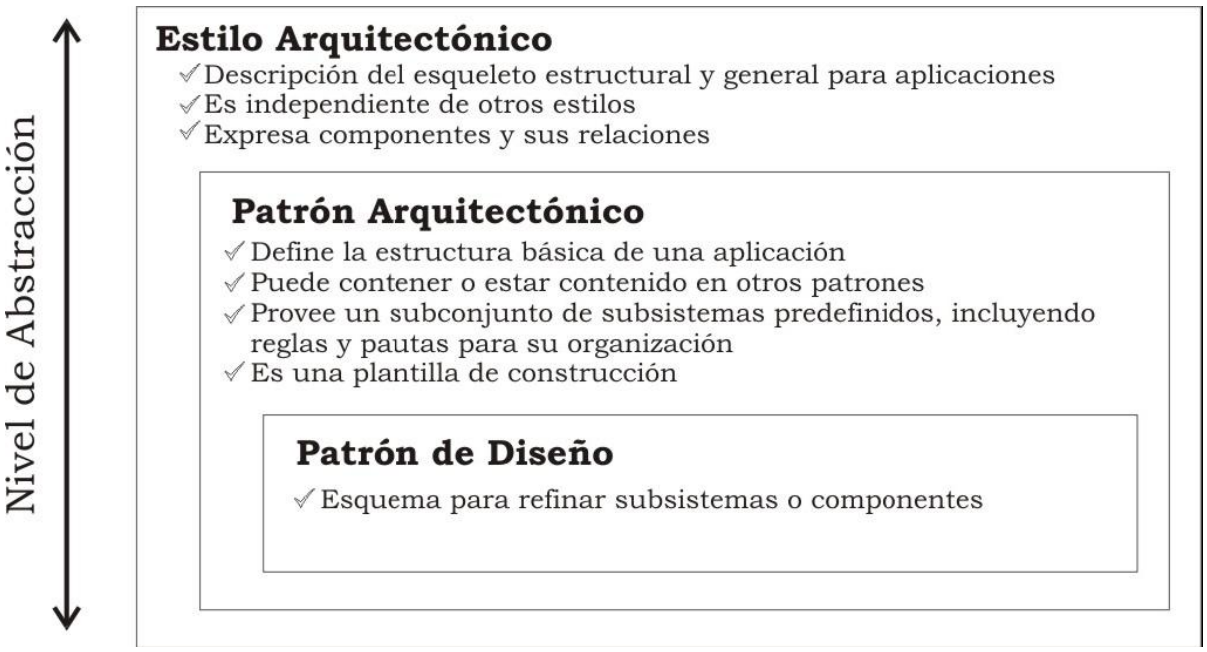


Figura 1. Relación de abstracción entre estilo arquitectónico, patrón arquitectónico y patrón de diseño.

Los estilos y patrones ayudan al arquitecto a definir la composición y el comportamiento del sistema de software, y una combinación adecuada de ellos permite alcanzar los requerimientos de calidad.
Además, la organización del sistema de software debe estar disponible para todos los involucrados en el desarrollo del sistema, ya que establece un mecanismo de comunicación entre los mismos. Tal objetivo se logra mediante la representación de la arquitectura en formas distintas, obteniendo así una descripción de la misma de forma que puede ser entendida y analizada por todos los involucrados, con miras a obtener un sistema de calidad. Estas descripciones pueden establecerse a través de las vistas arquitectónicas, las notaciones como UML y los lenguajes de descripción arquitectónica [13].

1.3.3 Tecnologías y Herramientas

1.3.3.1 Plataforma de desarrollo

Tanto JAVA como PHP y PYTHON, que son los entornos de desarrollo más utilizados, y por tanto está probada su eficiencia y potencia en el desarrollo de software, pueden ser utilizados para programar orientado a objeto, están liberados bajo licencia GPL, cuentan con muy buena documentación, son multiplataformas, poseen Frameworks que facilitan el desarrollo y garantizan la seguridad e integridad de los datos, así como el trabajo con wizards. Los tres disponen de Eclipse como Entorno de Desarrollo Integrado (IDE) y en el caso de JAVA también se encuentra NetBeans.

Se puede decir que tanto JAVA como PHP y PYTHON pueden ser utilizados para el desarrollo del software, ninguno puede ser descartado del todo desde el punto de vista de los requisitos de la aplicación Web, pero es importante también tener en cuenta la necesidad de desarrollar una aplicación de escritorio para recopilar la información requerida por el CNBA en los Entes que tienen los datos en formato digital y con este nuevo requisito el entorno de desarrollo más apropiado es Java, porque presenta IDEs de desarrollo como NetBeans y Eclipse, los cuales ofrecen grandes facilidades con editores gráficos para programar tanto aplicaciones de escritorio como aplicaciones Web. Teniendo en cuenta todo lo anteriormente planteado y que el equipo de desarrollo tiene experiencia en JAVA, es que se toma la decisión de usar esta plataforma de desarrollo para implementar la aplicación.

La plataforma Java ofrece las siguientes ventajas:

- Brinda herramientas libres, lo cual resulta de gran valor producto al interés que existe por parte del Ministerio de migrar hacia software libre.
- Es multiplataforma.

- Tiene un API bien documentado, facilitando el desarrollo así como las futuras acciones de mejora y mantenimiento de las aplicaciones. Además, el lenguaje de programación está basado en el paradigma orientado a objeto, el cual brinda un conjunto de bondades para la construcción del software.
- Plataforma muy completa para el desarrollo de aplicaciones del lado del cliente y del lado del servidor.
- Es robusta, bien probada y segura.
- Cumple con todos los aspectos esenciales para el desarrollo del proyecto [1].

Luego de establecer el entorno de desarrollo es importante el estudio de los Frameworks que acompañarán la implementación, en este caso se tienen:

Spring: Spring es un Framework de código abierto de desarrollo de aplicaciones para la plataforma Java, cuenta con un conjunto de librerías en las que podemos escoger aquellas que faciliten el desarrollo de nuestra aplicación. Entre sus posibilidades más potentes está su contenedor de Inversión de Control (Inversión de Control, también llamado Inyección de Dependencias, es una técnica alternativa a las clásicas búsquedas de recursos vía JNDI. Permite configurar las clases en un archivo XML y definir en él las dependencias. De esta forma la aplicación se vuelve muy modular y a la vez no adquiere dependencias con Spring), la introducción de aspectos, plantillas de utilidades para Hibernate, iBatis y JDBC así como la integración con JSF.

Es uno de los proyectos más sorprendentes en el panorama actual en Java en el grado en que ayuda a que los diferentes componentes que forman una aplicación trabajen entre sí; pero no establece apenas dependencias consigo mismo. Esta es la primera característica de este Framework. Sería posible retirarlo sin prácticamente cambiar líneas de código. Lo único necesario es, lógicamente, añadir la funcionalidad que provee, ya sea con otro Framework similar o mediante nuestro código.

Spring MVC: Una de las alternativas a Struts, pero mucho más elegante, que además ha incorporado una lógica de diseño más sencilla y que cuenta con todo el conjunto de librerías de Spring Framework.

Hibernate: Hibernate es un motor de persistencia de código abierto. Permite mapear un modelo de clases a un modelo relacional sin imponer ningún tipo de restricción en ambos diseños. Cuenta con una amplia documentación, tanto a nivel de libros publicados como disponibles gratuitamente en su

Web. A nivel comercial está respaldado por JBoss, que proporciona servicios de soporte, consultoría y formación en el mismo.

Hoy en día, en su versión 3.1, ya soporta el estándar EJB 3 (su autor es uno de los principales integrantes del JCP que está definiendo esta especificación) por lo que se puede elegir desarrollar aplicaciones empleando EJB 3 -que correrán en cualquier contenedor de EJB's que soporte J2EE 5- o aplicaciones independientes. Esto no solo brinda la inversión de tiempo en Hibernate de cara al futuro, sino que, de ser útil, nos hace totalmente independientes del mismo.

Acegi: es un gestor de seguridad que está diseñado fundamentalmente para ser usado con Spring y en el que destaca su versatilidad. Acegi proporciona una capa que envuelve diversos estándares de seguridad presentes en Java y ofrece una forma unificada de configuración a través de un descriptor en XML.

Cubre la capa web y la de negocio. A nivel Web captura todas las peticiones mediante la implementación de un filtro y a nivel de métodos mediante interceptación a través de AOP. En ambos casos permite aplicar los criterios de seguridad que trae por defecto o añadir nuevas opciones de forma sencilla implementando las interfaces diseñadas a tal fin.

Acegi se ha elegido como opción frente a los sistemas propietarios de los diferentes vendedores por su universalidad de uso –no es necesario cambiar nada si se cambia de proveedor en los servidores- así como por su potencia –que engloba los APIs de seguridad de Java-.

También es relevante definir el servidor de aplicaciones sobre el cual va a correr la aplicación Web, en este caso se decidió que fuera Apache Tomcat el cual no es más que un contenedor de Servlets con un entorno JSP [29]. Dado que Apache Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

1.3.3.2 Sistema Gestor de Bases de Datos

El Sistema Gestor de Bases de Datos (SGBD) es una aplicación que permite a los usuarios definir, crear y mantener la base de datos y proporciona un acceso controlado a la misma. Proporciona los servicios de creación y definición de la base de datos, manipulación de los datos, acceso controlado a

los datos mediante mecanismos de seguridad, mantener integridad y consistencia de los datos, acceso compartido a la base de datos y mecanismos de copias de respaldo y recuperación.

Asumiendo la importancia que tiene el repositorio final de la información en todo sistema informático, además, en este caso, la cantidad de información que se maneja es muy grande y necesita estar en un contenedor muy seguro por la importancia y trascendencia que origina, que va a ser consultado posiblemente de modo concurrente por un gran número de usuarios, es necesario hacer una comparación profunda para decidir cuál de los SGBD existentes se utilizará, el balance será hecho entre los SGBD más usados y mejor documentados, MySQL, PostgreSQL y Oracle.

MySQL es un sistema de gestión de bases de datos relacional, utiliza el lenguaje de programación Structured Query Language (SQL) y entre sus características se destacan [14]:

- En multiplataforma.
- Tiene APIs disponibles para C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, y Tcl.
- Proporciona sistemas de almacenamientos transaccionales y no transaccionales.
- Presenta un sistema de privilegios y contraseñas que es muy flexible y seguro, y permite verificación basada en el host. Las contraseñas son seguras porque todo el tráfico de contraseñas está encriptado cuando se conecta con un servidor.
- Los clientes pueden conectar con el servidor MySQL usando sockets TCP/IP en cualquier plataforma.
- La interfaz para el conector J MySQL proporciona soporte para clientes Java que usen conexiones JDBC. Estos clientes pueden ejecutarse en Windows o Unix y es un código abierto.
- El tamaño efectivo máximo para las bases de datos en MySQL usualmente lo determinan los límites de tamaño de ficheros del sistema operativo, y no límites internos de MySQL.

PostgreSQL es un sistema objeto-relacional ya que incluye características de la orientación a objetos como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. A pesar de esto, PostgreSQL no es un sistema de gestión de bases de datos puramente orientado a objetos. Algunas de sus principales características son [15]:

- Implementación del estándar SQL92/SQL99.

- Soporta distintos tipos de datos: además del soporte para los tipos base, también soporta datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes (MAC, IP), cadenas de bits, etc. También permite la creación de tipos propios.
- Incorpora una estructura de datos array y funciones de diversa índole: manejo de fechas, geométricas, orientadas a operaciones con redes, etc.
- Permite la declaración de funciones propias, así como la definición de disparadores.
- Soporta el uso de índices, reglas y vistas.
- Incluye herencia entre tablas (aunque no entre objetos, ya que no existen).
- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.
- Posee una gran escalabilidad. Es capaz de ajustarse al número de CPUs y a la cantidad de memoria que posee el sistema de forma óptima, haciéndole capaz de soportar una mayor cantidad de peticiones simultáneas de manera correcta.
- Implementa el uso de rollback's, subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz, y ofreciendo soluciones en campos en las que MySQL no podría.

Oracle es un uno de los mejores SGBD pero es comercial, por lo que necesita de licencias para poder usarlo, y las limitaciones que impone el software propietario en contraste con las bondades que brinda el software libre hacen de este SGBD débil en comparación con PostgreSQL.

Al finalizar la comparación se llegó a la conclusión de utilizar PostgreSQL como SGBD debido a su condición de ser multiplataforma, por estar licenciado bajo GNU y a pesar de esto tener gran número de características que normalmente sólo se encontraban en las bases de datos comerciales tales como Oracle y a la superioridad que demuestra respecto a las necesidades del problema planteado sobre su principal rival MySQL, entre ellas se encuentran: soporta una capacidad de almacenamiento en el orden de los TB (terabytes), posee gran escalabilidad y por tanto puede soportar una mayor cantidad de peticiones concurrentes y tiene la capacidad de comprobar la integridad referencial, así como también la de almacenar procedimientos en la propia base de datos.

1.3.3.3 Herramienta para el Control de Versiones

La mayoría de los proyectos de desarrollo del software involucran a muchos desarrolladores trabajando concurrentemente con una colección variada de archivos durante un largo período de tiempo. Por consiguiente, es crítico que los cambios hechos por estos desarrolladores sean rastreados con el fin de conocer el responsable de cada cambio. Además es muy importante poder combinar las contribuciones de todos los desarrolladores.

Una buena herramienta para el control de las versiones ayuda a proteger la integridad de los datos, manteniendo una historia de revisión, no hay que preocuparse de cuestiones como por ejemplo, que el código fue borrado en una edición de un día; un repositorio central del proyecto también puede ayudar con la copia de respaldo de los datos.

Entonces, el control de versiones es en esencia el rastreo, controlante y combinado de versiones diferentes (las llamadas revisiones) de un proyecto con el paso del tiempo.

Subversion y CVS

Subversion es patrocinado (como un proyecto de código abierto) por tigris.org, una comunidad en línea diseñada para promover el desarrollo de herramientas de código abierto para la Ingeniería de Software. La naturaleza abierta de Subversion también le hace muy integrable, el centro de Subversion es, de hecho, una colección de bibliotecas con una API abierta y muy bien documentada, esta API le permite a los desarrolladores integrarlo en otra herramienta e incluso automatizar una parte del proceso de interacción con un repositorio SVN, además provee una Interfaz Gráfica del Usuario (GUI).

Usa transacciones cada vez que modifica la base de datos, es decir, cuando se realiza un commit, Subversion marca al estado actual de la base de datos y luego hace sus modificaciones, de ese modo, si una colisión interrumpe al commit, no hay riesgo que la base de datos sea corrompida, pues la base de datos automáticamente será restaurada a su estado antes de que el commit comenzase, esta es otra característica que carecen muchos VCSs anteriores, como CVS o Visual SourceSafe.

Si un conflicto ocurre, SVN además de darle un tratamiento similar al que realiza CVS, también suma versiones temporales del archivo haciendo posible que el conflicto se resuelva de un modo más fácil y rápido.

Una de características poderosas y únicas de Subversion es su soporte para metainformación de archivo y de directorio en forma de propiedades, que dejan al usuario almacenar palabras claves

arbitrarias. Adicionalmente, Subversion define varias propiedades especiales que pueden usarse internamente para proveer alguna funcionalidad adicional.

El API está disponible para un gran número de lenguajes de programación como C, C + +, Java, Perl, y Pitón [16].

Entonces SVN se usará como herramienta para el control de versiones por todas las razones que la hacen superior a la herramienta CVS y fueron explicadas anteriormente.

1.4 Metodología de desarrollo

En el proceso de desarrollo de software la metodología define *Quién* debe hacer *Qué*, *Cuándo* y *Cómo* debe hacerlo para obtener los distintos productos parciales y finales; la metodología de desarrollo que será seguida en el trabajo es RUP ya que es la metodología más acorde, dada la complejidad y amplitud del proyecto, además, es una metodología muy bien documentada y ejemplificada, su ciclo de vida se caracteriza por estar dirigido por casos de uso, centrado en la arquitectura y ser iterativo e incremental.

Ella establece que la arquitectura se desarrolla mediante iteraciones, principalmente durante la fase de elaboración. Cada iteración se desarrolla comenzando con los requisitos y siguiendo con el análisis, diseño, implementación y pruebas, pero centrándose en los casos de uso relevantes desde el punto de vista de la arquitectura y en otros requisitos. El resultado final de la fase de elaboración es una línea base de la arquitectura –un esqueleto del sistema con pocos “músculos” de software [17].

Define la descripción de la arquitectura como una vista de los modelos del sistema, de los modelos de casos de uso, análisis, diseño, implementación y despliegue, que describe las partes del sistema que es importante que comprendan todos los desarrolladores y otros interesados, en otras palabras el modelo de las 4+1 vistas propuestas por Kruchten en 1999:

- La vista de la arquitectura del modelo de casos de uso: Presenta los actores y los casos de uso del sistema más importantes (o escenarios de esos casos de uso).
- La vista de la arquitectura del modelo de diseño: Presenta los clasificadores más importantes para la arquitectura pertenecientes al modelo de diseño: los subsistemas e interfaces más importantes, así como algunas pocas clases muy importantes, fundamentalmente las clases

activas. También presenta cómo se realizan los casos de uso en términos de esos clasificadores, por medio de realizaciones de caso de uso.

- La vista de la arquitectura del modelo de despliegue: Define la arquitectura física del sistema por medio de los nodos interconectados. Estos nodos son elementos hardware sobre los cuales pueden ejecutarse los elementos software. Con frecuencia conocemos cómo será la arquitectura física del sistema antes de comenzar su desarrollo. Por tanto, podemos modelar los nodos y las conexiones del modelo de despliegue tan pronto como comience el flujo de trabajo de los requisitos. Estos diagramas también pueden mostrar cómo se asignan los componentes ejecutables a los nodos.

- La vista de la arquitectura del modelo de implementación: Es una correspondencia directa de los modelos de diseño y de despliegue. Cada subsistema de servicio del diseño normalmente acaba siendo un componente por cada tipo de nodo en el que deba instalarse –pero no siempre es así-. A veces el mismo componente puede instanciarse y ejecutarse sobre varios nodos.

- La vista de la arquitectura de procesos: Incluye la descripción de las tareas (procesos, hilos, tareas programadas, eventos y notificaciones) involucrados en la ejecución del sistema. Además de incluir la ubicación de las clases y objetos necesarios para estas tareas de ser necesario. Es opcional si no se tienen procesos concurrentes.

1.4.1 Responsabilidades del arquitecto de software

Crear AS es un empeño difícil, y el arquitecto del software tiene uno de los trabajos más difíciles en un proyecto de desarrollo de software. El arquitecto debe poder comunicarse con todos los implicados en el proceso de desarrollo de software. Debe tener excelentes habilidades de diseño, tecnología, y un conocimiento de las mejores prácticas de ingeniería del software. Debe controlar el curso a través de políticas organizativas directas para lograr terminar el proyecto correctamente y a tiempo. El arquitecto del software debe ser un líder, un mentor, y una persona que toma las decisiones enérgicamente.

La metodología RUP señala que los arquitectos moldean el sistema para darle una forma. Es esta la forma: la arquitectura, que debe diseñarse para permitir que el sistema evolucione, no solo en su desarrollo inicial, sino también a lo largo de las futuras generaciones. Para encontrar esa forma, los arquitectos deben trabajar sobre la comprensión general de las funciones claves, es decir, sobre los casos de uso claves del sistema. De manera resumida, señala que el arquitecto [17]:

- Crea un esquema en borrador de la arquitectura, comenzando por la parte de la arquitectura que no es específica de los casos de uso (por ejemplo: la plataforma). Aunque esta parte de la arquitectura es independiente de los casos de uso, el arquitecto debe poseer una comprensión general de los casos de uso antes de comenzar la creación del esquema arquitectónico.
- A continuación, el arquitecto trabaja con un subconjunto de los casos de uso especificados, con aquellos que representen las funciones clases del sistema en desarrollo. Cada caso de uso seleccionado se especifica en detalle y se realiza en términos de subsistemas, clases y componentes.
- A medida que los casos de uso se especifican y maduran, se descubre más de la arquitectura. Esto a su vez, lleva a la maduración de más casos de uso. Este proceso continúa hasta que se considere que la arquitectura es estable.

1.4.2 Herramienta y lenguaje para el modelado

RUP promueve el uso de UML para modelar la AS, sin embargo, los académicos han cuestionado a UML como un instrumento para modelar la arquitectura del software. Esto es en mayor parte porque las versiones anticipadas de UML no tuvieron notaciones para componentes y conectores como elementos de primeras clases. Otras construcciones son importantes al describir la arquitectura de un software, como puertos o puntos de interacción con un componente y los roles o los puntos de interacción con un conector. UML 2.0 fue liberado en junio del 2003 y se ocupa de muchos de estos asuntos en el lenguaje. Por ejemplo en esta versión UML mueve componentes y conectores a través del ciclo de vida. Un componente puede ser descrito para usar una interfaz y proveer un puerto. Un puerto puede ser un puerto complicado que provee y consume múltiples interfaces. En fin, UML 2.0 hace grandes adelantos para convertirse en la notación estándar para describir arquitecturas.

UML provee una notación para la descripción de la proyección de los componentes de software en el hardware. Esto corresponde a la vista física del modelo 4+1. La proyección de los componentes de software permite a los ingenieros de software hacer mejores estimaciones cuando se intenta medir la calidad del sistema implementado, lo cual contribuye a la búsqueda de la mejor solución para un sistema específico. Esta notación puede ser extendida con mayor nivel de detalle y los componentes pueden ser conectados entre sí con el uso de las bondades del lenguaje UML.

Por último, los patrones y Frameworks están también soportados por UML, mediante el uso combinado de paquetes, componentes y colaboraciones, entre otros. Booch et al. [18] proponen de forma

detallada todos los aspectos que hacen de UML un lenguaje conveniente para la representación de las arquitecturas de software. Utilizar UML tiene la ventaja bien definida de ser una notación estándar para diseño del software, entonces se utilizaría el mismo lenguaje para la descripción de la arquitectura y el diseño del software.

A pesar de la creación de una gran variedad de Lenguajes de Descripción de Arquitectura (ADLs) que tienen la colección de elementos claves para la representación de primera clase de los intereses arquitectónicos y enfocan la atención en una descripción precisa de arquitectura del software. La realidad indiscutible es que no hay notación estándar para documentar arquitectura del software. Los criterios cruciales al escoger un lenguaje modelador – si es UML, un ADL, o su propia notación – es que debería propiciar la comprensión de la arquitectura por los que lo leen. El modelo debe lograr esta meta primaria a pesar de la notación usada para documentarla.

Con este punto de partida, la decisión para describir la AS con la tecnología J2EE y el Framework Spring es utilizar el lenguaje de modelado UML, que es el propuesto por la metodología de desarrollo a seguir y que además es la notación que dominan los diseñadores de la aplicación, y utilizar el mismo lenguaje favorece el entendimiento y minimiza los gastos de tiempo en el aprendizaje de otro lenguaje para la comprensión de la arquitectura.

Existen software para el modelado en UML, son las nombradas herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador), estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

El **Rational Rose** es la herramienta CASE que comercializan los desarrolladores de UML y líder en el mundo de modelación visual de sistemas que permite especificar, analizar y diseñar el sistema antes de codificarlo, sin embargo posee limitantes que la hacen débil en comparación a otras herramientas con las mismas facilidades de modelado, estas debilidades radican en la dependencia de la plataforma Windows y la integración solo con herramientas que estén en el mismo grupo de software propietario, una de las herramientas que supera a Rational Rose en estos relevantes puntos es **Visual Paradigm para UML** (VP-UML) que se caracteriza entre otras cosas por [19]:

- Permitir el intercambio de diagramas y modelos UML con otras herramientas con representación del estándar industrial.

- Generar documentación en múltiples formatos como PDF, HTML y Microsoft Word.
- Generar código e ingeniería inversa en más de 10 lenguajes de programación.
- Soportar el desarrollo completo del ciclo de vida del software, de análisis a diseño, y de diseño a implementación, en el IDE que el usuario escoja.
- Realizar mapeo Objeto-Relacional.

Debido al análisis anterior, que demuestra su superioridad, y facilidades de uso, se decidió utilizar VP-UML como herramienta de modelado para la descripción de la arquitectura propuesta en el presente trabajo.

1.5 Arquitectura seleccionada

En epígrafes anteriores se hace referencia al Modelo Vista Controlador, reconocido como estilo arquitectónico por Taylor y Medvidovic [9], muy rara vez mencionado en las encuestas estilísticas usuales, y además considerado una micro-arquitectura por Robert Allen y David Garlan [30].

El MVC ha sido propio de las aplicaciones en Smalltalk[2], por lo menos desde 1992, antes que se generalizaran las arquitecturas en capas múltiples. En ocasiones se lo define más bien como un patrón de diseño o como práctica recurrente, y en estos términos es referido en el marco de la estrategia arquitectónica de Microsoft.

Por todas estas razones, para el desarrollo del presente trabajo se tomó al Modelo Vista Controlador como estilo y patrón ya que abarca ambas definiciones independientemente del nivel de abstracción donde se aplique. Dentro de los principales beneficios que brinda se encuentran:

- Menor acoplamiento, porque desacopla las vistas de los modelos y desacopla los modelos de la forma en que se muestran e ingresan los datos.
- Mayor cohesión debido a que cada elemento del patrón está altamente especializado en su tarea (la vista en mostrar datos al usuario, el controlador en las entradas y el modelo en su objetivo de negocio).

[2] Sistema informático que permite realizar tareas de computación mediante la interacción con un entorno de objetos virtuales.

- Permite que las vistas provean al desarrollador de una mayor flexibilidad y agilidad, porque se pueden crear múltiples vistas de un modelo, se pueden crear, añadir, modificar y eliminar nuevas vistas dinámicamente y las vistas pueden anidarse.
- Cambia el modo en que una vista responde al usuario sin cambiar su representación visual, sincronizar las vistas y puede concentrarse en diferentes aspectos del modelo.
- Permite una mayor facilidad para el desarrollo de clientes ricos en múltiples dispositivos y canales, más claridad de diseño, facilita el mantenimiento y la escalabilidad.

Entre los patrones de diseño seleccionados se encuentra el de Fachada pues con su uso se logra ocultar la complejidad de algunos componentes y aumenta la capacidad del sistema para evolucionar fácilmente, o sea, fortalece su modificabilidad. La Inyección de Dependencia es otro, y particularmente constituye una de las potencialidades del Framework Spring, en el que, como el servicio se identifica y localiza por medio de mecanismos no programáticos, o sea, externos al código (por un archivo XML), las dependencias con respecto a los servicios son explícitas y no están en el código, ganando con ello facilidad de prueba, mantenimiento y bajo acoplamiento. También se utilizará el patrón DAO por la transparencia en el acceso a los datos, y porque permitiría, de ser necesario en un futuro, la migración más fácil a un gestor de base de datos diferente, y por tanto, a una implementación de la capa de acceso a datos diferente, también por su aporte a la reducción de la complejidad del código de los objetos de negocio y la centralización de todos los accesos a datos en una capa independiente.

Como se ha planteado con anterioridad, los patrones seleccionados determinan un Bajo Acoplamiento en el sistema, pues los componentes no se afectan por cambios en otros, son fáciles de entender por separado y fáciles de reutilizar, además, como se describió en el estudio del arte, el grado de acoplamiento no puede verse aislado de otros patrones como la Alta Cohesión, que mejora la claridad y facilidad con que se entiende el diseño, simplifica el mantenimiento y las mejoras de funcionalidad, y soporta mayor capacidad de reutilización.

Además se decide utilizar las Hojas de Estilo en Cascada (Cascading Style Sheets, CSS), lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML que permite a los desarrolladores Web controlar el estilo y el formato de múltiples páginas Web al mismo tiempo, potenciando además la modificabilidad y mantenibilidad ya que de no utilizar estos estilos sería necesario replicar el código del formato cuantas veces fuera necesario y en todas las páginas en las que se necesitara el mismo formato, mientras que con el uso de los estilos CSS cualquier cambio en el estilo marcado para un elemento en la CSS afectará a todas las páginas vinculadas a esa CSS en las

que aparezca ese elemento. Además su uso tiene otras mejoras como por ejemplo, aplicar al documento formato de modo mucho más exacto y la definición de atributos en las páginas utilizando muchas más unidades como pulgadas (in), puntos (pt) y centímetros (cm), además de píxeles (px) y porcentaje (%).

También es importante señalar el uso de Java Script, que es un lenguaje de programación interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas Web; este lenguaje se utilizará por las potencialidades que ofrece ya que mediante su código se pueden implementar funciones que se ejecuten del lado del cliente aumentando el rendimiento de la aplicación pues no es necesario que se ejecuten en el servidor y sus resultados regresen al cliente, ahorrando tiempo en el proceso, este es el caso por ejemplo de validaciones ligeras.

1.5.1 Tecnologías asociadas

- PostgreSQL 8.1
- Java 6.0
    - Eclipse 3.1
    - Spring 2.5
        - Acegi Security 1.0.7
    - Hibernate 3.0
    - Servlet 2.5
    - Java Server Pages 2.1
- Apache Tomcat 6.0
- Subversion 1.5

1.6 Métodos de evaluación de la arquitectura

El desarrollo de formas para relacionar atributos de calidad de un sistema a su arquitectura provee una base para la toma de decisiones objetivas sobre acuerdos de diseño y permite a los ingenieros realizar predicciones razonablemente exactas sobre los atributos del sistema que pueden representar una debilidad o no. El objetivo de fondo es lograr la habilidad de evaluar y llegar a acuerdos entre múltiples atributos de calidad para alcanzar un mejor sistema de forma global.

Según Barbacci et al. [20] la calidad de software se define como el grado en el cual el software posee una combinación deseada de atributos. Tales atributos son requerimientos adicionales del sistema, que hacen referencia a características que éste debe satisfacer, diferentes a los requerimientos funcionales. Estas características o atributos se conocen con el nombre de atributos de calidad, los cuales se definen como las propiedades de un servicio que presta el sistema a sus usuarios.

Teniendo en cuenta la estrecha relación que existe entre la arquitectura del software y los atributos de calidad, como expresan Bass et al. [21] en la afirmación: **"cada decisión incorporada en una arquitectura de software puede afectar potencialmente los atributos de calidad",** es un punto de vital importancia la evaluación de la arquitectura para garantizar la calidad del sistema y estar en condiciones de tomar decisiones acertadas sobre ella.

El primer paso para la evaluación de una arquitectura es conocer qué se quiere evaluar. De esta forma, es posible establecer la base para la evaluación, puesto que la intención es saber qué se puede evaluar y qué no. Luego el interés se centra en determinar el momento propicio para efectuar la evaluación de una arquitectura, en este sentido es posible hallar muchos criterios en la bibliografía consultada pero de forma general es certero plantear que es posible realizarla en cualquier momento, pero en particular existen dos variaciones útiles: temprana y tardía. La evaluación temprana no tiene que esperar a que la arquitectura esté totalmente especificada por lo que puede ser utilizada en cualquier etapa del proceso de creación de la arquitectura mientras que la evaluación tardía toma lugar no solo cuando la arquitectura está terminada, también cuando la implementación está completa.

En fin, una evaluación debe realizarse cuando hay suficiente de la arquitectura como para justificarlo, un buen criterio a seguir puede ser realizar una evaluación cuando el equipo de desarrollo empieza a tomar decisiones que dependen de la arquitectura y el costo de deshacerlas es mayor que el costo de realizar la evaluación.

Seguidamente corresponde la evaluación de la arquitectura, Bosch [22] afirma que se pretenden medir propiedades del sistema en base a especificaciones abstractas, como por ejemplo los diseños arquitectónicos. Por ello, la intención es más bien **la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos**. Las mediciones que se realizan sobre una AS pueden tener distintos objetivos, dependiendo de la situación en la que se encuentre el arquitecto y la aplicabilidad de las técnicas que emplea. Los objetivos que menciona son tres: *cualitativos*, *cuantitativos* y *máximos y mínimos teóricos*.

La medición *cualitativa* se aplica para la comparación entre arquitecturas candidatas y tiene relación con la intención de saber la opción que se adapta mejor a cierto atributo de calidad. Este tipo de medición brinda respuestas afirmativas o negativas, sin mayor nivel de detalle. La medición *cuantitativa* busca la obtención de valores que permitan tomar decisiones en cuanto a los atributos de calidad de una AS. El esquema general es la comparación con márgenes establecidos, como lo es el caso de los requerimientos de desempeño, para establecer el grado de cumplimiento de una arquitectura candidata, o tomar decisiones sobre ella.

Este enfoque permite establecer comparaciones, pero se ve limitado en tanto no se conozcan los valores teóricos máximos y mínimos de las mediciones con las que se realiza la comparación. Por último, la *medición de máximo y mínimo teórico* contempla los valores teóricos para efectos de la comparación de la medición con los atributos de calidad especificados. El conocimiento de los valores máximos o mínimos permite el establecimiento claro del grado de cumplimiento de los atributos de calidad.

De forma general el planteamiento anterior presenta los objetivos para efectos de la medición de los atributos de calidad. Sin embargo, en ocasiones, la evaluación de una AS no produce valores numéricos que permiten la toma de decisiones de manera directa. Ante la posibilidad de efectuar evaluaciones a cualquier nivel del proceso de diseño, con distintos niveles de especificación, Kazman et al. [23] proponen un esquema general en relación a la evaluación de una arquitectura con respecto a sus atributos de calidad. En este sentido, Kazman et al. afirman que de la evaluación de una arquitectura no se obtienen respuestas del tipo “si - no”, “bueno – malo” o “6.75 de 10”, sino que explica **cuáles son los puntos de riesgo del diseño evaluado**.

Una de las diferencias principales entre los planteamientos de Bosch [22] y Kazman et al. [23] es el enfoque que utilizan para efectos de la evaluación. El método de diseño de arquitecturas planteado por Bosch tiene como principal característica la evaluación explícita de los atributos de calidad de la

arquitectura durante el proceso de diseño de la misma. El autor afirma que el enfoque tradicional en la industria de software es el de implementar el sistema y luego establecer valores para los atributos de calidad del mismo. Este enfoque, según su experiencia, tiene la desventaja de que se destina gran cantidad de recursos y esfuerzo en el desarrollo de un sistema que no satisface los requerimientos de calidad. En este sentido, Bosch plantea las técnicas de evaluación: basada en escenarios, en simulación, en modelos matemáticos y en experiencia.

Por su parte, Kazman et al. [23] proponen que resulta de poco interés la caracterización o medición de atributos de calidad en las fases tempranas del proceso de diseño, dado que estos parámetros son, por lo general, dependientes de la implementación. Su enfoque se orienta hacia la mitigación de riesgos, ubicando dónde un atributo de calidad de interés se ve afectado por decisiones arquitectónicas. En su estudio, Kazman et al. presentan tres métodos de evaluación de arquitecturas de software, que son el Architecture Trade-off Análisis Method (ATAM), el Software Architecture Analysis Method (SAAM) y el Active Intermediate Designs Review (ARID).

Ambos planteamientos indican la importancia de la especificación exhaustiva de los atributos de calidad como base para efectos de la evaluación de una AS. Descripciones tales como *"el sistema debe ser robusto"* o *"el sistema debe exhibir un desempeño aceptable"* resultan ambiguas, puesto que lo que se entiende de ellos puede ser diferente para los distintos involucrados con el sistema. El punto es entonces definir los atributos de calidad en función de sus metas y su contexto, y no como cantidades absolutas, según Kazman et al.

De los planteamientos de evaluación establecidos por Bosch y Kazman et al. , se tiene que la evaluación de las arquitecturas de software puede ser realizada mediante el uso de diversas técnicas y métodos, de los cuales se profundiza en los métodos, puesto que el criterio de la evaluación cualitativa que pone al relieve los puntos débiles y las fortalezas de la arquitectura, es el que se considera más acertado para la evaluación de una AS, o sea, se coincide con el planteamiento de Kazman.

Software Architecture Analysis Method (SAAM) fue el primer método de evaluación basado en escenarios que surgió, centra su atención en la modificabilidad, y puede ser usado para evaluar una arquitectura o evaluar y comparar varias.

Este método se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro. Como entrada principal es necesaria

alguna forma de descripción de la arquitectura a ser evaluada. De acuerdo con Kazman et al. [23], las salidas de la evaluación del método SAAM son las siguientes:

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

Con la aplicación de este método, si el objetivo de la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad. Para el caso en el que se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones.

Architecture Trade-off Analysis Method (ATAM) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. El nombre del método 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; o sea, los tipos de *acuerdos (trade-off)* que se establecen entre ellos.

El método se concentra en la identificación de los estilos arquitectónicos ya que 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.

El ATAM confía fuertemente en la identificación de los puntos de sensibilidad y los riesgos de la arquitectura, confía en que los puntos de sensibilidad no solo descubrirán potenciales problemas (riesgos), sino también fortalezas de la arquitectura. Un aspecto del método es que no solo encuentra riesgos, sino que también encuentra no riesgos, ya que tanto los no riesgos como los riegos están relacionados con las respuestas arquitectónicas provenientes de las decisiones arquitectónicas. Pero con los no riesgos se puede decir que las decisiones arquitectónicas son apropiadas, es decir, el diseño de la arquitectura satisface los requerimientos de los atributos de calidad. Interesa registrar esta información, como se registra información sobre los riesgos, porque si esta decisión arquitectónica alguna vez cambia, es necesario examinar si el no riesgo se transforma en un riesgo.

Un punto de sensibilidad es una propiedad de uno o más componentes (y/o relaciones entre componentes) que son críticos para lograr una respuesta de un atributo de calidad particular. ATAM, sugiere que cada punto de sensibilidad sea clasificado como riesgo o no riesgo, dependiendo si la respuesta deseada es lograda o no.

Al finalizar el desarrollo del método se obtienen los siguientes resultados:

- El documento de propuestas arquitectónicas.
- El conjunto de escenarios priorizados.
- El conjunto de preguntas basadas en los atributos.
- El árbol de utilidad.
- Los riesgos descubiertos.
- Los no riesgos documentados.
- Los puntos de sensibilidad y de trade-off encontrados.

Active Reviews for Intermediate Designs (ARID) de acuerdo con Kazman et al. [23] el método ARID es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura. Según los autores, ARID es un híbrido entre Active Design Review (ADR) y Architecture Trade-Off Method (ATAM), descrito anteriormente.

Tomando en cuenta la importancia de la evaluación de la AS y profundidad de las técnicas y métodos mencionados en este epígrafe, se certificarán las decisiones arquitectónicas de la descripción propuesta mediante el método de evaluación Architecture Tradeoff Analysis Method (ATAM) porque demostró ser el más acorde a las necesidades del trabajo, el más documentado y con mejores explicaciones en los casos de estudio; además, se respaldarán los resultados mediante la implementacion de la aplicación Web para la captura de la información requerida por el CNBA, como aplicación real de las disposiciones del diseño de la arquitectura.

1.7 Atributos de calidad en la arquitectura de software

Conociendo la definición de atributo de calidad y la trascendencia que tienen las decisiones arquitectónicas sobre ellos, es importante determinar cuáles atributos de calidad deben alcanzarse en la descripción arquitectónica y que serán el centro de la evaluación posterior para valorar en qué grado se logró el cumplimiento de ellos en la propuesta arquitectónica. En este sentido es importante resaltar que no puede lograrse la satisfacción de ciertos atributos de calidad de manera aislada. Lograr un atributo de calidad puede tener efectos positivos o negativos sobre otros atributos que, de alguna manera, también se desean alcanzar.

Para determinar los atributos de calidad que debe lograr la AS con la tecnología J2EE y el Framework Spring fue necesario guiarse por los objetivos y restricciones arquitectónicas que el negocio impone y por los modelos de calidad, es decir, las diferentes formas en que los atributos de calidad se pueden organizar y descomponer; ya que Pressman [24] indica que los factores que afectan a la calidad del software no cambian, por lo que resulta útil el estudio de los modelos de calidad que han sido propuestos en este sentido desde los años 70. Dado que los factores de calidad presentados para ese entonces siguen siendo válidos, se estudiarán algunos de los modelos más importantes propuestos hasta ahora: ISO/IEC 9126 (1991) e ISO/IEC 9126 adaptado para arquitecturas de software, propuesto por Losavio et al [25].

El estándar **ISO/IEC 9126** ha sido desarrollado en un intento de identificar los atributos clave de calidad para un producto de software [24]. Este estándar es una simplificación del Modelo de McCall [25], e identifica seis características básicas de calidad que pueden estar presentes en cualquier producto de software. El estándar provee una descomposición de las características en subcaracterísticas, que se muestran en la siguiente tabla.

Tabla 2. Atributos de calidad - Modelo ISO/IEC 9126.

Característica	Subcaracterística
Funcionalidad	Adecuación Exactitud Interoperabilidad Seguridad
Confiabilidad	Madurez Tolerancia a fallas Recuperabilidad
Usabilidad	Entendibilidad Capacidad de aprendizaje Operabilidad

Eficiencia	Comportamiento en tiempo Comportamiento de recursos
Mantenibilidad	Analizabilidad Modificabilidad Estabilidad Capacidad de pruebas
Portabilidad	Adaptabilidad Instalabilidad Reemplazabilidad

Es interesante destacar que los factores de calidad que contempla el estándar ISO/IEC 9126 no son necesariamente usados para mediciones directas [24], pero proveen una valiosa base para medidas indirectas, y una excelente lista para determinar la calidad de un sistema.

ISO/IEC 9126 adaptado para arquitecturas de software: Losavio et al. [25] proponen una adaptación del modelo ISO/IEC 9126 de calidad de software para efectos de la evaluación de AS. El modelo se basa en los atributos de calidad que se relacionan directamente con la arquitectura: funcionalidad, confiabilidad, eficiencia, mantenibilidad y portabilidad. Los autores plantean que la característica usabilidad propuesta por el modelo ISO/IEC 9126 puede ser refinada para obtener atributos que se relacionan con los componentes de la interfaz con el usuario. Dado que estos componentes son independientes de la arquitectura, no son considerados en la adaptación del modelo.
La tabla que se muestra a continuación presenta los atributos de calidad planteados por Losavio et al. [25], que poseen subcaracterísticas asociadas con elementos de tipo arquitectónico.

Tabla 3. Atributos de calidad - Modelo ISO/IEC 9126 adaptado para arquitectura de software.

Característica	**Subcaracterística**	**Elementos de tipo arquitectónico**
Funcionalidad	Adecuación	Refinamiento de los diagramas de secuencia
	Exactitud	Identificación de los componentes con las funciones responsables de los cálculos
	Interoperabilidad	Identificación de conectores de comunicación con sistemas externos
	Seguridad	Mecanismos o dispositivos que realizan explícitamente la tarea
Confiabilidad	Tolerancia a fallas	Existencia de mecanismos o dispositivos de software para manejar excepciones
	Recuperabilidad	Existencia de mecanismos o dispositivos de software para reestablecer el nivel de

		desempeño y recuperar datos
Eficiencia	Desempeño	Componentes involucrados en un flujo de ejecución para una funcionalidad
	Utilización de recursos	Relación de los componentes en términos de espacio y tiempo
Mantenibilidad	Acoplamiento	Interacciones entre componentes
	Modularidad	Número de componentes que dependen de un componente
Portabilidad	Adaptabilidad	Presencia de mecanismos de adaptación
	Instalabilidad	Presencia de mecanismos de instalación
	Coexistencia	Presencia de mecanismos que faciliten la coexistencia
	Reemplazabilidad	Lista de componentes reemplazables para cada componente

Debido a la actualidad del modelo, a las mejoras que formula para efectos de la evaluación de la AS y a la correspondencia de los atributos que propone con las necesidades del sistema, se determinaron los atributos que propone este modelo como los atributos de calidad que deben alcanzarse con la arquitectura propuesta, y serán los mismos en base a los cuales se evaluará el diseño arquitectónico.

1.8 Conclusiones

Luego de terminar el estudio minucioso de los temas relacionados con la AS se ha concluido la primera de las tareas de la investigación y se obtuvieron los conocimimientos que servirán de base para el desarrollo del trabajo, en este capítulo además, quedaron definidos aspectos tan importantes para la descripción de la arquitectura como son: el estilo y patrón arquitectónico que se utilizará, los patrones de diseño; el lenguaje de modelado con el cual quedará documentada, en este caso se escogió UML, que a pesar de no ser un ADL permite muy bien describir la arquitectura; la herramienta CASE para realizar el modelado y los procedimientos para la evaluación de la arquitectura.

Capítulo 2: Descripción de la arquitectura de software

En este capítulo encontrará íntegramente la descripción de la AS con la tecnología J2EE y el Framework Spring que podría ser utilizada en el desarrollo de aplicaciones Web que utilicen estas tecnologías, también se incluyen decisiones importantes acerca del estilo de codificación a seguir por los desarrolladores y los patrones de diseño propuestos para elevar la capacidad del diseño que se traduce en calidad del software.

2.1 Representación arquitectónica

Este documento presenta la arquitectura como una serie de vistas: la vista de casos de uso, la vista lógica, la vista de implementación y la vista de despliegue. Esas vistas están representadas mediante modelos de VP-UML y usa UML como lenguaje de modelado.

2.2 Objetivos y restricciones arquitectónicas

Las restricciones del sistema que tienen consecuencias relevantes en la arquitectura están dadas porque:

1. Se deben recopilar los datos de varios Entes que se encuentran distribuidos geográficamente por todo el país.
2. Los datos que suministran la mayoría de los Entes están en copia dura, o sea, documentos impresos; y los Entes que manejan la información de forma digital la almacenan con su propio formato, es decir, no está estandarizado el formato de los datos.
3. Los datos con los cuales trabaja el CNBA son de vital importancia y de carácter estratégico para la seguridad alimentaria del país por lo cual es necesario garantizar su seguridad.
4. Debe existir un único repositorio central para almacenar todos los datos recopilados.
5. La cantidad de información que se maneja es muy grande.

Partiendo de las limitantes planteadas anteriormente, se brinda como solución informática una Aplicación Web para los Entes que tienen los datos a suministrar en copia dura, es decir, no están digitalizados. Puesto que para ingresar los datos manualmente la mejor forma de hacerlo es a través de un cliente ligero como lo es un Navegador (Internet Explorer, Opera o FireFox) que acceda a la

aplicación Web aprovechando las facilidades de despliegue que brindan y los pocos requerimientos de hardware que demandan.

2.3 Vistas arquitectónicas

Siguiendo la metodología RUP tal y como se expuso en el Capítulo1, a continuación se desarrolla la descripción de la arquitectura a través de las 4+1 vistas propuestas por Kruchten, es importante destacar que una de las vistas propuestas por este modelo, la Vista de Procesos, no es necesaria ya que el sistema no maneja hilos de procesos concurrentes.

2.3.1 Vista de Casos de Uso

Al analizar los casos de uso del sistema, se determinaron como casos de uso arquitectónicamente significativos:

Caso de uso Autenticar Usuario: Se inicia cuando el Usuario introduce los datos para autenticarse en el sistema. El sistema verifica que todos los datos necesarios hayan sido insertados y de forma correcta, dando acceso al sistema según nivel de acceso, finalizando el caso de uso.

Caso de uso Gestionar Usuarios: Se inicia cuando el Administrador indica registrar, modificar o eliminar un usuario del sistema. El sistema muestra las interfaces correspondientes a cada una de estas opciones. El administrador realiza las acciones necesarias para que se registre, modifique o elimine un usuario, finalizando el caso de uso.

Caso de uso Registrar Producción por Rubro: Inicia cuando el Responsable de Producción Nacional escoge Registrar Información por Rubro. El sistema muestra interfaz para registrar datos de la producción, dando la posibilidad de registrar la producción de cada rubro, real o estimada, ya sea a nivel nacional, por estado o municipio.

Caso de uso Registrar Permisología: Inicia cuando el Responsable Permisología escoge Registrar Información de permisología. El sistema muestra interfaz para registrar datos de la permisología. El caso de uso termina cuando el Responsable Permisología registra todos los permisos que han sido solicitados.

Caso de uso Registrar CSP por Rubro: Inicia cuando el Responsable CSP selecciona la opción de Registrar CSP por Rubro, donde el mismo introduce los datos correspondientes para realizar el registro y el sistema a su vez realiza un grupo de acciones requeridas por el Responsable CSP para almacenar la información de la manera deseada. El caso de Uso finaliza cuando el Responsable CSP completa el

registro y el sistema ha realizado todas las validaciones pertinentes o cuando el Responsable CSP escoge otra opción del sistema.

Caso de uso Registrar Importación: Inicia cuando el Responsable Entes Privados escoge Registrar Información de importación. El sistema muestra interfaz para registrar datos de la importación. El CU termina cuando se registran todas las importaciones deseadas.

Caso de uso Registrar Exportación: Inicia cuando el Responsable Entes Privados escoge Registrar Información de exportación. El sistema muestra interfaz para registrar datos de la exportación. El CU termina cuando se registran todas las exportaciones deseadas.

Caso de uso Registrar Inventario por Rubro: Inicia cuando el Responsable Entes Privados selecciona la opción de Registrar Inventario por Rubro, donde el mismo introduce los datos correspondientes para realizar el registro y el sistema a su vez realiza un grupo de acciones requeridas por el cliente para almacenar la información de la manera deseada. El caso de uso finaliza cuando el Responsable Entes Privados completa y confirma el registro y el sistema realiza todas las validaciones necesarias, o cuando el Responsable Entes Privados escoge otra opción del sistema.

La realización de estos casos de usos así como documentación ampliada de sus especificaciones se puede encontrar en [1] y [26].

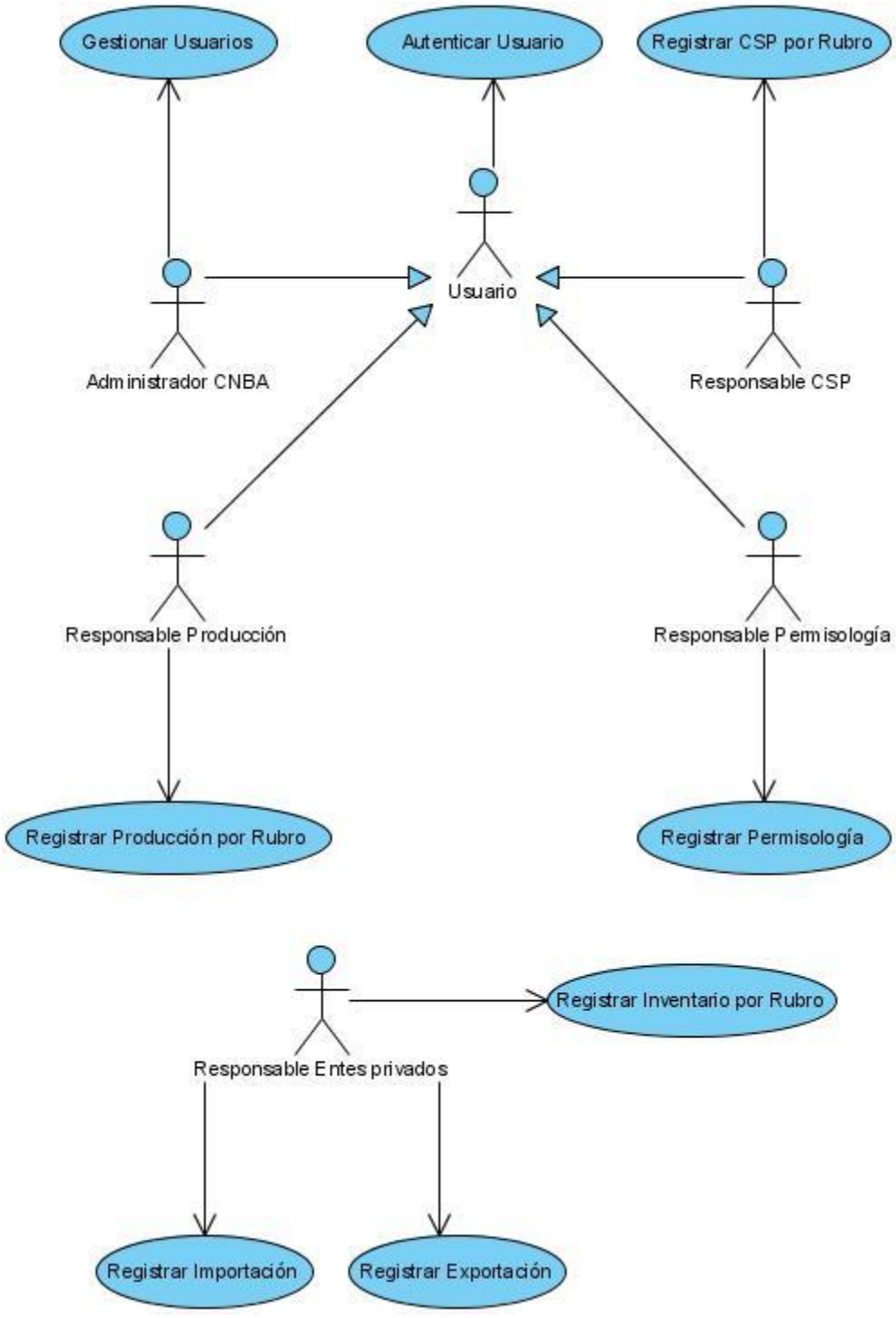


Figura 2. Vista de Casos de Uso.

2.3.2 Vista Lógica

En la ***Aplicación WEB*** los principales paquetes son:

Figura 3. Principales paquetes de la Aplicación Web.

minnpalCnba.web: contiene las clases y formularios para la aplicación Web agrupados en seis paquetes para cada uno de los seis módulos (Producción Nacional, Importaciones y Exportaciones, Permisología, Consumo Social Priorizado, Administración e Inventario) y un séptimo paquete con las funcionalidades comunes a todos los módulos.

minppalCnba.web.<modulo>: representa a un módulo determinado, cuyo nombre reemplaza la palabra "<modulo>". Contiene las **vistas**, que internamente tienen a los "**tiles**" para definir la estructura de las páginas (*Layout*), los *Java Server Pages* en el subpaquete "**jsp**" y la **internacionalización** que contiene los mensajes de texto según el idioma. Además de las vistas están los **controladores** y los **servicios** que dan respuesta a la lógica de negocio.

minppalCnba.web.comun: contiene las vistas, controladores y servicios comunes a todos los módulos. Su estructura interna es similar a la del paquete **minppalCnba.web.<modulo>**.

En la ***Capa de Acceso a Datos*** los principales paquetes son:

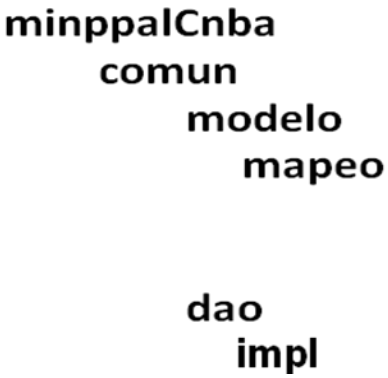


Figura 4. Principales paquetes de la Capa de Acceso a Datos.

minppalCnba.comun: este paquete contiene las clases que responden a las entidades de negocio y a los DAO que se van a encargar de almacenar y recuperar las entidades de la Base de Datos.

minppalCnba.comun.modelo: contiene las clases que representan los datos o entidades de negocio. El subpaquete "**mapeo**" contiene ficheros XML de configuración que son necesarios para que la herramienta Hibernate establezca el mapeo de los objetos a la base de datos relacional.

minppalCnba.comun.dao: contiene las interfaces para los DAO y las implementaciones de esas interfaces se encuentran en el subpaquete "**impl**".

Elementos del modelo arquitectónicamente significantes:

- **Modelo Vista Controlador**

El diseño de la aplicación Web se realizó utilizando el patrón MVC como quedó planteado en el capítulo anterior, patrón que determina un alto grado de cohesión y se puede apreciar en la siguiente figura.

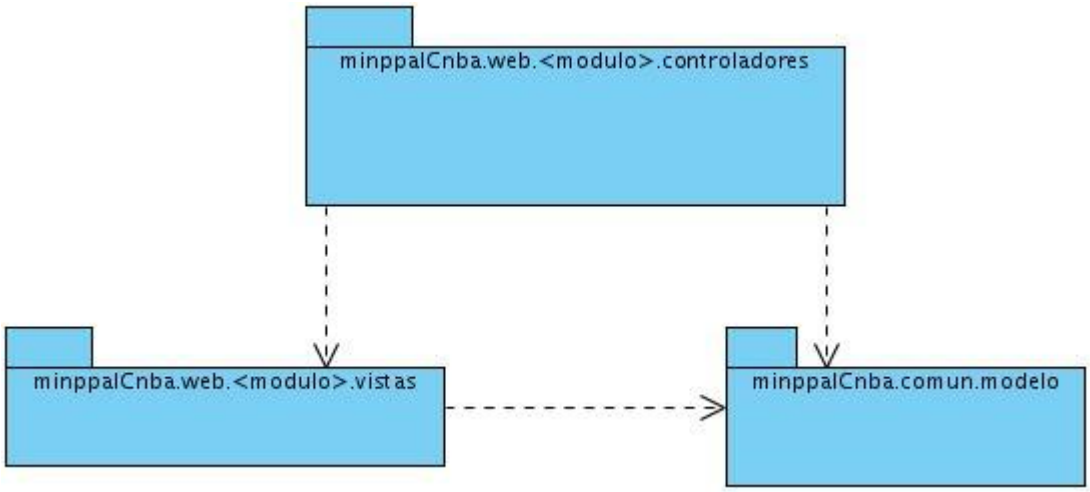


Figura 5. Vista Lógica - Patrón MVC.

- **Servicios de Fachada**

Se diseñaron varios servicios para la implementación de las reglas de negocio y para ocultar la complejidad de determinadas clases (Patrón Fachada) a aquellas que solicitan los servicios. Como se ve en la figura que aparece a continuación las clases que se agrupan en el paquete **minppalCnba.web.<modulo>.servicios** sirven de fachada entre las clases de los paquetes **minppalCnba.web.<modulo>.controladores** y **minppalCnba.web.<modulo>.servicios.impl.**

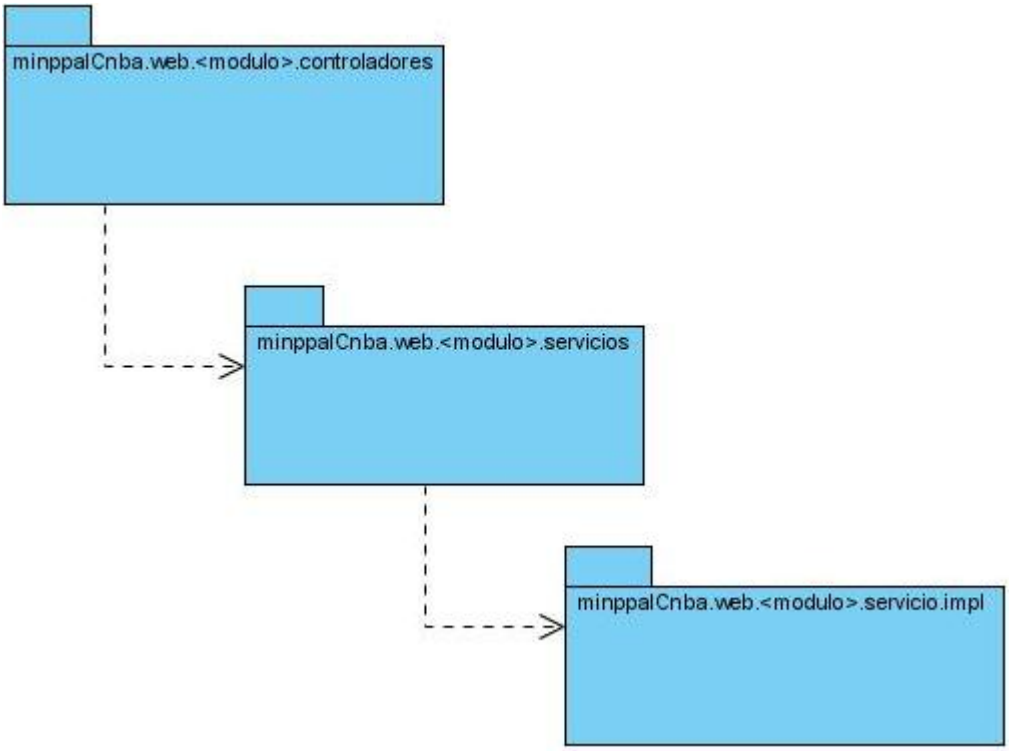


Figura 6. Vista Lógica - Patrón Fachada en la Aplicación Web.

También es posible apreciar el uso de este patrón en la Capa de Acceso a Datos, donde el paquete **minppalCnba.comun.dao** agrupa las interfaces que sirven de fachada a la relación entre las clases agrupadas en **minppalCnba.web.<modulo>.servicios.impl** y las contenidas en **minppalCnba.comun.dao.impl,** en este último se ubican las clases que contienen la complejidad de la implementación de la lógica del acceso a los datos.

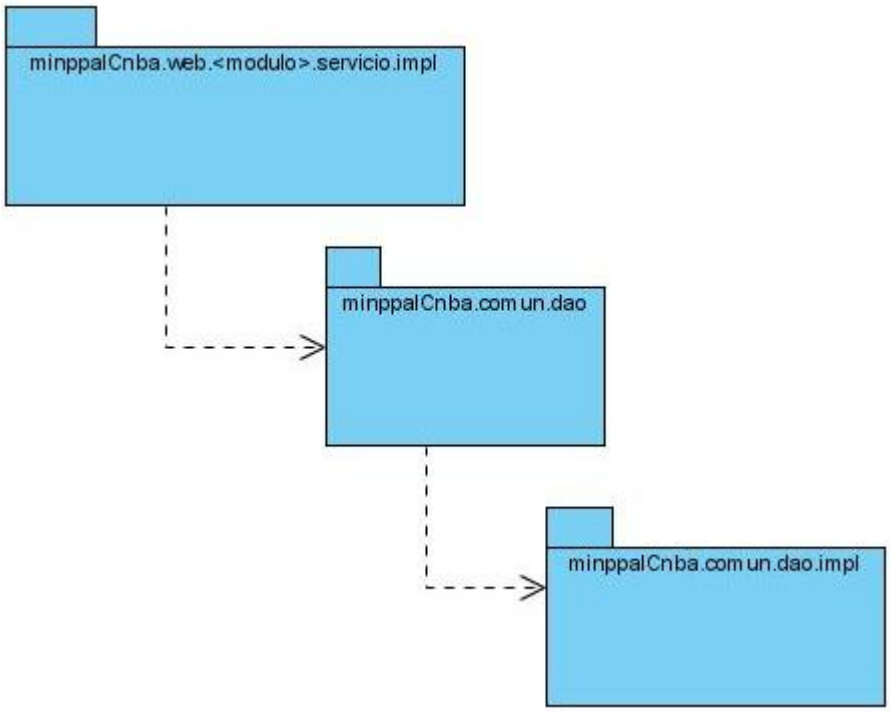


Figura 7. Vista Lógica - Patrón Fachada en la Capa de Acceso a Datos.

- **Bajo Acoplamiento e Inyección de Dependencia**

Se logró un bajo acoplamiento de clases al utilizar el patrón Inyección de Dependencia. Un conjunto de clases importan determinadas interfaces y no crean directamente instancias de las implementaciones de dichas interfaces. Las instancias de dichas interfaces se configuran en un fichero XML logrando un bajo acoplamiento o desacoplamiento sin necesidad de modificar el código fuente. En la figura posterior se representa como las clases agrupadas en el paquete **minppalCnba.web.<modulo>.controladores** importan las interfaces contenidas en **minppalCnba.web.<modulo>.servicios**, sin embargo, ellas no construyen directamente las instancias en su implementación, ya que se inyecta la dependencia a través de un fichero de configuración XML así como la relación que tienen con las clases que implementan estas interfaces, que como se plantea en la nota, no es directa pues las clases controladoras no conocen quienes implementan los servicios que ellas utilizan, todo esto determina el bajo acoplamiento entre ellas y la potencialidad de la aplicación ante modificaciones posteriores. Lo mismo sucede entre las clases de los paquetes **minppalCnba.web.<modulo>.servicios, minppalCnba.comun.dao y minppalCnba.comun.dao.impl.**

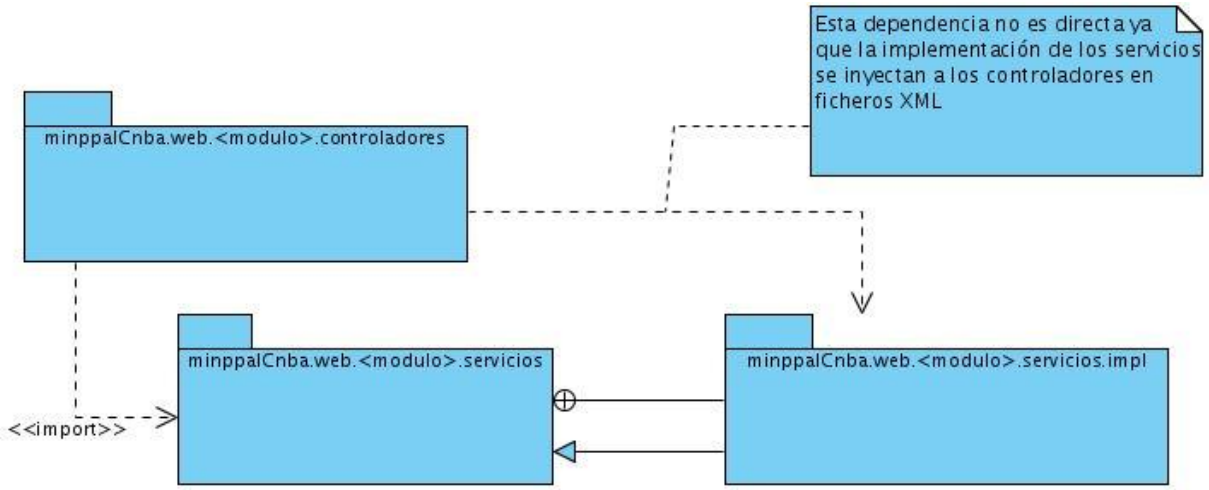


Figura 8. Vista Lógica - Patrón Inyección de dependencia y Bajo Acoplamiento en la Aplicación Web.

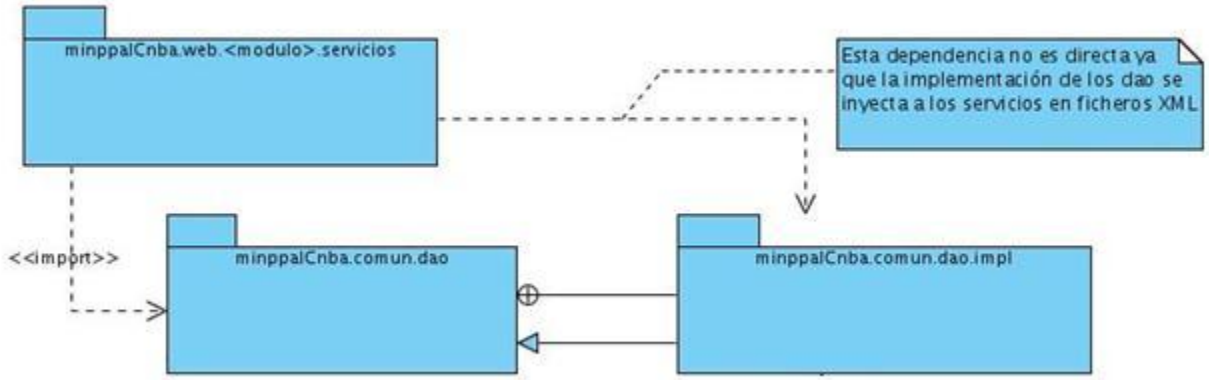


Figura 9. Vista Lógica - Patrón Inyección de dependencia y Bajo Acoplamiento en la Capa de Acceso a Datos.

- **DAO**

Se diseñó un componente para el acceso a los datos aplicando el patrón Data Access Object que le permite al sistema gran escalabilidad pues desacopla la lógica de negocio de la lógica de acceso a datos, de manera que se pueda cambiar la fuente de datos fácilmente, y además reduce la complejidad del código de los objetos de negocio. La imagen siguiente provee una visión de su aplicación, donde las clases de la lógica del negocio se encuentran agrupadas en el paquete **minppalCnba.comun.modelo,** dentro de este se encuentra el paquete **minppalCnba.comun.modelo.mapeo** que contiene todos los ficheros de mapeo del Framework

Hibernate, ficheros donde se mapean a tablas de la base de datos las clases que se van a persistir, o sea, las clases del negocio; separadamente se halla el paquete **minppalCnba.comun.dao,** que agrupa a todas las clases encargadas de la lógica del acceso a los datos**.**

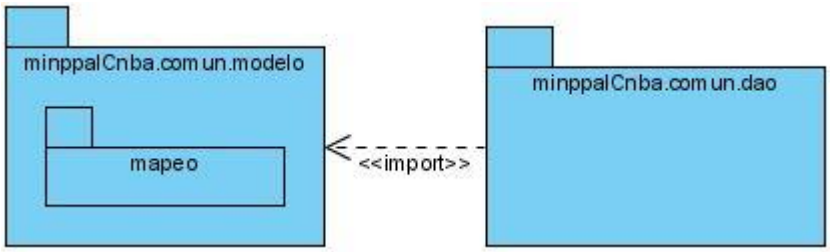


Figura 10. Vista Lógica - Patrón DAO.

Visión general de la arquitectura

A continuación se presenta una vista general de los paquetes de la aplicación, estos responden al patrón **Modelo Vista Controlador**, donde se integran todos los paquetes y sus relaciones, es importante resaltar que las relaciones que no se muestran en la figura correspondiente al paquete **minppalCnba.web.comun** son las mismas que se establecen entre sus homólogos en el paquete **minppalCnba.web.<modulo>** como se expresa en la nota, omitidas para lograr un mejor entendimiento de la figura.

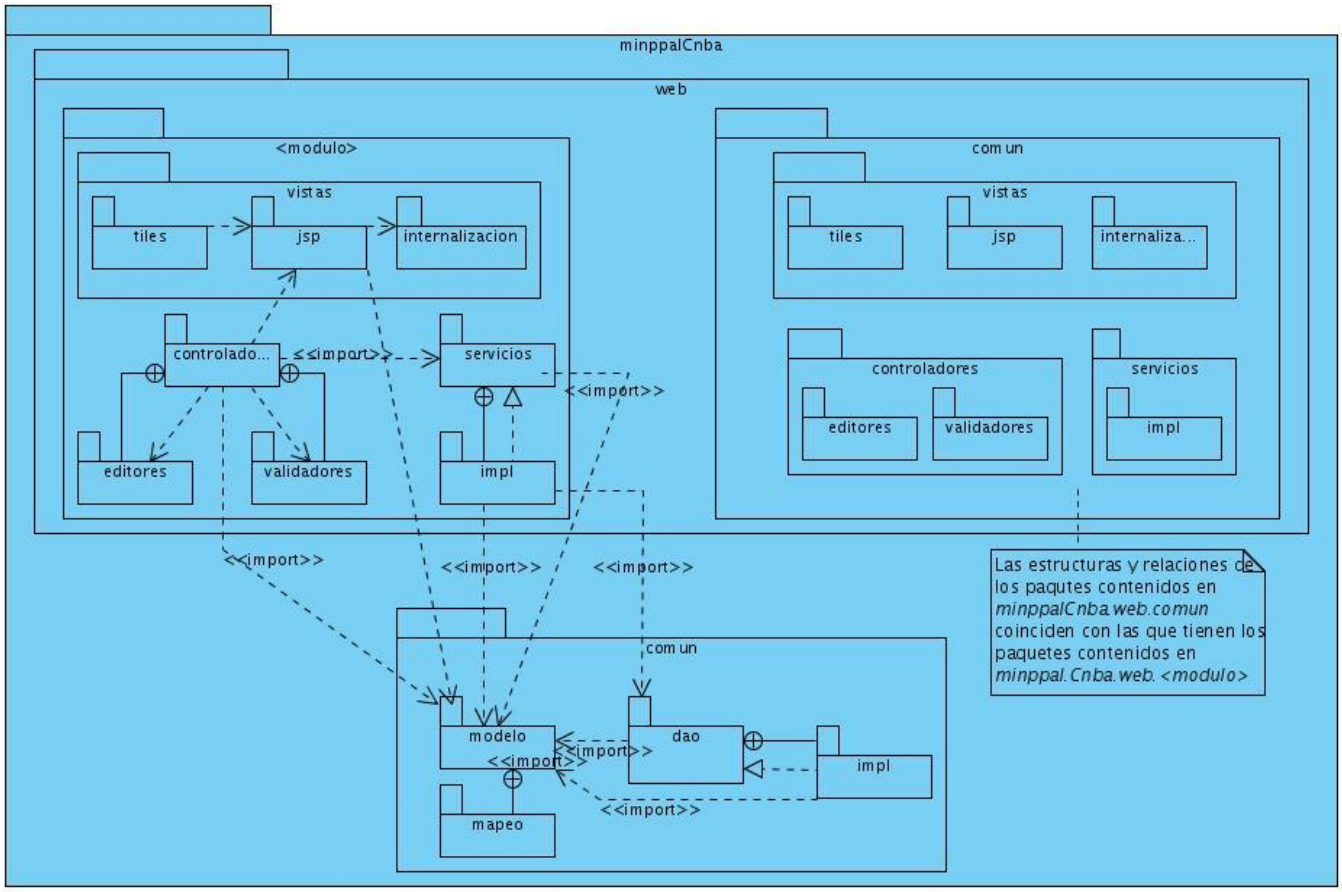


Figura 11. Vista Lógica.

2.3.3 Vista de Despliegue

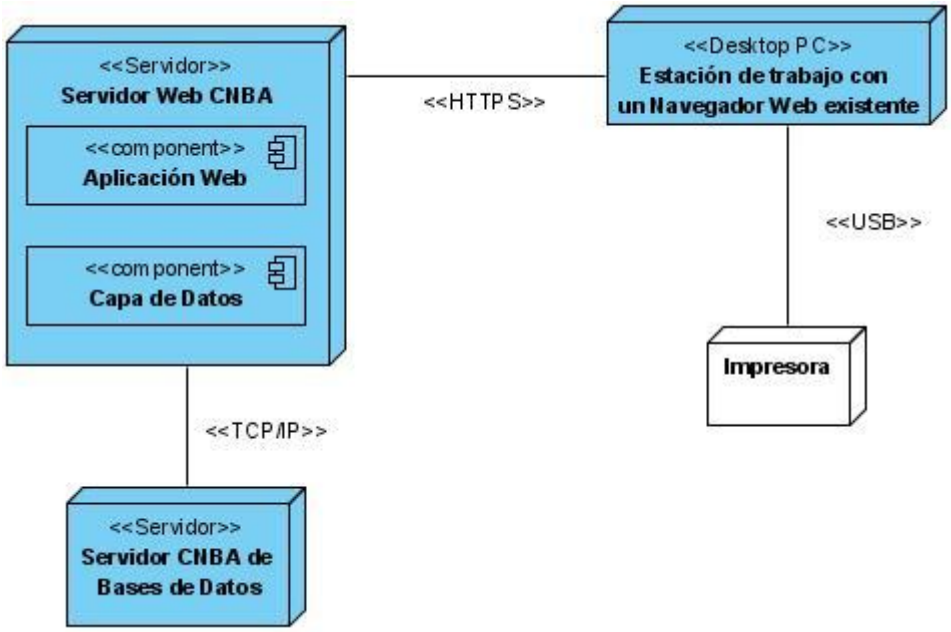


Figura 12. Vista de Despliegue.

Servidor CNBA de Base de Datos

Se refiere al servidor que radica en el CNBA donde van a estar centralizados los datos recopilados de los envíos de las diferentes estaciones de trabajo.

Servidor Web CNBA

Servidor Web que publica la aplicación Web para que diferentes actores transcriban los datos que necesita el CNBA, o sea, es la máquina donde se mantiene corriendo la aplicación Web a la que se conectan los clientes por medio de sus estaciones de trabajo para acceder a los datos o enviarlos, esta conexión es segura mediante el protocolo <<HTTPS>>.

Estaciones de trabajo con un navegador Web existente

Se refiere a las estaciones de trabajo que el usuario utilizará para acceder a la aplicación Web y transcribir sus datos a través de un navegador Web (Internet Explorer, Opera o FireFox) que debe estar previamente instalado en la máquina cliente.

Impresora

Este dispositivo estará a disposición de la estación de trabajo con la aplicación cliente para visualizar en una hoja impresa los datos enviados al CNBA, conectada a la PC cliente mediante un puerto <<USB>>.

2.3.4 Vista de Implementación

En esta sección se presenta la interacción de los principales componentes que fueron asignados a los nodos físicos en el despliegue.

Figura 13. Vista de Implementación.

Capa de datos

Contiene las clases que representan las entidades del negocio y los DAO. Para cada DAO existe una interfaz a la cual se le dio una implementación dentro del componente para dar persistencia a determinada entidad.

Aplicación Web

Se refiere a la implementación de todas las clases para cada uno de los seis módulos que conforman la aplicación Web para la captura de información requerida por el CNBA identificados en la fase de análisis y diseño. Se relaciona con el componente de la Capa de Datos utilizando las interfaces que este exporta.

2.4 Lineamientos de diseño e implementación

En esta sección se incluyen todas las restricciones impuestas por el arquitecto de software para el correcto desarrollo y evolución de la solución, tanto desde el punto de vista de diseño, como de la implementación.

2.4.1 Patrones de diseño propuestos

Como quedó manifestado en la Fundamentación Teórica y como pudo apreciarse en la Vista Lógica de la AS propuesta, los patrones de diseño sugeridos fueron:

- Inyección de Dependencia.
- Data Access Object.
- Fachada.
- Bajo Acoplamiento.
- Alta Cohesión.

2.4.2 Estándares de codificación

Como se ha expuesto anteriormente, el arquitecto de software tiene la difícil tarea de lograr el entendimiento entre todas las personas que conforman el equipo de trabajo, de forma tal que el trabajo de uno pueda ser comprendido por el resto del equipo, con esta misión, otra de las decisiones importantes que toma el arquitecto es el establecimiento de un estilo de codificación por el cual se rijan todos los desarrolladores.

Un estilo de codificación puede definirse como reglas generales que indican cómo construir técnicamente el software, una guía por la que todos los desarrolladores deben seguir su trabajo de forma tal que sea comprensible por todos, los estilos de codificación no son patrones preestablecidos que deben seguirse de forma cabal, sino que, aunque ya existen algunos que son seguidos de acuerdo al lenguaje de programación en el que se implementa la aplicación, es posible establecer su propio estilo de codificación, lo importante es que el desarrollo se guíe por alguno de ellos puesto que facilita la comprensión de los programas, disminuyendo el número de errores durante el desarrollo y el mantenimiento, colaborando así con atributos de calidad tan importantes como la modificabilidad, pues el software debe escribirse de tal forma que pueda evolucionar para cumplir las necesidades de cambio de los clientes, y la reusabilidad, pues el software debe desarrollarse de manera que pueda ser reutilizado total o parcialmente en cualquier otro software, contribuye a la documentación del código y al ahorro de tiempo y recursos en la comprensión de lo escrito, ya que cualquier otra persona debería ser capaz de leer el código y entender su funcionamiento y una buena codificación permite producir un mejor análisis y por lo tanto, los resultados de funcionalidad se mejoran y cuanto menos tiempo se tarde en leer el código, más tiempo de análisis se tiene.

Las reglas que define el estilo de codificación adoptado en el presente trabajo se exponen íntegramente en el Anexo 1.

2.5 Conclusiones

Al conluir el presente capítulo se ha dado una terminación exitosa a la mayoría de las tareas de la investigación y a la más importante, la descripción de la arquitectura, donde se han aplicado satisfactoriamente los conocimientos adquiridos durante el proceso de investigación del capítulo anterior y se ha dado solución al problema planteado.

Capítulo 3: Evaluación de la arquitectura propuesta

Ya ha sido abordada y justificada en el documento la importancia que representa la AS para el proceso de desarrollo del software, y como ha podido apreciarse, el diseño arquitectónico incluye muchas decisiones en pos de lograr no solo la culminación exitosa del software, sino su fácil evolución futura. Para alcanzar este objetivo es decisivo mantener una rigurosa evaluación del diseño arquitectónico con el fin de reconocer a tiempo una debilidad y tomar medidas respecto a ello. En este capítulo encontrará el proceso de evaluación de la arquitectura propuesta no solo por uno de los métodos sugeridos en la literatura sino reafirmada por la implementación de la aplicación Web para la captura de la información requerida por el CNBA, que constituye la puesta en práctica de la AS propuesta y que permitirá verificar, en base a la implementación concreta y tangible de las decisiones arquitectónicas tomadas, el grado de cumplimiento de los atributos de calidad que se trazaron como meta de calidad del software.

3.1 Calidad de la arquitectura

Como quedó expuesto en el Capítulo 1 los atributos de calidad que se desean alcanzar con el diseño arquitectónico son los planteados en el Modelo ISO/IEC 9126 adaptado para AS, para lograr cada uno de ellos se tomaron las siguientes decisiones arquitectónicas:

Tabla 4. Decisiones arquitectónicas que responden a los atributos de calidad deseados.

Atributo de calidad	Decisión arquitectónica para lograrlo.
Funcionalidad	La habilidad del sistema para realizar el trabajo para el cual fue concebido se garantiza con el uso de la tecnología J2EE y el Framework Spring que permiten el desarrollo de aplicaciones Web de probada calidad y seguridad gracias al Framework Acegi Security de Spring, junto con el estilo arquitectónico en capas y su especificación en patrón arquitectónico MVC, también es relevante el SGBD PostgreSQL dada la importancia que tiene para la aplicación la cantidad de datos que se manejan y las conexiones concurrentes que puedan hacerse en un momento dado.
Confiabilidad	La medida de la habilidad del sistema a mantenerse operativo a lo largo del tiempo se avala con el uso del servidor de aplicación Apache Tomcat que es el encargado de restablecer el nivel de desempeño del sistema, también el

	uso del Framework Acegi Security de Spring que garantiza el mecanismo de software para manejar las excepciones.
Eficiencia	Una de las ventajas que propició la decisión arquitectónica de desarrollar una aplicación Web fue los pocos recursos que esta demanda de la PC cliente, ya que la aplicación realmente está corriendo en el servidor Web, y allí las decisiones arquitectónicas para asegurar el desempeño del sistema como el grado en el cual cumple con las funciones designadas dentro de ciertas restricciones como velocidad, exactitud o uso de memoria, están tomadas con la elección del SGBD PostgreSQL y el Framework Hibernate para el acceso a los datos.
Mantenibilidad	La capacidad de someter al sistema a reparaciones y evoluciones de manera rápida y a bajo costo se certifica con el uso de los patrones de diseño DAO, Inyección de Dependencia, Fachada, Bajo Acoplamiento y Alta cohesión. También el patrón de arquitectura MVC contribuye a lograr este objetivo. Y aunque no son iguales Mantenibilidad y Modificabilidad, pero por la relación muy estrecha entre los conceptos, y la importancia que representa la habilidad de realizar cambios futuros al sistema, no se puede dejar de mencionar que los patrones seleccionados para darle soporte a la capacidad de mantenibilidad del sistema son los mismos que propician la habilidad de modificabilidad del mismo. También el uso de las Hojas de Estilo CSS es otra de las decisiones que garantizan el logro del atributo de calidad.
Portabilidad	La habilidad del sistema para ser ejecutado en diferentes ambientes de computación se avala por la tecnología usada en su implementación ya que todas fueron escogidas bajo muchos criterios pero uno de los principales fue la habilidad de ser multiplataforma, este es el caso de J2EE, PostgreSQL y Tomcat. Es necesario hacer la aclaración de que estas herramientas son necesarias en el punto del despliegue de la aplicación en el servidor Web, pero la propia aplicación puede ser accedida por cualquier computador que tenga instalado un navegador Web independientemente del Sistema Operativo en el que se trabaje; y para que sin importar el navegador que se utilice, no haya cambios en la interfaz del usuario, se decidió utilizar el

	lenguaje de programación Java Script y las Hojas de Estilo CSS.

3.2 Aplicación de los métodos de evaluación utilizados

3.2.1 Architecture Tradeoff Analysis Method

En la Fundamentación Teórica quedó dispuesto que el método de evaluación de la AS propuesta sería el Architecture Tradeoff Analysis Method (ATAM) desarrollado por el SEI, su estrategia general consiste en elaborar y filtrar escenarios de calidad y relacionarlos con aspectos específicos de la arquitectura que ayudan a cumplirlos; plantea los roles:

- *Grupo de evaluación de la arquitectura*: grupo de entre 3-5 personas pertenecientes a la misma organización del grupo de desarrolladores o consultores. Este grupo estaría compuesto por los desarrolladores de la aplicación.
- *Tomadores de decisión del proyecto (decision makers)*: responsables del proyecto en capacidad de solicitar cambios, incluyen gerente de proyecto, usuarios del sistema y arquitecto de la solución. En este caso el grupo se compone del líder del proyecto, los arquitectos de la aplicación, y los usuarios del sistema.
- *Stakeholders de la arquitectura*: articulación de las metas de los atributos de calidad en el sistema para que sea exitoso, incluye ingenieros desarrolladores, integradores, el grupo de pruebas, usuarios, etc. Este grupo está compuesto por el dirigente del grupo de calidad, los desarrolladores, y los usuarios.

Comprende nueve pasos agrupados en cuatro fases, que de forma general se resumen de la siguiente forma:

- Presentación, donde la información es intercambiada.
- Investigación y análisis, donde se valoran los atributos claves de calidad requeridos, uno a uno con las propuestas arquitectónicas.
- Pruebas, donde se revisan los resultados obtenidos contra las necesidades relevantes de los stakeholders.
- Informes, donde se presentan los resultados del ATAM.

A continuación se presenta la evaluación de la arquitectura propuesta:

Fase 1: Presentación

Paso 1. Presentación del ATAM: El líder de evaluación describe el método a los participantes, trata de establecer las expectativas y responde las preguntas propuestas.

Paso 2. Presentación de las metas del negocio: Se realiza la descripción de las metas del negocio que motivan el esfuerzo, y aclara que se persiguen objetivos de tipo arquitectónico.

En este paso, el líder del proyecto presenta el sistema desde la perspectiva del negocio, donde quedan expuestos como principales funciones del sistema:

1. Autenticar Usuario
2. Gestionar Usuarios
3. Registrar Producción por Rubro
4. Registrar Permisología
5. Registrar CSP por Rubro
6. Registrar Importación
7. Registrar Exportación
8. Registrar Inventario por Rubro

Como restricciones del sistema, relevantes arquitectónicamente, quedaron identificadas:

1. Se deben recopilar los datos de varios Entes que se encuentran distribuidos geográficamente por todo el país.
2. Los datos que suministran la mayoría de los Entes están en copia dura, o sea, documentos impresos; y los Entes que manejan la información de forma digital la almacenan con su propio formato, es decir, no está estandarizado el formato de los datos.
3. Los datos con los cuales trabaja el CNBA son de vital importancia y de carácter estratégico para la seguridad alimentaria del país por lo cual es necesario garantizar su seguridad.
4. Debe existir un único repositorio central para almacenar todos los datos recopilados.
5. La cantidad de información que se maneja es muy grande.

Y como metas de los atributos de calidad que dan forma a la arquitectura:

1. Lograr la habilidad del sistema para realizar el trabajo para el cual fue concebido.
2. Lograr la habilidad del sistema a mantenerse operativo a lo largo del tiempo.
3. Lograr que el grado con el que el sistema cumpla sus funciones designadas, dentro de ciertas restricciones como velocidad, exactitud o uso de memoria, sea satisfactorio.
4. Lograr la capacidad de que de ser necesario someter al sistema a reparaciones y evoluciones, estas se desarrollen de manera rápida y a bajo costo.
5. Lograr la habilidad de realizar cambios futuros al sistema sin mayores afectaciones a lo que ya está hecho.
6. Lograr la habilidad del sistema para ser ejecutado en diferentes ambientes computacionales.

Paso 3. Presentación de la arquitectura: El arquitecto describe la arquitectura, enfocándose en cómo esta cumple con los objetivos del negocio.

Este paso es desarrollado por los arquitectos e incluye toda la descripción arquitectónica en detalle, no se presenta en esta sección toda la información concerniente a la arquitectura que está siendo evaluada ya que en el capítulo anterior se mostró completamente.

Fase 2: Investigación y análisis

Paso 4. Identificación de los enfoques arquitectónicos: Estos elementos son detectados, pero no analizados.

En este momento quedan identificadas todas las propuestas arquitectónicas que serán analizadas más tarde, o sea, se especifica la tecnología que va a ser utilizada, el patrón arquitectónico y los patrones de diseño, estos elementos no se muestran a continuación ya que se encuentran detallados en el Capítulo 1.

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

El árbol de utilidad se prioriza en dos dimensiones:

1. Por la importancia que cada escenario tiene para el éxito del sistema.
2. Por el grado de dificultad que posee el escenario para ser realizado, según la estimación del arquitecto.

Habitualmente la escala utilizada para ambas dimensiones es High, Medium, Low, pero en este caso se utilizará Alta (A), Media (M) y Baja (B).

En el ATAM hay tres tipos de escenarios: *escenarios de caso de uso* (típicamente usos del sistema existente), *escenarios de crecimiento* (cambios anticipados al sistema), y *escenarios exploratorios* (cambios extremos que se espera que estresen el sistema). Estos diferentes tipos de escenarios son utilizados para probar el sistema desde diferentes ángulos, optimizando las oportunidades de que aparezcan riesgos en las decisiones arquitectónicas.

Terminado el paso actual se obtuvo el siguiente Árbol de Utilidad:

- Utilidad
  - Funcionalidad
    - Seguridad
      - (A, M) Las funcionalidades del sistema se muestran de acuerdo al usuario que este activo.
      - (A, A) Tener protección contra ataques externos que puedan afectar la integridad de los datos almacenados.
      - (A, M) Encriptar todo el tráfico de información.
      - (M, B) Autenticar los usuarios para permitir el uso adecuado de la aplicación.
  - Confiabilidad
    - Recuperabilidad
      - (M, A) En caso de fallo, recuperar el sistema y los datos entrados sin enviar.
  - Mantenibilidad
    - Modificabilidad
      - (A, A) Diseño con capacidad de adición de nuevas funcionalidades y modificaciones futuras.
  - Eficiencia
    - Desempeño
      - (M, M) Búsqueda de información de forma rápida.
  - Portabilidad
    - Adaptabilidad
      - (A, B) Usar la aplicación en los sistemas operativos Windows y GNU/Linux.
      - (A, B) Acceder al sistema desde cualquier parte del país.

En este paso se obtiene el Anexo 2 que basado en los escenarios identificados en el paso 5 y las propuestas arquitectónicas que cumplen con estos escenarios, documenta detalladamente la medida en la cual son adecuados el uno para el otro, y logra la meta del equipo de evaluación de estar convencidos que la propuesta instanciada en la arquitectura que se está evaluando es la apropiada para satisfacer los requerimientos de un atributo específico.

Fase 3: Pruebas

Paso 7. Lluvia de ideas y establecimiento de prioridad de escenarios: Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios.

Mientras que la generación del árbol de utilidad es fundamental para entender cómo el arquitecto percibe y maneja las guías arquitectónicas de los atributos de calidad, el propósito de la lluvia de ideas de escenarios es tomarle el pulso a una gran comunidad de stakeholders.

La lista de los escenarios priorizados resultantes de la lluvia de ideas, recogidos en el Anexo 3, fue comparada con los generados por el Árbol de Utilidad, y al descubrir nuevos escenarios fueron adicionados al Árbol de Utilidad obtenido en el paso 5 y este nuevo Árbol de Utilidad que recoge todos los escenarios identificados durante el proceso de evaluación constituye una de las salidas del método de evaluación Anexo 4.

Paso 8. Análisis de los enfoques arquitectónicos: Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.

Al desarrollar el presente paso, luego de mapear los escenarios recientemente generados con ranking más alto en los artefactos arquitectónicos, se logra la documentación de otros escenarios presentada en el Anexo 2.

Fase 4: Reporte

Paso 9. Presentación de los resultados: Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

Finalmente, la información recogida durante el proceso de evaluación del ATAM se resume y presenta, las más importantes salidas son:

- El conjunto de escenarios priorizados: Anexo 2.
- El árbol de utilidad: Anexo 4.
- Los riesgos descubiertos: Anexo 5.
- Los no riesgos descubiertos: Anexo 6.
- Los puntos de sensibilidad y de tradeoff encontrados: Anexo 7.

3.2.2 Elementos de la implementación de la aplicación Web para la captura de información requerida por el CNBA que respaldan la evaluación

Dado que la mejor forma de demostrar la veracidad de los planteamientos arquitectónicos hechos en el capítulo anterior es la aplicación de todos ellos en un software real implementado bajo sus disposiciones, y para respaldar con más fuerza los resultados de la evaluación hecha con el método ATAM, que en ocasiones, algunos de los escenarios definidos por los usuarios no tenían respaldo arquitectónico que garantizara su funcionalidad en el sistema y sin embargo, en la implementación si se demuestra el cumplimiento de estos requisitos, se procede a realizar una valoración de la arquitectura a través de la aplicación Web para la captura de la información requerida por el CNBA, esta aplicación ya fue probada y aprobada por el grupo de Calidad UCI, y actualmente se encuentra desplegada en el CNBA desarrollando satisfactoriamente las funciones para las cuales fue diseñada.

Con esta valoración se persigue como objetivo, demostrar con más argumentos que la arquitectura propuesta cumple con todas las condiciones para ser adoptada como arquitectura de cualquier sistema informático para la captura de información que utilice las mismas tecnologías y persiga los mismos atributos de calidad.

Debido a que los atributos de calidad que se desean alcanzar con el diseño arquitectónico son los contemplados en el Modelo de Calidad ISO/IEC 9126 adaptado para AS [25], será este el procedimiento a seguir para exponer a través de elementos de la implementación de la aplicación Web para la captura de la información requerida por el CNBA que se alcanzaron las metas de calidad.

Tabla 5. Correspondencia entre características y subcaracterísticas con los elementos de la implementación que garantizan su alcance.

Características	Subcaracterísticas	Elementos de la implementación
Funcionalidad	Exactitud	Los componentes responsables de los cálculos están bien definidos e identificados. Se localizan en el método `processFinish()` de los controladores que realizan operaciones de cálculo para la conversión de unidades de medida y precios [Anexo 8.1]; también se encuentran funciones Java Script responsables de mostrar las conversiones al mismo tiempo que se escribe el valor que será convertido, este fue uno de los escenarios pedidos por los stakeholders que no tenía una justificación desde el punto de vista arquitectónico y queda validado su cumplimiento por este elemento de la implementación [Anexo 8.2].
	Interoperabilidad	Los conectores de comunicación con sistemas externos como es el caso de PostgreSQL se encuentran correctamente identificados y configurados en ficheros XML [Anexo 9].
	Seguridad	Sí existe el mecanismo que desarrolla explícitamente la tarea, es el caso del Framework Acegi Security del Framework Spring, que se encarga de garantizar la autenticación de usuarios y la autorización al acceso de los recursos.
Confiabilidad	Tolerancia a fallas	Si existe el mecanismo de software para manejar las excepciones, es implementado directamente en la aplicación mediante el Framework Spring. [Anexo 10]
	Recuperabilidad	El nivel de desempeño se restablece por medio del Servidor Web Tomcat, y no existe el mecanismo para la recuperación de datos.
Eficiencia	Desempeño	La implementación se desarrolló teniendo en cuenta la importancia del buen desempeño de la aplicación traducido en la velocidad de respuesta a las peticiones del usuario, ejemplo de ello es el uso de las caché del Framework Hibernate.
Mantenibilidad	Acoplamiento	La relación entre componentes es baja, logrado por la aplicación de los patrones utilizados en el diseño de la arquitectura.
	Modularidad	La implementación está separada por módulos y estos a su vez por componentes, el módulo de acceso a datos es común para el resto de los módulos, y cada uno tiene su propia estructura y para modificarlo no es necesario modificar el

		resto.
Portabilidad	Adaptabilidad	Se implementaron funciones Java Script y Hojas de Estilos CSS que permiten la adaptación para que pueda utilizarse cualquier navegador sin afectaciones en la interfaz de usuario. [Anexo 11]
	Instalabilidad	La aplicación finalmente se compila para un ".war" (Web Archive), especificación desarrollada por Sun que permite agrupar un conjunto de clases y documentos que conforman una aplicación Web en Java**,** que se despliega en el servidor de aplicación.
	Coexistencia	No existen mecanismos.
	Reemplazabilidad	Los componentes de la aplicación Web como la implementación de las clases de acceso a datos y de los servicios son reemplazables o pueden ser reemplazadas por otras implementaciones sin que esto afecte el resto de la aplicación.

3.3 Conclusiones

Al terminar el presente capítulo se han cumplido exitosamente todas las tareas propuestas para alcanzar los objetivos planteados en el trabajo, y se ha logrado con la evaluación de la arquitectura, descubrir las decisiones arquitectónicas que constituyen fortalezas o puntos débiles, así como documentar detalladamente cómo estas decisiones responden a los atributos de calidad y cómo colaboran entre sí para alcanzarlos; con el desarrollo de este capítulo se logró comprobar que la arquitectura propuesta puede ser utilizada como referencia para aplicaciones web que utilicen la tecnología J2EE y el Framework Spring y que tengan las mismas prioridades hacia los atributos de calidad planteados en el presente trabajo, considerando siempre los riesgos identificados durante la evaluación, que han quedado claramente definidos y argumentados en conjunto con los no riesgos, los puntos de sensibilidad y los puntos de tradeoff.

Conclusiones

Al finalizar el desarrollo del trabajo se pudo llegar a las siguientes conclusiones:

- La arquitectura propuesta cumple con todos los requisitos de calidad pedidos por el cliente.
- El trabajo constituye una fuente de conocimientos sólidos sobre los métodos de evaluación de arquitectura de software, especialmente de la aplicación del Architecture Trade-off Análisis Method.
- En base a los atributos de calidad que se propusieron alcanzar, la arquitectura propuesta puede ser utilizada como referencia para aplicaciones web que utilicen la tecnología J2EE y el Framework Spring y que tengan las mismas prioridades hacia los atributos de calidad planteados en el trabajo.
- La arquitectura de software descrita da solución al problema planteado ya que las decisiones en las que se basa garantizan la implementación y evolución exitosa de la aplicación Web requerida por el CNBA.

Recomendaciones

Se recomienda estudiar las decisiones arquitectónicas que constituyen riegos según el método de evaluación ATAM y valorar su sustitución por otras decisiones que no pongan en peligro la aplicación como posible mejora a la solución dada en el presente trabajo.

Además se recomienda adicionar la actividad “Evaluación de la arquitectura” al proceso de desarrollo del software de los proyectos productivos de la Universidad teniendo en cuenta las mejoras que le puede aportar.

Referencias Bibliográficas

1. **Longendri Aguilera Mendoza, Lázaro Cánova Amador, Ana Lupe Delgado Montero**. *Solución Informática Para el Centro Nacional de Balance Alimentario y Suministros*. Albet Ingeniería y sistemas. Proyecto Técnico CNBA, 2008.

2. Citado el 03-02-2009: http://www.sei.cmu.edu/architecture/definitions.html

3. **Paul Clements***. A Survey of Architecture Description Languages.* Proceedings of the International Workshop on Software Specification and Design, Alemania 1996.

4. **David Garlan, Mary Shaw**. *An introduction to Software Architecture*. School of Computer Science Carnegie Mellon University, Pittsburgh, Enero 1994 [En línea citado el 20-02-2009: http://www.comp.nus.edu.sg/~liuyang/2103/4.pdf]

5. **Mary Shaw, David Garlan**. *Software Architecture: Perspectives on an emerging discipline.* Upper Saddle River, Prentice Hall 1996.

6. **Len Bass, Paul Clements, Rick Kazman.** *Software Architecture in Practice, Second Edition.* Addison Wesley. Abril 2003.

7. **Mary Shaw, David Garlan**. *Introduction to Software Architectures. New perspectives on an emerging discipline*. Prentice Hall 1996.

8. **Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal.** *Pattern – Oriented Software Architecture. A System of Patterns.* John Wiley & Sons, Inglaterra 1996.

9. **Richard Taylor, Nenad Medvidovic, Kenneth Anderson, James Whitehead, Jason Robbins, Kari Nies, Peyman Oreizy, Deborah Dubrow**. *A Component- and Message-Based Architectural Style for GUI Software. Proceedings of the 17th International Conference on Software Engineering, Seattle, ACM Press, pp. 295-304, Abril 1995.*

10. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns (Elements of Reusable Object-Oriented Software)*.

11. **David Garlan, Mary Shaw.** "*An introduction to software architecture*". CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21, 1994.

12. **Steve Burbeck**. *"Application programming in Smalltalk-80*: How to use Model-View-Controller (MVC)". University of Illinois in Urbana-Champaign, Smalltalk Archive, [En línea citado el 10-02-2009: http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html]

13. **Peter Bengtsson**. *Design and Evaluation of Software Architecture*. University of Karlskrona/Ronneby, Department of Software Engineering and Computer Science, Karlskrona 1999.

14. Citado 10-02-2009: http://dev.mysql.com/doc/refman/5.0/es/features.html

15. Citado 10-02-2009: http://www.netpecos.org/docs/mysql_postgres/x15.html

16. Citado 03-02-2009: http://svnbook.red-bean.com/index.es.html

17. **Ivar Jacobson, Grady Booch, James Rumbaugh**. *El Proceso Unificado De Desarrollo De Software (Volumen 1)*. Editorial Félix Varela, La Habana 2004.

18. **Grady Booch, James Rumbaugh, Ivar Jacobson.** *The UML Modeling Language User Guide*. Adisson-Wesley, 1999.

19. Citado 24-02-2009: http://www.visual-paradigm.com/product/vpuml/

20. **Mario Barbacci, Mark Klein, Thomas Longstaff, Charles Weinstock**. *Quality Attributes*. Carnegie Mellon University. Technical Report. 1995 [En línea citado el 15-02-2009: http://www.sei.cmu.edu/publications/documents/95.reports/95.tr.021.html]

21**. Len Bass, Mark Klein, Felix Bachmann**. *Quality Attribute Design Primitives*. Software Engineering Institute, Carnegie Mellon University 2000. Citado el 15-02-2009 de: http://www.sei.cmu.edu/publications/documents/00.reports/00tn017.html

22. **Bosch, J**. *Design & Use of Software Architectures*. Addison-Wesley 2000.

23**. Rick Kazman, Paul Clements, Mark Klein**. *Evaluating Software Architectures. Methods and case studies*. Addison Wesley. 2001

24**. Roger S. Pressman.** *Ingeniería del Software: Un enfoque práctico* Parte1. La Habana, 2005.

25**. Francisca Losavio, Ledis Chirinos, Nicole Lévy, Amar Ramdane-Cherif**. *Quality Characteristics for Software Architecture*. JOT 2003.[En línea citado el 12-03-2009: http://www.jot.fm/issues/issue_2003_03/article2/].

26. **Yamila Mateu, Marta Hernández, Marisleydis Hondar**. *Solución informática para el Centro Nacional de Balance Alimentario: análisis y diseño de la aplicación Web para la captura y envío de los datos requeridos por el CNBA*. Universidad de las Ciencias Informatiza, La Habana, Mayo 2008.

27. Citado el 05-03-2009: java.sun.com/**j2ee**/overview.html

28. Citado el 05-03-2009: http://www.programacion.com/java/tutorial/patrones2/8/

29. Citado el 05-03-2009: http://www.programacion.com/tutorial/tomcatintro/1/

30. **Robert Allen y David Garlan**, *The Wright Architectural Description Language, Technical Report, Carnegie Mellon University. 1996.*

Bibliografía

1. **Paul Clements**. *Coming Attractions In Software Architecture*. Technical Report, CMU/SEI-96-TR-008, ESC-TR-96-008. Enero 1996.

2. **Paul Clements, Linda Northrop**. *Software Architecture: An Executive Overview*. Technical Report, CMU/SEI-96-TR-003, ESC-TR-96-003. Febrero 1996.

3. **Roger S. Pressman.** *Ingeniería Del Software: Un Enfoque Práctico* Parte1. La Habana 2005.

4. **Ivar Jacobson, Grady Booch, James Rumbaugh.** *El Proceso Unificado De Desarrollo De Software (Volumen 1)*. Editorial Félix Varela. La Habana 2004.

5. **Grady Booch, James Rumbaugh, Ivar Jacobson**. *The UML Modeling Language User Guide*. Addison-Wesley 1999.

6. **Len Bass, Paul Clements, Rick Kazman.** *Software Architecture In Practice, Second Edition.* Addison-Wesley. Abril 2003.

7. **Erika Camacho, Fabio Cardeso, Gabriel Núñez.** *Arquitecturas De Software. Guía De Estudio***.** Abril – 2004 [En línea citado el 10-03-2009 Http://Prof.Usb.Ve/Lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.Pdf]

8. **Philippe Kruchten.** *The Rational Unified Process*. Reading, Addison Wesley Longman, Inc. 1999.

9. **Rick Kazman, Mark Klein, Paul Clements**. *ATAM: Method for Architecture Evaluation.* Pittsburgh, 2009 [En línea citado el 10-03-2009 http://www.sei.cmu.edu/publications/documents/00.reports/00tr004.html]

10. **Longendri Aguilera Mendoza, Lázaro Cánova Amador, Ana Lupe Delgado Montero**. *Solución Informática Para el Centro Nacional de Balance Alimentario y Suministros*. Albet Ingeniería y sistemas. Proyecto Técnico CNBA, 2008.

11. **Paul Clements***. A Survey of Architecture Description Languages.* Proceedings of the International Workshop on Software Specification and Design, Alemania 1996.

12. **David Garlan, Mary Shaw**. *An introduction to Software Architecture*. School of Computer Science Carnegie Mellon University, Pittsburgh, Enero 1994 [En línea citado el 20-02-2009: http://www.comp.nus.edu.sg/~liuyang/2103/4.pdf]

13. **Mary Shaw, David Garlan**. *Software Architecture: Perspectives on an emerging discipline.* Upper Saddle River, Prentice Hall 1996.

14. **Mary Shaw, David Garlan**. *Introduction to Software Architectures. New perspectives on an emerging discipline*. Prentice Hall 1996.

15. **Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal.** *Pattern – Oriented Software Architecture. A System of Patterns.* John Wiley & Sons, Inglaterra 1996.

16. **Richard Taylor, Nenad Medvidovic, Kenneth Anderson, James Whitehead, Jason Robbins, Kari Nies, Peyman Oreizy, Deborah Dubrow**. *A Component- and Message-Based Architectural Style for GUI Software. Proceedings of the 17th International Conference on Software Engineering, Seattle, ACM Press, pp. 295-304, Abril 1995.*

17. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns (Elements of Reusable Object-Oriented Software)*.

18. **David Garlan, Mary Shaw.** "*An introduction to software architecture*". CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21, 1994.

19. **Steve Burbeck**. *"Application programming in Smalltalk-80*: How to use Model-View-Controller (MVC)". University of Illinois in Urbana-Champaign, Smalltalk Archive, [En línea citado el 10-02-2009: http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html]

20. **Peter Bengtsson**. *Design and Evaluation of Software Architecture*. University of Karlskrona/Ronneby, Department of Software Engineering and Computer Science, Karlskrona 1999.

21. **Mario Barbacci, Mark Klein, Thomas Longstaff, Charles Weinstock**. *Quality Attributes*. Carnegie Mellon University. Technical Report. 1995 [En línea citado el 15-02-2009: http://www.sei.cmu.edu/publications/documents/95.reports/95.tr.021.html]

22**. Len Bass, Mark Klein, Felix Bachmann**. *Quality Attribute Design Primitives*. Software Engineering Institute, Carnegie Mellon University 2000. Citado el 15-02-2009 de: http://www.sei.cmu.edu/publications/documents/00.reports/00tn017.html

23. **Bosch, J**. *Design & Use of Software Architectures*. Addison-Wesley 2000.

24**. Rick Kazman, Paul Clements, Mark Klein**. *Evaluating Software Architectures. Methods and case studies*. Addison Wesley. 2001

25**. Francisca Losavio, Ledis Chirinos, Nicole Lévy, Amar Ramdane-Cherif**. *Quality Characteristics for Software Architecture*. JOT 2003.[En línea citado el 12-03-2009: http://www.jot.fm/issues/issue_2003_03/article2/].

26. **Yamila Mateu, Marta Hernández, Marisleydis Hondar**. *Solución informática para el Centro Nacional de Balance Alimentario: análisis y diseño de la aplicación Web para la captura y envío de los datos requeridos por el CNBA*. Universidad de las Ciencias Informatiza, La Habana, Mayo 2008.

27. **Robert Allen y David Garlan**, *The Wright Architectural Description Language, Technical Report, Carnegie Mellon University. 1996.*

Anexos

Anexo 1. Estilo de codificación:

Declaración de variables:

- Codificar cada variable con un nombre que intente describir lo más cercano posible la utilidad que va a tener.
- Deben empezar con minúscula y en caso de nombres compuestos la siguiente también se escribe con mayúscula.
- En el caso de las variables que constituyen instancias de clases persistentes deben llamarse del mismo modo que la clase que representa empezando con minúscula.

Nombre de métodos:

- El nombre debe describir lo más posible la utilidad que va tener.
- Deben empezar con mayúscula y en caso de nombres compuestos la siguiente también se escribe con mayúscula.

Nombre de clases:

- Las clases controladoras deben llamarse <Funcionalidad que juega con mayúscula>Controlador.
- Las interfaces de los servicios deben llamarse <Nombre del Módulo>Service.
- Las implementaciones de las interfaces deben llamarse < Nombre del Módulo>ServiceImpl.
- Los editores deben llamarse <Nombre de la Clase>PropertyEditor.
- Los validadores deben llamarse <Nombre del controlador que validan>Validator.

Corrección de errores:

- Si se encuentran errores, estos deben ser corregidos inmediatamente donde se encontraron y no a través de un parche más adelante en el código o en el flujo del programa.

Escribir limpiamente:

- Todo lo que reduce la ambigüedad es bueno. Por ejemplo los paréntesis.
- Un adecuado espaciado en el código lo hace mucho más legible. Esto incluye espaciado vertical y horizontal en las sentencias y el identado.

Regla:

- Incluir comentarios para cada línea de código que no sean entendibles por sí sola.
- Identación de 8 espacios para cada nivel (equivalente a un Tab).

Líneas en blanco:

- Antes bloque o línea de comentario, a menos que sea la primer línea de un bloque
- Entre declaración de métodos; declaración de la variable y declaración de primer método.

La siguiente figura muestra un fragmento de código que sirve de ejemplo del uso del estilo en la implementación de la aplicación Web requerida por el CNBA, en ella se pueden apreciar algunas de las disposiciones planteadas anteriormente.

```
public class AdministracionServiceImpl implements AdministracionService {

    private UnidadMedidaDAO unidadMedidaDAO;
    private PermisoDAO permisoDAO;

    @Override
    public boolean existeUnidadMedida(String unidadMedida,
            Integer idUnidadMedida) {
        return unidadMedidaDAO.existeUnidadMedida(unidadMedida, idUnidadMedida);
    }

    @Override
    public List<UnidadMedida> obtenerTodasUnidadMedida() {
        return unidadMedidaDAO.obtenerTodos();
    }
```

Figura 14. Fragmento de código de la clase AdministraciónServiceImpl.

Anexo 2. Salidas del método de evaluación ATAM: Análisis de propuestas arquitectónicas.

Escenario #: 1	Escenario: Autenticar los usuarios para permitir el uso adecuado de la aplicación.			
Atributo(s)	Seguridad			
Ambiente	Operación normal.			
Estímulo	Persona o usuario no autenticado intenta acceder a las funcionalidades del sistema.			
Respuesta	Muestra página de autenticación para que introduzca sus credenciales.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de tradeoff	Riesgo	No Riesgo
Utilización del Framework Spring				N1
Utilización del Framework Acegi Security	S1			N2
Explicación	Las decisiones arquitectónicas que responden a la seguridad en el acceso a la aplicación y permitir por consiguiente su uso solo por personas autorizadas son: • Framework Spring: este Framework brinda las facilidad de implementar la seguridad utilizando uno de los Framework de los que se compone: • Framework Acegi Security: proporciona la funcionalidad necesaria para adoptar mecanismos de seguridad en aplicaciones Java utilizando características de programación orientada a aspectos, de forma transparente para el desarrollador, utilizando para ello el soporte prestado por el Framework Spring; este Framework permite definir reglas de acceso basadas en roles o tipos de usuario para los recursos y funcionalidades de la aplicación.			

<table>
<tr><td>Escenario #: 2</td><td colspan="4">Escenario: Las funcionalidades del sistema se muestran de acuerdo al usuario que esté autenticado.</td></tr>
<tr><td>Atributo(s)</td><td colspan="4">Seguridad</td></tr>
<tr><td>Ambiente</td><td colspan="4">Operación normal.</td></tr>
<tr><td>Estímulo</td><td colspan="4">Usuario entra al sistema.</td></tr>
<tr><td>Respuesta</td><td colspan="4">Solo se muestran las funcionalidades a las que tiene acceso según su tipo de usuario.</td></tr>
<tr><td>Decisiones arquitectónicas</td><td>Punto de sensibilidad</td><td>Punto de tradeoff</td><td>Riesgo</td><td>No Riesgo</td></tr>
<tr><td>Utilización del Framework Spring</td><td></td><td></td><td></td><td>N3</td></tr>
<tr><td>Utilización del Framework Acegi Security</td><td>S2</td><td></td><td></td><td>N4</td></tr>
<tr><td>Explicación</td><td colspan="4">Las funcionalidades del sistema deben mostrarse de acuerdo al usuario que este activo, garantizando de este modo que la aplicación sea utilizada en correspondencia con los privilegios permitidos a cada tipo de usuario, las disposiciones arquitectónicas que aseguran el cumplimiento de lo planteado anteriormente son:

• Framework Spring: este Framework brinda las facilidad de implementar la seguridad utilizando uno de los Framework de los que se compone:

• Framework Acegi Security: proporciona la funcionalidad necesaria para adoptar mecanismos de seguridad en aplicaciones Java utilizando características de programación orientada a aspectos, de forma transparente para el desarrollador, sin necesidad de desarrollar código, utilizando para ello el soporte prestado por el Framework Spring.</td></tr>
</table>

Escenario #: 3	Escenario: Encriptar todo el tráfico de información.			
Atributo(s)	Seguridad			
Ambiente	Operación normal.			
Estímulo	Envío de la información al CNBA.			
Respuesta	Encripta la información y la envía al CNBA de forma segura.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de tradeoff	Riesgo	No Riesgo
Protocolo seguro HTTPS	S3			N5
Explicación	La seguridad de la información que es enviada a través de la red se garantiza con el uso del: • Protocolo HTTPS: versión segura del protocolo HTTP que implementa un canal de comunicación seguro basado en SSL (Secure Socket Layers) entre el navegador del cliente y el servidor HTTP.			

Escenario #: 4	Escenario: Tener protección contra ataques externos que puedan afectar la integridad de los datos almacenados.			
Atributo(s)	Seguridad.			
Ambiente	Operación normal.			
Estímulo	Agente externo intenta atacar la BD directamente o a través de la aplicación.			
Respuesta	No tiene efecto el intento de ataque.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de tradeoff	Riesgo	No Riesgo
Framework Hibernate	S4	T1		N6
SGBD PostgreSQL	S5	T2	R1	
En el despliegue el			R2	

servidor Web es una ubicación física aparte del Servidor de BD.				
Explicación	La integridad de los datos almacenados en la BD es uno de los puntos más sensibles en la seguridad del sistema debido a la relevancia que tienen, como se ha expuesto con anterioridad, por tanto muchas son las decisiones arquitectónicas que responden a la protección de estos contra ataques externos: • Framework Hibernate: Las fallas de inyección, en particular la inyección SQL, son ataques comunes en aplicaciones Web. La inyección ocurre cuando los datos proporcionados por el usuario son enviados e interpretados como parte de una orden o consulta. Las fallas de inyección permiten a un atacante crear, modificar o borrar cualquier información arbitraria disponible en la aplicación. El mecanismo clave para evitar inyecciones es el uso de APIs seguras, como consultas con parámetros de asignación rigurosa y bibliotecas de mapeo relacional de objetos (ORM) como Hibernate. • PostgreSQL: La seguridad de la base de datos está implementada en varios niveles y todos los ficheros almacenados en la base de datos están protegidos contra escritura por cualquier cuenta que no sea la del superusuario de Postgre. • Despliegue: La distribución física del sistema en distintos servidores contribuye a la protección de los datos ya que no se encuentran en el mismo ordenador la aplicación y la BD, y el atacante probablemente desconoce cuál es la ubicación real del servidor de BD.			

<table>
<tr><td>Escenario #: 5</td><td colspan="4">Escenario: Búsqueda de información de forma rápida.</td></tr>
<tr><td>Atributo(s)</td><td colspan="4">Eficiencia</td></tr>
<tr><td>Ambiente</td><td colspan="4">Múltiples usuarios conectados concurrentemente.</td></tr>
<tr><td>Estímulo</td><td colspan="4">El usuario selecciona los parámetros y realiza la búsqueda.</td></tr>
<tr><td>Respuesta</td><td colspan="4">Muestra los datos rápidamente.</td></tr>
<tr><td>Decisiones arquitectónicas</td><td>Punto de sensibilidad</td><td>Punto de tradeoff</td><td>Riesgo</td><td>No Riesgo</td></tr>
<tr><td>SGBD PostgreSQL</td><td>S6</td><td>T2</td><td></td><td>N7</td></tr>
<tr><td>Framework Hibernate</td><td>S7</td><td>T1</td><td></td><td>N8</td></tr>
<tr><td>Explicación</td><td colspan="4">Bajo una condición de stress como la conexión concurrente de múltiples usuarios a la aplicación haciendo diferentes o las mismas peticiones, las decisiones arquitectónicas que garantizan la respuesta satisfactoria del sistema son:
• El SGBD PostgreSQL, capaz de manejar las conexiones concurrentes de muchos usuarios de forma eficiente.
• El Framework Hibernate, que con su caché de dos niveles ofrece una gran flexibilidad y rendimiento ya que al almacenar las estructuras de objetos en memoria, evita volver a buscar dichos objetos en la base de datos ahorrando multitud de accesos, y por consiguiente tiempo. Además incluye una caché de consultas que da la posibilidad de obtener rápidamente resultados que ya habían sido consultados previamente.</td></tr>
</table>

Escenario #: 6	Escenario: *Desplegar la aplicación en los sistemas operativos Windows y GNU/Linux.*			
Atributo(s)	Portabilidad.			
Ambiente	Operación normal.			
Estímulo	El webmaster cambia el sistema operativo del servidor Web.			
Respuesta	No hay cambios en la funcionalidad de la aplicación.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de tradeoff	Riesgo	No Riesgo
Tecnología multiplataforma	S8			N9
Explicación	Utilizar Tecnología multiplataforma fue una de las prioridades entre las decisiones arquitectónicas porque garantizan la portabilidad del sistema independientemente del sistema operativo instalado en el servidor Web, por este y otros motivos se escogió el uso del SGBD PostgreSQL y Java.			

Escenario #: 7	Escenario: Acceder al sistema desde cualquier parte del país.			
Atributo(s)	Portabilidad.			
Ambiente	Operación normal.			
Estímulo	El usuario desea utilizar el sistema desde cualquier estado.			
Respuesta	La aplicación se encuentra disponible a través de la web.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de tradeoff	Riesgo	No Riesgo
Tecnología Web.	S9			N10
Explicación	La decisión arquitectónica de utilizar tecnología Web se basa en la necesidad de garantizar el acceso al sistema desde cualquier geografía del país, y dada la marcada diferencia tecnológica que existe, una aplicación Web garantiza el uso satisfactorio del sistema ya que no demanda muchos requerimientos de hardware y software de la PC en la que se trabaja, solo necesita estar conectado a Internet y tener instalado algún navegador Web, como ejemplo de la tecnología Web			

	utilizada está J2EE y el Framework Spring.

Escenario #: 8	Escenario: Diseño con capacidad para la adición de nuevas funcionalidades y modificaciones futuras.			
Atributo(s)	Mantenibilidad			
Ambiente	Operación normal.			
Estímulo	Necesidad de modificar una funcionalidad del sistema o adicionar una nueva.			
Respuesta	El sistema modificado no sufre impactos en el resto de las funcionalidades.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de tradeoff	Riesgo	No Riesgo
Patrón arquitectónico MVC	S10			N11
Patrón de diseño DAO	S11			N12
Patrón de diseño Fachada				N13
Patrón de diseño Inyección de Dependencia	S12			N14
Patrón de diseño Bajo Acoplamiento y Alta Cohesión	S13			N15
Hojas de Estilo CSS		T3		N16
Explicación	Estas decisiones arquitectónicas garantizan el alto grado de mantenibilidad y modificabilidad en la arquitectura propuesta ya que:			

	• Gracias al patrón Inyección de Dependencia cuando se requiere reemplazar o adicionar una clase de la cual depende otra, no es necesario conocer los detalles de la inicialización de la nueva clase pues no es instanciada en la clase dependiente, para lograr el cambio satisfactoriamente, solo se requiere actualizar la configuración del fichero XML. • Por el patrón DAO la migración a un gestor de base de datos diferente sería más fácil pues oculta los detalles de implementación de la fuente de datos a sus clientes; como la interface expuesta por el DAO no cambia cuando cambia la implementación de la fuente de datos subyacente, este patrón permite al DAO adaptarse a diferentes esquemas de almacenamiento sin que esto afecte a sus clientes o componentes de negocio. • Con el patrón Fachada se logra ocultar la complejidad de determinadas clases y disminuir el acoplamiento, lo que facilita el entendimiento de las clases por separado y evita que los cambios en unas afecten a otras. • Mediante el patrón MVC se alcanza una mayor cohesión entre los elementos que se especializan en sus funciones, y un bajo acoplamiento entre ellos, por tanto los cambios son fáciles de implementar. • Al utilizar los estilos CSS se aumenta la capacidad del sistema de evolucionar fácilmente y responder a los cambios de forma rápida, ya que esta decisión tiene la ventaja de ofrecer a los desarrolladores la facilidad de cambiar el formato de las páginas con solo cambiar el elemento que desea en el estilo, y todas las páginas que usan ese estilo automáticamente cambian su formato sin necesidad de ir a cada una de ellas.

Escenario #: 9	Escenario: *En caso de fallo, recuperar el sistema y los datos entrados sin enviar.*			
Atributo(s)	Confiabilidad			
Ambiente	Operación normal.			
Estímulo	Error del sistema operativo que reinicia el ordenador de la PC del cliente.			
Respuesta	No hay recuperación de los datos.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de tradeoff	Riesgo	No Riesgo
No hay			R3	
Explicación	En este escenario se da el mayor de los riesgos que tiene la arquitectura propuesta, no existen decisiones arquitectónicas que respondan a la recuperación del sistema luego de un fallo externo por falta de fluido eléctrico, error del sistema operativo o cualquier error ajeno a la aplicación en la máquina del cliente.			

El siguiente escenario es adicionado a partir de los escenarios priorizados de la lluvia de ideas obtenidos en el paso 7.

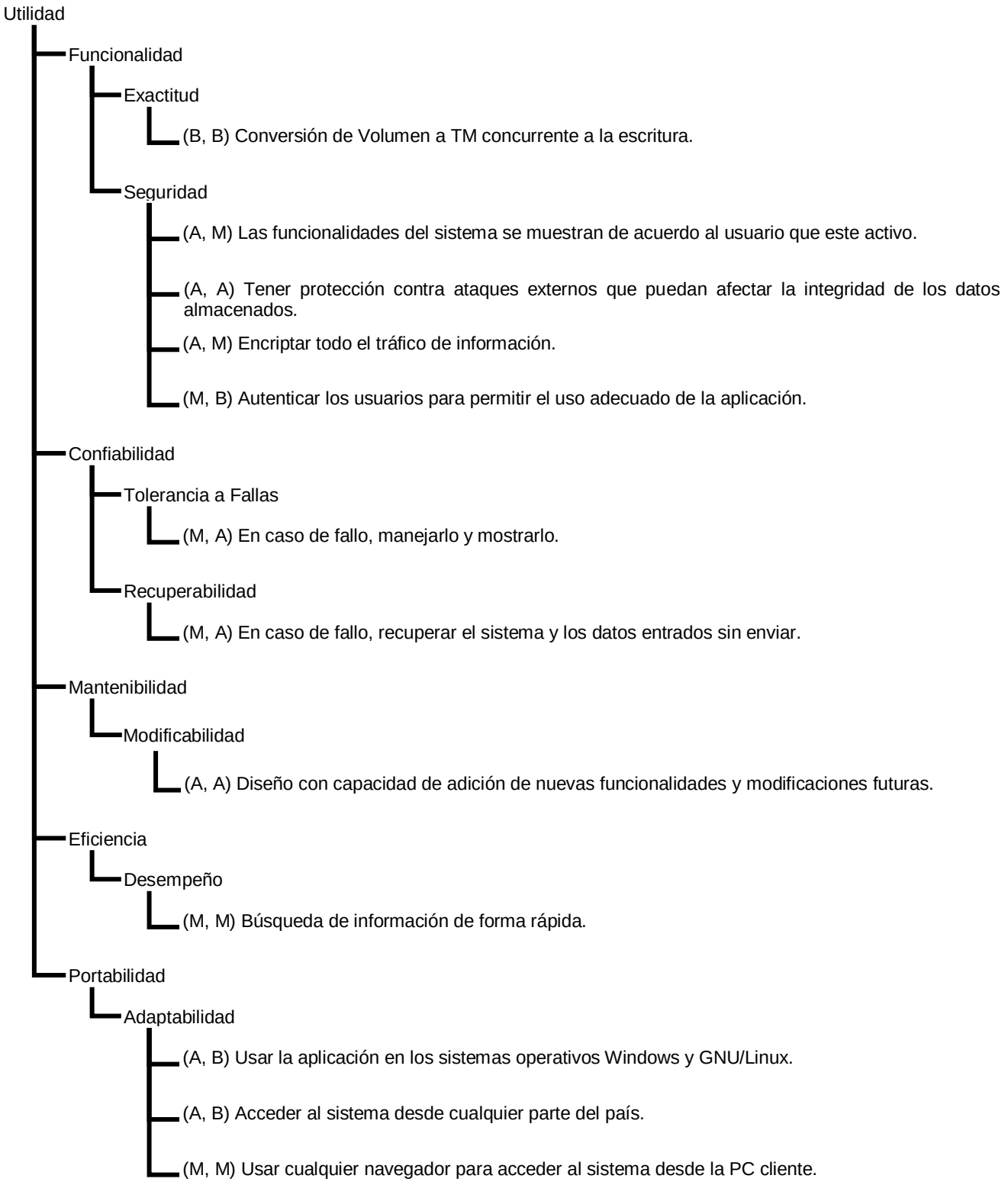
Escenario #: 11	Escenario: Usar cualquier navegador para acceder al sistema desde la PC cliente.			
Atributo(s)	Portabilidad.			
Ambiente	Operación normal.			
Estímulo	El usuario utiliza cualquier navegador web para acceder al sistema.			
Respuesta	La aplicación funciona correctamente sin ocurrir cambios en la interfaz de usuario.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de tradeoff	Riesgo	No Riesgo
Java Script				N17

Hojas de Estilo CSS		T3		N18
Explicación	Las siguientes decisiones arquitectónicas garantizan el cumplimiento del uso del sistema independientemente del navegador Web que se utilice: • Usar el lenguaje de programación Java Script permite implementar funciones que permitan reconocer el navegador que se utiliza para acceder al sistema y en dependencia del mismo hacer las adaptaciones necesarias para que no haya cambios en la interfaz del usuario. • Las Hojas de Estilo CSS permite aplicar el formato de las páginas Web en dependencia del navegador Web que utiliza el cliente para conectarse al sistema de forma rápida y eficiente.			

Anexo 3. Lista de escenarios priorizados de la lluvia de ideas:

Nro.	Escenario	Votos	Atributo de Calidad
10	Conversión de Volumen a TM concurrente a la escritura.	2	Funcionalidad
11	Usar cualquier navegador para acceder al sistema desde la PC cliente.	5	Portabilidad
12	En caso de fallo, manejarlo y mostrarlo.	3	Confiabilidad

Anexo 4. Salidas del método de evaluación ATAM: Árbol de Utilidad.



Anexo 5. Salidas del método de evaluación ATAM: Riesgos.

R1. La naturaleza libre del software PostgreSQL y la consecuente publicidad de sus características, fortalezas y debilidades, además de su código fuente; le ofrece la posibilidad a muchas personas de descubrir o introducir formas de acceso no controladas a la BD y por lo tanto compromete la seguridad de la información. El SGBD PostgreSQL al ser un software libre puede resultar insuficiente en la protección contra ataques externos que puedan afectar la integridad de los datos almacenados.

R2. La distribución física de los componentes en locaciones diferentes puede contribuir a la falta de seguridad ya que, un atacante puede acceder al Servidor de BD sin tener que acceder al Servidor Web y evadir los mecanismos de seguridad aplicados a ese nivel. En el despliegue al estar el servidor Web es una ubicación física aparte del Servidor de BD se puede dar como resultado facilidades de acceso clandestino al Servidor de BD.

R3. Las decisiones arquitectónicas para la recuperación del sistema luego de un fallo externo no han sido definidas, esto podría dar como resultado una pérdida de la información tecleada en el sistema y comprometer su confiabilidad. La falta de medidas para contrarrestar fallos externos se traduce en la perdida de los datos aún no enviados a la BD.

Anexo 6. Salidas del método de evaluación ATAM: No Riesgos.

N1. Al utilizar el Framework Spring la seguridad del sistema es tratada como un requisito no funcional implementado como un aspecto (AOP) a través del Framework Acegi Security el cual garantiza la seguridad al sistema mediante el potente mecanismo de control de usuarios que posee.

N2. La utilización del Framework Acegi Security garantiza la seguridad en el acceso al sistema pues proporciona un probado y eficiente mecanismo de control de usuarios.

N3. La utilización del Framework Spring garantiza la seguridad del sistema a través del Framework Acegi Security el cual mediante su potente y probado mecanismo de autentificación y autorización de usuarios permite que los usuarios que se autentiquen en el sistema solo tengan acceso a las operaciones que tienen definidas.

N4. Al utilizar el Framework Acegi Security se garantiza la seguridad del sistema, permitiendo que cada usuario solo pueda realizar las operaciones que le son concedidas, todo esto es posible mediante el eficiente mecanismo de autentificación y autorización de usuarios que posee dicho Framework.

N5. La transferencia de información bajo el protocolo HTTPS avala la seguridad con la que la información viaja por la red ya que este protocolo utiliza mecanismos de encriptación basado en las SSL para crear un canal cifrado y seguro.

N6. El uso del Framework Hibernate constituye una defensa ante intentos de ataques como la Inyección SQL potenciando la seguridad de la aplicación. Hibernate al ser un ORM es uno de los mecanismos claves para lograr la invulnerabilidad ante las comunes Inyecciones SQL.

N7. Al utilizar el SGBD PostgreSQl la búsqueda de información se realiza de forma rápida porque permite que no se bloquean las tablas, ni siquiera las filas, cuando un proceso interactúa con la BD, todo esto se hace posible por el sofisticado analizador/optimizador de consultas que posee y el uso de la tecnología Multi-Version Concurrency Control (MVCC) para evitar los bloqueos innecesarios, además para el manejo eficiente de estos procesos usa una arquitectura cliente/servidor proceso por usuario la cual consiste en un proceso maestro que se ramifica proporcionando conexiones adicionales para cada cliente que intente conectar a PostgreSQL.

N8. El Framework Hibernate es un componente que aumenta la rapidez de las búsquedas que se realizan garantizando así la eficiencia del sistema ya que con su sistema de caché logra mejorar el tiempo de las búsquedas y por tanto el rendimiento de la aplicación.

N9. Seleccionar la tecnología teniendo en cuenta su independencia a los sistemas operativos es una decisión que determina el alto grado de portabilidad del sistema ya que no está sujeto a ninguna plataforma.

N10. Utilizar tecnología Web para el desarrollo de la aplicación responde positivamente a la necesidad de que el sistema sea accedido desde cualquier parte del país y dada la marcada diferencia tecnológica y los pocos recursos que requiere la aplicación, con ella se garantiza su portabilidad.

N11. Al utilizar el patrón arquitectónico Modelo Vista Controlador (MVC), el cual separa el modelo, la vista y los controladores; esta separación permite construir y probar el modelo independientemente de la representación visual por lo que posibilita en un futuro agregar nuevos módulos y funcionalidades al sistema.

N12. Cuando se utiliza el patrón de diseño DAO el cual se encarga de ocultar completamente los detalles de implementación de la fuente de datos al cliente, además provee una interface que no cambia cuando cambia la implementación de la fuente de datos subyacente, provoca que se puedan agregar nuevas funcionalidades al sistema a través de las interfaces sin tener cambios significativos.

N13. El uso del patrón de diseño Fachada garantiza el diseño con capacidad para la adición de nuevas funcionalidades y modificaciones futuras porque nos provee de una interfaz unificada sencilla que hace de intermediaria entre una interfaz o grupo de interfaces más complejas y un cliente, reduciendo en gran medida las dependencias entre clases, ya que ofrece un punto de acceso al resto de clases, al realizar cambios o sustituciones en las clases solo hay que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones cliente y de esta forma se garantiza la mantenibilidad de sistema.

N14. El uso del patrón de diseño Inyección de Dependencia permite que se puedan agregar nuevas funcionalidades o cambios al sistema porque este patrón establece que un objeto no es el responsable de setear sus dependencias, sino que las mismas deben ser seteadas por un tercero, o sea se inyectan objetos a una clase en lugar de ser la propia clase quien cree el

objeto y de esta manera ayuda a la mantenibilidad del sistema porque es una forma para mitigar la proliferación de dependencias y de fomentar el bajo acoplamiento entre los componentes.

N15. Al utilizar el patrón de diseño Bajo Acoplamiento y Alta Cohesión nos da una medida de la fuerza con que una clase está conectada a otras clases y de cuán relacionadas y enfocadas están las responsabilidades de una clase, por lo que un acoplamiento bajo significa que una clase no depende de muchas clases y una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme, todo esto permite que se le puedan agregar nuevas funciones y realizar modificaciones al sistema sin causar cambios en los diferentes componentes de la aplicación.

N16. Elegir el uso de las Hojas de Estilo CSS es una decisión que garantiza la fácil evolución del sistema ante cambios aumentando su poder de mantenibilidad y modificabilidad, pues los formatos de las páginas de la aplicación son fácilmente cambiables, solo es necesario actualizar los elementos de los estilos que se desean cambiar y todas las páginas que utilicen esos estilos cambiarán su formato automáticamente.

N17. Al utilizar el lenguaje de programación Java Script se pueden implementar funciones que garanticen el reconocimiento del navegador Web que se utiliza al acceder a la aplicación y entonces poder hacerle los ajustes necesarios para que se pueda usar cualquier navegador para acceder al sistema desde la PC cliente sin que haya cambios en la interfaz de usuario y por tanto garantizar la portabilidad de la aplicación.

N18. La decisión arquitectónica de usar Hojas de Estilo CSS garantiza la portabilidad del sistema ya que a través de este lenguaje se definen los estilos que se utilizarán para dar el formato a las páginas Web de la aplicación en dependencia del navegador Web que se utilice para acceder a esta.

Anexo 7. Salidas del método de evaluación ATAM: Puntos de Sensibilidad y deTradeoff.

S1. La utilización del Framework Acegi Security es un punto de sensibilidad al autenticar los usuarios para permitir el uso adecuado de la aplicación.

S2. La utilización del Framework Acegi Security es un punto de sensibilidad para lograr que las funcionalidades del sistema se muestren de acuerdo al usuario que esté autenticado.

S3. El uso del protocolo seguro HTTPS constituye un punto de sensibilidad al encriptar todo el tráfico de información.

S4. Utilizar el Framework Hibernate es un punto de sensibilidad para tener protección contra ataques externos que puedan afectar la integridad de los datos almacenados.

S5. Utilizar el SGBD PostgreSQL es un punto de sensibilidad para tener protección contra ataques externos que puedan afectar la integridad de los datos almacenados.

S6. Emplear el Framework Hibernate constituye un punto de sensibilidad para realizar la búsqueda de información de forma rápida.

S7. Emplear el SGBD PostgreSQL constituye un punto de sensibilidad para realizar la búsqueda de información de forma rápida.

S8. Seleccionar la tecnología empleada entre otros criterios por su capacidad multiplataforma constituye un punto de sensibilidad para desplegar la aplicación en los sistemas operativos Windows y GNU/Linux.

S9. La decisión de utilizar tecnología Web es un punto de sensibilidad para poder acceder al sistema desde cualquier parte del país.

S10. Elegir el patrón arquitectónico MVC constituye un punto de sensibilidad para lograr un diseño con capacidad para la adición de nuevas funcionalidades y modificaciones futuras.

S11. El patrón de diseño DAO establece un punto de sensibilidad para lograr un diseño con capacidad para la adición de nuevas funcionalidades y modificaciones futuras.

S12. Utilizar el patrón de diseño Inyección de Dependencia es un punto de sensibilidad para lograr un diseño con capacidad para la adición de nuevas funcionalidades y modificaciones futuras.

S13. Decidir el uso de los patrones de diseño Bajo Acoplamiento y Alta Cohesión constituye un punto de sensibilidad para lograr un diseño con capacidad para la adición de nuevas funcionalidades y modificaciones futuras.

T1. El uso del Framework Hibernate es un punto de trade-off porque es una decisión que interviene en la respuesta del sistema ante los atributos seguridad y eficiencia.

T2. Utilizar el SGBD PostgreSQL es un punto de trade-off ya que es una decisión que interviene en la respuesta del sistema ante los atributos seguridad y eficiencia.

T3. Usar Hojas de Estilo CSS es constituye un punto de trade-off pues la decisión influye en la respuesta de la aplicación ante los atributos de modificabilidad y portabilidad.

Anexo 8. Componentes de Cálculo:

Anexo 8.1:

```
Double fc = inventario.getUnidadMedida().getConversionTm();
for (DescripcionRubroXUnidadMedida um : inventario
        .getDescripcionRubro().getDescripcionRubroXUnidadMedidas()) {
    if (um.getUnidadMedida().getIdUnidadMedida().equals(
            inventario.getUnidadMedida().getIdUnidadMedida())
            && um.getFactorConversion() != null) {
        fc = um.getFactorConversion();
        break;
    }
}

if (fc != null)
    inventario.setInventarioTm(inventario.getInventario() / fc);
else
    inventario.setInventarioTm(inventario.getInventario());

inventario
        .setPrecioNivelAlmacenTm(inventario.getPrecioNivelAlmacen() * 1000);
```

Figura 15. Fragmento de código del controlador "InsertarInventarioControlador".

Anexo 8.2:

```
function cspTM()
{
    var num= document.forms.registrarCSP2.csp.value;
    var fc = document.forms.registrarCSP2.unidadMedida.value;
    fc = fc.substring(fc.indexOf('-')+1);
    var indice = document.forms.registrarCSP2.unidadMedida.selectedIndex;
    var textoEscogido = document.forms.registrarCSP2.unidadMedida.options[indice].text;
    var resp;
    if(num == "")
        resp =0;
        else if(fc != " " && fc != "")
            resp = num / fc;
            else
                resp = num*1;

    valor = parseInt(resp)
    if(resp < 0){
        resp = 'Solo se aceptan n&uacute;meros positivos.';
        document.getElementById("cspTm").style.color="red";
    } else if(isNaN(valor)){
        resp = 'Solo se aceptan n&uacute;meros.';
        document.getElementById("cspTm").style.color="red";
    }else {
        document.getElementById("cspTm").style.color="black";
    }
    capa = document.getElementById("cspTm");
    capa.innerHTML = resp;
}
```

Figura 16. Fragmento de código del fichero betty.js.

Anexo 9. Comunicación con sistemas externos:

```
<bean id="propertyConfigurer"
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location">
        <value>/WEB-INF/classes/hibernate.properties</value>
    </property>
</bean>

<bean id="dataSource"
    class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">

    <property name="url">
        <value>${hibernate.connection.url}</value>
    </property>
    <property name="driverClassName">
        <value>${hibernate.connection.driver_class}</value>
    </property>
    <property name="username">
        <value>${hibernate.connection.username}</value>
    </property>
    <property name="password">
        <value>${hibernate.connection.password}</value>
    </property>
```

Figura 17. Fragmento de código del fichero de configuración dao-base.xml.

```
hibernate.connection.url =jdbc:postgresql://localhost/minppal-cnba

hibernate.connection.driver_class=org.postgresql.Driver
hibernate.connection.username=test
hibernate.connection.password=test
hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

hibernate.hbm2ddl.auto=none
hibernate.show_sql=false
```

Figura 18. Fragmento del fichero de configuración hibernate.properties.

Anexo 10. Mecanismos para el manejo de excepciones:

```
public class MinppalCnbaException extends RuntimeException {

    private String errorCode;
    private String[] params;

    public MinppalCnbaException(String errorCode) {
        super("minppal_exception");
        this.errorCode = errorCode;
    }

    public MinppalCnbaException(String errorCode, String... params) {
        super("minppal_exception");
        this.errorCode = errorCode;
        this.params = params;
    }
```

Figura 19. Fragmento de código de la clase MinppalCnbaException.

```
Municipio mm = ubicacionService.obtenerMunicipio(mun
        .getId());

if (msg != null && msg.length() > 0)
    throw new MinppalCnbaException(
            "exception.municipio.modificar", mm
                    .getNombreMunicipio(), mm.getEstado()
                    .getNombreEstado(), mun.getEstado()
                    .getNombreEstado(), msg);
else
    throw new MinppalCnbaException(
            "exception.municipio.modificar.corto", mm
                    .getNombreMunicipio(), mm.getEstado()
                    .getNombreEstado(), mun.getEstado()
                    .getNombreEstado());
```

Figura 20. Fragmento de código de la clase MunicipioFormController donde se lanzan excepciones de tipo MinppalCnbaException.

```
<span class="error_page">

    <c:choose>
        <c:when test="${param.error_code == 'no_access'}">
            <authz:authorize ifNotGranted="ROLE_ANONYMOUS">
                Atención '<c:out value="${user.nombreUsuario}" />', no se le permite el acceso a este módulo.
            </authz:authorize>
            <authz:authorize ifAllGranted="ROLE_ANONYMOUS">
                Acceso denegado.
            </authz:authorize>
        </c:when>

        <c:when test="${param.error_code == 404}">
            La página no se encuentra. Por favor, utilice las opciones del menú.
        </c:when>

        <c:when test="${exception.message == 'minppal_exception'}">
            <fmt:message key="${exception.errorCode}">
                <c:forEach items="${exception.params}" var="pr">
                    <fmt:param value="${pr}"/>
                </c:forEach>
            </fmt:message>
        </c:when>
```

Figura 21. Fragmento de código de la página de error: error.jsp

Anexo 11. Mecanismos de adaptación:

```
var BrowserDetect = {
        init: function () {
            this.browser = this.searchString(this.dataBrowser) || "An unknown browser";
            this.version = this.searchVersion(navigator.userAgent)
                || this.searchVersion(navigator.appVersion)
                || "an unknown version";
            this.OS = this.searchString(this.dataOS) || "an unknown OS";
        },
    searchString: function (data) {
        for (var i=0;i<data.length;i++) {
            var dataString = data[i].string;
            var dataProp = data[i].prop;
            this.versionSearchString = data[i].versionSearch || data[i].identity;
            if (dataString) {
                if (dataString.indexOf(data[i].subString) != -1)
                    return data[i].identity;
            }
            else if (dataProp)
            return data[i].identity;
        }
    },
```

Figura 22. Fragmento de código de la función que detecta el navegador Web que utiliza la PC cliente.

```
if(BrowserDetect.browser == 'Opera'){

    document.getElementById('menu_style').href="<c:url value="/styles/menu-estilos-op.css"/>";
}else if(BrowserDetect.browser == 'Explorer'){

    document.getElementById('menu_style').href="<c:url value="/styles/menu-estilos-ie.css"/>";
}
```

Figura 23. Fragmento de código de la función JS que aplica uno u otro estilo CSS en dependencia del navegador Web que esté utilizando el cliente.

Anexo 12. Actas de validación de la arquitectura propuesta:

A continuación se muestran los documentos que fueron firmados como constancia de la conformidad del cliente con la arquitectura diseñada y la aplicación implementada, también son constancia de la funcionalidad sin errores del sistema desarrollado; imágenes que sirven de validación a todos los planteamientos hechos en el presente trabajo, los documentos originales se encuentran en ALBET Ingeniería y Sistemas.

Debido a que en agosto del 2008 el nombre del Centro Nacional de Balance Alimentario (CNBA) fue cambiado por Sistema de Análisis de Indicadores de Cadenas Agroalimentarias (SAICA), fue este el nombre dado a la aplicación desarrollada y los documentos que a continuación se presentan llevan ese nombre en vez del nombre del CNBA.

Además se muestra la imagen del Premio al Rector a nivel de Facultad otorgado a la Arquitectura del proyecto MINPPAL, este premio es el más alto reconocimiento otorgado en la Universidad de las Ciencias Informáticas, en este caso fue entregado en el nivel de Facultad y constituye un elemento más que valida la arquitectura propuesta.

Acta de Aceptación

Producto: Documento "Informe Técnico de Captura de Requisitos del MINPPAL_CNBA".

Categoría de las pruebas: Pruebas de Aceptación a la Documentación.

Fecha de la conciliación: 04 de junio de 2008.

Involucrados en el proceso:

- **Por la parte del Cliente (MINPPAL):** Tomás John
- **Por la parte Suministradora (ALBET):** José A. Castaño
- **Observador Independiente (CALISOFT):** Yeniset León

Observaciones del proceso:

Durante el proceso de pruebas se detectaron un conjunto de defectos que quedaron registrados adecuadamente en el correspondiente documento de No Conformidades detectadas, con sus respectivas observaciones. Teniendo en cuenta que las No Conformidades han sido debidamente resueltas por el Equipo de Desarrollo y validada la eficacia de la corrección por los clientes, se ha tomado el acuerdo de aceptar el documento **"Informe Técnico de Captura de Requisitos del MINPPAL_CNBA"** para la versión 1.3, con fecha 04 de junio de 2008, el cual se anexa a esta acta.

Para que conste la Aceptación de los resultados de las pruebas y por tanto la Aceptación del Documento **"Informe Técnico de Captura de Requisitos del MINPPAL_CNBA"** dando fe del acuerdo firman la presente los principales representantes de las partes.

Tomás John
Representante parte cliente
(MINPPAL)

José A. Castaño
Representante parte
suministradora
(ALBET)

Yeniset León Perdomo
Observador independiente
(CALISOFT)

Figura 24. Acata de Aceptación del Informe Técnico de Captura de Requisitos.

Acta de Aceptación

calisoft

Producto: Documento "Informe Técnico de Documentacion de Diseno" del MINPPAL_CNBA

Categoría de las pruebas: Pruebas de Aceptación a la Documentación.

Fecha de la conciliación: 7 de Agosto de 2008.

Involucrados en el proceso:

- **Por la parte del Cliente (MINPPAL):** Johana Olarte
- **Por la parte Suministradora (ALBET):** José A. Castaño
- **Observador Independiente (CALISOFT):** Yeniset León

Observaciones del proceso:

Por acuerdo entre las partes involucradas en el proceso de Pruebas de Aceptación se ha tomado el acuerdo de aceptar el documento "**Informe Técnico de Documentación de Diseño**" **del MINPPAL_CNBA** para la versión 1.3, con fecha 7 de Agosto de 2008.

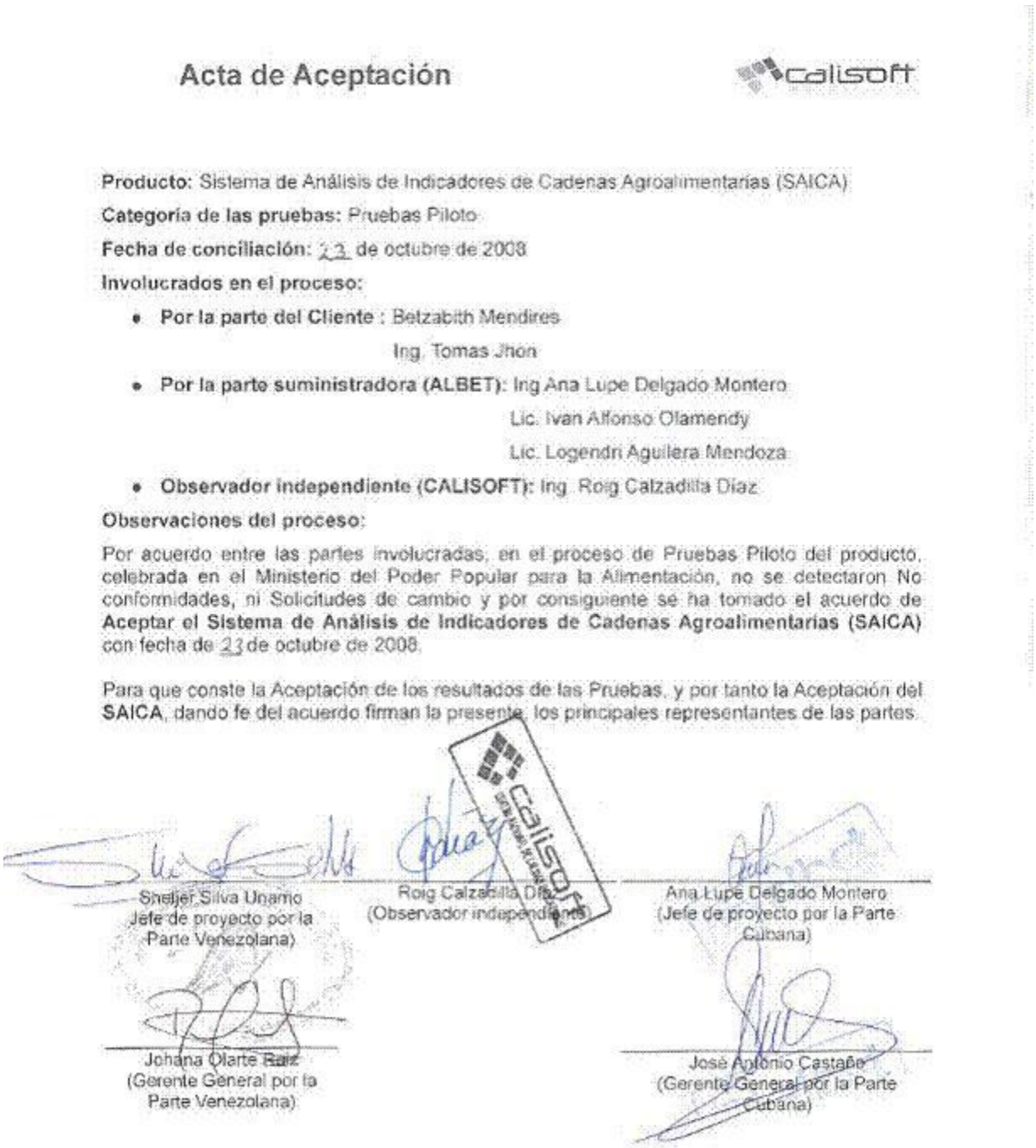
Para que conste la Aceptación de los resultados de las pruebas y por tanto la aceptación del documento **"Informe Técnico de Documentación de Diseño" del MINPPAL_CNBA** para la versión 1.3, dando fe del acuerdo firman la presente los principales representantes de las partes.

Johana Olarte
Representante parte cliente
(MINPPAL_CNBA)

José A. Castaño
Representante parte
suministradora
(ALBET)

Yeniset León Perdomo
Observador independiente
(CALISOFT)

Figura 25. Acta de Aceptación del Informe Técnico de Documentación de Diseño.



Acta de Aceptación

calisoft

Producto: Sistema de Análisis de Indicadores de Cadenas Agroalimentarias (SAICA)

Categoría de las pruebas: Pruebas Piloto

Fecha de conciliación: 23 de octubre de 2008

Involucrados en el proceso:

- **Por la parte del Cliente :** Betzabith Mendires
  Ing. Tomas Jhon
- **Por la parte suministradora (ALBET):** Ing Ana Lupe Delgado Montero
  Lic. Ivan Alfonso Olamendy
  Lic. Logendri Aguilera Mendoza
- **Observador independiente (CALISOFT):** Ing. Roig Calzadilla Díaz

Observaciones del proceso:

Por acuerdo entre las partes involucradas, en el proceso de Pruebas Piloto del producto, celebrada en el Ministerio del Poder Popular para la Alimentación, no se detectaron No conformidades, ni Solicitudes de cambio y por consiguiente se ha tomado el acuerdo de **Aceptar el Sistema de Análisis de Indicadores de Cadenas Agroalimentarias (SAICA)** con fecha de 23 de octubre de 2008.

Para que conste la Aceptación de los resultados de las Pruebas, y por tanto la Aceptación del **SAICA**, dando fe del acuerdo firman la presente, los principales representantes de las partes.

Sneljer Silva Unamo
Jefe de proyecto por la Parte Venezolana)

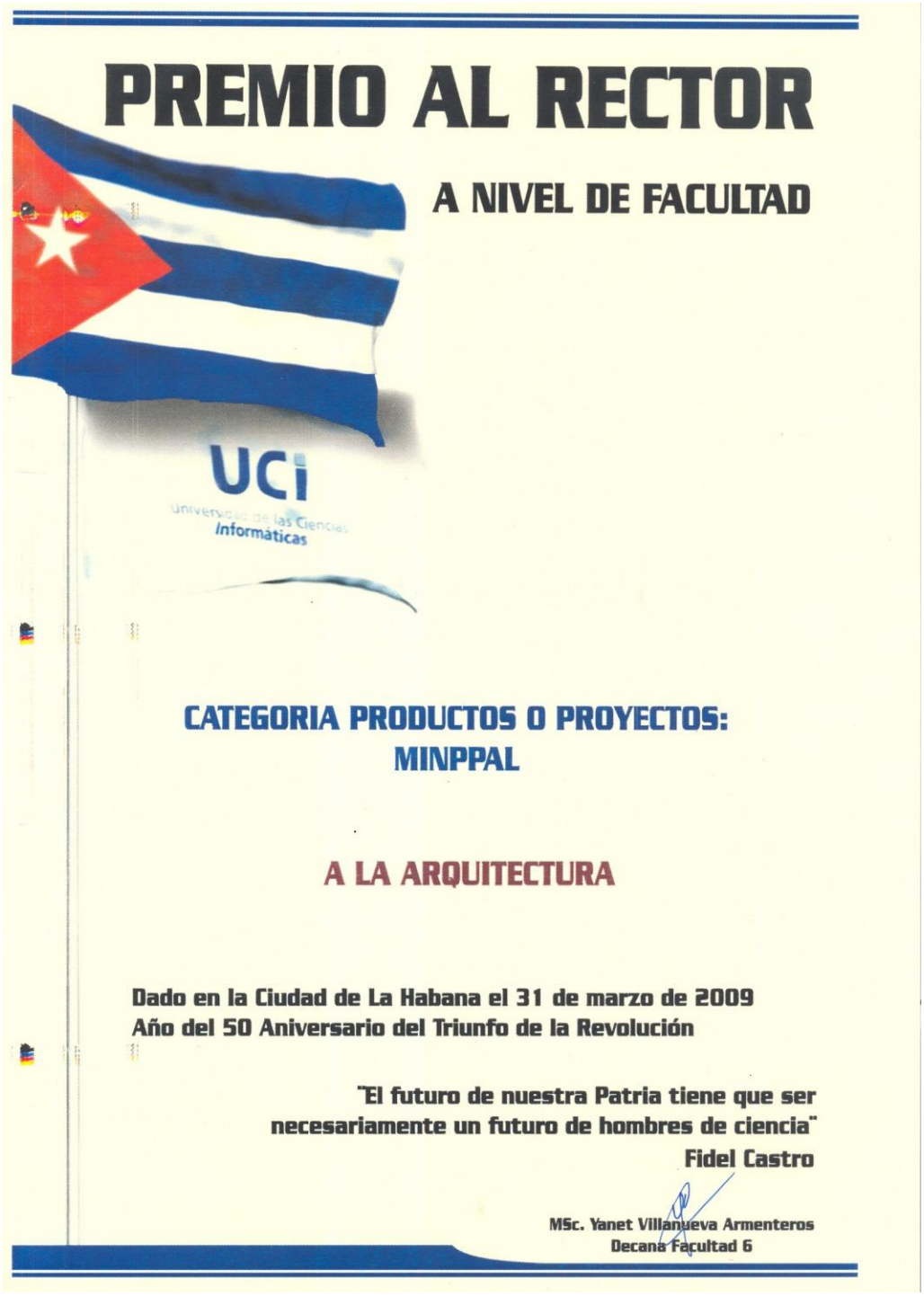
Roig Calzadilla Díaz
(Observador independiente)

Ana Lupe Delgado Montero
(Jefe de proyecto por la Parte Cubana)

Johana Olarte Ruiz
(Gerente General por la Parte Venezolana)

José Antonio Castaño
(Gerente General por la Parte Cubana)

Figura 26. Acta de Aceptación en la categoría de Pruebas Piloto.



PREMIO AL RECTOR

A NIVEL DE FACULTAD

UCI
Universidad de las Ciencias Informáticas

CATEGORIA PRODUCTOS O PROYECTOS:
MINPPAL

A LA ARQUITECTURA

Dado en la Ciudad de La Habana el 31 de marzo de 2009
Año del 50 Aniversario del Triunfo de la Revolución

"El futuro de nuestra Patria tiene que ser necesariamente un futuro de hombres de ciencia"
Fidel Castro

MSc. Yanet Villanueva Armenteros
Decana Facultad 6

Figura 27. Premio al Rector a nivel de Facultad otorgado a la Arquitectura del Proyecto MINPPAL.

Glosario de términos

AOP (Aspect-Oriented Programming): es un paradigma de programación cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos.

API (Application Programming Interface): es el conjunto de funciones y procedimientos (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Árbol de Utilidad: es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno.

Casos de uso arquitectónicamente relevantes: son aquellos que nos ayudan a mitigar los riesgos más importantes, aquellos que son los más importantes para los usuarios del sistema, y aquellos que nos ayudan a cubrir todas las funcionalidades significativas.

Contenedor de Servlets: es un SHELL de ejecución que maneja e invoca Servlets por cuenta del usuario.

*CVS (*Concurrent Versions System*):* es una aplicación informática que implementa un sistema de control de versiones.

EJB (Enterprise JavaBeans): son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE. Su especificación detalla cómo los servidores de aplicaciones proveen objetos desde el lado del servidor que son, precisamente, los EJB.

GNU: proyecto iniciado por Richard Stallman con el objetivo de crear un sistema operativo completamente libre. Actualmente la Fundación para el Software Libre (FSF - Free Software Foundation) institución dedicada a eliminar las restricciones de uso, copia, modificación y distribución del software, pone a disposición de todo el mundo un completo e integrado sistema de software llamado GNU.

GPL (General Public License): es una licencia creada por la Free Software Foundation y está orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

GRASP (General Responsability Assignment Software Patterns): son patrones generales de software para asignación de responsabilidades.

iBatis: es un Framework de código abierto basado en capas desarrollado por Apache Software Foundation, que se ocupa de la capa de Persistencia (se sitúa entre la lógica de Negocio y la capa de la Base de Datos).

JBoss: es un servidor de aplicaciones J2EE de código abierto. Al estar basado en Java, JBoss puede ser utilizado en cualquier sistema operativo que lo soporte. Implementa todo el paquete de servicios de J2EE.

JCP (Java Community Process): es un proceso formalizado que permite a las partes interesadas a involucrarse en la definición de futuras versiones y características de la plataforma Java.

JDBC (Java Database Connectivity): es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java.

JNDI (Java Naming and Directory Interface): es una API para servicios de directorio.

JSF (Java Server Faces): es un Framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE.

Línea base de la arquitectura: es una versión interna del sistema, que está centrada en la descripción de la arquitectura.

SHELL: intérprete de comandos, consola o interfaz de ejecución.

XML (Extensible Markup Language): es un metalenguaje extensible de etiquetas.