Universidad de las Ciencias Informáticas
FACULTAD 6

Título: "Propuesta de arquitectura para el Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela."

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autores: Eneysi Osorio Castro

Asdrúbal Torres Camilo

Tutores: Ing. Andrés Ballester Marsal

Ing. Reynaldo Alvarez Luna

Ciudad de La Habana, Cuba.

Junio, 2010.

"Año 52 de la Revolución"

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 _____.

Asdrúbal Torres Camilo

_____
Firma del Autor

Ing. Andrés Ballester Marsal

_____
Firma del Tutor

Eneysi Osorio Castro

_____
Firma del Autor

Ing. Reynaldo Alvarez Luna

_____
Firma del Tutor

DATOS DE CONTACTO

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

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

Agradecimientos

Ante todo agradezco a mis padres por guiarme con su ejemplo de sacrificio y firmeza. A mis hermanas y hermanos por brindarme su ayuda todo el tiempo. A mi familia por confiar en mí. A mi novio Yanel Machado Machado por apoyarme siempre y demostrarme que con esfuerzo y sacrificios todo es posible. A mis cuñados Ismael y Erislandi, por su apoyo y a mi cuñada Carmen Luisa por estar presente siempre. A mi otra madre Diosdada. Al Comandante Fidel Castro y la Revolución por darme la oportunidad de estudiar en esta universidad de excelencia. A todos mis amigos que en diferentes momentos desempeñaron un papel importante en mi vida: a esa hermana de siempre Yulainne (Yuly), Karelia, a mi buena consejera Yanet Hernández, Yaima (Yayi), Lianet Hernández, Daynelis, Lidiannis (Lili), mi hermanito Yunier (Yuyo)… A mi compañero de tesis Asdrúbal por soportarme durante el desarrollo de este arduo trabajo. A mis tutores por su paciencia y entrega. A mis compañeros y profesores del proyecto Prevención del Delito por lograr llegar a ser la familia que hoy somos. En fin a todos aquellos que contribuyeron a mi formación profesional.

Eneysi

En primer lugar agradezco a mi familia, fuente de mi educación y aprendizaje. A todos los que me han brindado la mano cuando la he necesitado. En especial a toda mi familia de la habana que siempre me ayudó, a Pepito, Marlenis, Iliana, Roberto, Roy, Juan Carlos, Evismaris, Dixan. También a Zoraida y Gloria que siempre me trataron como otro miembro de la familia. A los tutores por todas las horas dedicadas para que este trabajo se hiciera realizad. A todos mis amigos de la universidad durante estos 5 largos años de estudio y sacrificio. Muy especial a todos los del grupo 6105. A mis amigos de siempre Asencio, Jorge y Marianela. A mis compañeros de trabajo Yudiel y Lianet. Gracias a todos los profesores del proyecto Genética Médica. A todos los miembros del proyecto Prevención del Delito. A mis compañeros de tesis por trabajar siempre juntos y como verdaderos hermanos. Agradezco a Eneysi por ser mi dúo de tesis, y haber compartido estos días de stress y trabajo juntos. Y finalmente agradezco a todos los profesores que han participado en mi formación profesional durante estos 5 años, y en especial, a la Universidad de las Ciencias Informáticas y a la Revolución, que me dio la oportunidad de cursar esta carrera, haciendo mis sueños realidad.

Asdrúbal

Dedicatoria

A mi madre, por su incansable educación, su confianza, por su comprensión y el amor que me brindó siempre.

A mi padre, por su constante ejemplo, por tenerlo ahí cada instante de mi vida, por su sabiduría, su firmeza y la educación que supo darme.

A mis hermanas Aliuska y Daniuska, por estar siempre conmigo dándome su apoyo y por ser las mejores amigas de todas.

A mis hermanos Yuri y Yaismel, por ser ejemplos siempre y por darme tanta fuerza para seguir adelante.

A mi novio Yanel Machado Machado, por su paciencia, por el amor tan puro que supo ofrecerme, por su apoyo, comprensión y por darme las fuerzas para seguir adelante en las más difíciles de las situaciones.

A mis sobrinitos Isabelita, Dayanis y Yonatan, por ser unos de mis grandes amores y aunque son muy pequeñitos son fuentes de mi inspiración.

A toda mi familia, por su confianza y contribuir a que sea lo que soy hoy.

A la Revolución, por darme esta gran oportunidad de ser una más de sus profesionales.

Eneysi

A mi madre por ser mi inspiración diaria, mi musa de la confianza, mi aliento para seguir siempre adelante.

A mi padre por exigirme cada día ser mejor, guiándome con su ejemplo de voluntad, empeño y sacrificio. A mi familia, creadora de todo lo que soy hoy.

A la Revolución y a Fidel por permitirme cumplir este sueño.

Asdrúbal

Resumen

El presente trabajo se realizó a raíz de la necesidad de lograr una mejor integración de los componentes del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. Para darle solución a esta situación se decidió definir la Arquitectura de Software del sistema.

Dicha arquitectura cuenta con los elementos necesarios para el desarrollo del sistema. Los mismos tuvieron su éxito por las decisiones arquitectónicas que fueron tomadas a lo largo del ciclo de vida del desarrollo del sistema. Otro aspecto que garantizó el éxito fue la descripción de la arquitectura, representada por medio de las vistas de la arquitectura, definidas por la metodología de desarrollo RUP. Con el objetivo de que las principales tecnologías y herramientas garanticen una realización exitosa del sistema se llevó a cabo un profundo estudio de las mismas, teniéndose en cuenta principalmente que estuviesen bajo licencia libre. Fue utilizada como plataforma de desarrollo Java Enterprise Edition (JEE). Se desarrolló el sistema empleando los patrones arquitectónicos Arquitectura en Capas y Modelo Vista Controlador (MVC), los mismos fueron implementados haciendo uso de los frameworks Dalas y Spring fundamentalmente.

Finalmente, fue evaluada la arquitectura haciendo uso del método de evaluación Architecture Tradeoff Analisys Method (ATAM), obteniéndose resultados satisfactorios.

PALABRAS CLAVE: Arquitectura de Software, ATAM, Dalas, JEE, Spring.

Tabla de Contenidos

AGRADECIMIENTOS III
DEDICATORIA IV
RESUMEN V
INTRODUCCIÓN 1
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 4
1.1 Descripción del entorno 4
1.2 Arquitectura de Software 5
1.2.1 Estilo Arquitectónico 6
1.2.2 Patrón Arquitectónico 7
1.3 Estado del arte de la Arquitectura de Software 8
1.3.1 Estilos arquitectónicos 8
1.3.2 Patrones arquitectónicos 9
1.3.3 Patrones de Diseño 10
1.3.4 Relación entre los niveles de abstracción 11
1.4 Atributos de calidad en la Arquitectura de Software 12
1.5 Tecnologías y Herramientas 13
1.5.1 Plataforma de desarrollo 13
1.5.2 Framework de desarrollo 14
1.5.3 Entorno de Desarrollo Integrado 21
1.5.4 Servidor Web 23
1.5.5 Sistema Gestor de Bases de Datos 23
1.5.6 Herramienta para el Control de Versiones 25
1.6 Metodología de desarrollo 26
1.6.1 Responsabilidades del arquitecto de software 28
1.6.2 Lenguaje y herramienta para el modelado 29
1.7 Métodos de evaluación de la Arquitectura de Software 31
1.8 Conclusiones 32
CAPÍTULO 2: DESCRIPCIÓN DE LA ARQUITECTURA 33
2.1 Arquitectura seleccionada 33
2.2 Representación arquitectónica 34

Tabla de contenidos

2.3 Objetivos y restricciones arquitectónicas 34
2.4 Vistas arquitectónicas 37
2.4.1 Vista de Casos de Uso 37
2.4.2 Vista Lógica 39
2.4.3 Vista de Implementación 44
2.4.4 Vista de Despliegue 47
2.5 Lineamientos de diseño e implementación 50
2.5.1 Patrones de diseño propuestos 50
2.5.2 Estándares de codificación 50
2.6 Conclusiones 50
CAPÍTULO 3: EVALUACIÓN DE LA ARQUITECTURA PROPUESTA 51
3.1 Calidad de la arquitectura 51
3.2 Aplicación del método de evaluación seleccionado. 53
3.3 Conclusiones 57
CONCLUSIONES 58
RECOMENDACIONES 59
REFERENCIAS BIBLIOGRÁFICAS 60
BIBLIOGRAFÍA 62
ANEXOS 65
GLOSARIO 69

INTRODUCCIÓN

Con el triunfo del presidente venezolano Hugo Rafael Chávez Frías, se prioriza una amplia agenda social, en la que se encuentran sectores como la salud, la educación, el empleo así como la seguridad ciudadana. Para garantizar la paz y el bienestar de la sociedad se hace necesario disminuir la delincuencia que presenta el pueblo venezolano, por lo que se buscan vías de solución. Como parte de la cooperación entre Cuba y la República Bolivariana de Venezuela, se desarrolla el proyecto Prevención del Delito por parte de la Universidad de las Ciencias Informáticas (UCI) conjuntamente con el Ministerio del Poder Popular para las Relaciones Interiores y de Justicia (MPPRIJ) de la República Bolivariana de Venezuela.

El MPPRIJ, teniendo en cuenta su misión institucional de garantizar la seguridad ciudadana, promueve la formulación y puesta en marcha del Proyecto Implantación de Solución Integral para el perfeccionamiento del Sistema de Prevención del Delito de la República Bolivariana de Venezuela, con el propósito de brindar a las Coordinaciones Regionales una tecnología que permita un mejor desempeño y de esta manera brindar una asistencia de mayor calidad al ciudadano.

La Dirección General de Prevención del Delito (DGPD) está adscrita el Viceministerio de Seguridad Ciudadana del MPPRIJ de la República Bolivariana de Venezuela. Posee 22 Coordinaciones de Estado, 18 adscritas al MPPRIJ, ubicadas en los Estados: Anzoátegui, Aragua, Barinas, Bolívar, Carabobo, Cojedes, Falcón, Guárico, Lara, Mérida, Miranda, Monagas, Portuguesa, Táchira, Trujillo, Yaracuy, Zulia y en el Distrito Capital. Además, se han creado 4 de forma descentralizada en los estados: Delta Amacuro, Nueva Esparta, Sucre y Vargas, las cuales ejecutan los programas de la Dirección General y apoyan los planes de los Ejecutivos Regionales.

La DGPD, con el propósito de lograr una mejora en el procesamiento de las informaciones y el cumplimiento de las actividades a realizar, necesita de los datos de las distintas Coordinaciones Regionales ubicadas en diferentes puntos de la geografía venezolana, requiriéndose una integración entre dichos entes para el envío de la información. Existe además diversidad en la tecnología utilizada, lo que significa que algunos de los procesos se manejan en formato digital, aunque muchos de ellos son elaborados de manera manual y guardados en formato duro. Esto provoca que el envío de las informaciones sea más lento y que no exista una adecuada seguridad de las informaciones manejadas. Además, no se hace posible una eficiente toma de decisiones y en la DGPD no se cuenta con información

centralizada para elaborar informes referentes a la seguridad ciudadana, crear proyectos y programas orientados al crimen y la violencia.

Una solución viable a esta situación es desarrollar un Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito. El sistema a desarrollarse sobre la plataforma Java Enterprise Edition (JEE), debe cumplir con los requisitos indispensables de calidad, seguridad y escalabilidad, que le hagan una solución eficiente para cumplir con los objetivos de la DGPD. Debido a la magnitud y complejidad de este proyecto de desarrollo, la cantidad de personas involucradas, la necesidad de documentación temprana de las decisiones de diseño, el número de módulos a implementar, los atributos de calidad involucrados en el sistema, así como la importancia de trabajar sobre una línea común que permita alcanzar los objetivos del sistema de información y la evolución a futuras versiones, es vital el diseño de una **Arquitectura de Software** sólida y flexible, que permita el desarrollo y mantenimiento exitoso del sistema.

Partiendo de los argumentos anteriormente expuestos, el **problema científico** de la presente investigación es: ¿Cómo definir la organización estructural de las partes que componen el Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito?

Teniendo en cuenta lo anteriormente planteado como problema científico, se define como **objeto de estudio**: Arquitectura de Software para Sistemas de Gestión sobre JEE, y a partir del mismo se selecciona como **campo de acción**: Arquitectura de Software para el Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito.

Para dar respuesta al problema científico planteado se propone como **objetivo general**: Definir la Arquitectura de Software del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito. Del mismo se derivan los siguientes **objetivos específicos**:

- Definir la línea base de la arquitectura.
- Describir la arquitectura del sistema.
- Evaluar la arquitectura propuesta.

Con el objetivo de dar solución a los objetivos específicos se plantean las siguientes **tareas de la investigación**:

1. Estudio de los requerimientos funcionales y no funcionales del sistema.

2. Estudio del arte de la Arquitectura de Software teniendo en cuenta la tecnología JEE para el desarrollo de Sistemas de Gestión de Información.
3. Estudio y selección de las tecnologías y herramientas a utilizar.
4. Estudio y selección de los estilos y patrones arquitectónicos a utilizar.
5. Estudio y selección de las metodologías de desarrollo.
6. Descripción de la Arquitectura de Software con la tecnología JEE para el desarrollo del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito.
7. Diseño de las vistas de casos de uso, lógica, implementación y despliegue.
8. Definir estándares de codificación para el desarrollo del sistema.
9. Análisis de las técnicas y métodos de evaluación de la Arquitectura de Software.
10. Evaluación de la arquitectura definida.

El presente trabajo de diploma está estructurado en tres capítulos:

- ***Capítulo 1: Fundamentación Teórica,*** en este capítulo serán analizados aquellos aspectos que son de suma importancia para la Arquitectura de Software, además se hará un estudio del arte de la misma. Se podrá encontrar también las diferentes tecnologías y herramientas seleccionadas para dar cumplimiento al desarrollo del sistema, así como la metodología de desarrollo que se seguirá, unido al lenguaje y la herramienta de modelado.
- ***Capítulo 2*****: *Descripción de la arquitectura***, en este capítulo se plantea cual será la arquitectura seleccionada para el desarrollo del sistema. Se especifica cómo será la representación arquitectónica y se presentan las diferentes vistas arquitectónicas que componen la descripción de la arquitectura planteada por la metodología de desarrollo utilizada. Además, podrá encontrar los estilos de codificación que deben seguir los desarrolladores y los patrones de diseño propuestos para elevar la calidad del software.
- ***Capítulo 3*****: *Evaluación de la arquitectura propuesta,*** en este capítulo se hará una evaluación de la arquitectura propuesta haciendo uso de los atributos de calidad y el método de evaluación seleccionado, describiéndose posteriormente los resultados obtenidos.

Capítulo 1: Fundamentación Teórica

En este capítulo de hace un estudio de aquellos temas que están estrechamente relacionados con la Arquitectura de Software, y que resultan esenciales para un mejor entendimiento de la solución propuesta, como los patrones y estilos arquitectónicos, patrones de diseño, lenguajes y herramientas de modelado. Además, se plantean las definiciones de Arquitectura de Software así como las tecnologías y herramientas necesarias para el desarrollo del sistema.

1.1 Descripción del entorno

La Dirección General de Prevención del Delito (DGPD) es el ente encargado de dar asesoría a los ciudadanos del país y sirve de puente para viabilizar problemas de las comunidades a través de las Coordinaciones Regionales de cada territorio. Igualmente la DGPD maneja diversos programas que llegan a esferas como las educacionales, políticas y sociales. Da orientación y ayuda a las personas que así lo necesitan y guía a las Coordinaciones Regionales en su trabajo.

La DGPD se encarga de recibir informaciones de las Coordinaciones Regionales que se encuentran ubicadas geográficamente por todo el país, con el fin de controlar los procesos clave, orientados a satisfacer la sociedad venezolana en materia de prevención del delito. Para trabajar con estas informaciones se hace mediante planillas, algunas veces de forma digital, observándose la marcada diferencia en las tecnologías utilizadas. Para el envío de las informaciones no se cuenta con una reglamentación o estructuración de cómo debe ser el mismo, por lo que para realizar este se hace por diferentes vías alternativas que pueden provocar la falta de seguridad de las informaciones enviadas. También se incurre en la demora de entrega. Además, no se hace posible una eficiente toma de decisiones y en la DGPD no se cuenta con información centralizada para elaborar informes referentes a la seguridad ciudadana.

Con el propósito de ayudar al mejoramiento de los diferentes procesos de la DGPD, es preciso un sistema que favorezca la mejora en el procesamiento de los datos que se recogen de las diferentes Coordinaciones. Un sistema que establezca un formato estándar para eliminar las demoras en la entrega de las informaciones así como la falta de seguridad en el envío de las mismas y la diferenciación de las tecnologías utilizadas para procesar los datos. Para lograr el desarrollo del sistema es precisa la integración de cada uno de los módulos por lo que debe estar compuesto. También se debe obtener una

solución altamente escalable y que cumpla con los atributos de calidad necesarios para el sistema. La calidad del sistema es necesaria, por lo que debe ser modificable, seguro y disponible, asegurando esto el mantenimiento, flexibilidad y reusabilidad del mismo. Para lograr estos requisitos es precisa una Arquitectura de Software robusta y confiable que se adapte a las necesidades del sistema.

1.2 Arquitectura de Software

A pesar del interés por la Arquitectura de Software como un campo de investigación, no existe una visión clara con respecto a la definición de la misma. La arquitectura, conformada por diferentes visiones del sistema, constituye un modelo de cómo está estructurado dicho sistema, sirviendo de comunicación entre las personas involucradas en el desarrollo y ayudando a realizar diversos análisis que orienten el proceso de toma de decisiones. Para que la arquitectura se convierta en una herramienta útil dentro del desarrollo de los sistemas de software, es necesario que se cuente con una manera precisa de representarla.

A continuación se muestran algunas de las principales definiciones de Arquitectura de Software estudiadas, que sirven de guía para el desarrollo del trabajo:

Clements considera que “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”. [1]

Otra definición es que la Arquitectura de Software consiste en la estructura o sistema de estructuras, que comprenden los elementos de software, las propiedades externas visibles de esos elementos y la relación entre ellos”. [2]

La definición de Arquitectura de Software más usada actualmente es de la IEEE Std 1471-2000, la cual 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”. [3]

La Arquitectura de Software abarca decisiones importantes sobre: la organización del sistema software, los elementos estructurales que compondrán el sistema y sus interfaces, junto con sus comportamientos,

tal y como se especifican en las colaboraciones entre estos elementos, además de la composición de los elementos estructurales y del comportamiento en subsistemas progresivamente más grandes.

Sin embargo, la arquitectura software está afectada no solo por la estructura y el comportamiento, sino también por el uso, la funcionalidad, el rendimiento, la flexibilidad, la reutilización, la facilidad de comprensión, las restricciones y compromisos económicos y tecnológicos y la estética. [4]

La Arquitectura de Software es una vista estructural de alto nivel; define un estilo o combinación de estilos para una solución, centra su atención en los requerimientos no funcionales. Se puede añadir que la misma está muy relacionada al trabajo en equipo y la correcta planificación y costo del software.

El objetivo principal de la Arquitectura del Software es aportar elementos que ayuden a la toma de decisiones y, al mismo tiempo, proporcionar conceptos y un lenguaje común que permitan la comunicación entre los equipos que participen en un proyecto. [5]

A pesar de las discrepancias en las diversas definiciones de Arquitectura de Software, los autores coinciden que la misma define la estructura del sistema.

La Arquitectura de Software posee gran importancia, para mencionar algunas se puede decir que la misma permite la comunicación entre las personas involucradas. Contribuye a obtener una documentación temprana de las decisiones de diseño. Permite adivinar determinadas cualidades del sistema. Además, es la base para el entrenamiento de nuevo personal.

1.2.1 Estilo Arquitectónico

Perry y Wolf establecen el razonamiento sobre estilos de arquitectura como uno de los aspectos fundamentales de la disciplina Arquitectura de Software. Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales. [6]

Estilo arquitectónico es 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. El estilo arquitectónico define las reglas

generales de organización en términos de un patrón y las restricciones en la forma y la estructura de un grupo de sistemas de software. Los mismos sirven para sintetizar estructuras de soluciones, así como definir los patrones posibles de las aplicaciones. Estos estilos precisan qué forma tienen los componentes, qué forma tiene la comunicación entre esos componentes y qué restricciones se ponen a esa comunicación.

1.2.2 Patrón Arquitectónico

Un patrón arquitectónico define la estructura básica de una aplicación, provee un subconjunto de subsistemas predefinidos, incluyendo reglas, lineamientos para conectarlos y pautas para su organización, además constituye una plantilla de construcción.

Buschmann [7] define 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. Ésta abarca:
  - Estructura con componentes y relaciones.
  - Comportamiento a tiempo de ejecución: aspectos dinámicos de la solución, como la colaboración entre componentes y la comunicación entre ellos.

Además, propone los patrones arquitectónicos como descripción de un problema particular y recurrente de diseño, que aparece en contextos de diseño específicos, y presenta un esquema genérico demostrado con éxito para su solución.

Los patrones arquitectónicos son los que definen la estructura de un sistema software, los cuales a su vez se componen de subsistemas con sus responsabilidades. También tienen una serie de directivas para organizar los componentes del sistema, con el objetivo de facilitar la tarea del diseño del mismo. El uso de patrones permite la reutilización de soluciones arquitectónicas de calidad, facilita la documentación de diseños arquitectónicos y proporciona un vocabulario común que mejora la comunicación entre diseñadores.

1.2.3 Patrones de Diseño

Los patrones de diseño son soluciones a problemas específicos, basados en la experiencia y que se ha demostrado que funcionan. Estos patrones no son fáciles de entender, pero una vez entendido su funcionamiento, los diseños serán mucho más flexibles, modulares y reutilizables. Un patrón de diseño es una solución a un problema de diseño no trivial que es efectiva y reusable, lo que significa, que se puede aplicar a diferentes problemas de diseño en distintas circunstancias. Además, facilitan la comunicación entre diseñadores. Por otro lado, los patrones de diseño, facilitan el aprendizaje al programador inexperto, pudiendo establecer parejas problema-solución.

Dentro de las ventajas que nos proporcionan estos patrones se puede decir que los mismos son soluciones técnicas, se aplican en situaciones muy comunes, son soluciones simples, facilitan la reutilización de clases y del propio diseño.

1.3 Estado del arte de la Arquitectura de Software

Dentro del extenso mundo de la Arquitectura de Software, en la actualidad existe una amplia gama de estilos y patrones arquitectónicos así como patrones de diseño, los mismos enmarcan todas las buenas prácticas que se utilizan para un buen desarrollo de aplicaciones informáticas. En la presente investigación no se pretende describirlos en su totalidad, sino apenas los más representativos y vigentes que responden a las características que debe cumplir el sistema.

1.3.1 Estilos arquitectónicos

Seguidamente se presentan los estilos arquitectónicos más utilizados, los cuales constituyen una generalización y abstracción de los patrones.

Estilos de Llamada y Retorno: Los sistemas basados en este estilo utilizan un programa principal que tiene el control del sistema y varios subprogramas que se comunican con éste mediante el uso de llamadas. Esta familia de estilos enfatiza el uso de la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala.

Estilo Modelo Vista Controlador (MVC): El estilo arquitectónico MVC separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. Para ello utiliza tres clases diferentes: Modelo, Vista y Controlador. Este estilo tiene soporte de vistas múltiples, dado que la vista se halla separada del modelo y no hay dependencia directa del

modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. Además, posee adaptación al cambio.

Estilo Arquitecturas en Capas: Garlan y Shaw [8] 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.

Una especialización muy usada de la arquitectura en capas es la arquitectura de tres capas donde se observan muy bien delimitadas las responsabilidades de cada funcionalidad en la aplicación. Las capas de la aplicación pueden residir tanto en el mismo nodo físico como en nodos separados.

Estilos Centrados en Datos: Esta familia enfatiza en la integralidad, escalabilidad y la modificabilidad de los datos almacenados. Se aplica en sistema en los cuales cierto número de clientes acceden y actualizan los datos compartidos en un repositorio de manera frecuente.

1.3.2 Patrones arquitectónicos

Los patrones arquitectónicos expresan una organización estructural para un sistema. A continuación se muestran algunos de los más usados.

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: El *modelo* en el que se administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado y responde a instrucciones de cambiar el estado. La *vista* la cual maneja la visualización de la información. Y el *controlador* que interpreta las acciones del *mouse* (ratón) y el teclado, informando al modelo y/o a las vistas para que cambien según resulte apropiado. [9]

Patrón arquitectónico Arquitectura en Capas: Este patrón define cómo organizar el modelo de diseño a través de capas, que pueden estar físicamente distribuidas, lo que significa que los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores. Además, dicho patrón simplifica la comprensión y la organización del desarrollo de sistemas complejos, reduciendo las dependencias de forma que las capas más bajas no son conscientes de ningún detalle o interfaz de las superiores.

1.3.3 Patrones de Diseño

Es recomendable realizar un estudio de los distintos patrones de diseño teniendo en cuenta la estrategia que se pretenda seguir arquitectónicamente. A continuación se especifican algunos de ellos.

Patrón de diseño Fachada: Este patrón 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 las clases. Si estas cambian o se sustituyen por otras solo hay que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones clientes. Fachada no oculta las clases sino que ofrece una forma más sencilla de acceder a ellas.

Patrón de diseño Inyección de Dependencia: 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.

Patrón de diseño Bajo Acoplamiento: Es uno de los patrones de asignación de responsabilidades o GRASP. Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y disminuyendo la dependencia entre las clases.

Patrón de diseño Alta Cohesión: Mantiene la complejidad dentro de límites manejables, lo que significa que asigna una responsabilidad de modo que la cohesión siga siendo alta. 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.

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. El DAO oculta completamente los detalles de implementación de la fuente de datos a sus clientes. Como la interfaz 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. [10]

1.3.4 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 [7], que proponen el desarrollo de arquitecturas de software como un sistema de patrones, y distintos niveles de abstracción.

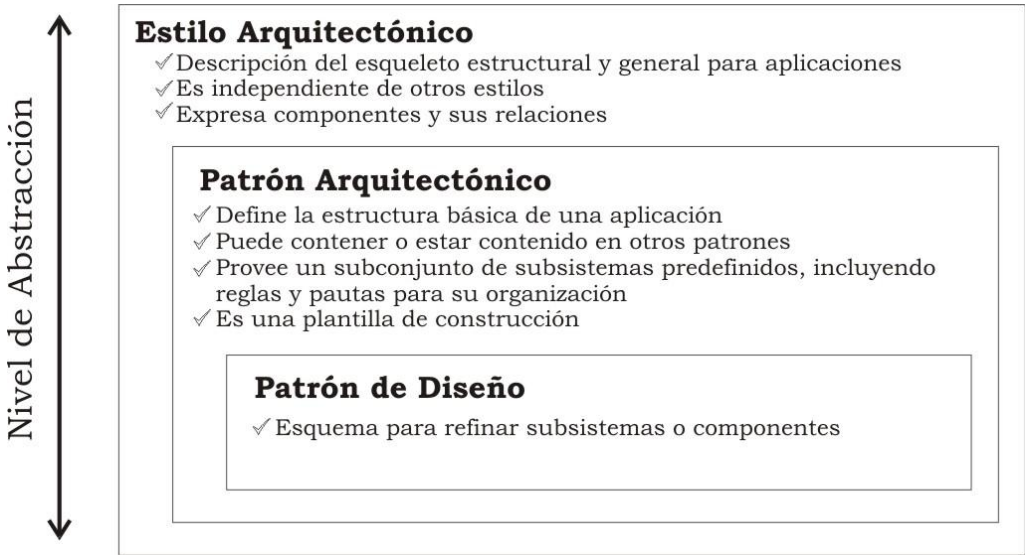

Figura 1. Relación de abstracción entre estilos y patrones.

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. [11]

1.4 Atributos de calidad en la Arquitectura de Software

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 de 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. [7]

Conociendo que se entiende por atributo de calidad, se hace necesario determinar cuáles deben alcanzarse en la Arquitectura de Software y que serán el centro de posterior evaluación para valorar en qué medida se logró el cumplimiento de estos atributos en la arquitectura propuesta. Es primordial tener en cuenta que no puede alcanzarse la satisfacción de ciertos atributos de calidad de manera aislada.

En su mayoría, los atributos de calidad se pueden organizar y descomponer de maneras diferentes, en lo que se conoce como modelos de calidad. Los modelos de calidad de software facilitan el entendimiento del proceso de la ingeniería de software. [7]

Para establecer los atributos de calidad que debe lograr la Arquitectura de Software con la tecnología JEE, fue preciso regirse por los modelos de calidad, ya que los factores que afectan a la calidad del software no cambian, por lo que es importante el estudio de los modelos de calidad que han sido propuestos en este sentido. Se analizarán algunos de los modelos más importantes como: McCall, ISO/IEC 9126 e ISO/IEC 9126 adaptado para arquitecturas de software.

McCall: El modelo de McCall describe la calidad como un concepto elaborado mediante relaciones jerárquicas entre factores de calidad, en base a criterios y métricas de calidad. [7]

En el modelo de McCall, los factores de calidad se agrupan en tres aspectos importantes de un producto de software: características operativas, capacidad de cambios y adaptabilidad a nuevos entornos. En este modelo, el término factor de calidad define características clave que un producto debe presentar. Los atributos del factor de calidad que define el producto son los nombrados criterios de calidad. Las métricas de calidad denotan una medida que puede ser utilizada para medir los criterios. Las métricas desarrolladas están relacionadas con los factores de calidad y la relación que se establece se mide en función del grado de cumplimiento de los criterios.

ISO/IEC 9126: Este estándar está pensado para los desarrolladores, personal que asegure la calidad y evaluadores independientes, y para los responsables de especificar y evaluar la calidad del producto software. Puede servir para validar la completitud de una definición de requisitos, identificar requisitos de calidad de software, objetivos de diseño y prueba y criterios de aseguramiento de la calidad.

Ha sido desarrollado en un intento de identificar los atributos clave de calidad para un producto de software. [12] Este estándar identifica seis características básicas de calidad que pueden estar presentes en cualquier producto de software. ISO/IEC 9126 provee una descomposición de las características en sub-características. Es interesante destacar que los factores de calidad que contempla el estándar ISO/IEC 9126 no son necesariamente usados para mediciones directas, 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 [13] proponen una adaptación del modelo ISO/IEC 9126 de calidad de software. El modelo se basa en los atributos de calidad que se relacionan directamente con la arquitectura: funcionalidad, confiabilidad, eficiencia, mantenibilidad y portabilidad.

Para alcanzar un atributo de calidad, es necesario tomar decisiones de diseño arquitectónico que requieren un pequeño conocimiento de funcionalidad. Cada decisión incorporada en una Arquitectura de Software puede afectar potencialmente los atributos de calidad. En la presente investigación se decidió usar los atributos definidos por el modelo ISO/IEC 9126 adaptado para arquitecturas de software, que deben alcanzarse con la arquitectura seleccionada y que serán usados en la evaluación de la arquitectura, además fueron seleccionados los mismos por estar en correspondencia con las necesidades del sistema.

1.5 Tecnologías y Herramientas

1.5.1 Plataforma de desarrollo

Según los atributos de calidad que regirán la evolución de la arquitectura se han seleccionado aquellas tecnologías y herramientas que permiten darle cumplimiento a los mismos. Por lo que como plataforma de desarrollo se utilizará JEE, la misma es seleccionada ya que es utilizada por gran cantidad de proyectos tanto a nivel nacional como internacional, incluyendo algunos de los proyectos del MPPRIJ. Además, en la UCI existe una consultoría de JEE que trabaja con esta plataforma.

La plataforma JEE ha sido diseñada para aplicaciones distribuidas con base en componentes o unidades funcionales de software que interactúan entre sí para formar parte de una aplicación empresarial JEE. Un

componente de esta plataforma debe formar parte de una aplicación y ser desplegado en un contenedor, o sea, en la parte del servidor JEE que le ofrece al componente ciertos servicios de bajo nivel y de sistema, tales como seguridad, manejo de concurrencia, persistencia y transacciones.

La plataforma JEE define un modelo de programación encaminado a la creación de aplicaciones basadas en n-capas. La lógica de la aplicación se divide en componentes de diferentes funciones que componen una aplicación JEE y que están distribuidos en dependencia de la capa en el ambiente multicapa JEE al cual la aplicación pertenece.

La plataforma ofrece un conjunto de Application Programming Interface (APIs) de Java para construir aplicaciones, las cuales definen un modelo de programación para las aplicaciones JEE. También ofrece una infraestructura de ejecución para albergar y gestionar aplicaciones. Entre las APIs utilizadas se destacan Java Server Pages (JSP) 2.1, Servlet 2.5, Java Database Connectivity (JDBC) 3.0, Java Servlet 2.5 y Java Persistence API (JPA) 2.0. Además, como contenedores que representan los períodos de ejecución para gestionar los componentes del sistema se encuentran JEE 6 y Apache Tomcat 6.0.20.

1.5.2 Framework de desarrollo

Luego de establecer la plataforma de desarrollo es importante el estudio de los frameworks que acompañarán la implementación, pero antes es necesario mencionar que son los componentes y los componentes de software.

Un componente es un bloque de construcción reusable de software: una pieza de código encapsulada de una aplicación que puede ser combinada con otros componentes y con códigos adicionales para producir una aplicación específica. Los componentes pueden ser simples o complejos. No existe un acuerdo general sobre qué es o qué no es un componente. Un componente puede ser muy pequeño, como lo es un GUI widget (por ejemplo, un botón), o éste puede implementar un complejo servicio de una aplicación, tal como una función para administrar la red. [14]

Una aplicación está compuesta por una colección de componentes. Las aplicaciones proveen un ambiente o un esqueleto de código, dentro del cual los componentes son insertados. Los componentes interactúan con su ambiente y pudieran interactuar con el sistema operativo de la computadora sobre la cual ellos son ejecutados. Cambiar una interfaz de un componente pudiera requerir cambiar el componente que usa esta interfaz. Sin embargo, cambiar la implementación del componente no debería requerir cambiar a los clientes de ese componente. [14]

Un componente es una parte no trivial, casi independiente, y reemplazable de un sistema que llena claramente una funcionalidad dentro de un contexto en una arquitectura bien definida. Un componente se conforma y provee la realización física por medio de un conjunto de interfaces. [15]

Un componente de software es un independiente paquete entregable de servicios de software, en el mayor sentido general. Un componente de software es una unidad de composición con interfaces contractualmente especificadas y explícitas sólo con dependencias dentro de un contexto. Un componente de software puede ser desplegado independientemente y es sujeto a la composición de terceros. [16]

El interés en reutilizar software ha sido cambiado de la reutilización de componentes simples a diseño de sistemas enteros o estructuras de aplicaciones. Un sistema software que pudiera ser reutilizado en este nivel para la creación de aplicaciones completas es llamado framework.

Los frameworks son basados en la idea que deberían permitir la producción fácil de un conjunto de sistemas específicos pero similares, dentro de un cierto dominio comenzando desde una estructura genérica. Brevemente, los frameworks son arquitecturas genéricas integradas por un extensible conjunto de componentes. Además, los frameworks pueden contener subframeworks los cuales representen subconjuntos de componentes de un sistema más grande. [14]

Un framework contiene clases o estructuras que implementan los componentes de una aplicación genérica también como componentes concretos que satisfacen tareas especializadas. Para desarrollar programas completos, los desarrolladores buscan e instancian los componentes apropiados. No hay una definición común para los frameworks, pero la mayoría tienen un tema común: la reutilización. Los framework que se utilizan y que son descritos a continuación permitirán ofrecer al sistema una alta confiabilidad y extensibilidad.

Framework Dalas 0.2

En la UCI debido a que diferentes proyectos ya elaborados habían utilizado tecnologías similares pero con arquitectura y modelos de trabajo distintos, no se había permitido la gestión del conocimiento en base a la experiencia adquirida ni se habían podido reutilizar los componentes desarrollados. Es por esto que el Centro de la Consultoría JEE, revisó el desarrollo de todos los proyectos elaborados sobre JEE en la universidad y tomó de ellos las mejores experiencias, definiendo entonces que DALAS era el framework base a seguir por los nuevos proyectos que desarrollan sobre JEE. Además, como Dalas no está en

contradicción con los requerimientos no funcionales y los atributos de calidad identificados para el sistema y la arquitectura de desarrollo a seguir, es que se escoge Dalas como uno de los framework para el desarrollo del sistema.

Dalas también ha sido seleccionado por ser este desarrollado en el Centro de Consultoría de la UCI y contar con buena documentación. Además, se cuenta con bastante experiencia por parte de los proyectos de la universidad que lo utilizan. Dalas ha sido utilizado por ejemplo en el Sistema de Gestión Penitenciaria Venezolano (SIGEP), así como en Reko (Réplica de Datos JEE). Dalas constituye un framework base de arquitectura para las aplicaciones que utilizan Spring Framework, que actúa como integrador y manejador de la aplicación y fundamenta su funcionamiento bajo la definición de estándares en todas las capas durante el proceso de desarrollo. Este framework permite la orquestación de una solución a través de la integración de componentes reutilizables y la gestión de módulos de negocio. La reducción del consumo de memoria es uno de los principales aportes del framework. Dalas responde a los patrones básicos de diseño Bajo acoplamiento y Alta cohesión.

Se utilizan para el desarrollo del sistema otros frameworks que también han sido propuestos por el Centro de la Consultoría JEE para así unificar el desarrollo sobre la universidad, estos framework no contradicen los requerimientos del sistema.

A partir de la organización de arquitectura a través del estilo arquitectónico definido, se utilizan estos frameworks distribuidos en las 3 capas del sistema con funcionalidades bien definidas y componentes implementados para cada una de estas, de la siguiente manera:

Tabla 1: Frameworks utilizados.

Capa de presentación	Capa de lógica de negocio	Capa de acceso a datos	Framework de soporte
Framework Spring MVC Framework Dojo 1.4.2	Spring Framework 3.0.2	Framework Hibernate 3.5	Spring Security 3.0.2. Hibernate Validator 4.0.1

A continuación se describen cada uno de los frameworks antes mencionados de forma sintetizada.

Framework Spring MVC

Spring MVC es uno de los módulos del framework de Spring e implementa una arquitectura Modelo - Vista - Controlador. Este framework presenta una lógica de diseño bastante sencilla y cuenta con todo el conjunto de librerías de Spring Framework. Dicho framework ofrece una división limpia entre

Controladores, Modelos (JavaBeans) y Vistas. Es muy flexible, ya que implementa toda su estructura mediante interfaces. Además, todas las partes del framework son configurables vía plugin en la interfaz. Provee interceptores también como controllers que permiten factorizar el comportamiento común en el manejo de múltiples requests. No obliga a utilizar JSP. Además, los controllers de Spring MVC se configuran mediante Inversión de Control como los demás objetos, lo cual los hace fácilmente testeables e integrables con otros objetos que estén en el contexto de Spring.

Framework Dojo 1.4.2

Dojo es un toolkit open source DHTML (*Dynamic HTML*) escrito en JavaScript. Su idea es la de abstraer al desarrollador de las complejidades del DHTML y de las discrepancias existentes entre navegadores, que hacen que el código JavaScript a utilizar sea diferente. Es un framework que contiene APIs y widgets (controles) para facilitar el desarrollo de aplicaciones Web. Los componentes que presenta este framework permiten mejorar la usabilidad y modificabilidad del sistema. Posee múltiples puntos de entrada, lo que significa que se puede empezar a utilizar al nivel que se desee. Además asegura soporte para el mayor número de plataformas y se encuentra bajo las licencias Berkeley Software Distribution (BSD) y AFL.

Entre los aspectos fundamentales de este framework se destacan que maneja incompatibilidades entre navegadores, posee soporte AJAX, oculta el manejo del XMLHttpRequest e incorpora un sistema de eventos orientado a aspectos. [17]

Spring Framework 3.0.2

Es un framework de aplicación de código abierto que ayuda a hacer el desarrollo en JEE mucho más fácil. El mismo interviene en todas las capas arquitectónicas de una aplicación JEE. Spring es modular, lo que significa que pueden utilizarse solo aquellas partes que se necesiten, sin necesidad de recurrir al resto. El mismo se basa en la Inversión de Control, Inyección de Dependencia y el trabajo con POJOs.

Esta versión del framework posee soporte para JSR 303 (Validación Bean). Cuenta con una documentación de referencia bastante completa. Además, se basa en Java 5 y es totalmente compatible con Java 6, JEE 1.4 y Java EE 6. Se encuentra bajo la licencia de apache 2.0.

Dentro de las características de esta versión de Spring Framework se destacan:

- **Lenguaje de Expresión de Spring:** Spring introduce un lenguaje de expresión que es similar a Unified EL en su sintaxis, pero que ofrece características más significativas. El lenguaje de expresión se puede utilizar cuando se están definiendo XML y anotaciones basado en definiciones de bean.

- **Mejoras en la Inversión de Control/Java basado en metadatos bean:** Algunas de las características fundamentales del proyecto JavaConfig se han añadido a Spring Framework. Por tanto, las siguientes anotaciones son directamente soportadas: @Configuration, @Bean, @Import, @ImportResource y @Value.
- **Sistema de conversión de tipo de propósito general y sistema de formato de campos:** Un formateador SPI ha sido introducido para dar formato a valores de campo. Esta SPI proporciona una alternativa más simple y robusta a PropertyEditors de JavaBean para su uso en entornos de cliente como Spring MVC.
- **Adiciones @MVC:** Se ha introducido un espacio de nombre mvc que simplifica enormemente la configuración de Spring MVC. Anotaciones adicionales se han añadido, tales como @CookieValue y @requestHeaders.
- **Soporte temprano para Java EE 6:** Se proporciona soporte para llamadas a métodos asincrónicos mediante el uso de la nueva anotación @Async (o en el caso de EJB 3.1 la anotación @Asynchronous). JSR 303, JSF 2.0 y JPA 2.0.
- **Soporte para base de datos embebidas:** Se proporciona soporte favorable para los motores embebidos de bases de datos en Java, incluyendo HSQL, H2, y Derby.

JPA 2.0

JPA es el estándar de Java para la persistencia, es decir, para objetos que guardan su estado en una base de datos. Es un marco ligero, basado en POJO de persistencia Java. Aunque el mapeo objeto-relacional es un componente principal del API, este además ofrece soluciones a los retos arquitectónicos de la integración de la persistencia en aplicaciones empresariales escalables.

Algunas de las características presentes en la versión 2.0 son: funciones adicionales de asignación, formas flexibles para determinar la forma en que el proveedor de acceso a la entidad estatal, la ampliación de la *Java Persistence Query Language* (QL JP), y un criterio de orientación a objetos de Java API para la creación dinámica de consultas.

El modelo de JPA es simple y elegante, potente y flexible. Es fácil de aprender, especialmente si se ha utilizado alguno de los productos de persistencia existentes en el mercado hoy en los cuales la API fue basada. Además, está implementado con Hibernate 3.5 y con anotaciones.

Hibernate Framework 3.5

Hibernate es un potente framework de mapeo objeto/relacional y servicio de consultas para Java. Es la solución ORM más popular en el mundo Java. Es una capa de persistencia objeto/relacional y un generador de sentencias SQL. Es open source y soporta la mayoría de los sistemas gestores de base de datos SQL y se integra de manera elegante y sin restricciones con los más populares servidores de aplicaciones JEE y contenedores web. Cada clase persistente necesita un fichero XML de mapeo. Posee una Caché, la cual puede aumentar significativamente el rendimiento de una aplicación, sobre todo en aquellas en las que leen muchos datos. Hibernate implementa la gestión de la API de la persistencia Java y el mapeado objeto-relacional.

Esta versión del framework incorpora una nueva arquitectura Interceptor/Callback, filtros definidos por el usuario, y el uso de anotaciones para definir la correspondencia conjuntamente con los archivos XML. Además permite especificar SQL escrito a mano, incluyendo procedimientos almacenados para todas las operaciones *create, update, delete y load.* Hibernate se puede administrar por medio de un servicio estándar JMX. JMX es el estándar JEE para la gestión de componentes Java. Este framework utiliza *Simple Logging Facade for Java* (SLF4J) con el fin de registrar varios eventos del sistema. SLF4J puede direccionar la salida de registro a varios marcos de trabajo de registro dependiendo del enlace escogido. Soporta el formato del mapeado de XML, diseñado para ser editado a mano y el mapeado basado en anotaciones. Además, de validación basada en anotaciones.

JSR 303

JSR 303 (Validación Bean), da la posibilidad de no tener que escribir múltiples veces las reglas de validación en las distintas capas de la aplicación, lo que hace posible ahorrar tiempo, simplificar el desarrollo y reducir el porcentaje de errores. Estandariza la declaración, definición y validación de restricción para la plataforma Java. Además, define un modelo de metadatos y API para validación de JavaBeans. El origen por defecto de los metadatos son las anotaciones, con la capacidad de desactivar y extender el metadato a través del uso de los descriptores de validación XML. Permite mediante anotaciones, especificar que validaciones queremos hacer sobre los beans. JSR 303 estandariza la validación, restricciones, declaración y los metadatos para la plataforma Java.

Hibernate Validator 4.0.1

Es una implementación del JSR 303 (Validación Bean), que permite definir validaciones usando XML y anotaciones, validación completa y recursiva de objetos, definición a medida de restricciones y validadores así como personalización de mensajes de error.

Dentro de las características de Hibernate Validator se destacan que: las validaciones son definidas con anotaciones lo cual facilita este proceso. Trae un conjunto predefinido de validaciones típicas. El sistema soporta internacionalización: trae mensajes de error traducidos a diez idiomas. Estos mensajes se pueden cambiar fácilmente, solo escribiendo un fichero de propiedades y sobre escribiendo los mensajes que interesen. Además se integra directamente con Hibernate, y en general con cualquier ORM, de forma que antes de hacer una inserción o actualización, se validarán los objetos.

Otros framework de apoyo, que contribuyen a un mejor desarrollo del sistema son:

Framework Spring Security 3.0.2

Spring Security es un framework que provee las declaraciones de seguridad de una aplicación basada en Spring. Ofrece una completa solución de seguridad, la manipulación de autenticación y autorización en la solicitud web y en la invocación de método. Es una API open source.

Cuando se aseguran aplicaciones web, Spring Security utiliza filtros Servlet que interceptan las solicitudes Servlet para llevar a cabo la autenticación y hacer cumplir la seguridad. Spring Security es capaz de gestionar seguridad en varios niveles: URLs que se solicitan al servidor, acceso a métodos y clases Java, y acceso a instancias concretas de las clases. Permite separar la lógica de las aplicaciones del control de la seguridad, utilizando filtros para las peticiones al servidor de aplicaciones o aspectos para la seguridad en clases y métodos. Además, la configuración de la seguridad es portable de un servidor a otro, ya que se encuentra dentro del WAR o el EAR de las aplicaciones.

Framework JasperReport 3.5.2

JasperReport es un framework bastante completo para desarrollar reportes tanto web como desktop en Java. Es una herramienta gratuita y open source que se compone de un conjunto de librerías Java para facilitar la generación de informes en las aplicaciones. Los informes se definen en un fichero xml el cual será compilado por las librerías jasper report y generarán un fichero .jasper que se utilizarán para rellenar y mostrar el informe final. Esta herramienta está completamente escrita en Java y puede usarse en una

gran variedad de aplicaciones, incluyendo JEE y aplicaciones WEB con la opción de generar contenido dinámico. La definición de los reportes creados por JasperReport persiste en un estándar de Web abierto basado en un formato XML conocido como JRXML.

1.5.3 Entorno de Desarrollo Integrado

Un entorno de desarrollo integrado o en inglés *Integrated Development Environment* (IDE) es un programa compuesto por un conjunto de herramientas para un ordenador. Generalmente incluyen en una misma suite un buen editor de código, enlace transparente a compiladores y debuggers e integración con sistemas controladores de versiones o repositorios. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Eclipse Galileo 3.5

El entorno de desarrollo Eclipse (también sus plugins) está desarrollado completamente en Java. Emplea módulos (plugins) para adicionar funcionalidades según las necesite el desarrollador. Para la gestión de la configuración y el control de versiones tiene soporte para CVS y Subversion. Además incluye plugins para realizar pruebas de unidad.

El IDE Eclipse es multiplataforma. Permite compilación incremental de código. Modifica e inspecciona valores de variables. Además, avisa de los errores cometidos mediante una ventana secundaria y depura código que resida en una máquina remota. Eclipse constituye una de las mejores herramientas para el desarrollo de aplicaciones libres.

Este IDE de desarrollo ha sido el seleccionado por ser totalmente libre y además contar con un robusto mecanismo orientado a plugins, que permite optimizar la calidad del desarrollo y reducir su tiempo de ejecución. No está en contradicción con los atributos de calidad de la arquitectura.

El IDE de desarrollo seleccionado contará con la inclusión de los plugins Web Tools Platform (WTP), Spring IDE, Hibernate Tools y Subclipse para el desarrollo del sistema. Seguidamente se plantean de manera breve algunos de los aspectos fundamentales de cada uno de estos plugin.

Web Tools Platform 3.0.5

La plataforma de herramientas web de Eclipse o Eclipse *Web Tool Platform* (WTP) provee varias APIs para desarrollo de aplicaciones sobre la Web y JEE. Éstas incluyen editores gráficos, de código fuente para una variedad de lenguajes, asistentes y aplicaciones incorporadas para simplificar el desarrollo de

servicios web, además de herramientas y APIs para soportar el despliegue, ejecución y prueba de aplicaciones. WTP soporta integración con servidores Web dentro de Eclipse como ambiente de ejecución de primera clase para aplicaciones web. También incluye la configuración de servidores y su asociación con los proyectos web, permitiendo la depuración sobre el servidor de los recursos y las clases. Se encuentra bajo los términos de la licencia *Eclipse Public License* (EPL).

Spring IDE 2.2.1

Spring IDE es un plugin que sirve como interfaz de usuario gráfica para la configuración de los archivos usados por Spring Framework. Permite el completamiento de etiquetas, valores de atributos y elementos en estos archivos de configuración. Es un plugin que ayudará a desarrollar con Eclipse aplicaciones que utilicen Spring. Muestra un árbol con todos los proyectos de Spring y sus archivos de configuración. Permite hacer búsquedas de beans en dichos archivos. Dentro de las funcionalidades básicas del plugin están que permite crear proyectos con naturaleza Spring, se puede indicar en qué ficheros xml van a estar declarados los bean de Spring y se pueden crear fácilmente. Además, se puede ver qué dependencias hay entre beans. Propiciando todo esto que el uso del plugin sea más sencillo.

Hibernate Tools 3.2.4

Es un conjunto de utilidades para Eclipse que facilitan el uso de Hibernate. Permiten generar gran parte del código necesario para el acceso a datos: los POJOs así como los ficheros de configuración.

Este plugin posee las siguientes características:

- **Editor de mapeos:** Es un editor para los archivos de mapeo XML de Hibernate, soportando autocompletamiento y sintaxis resaltada. Soporta incluso autocompletamiento semántico para nombres de clases, propiedades, tablas y columnas.
- **Consola:** La perspectiva de consola de Hibernate permite configurar conexiones a base de datos, provee visualización de clases y sus relaciones. Además, admite ejecutar preguntas en formato del lenguaje de preguntas de Hibernate (HQL) interactivamente contra la base de datos y mostrar los resultados.
- **Ingeniería inversa:** Permite generar las clases del modelo de dominio y los archivos de mapeo de Hibernate, documentación en formato HTML, a partir de una base de datos.

Subclipse 1.6.5

Subclipse es un plugin para Eclipse que adiciona integración para el control de versiones, permitiendo operaciones de sincronización y actualización. Permite bloqueos a recursos para que otros usuarios no puedan modificarlos. Con este plugin se consigue integrar las funcionalidades de Subversion en Eclipse. Muestra una vista del historial de versiones de los recursos con un conjunto de atributos de las acciones realizadas sobre el recurso. Soporta conectarse a varios repositorios de control de versiones al mismo tiempo, permitiendo hacer operaciones sobre el repositorio directamente. Es un plugin muy útil para el desarrollo colaborativo, en el que intervienen un conjunto de desarrolladores trabajando sobre el mismo proyecto, poniendo a disposición del equipo de desarrollo facilidades para el trabajo en equipo.

1.5.4 Servidor Web

Básicamente, un servidor web envía contenido estático a un navegador, carga un archivo y lo envía a través de la red al navegador de un usuario. El servidor Web es un programa que se ejecuta sobre el servidor que escucha las peticiones HTTP que le llegan y las satisface. Dependiendo del tipo de la petición, el servidor Web buscará una página Web o bien ejecutará un programa en el servidor.

Apache Tomcat 6.0.20

Para hospedar la aplicación se utilizará como servidor web a Apache Tomcat. El mismo es un software de código abierto implementado para las tecnologías Java Servlet y JSP. Apache Tomcat es desarrollado en un entorno abierto y participativo. Se encuentra publicado bajo la licencia del software de Apache.

El mismo es seleccionado por ser libre. Se ejecuta en varios Sistemas Operativos. Es un servidor configurable de diseño modular, con diversidad que permiten garantizar una elevada seguridad y buenas prestaciones. Además, brinda soporte para la plataforma de desarrollo JEE. Existe bastante documentación asociada al mismo, cuenta con una gran comunidad de desarrollo y es uno de los más usados internacionalmente.

1.5.5 Sistema Gestor de Base de Datos

Un Sistema Gestor de Base de Datos (SGBD) es una aplicación que permite a los usuarios definir, crear y mantener la base de datos. Proporciona los servicios de 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.

Teniendo en cuenta que se debe cumplir que cada una de las herramientas y tecnologías a utilizar deben ser software libres. Y debido a que la cantidad de información que se maneja para el desarrollo del sistema es muy grande. Además, dicha información 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 para decidir cuál de los SGBD existentes se utilizará, el balance será hecho entre algunos de los SGBD como MySQL y PostgreSQL.

MySQL 5.1.33

Existen diferentes arquitecturas para los sistemas de gestión de base de datos, pero la más extendida y la que más éxito ha tenido, es la arquitectura relacional MySQL. El mismo proporciona un servidor de base de datos SQL (*Structured Query Language*) muy rápido, multi usuario y robusto.

Este SGBD es, probablemente, el gestor más usado en el mundo del software libre. Esta gran aceptación es debida, en parte, a que existen infinidad de librerías y otras herramientas que permiten su uso a través de gran cantidad de lenguajes de programación, además de su fácil instalación y configuración. Este gestor se creó con la rapidez en mente, de modo que no tiene muchas de las características de los gestores comerciales más importantes.

Entre las características de MySQL se destacan: es multiplataforma, suministra 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. Además, los clientes pueden conectarse con el servidor MySQL usando sockets TCP/IP en cualquier plataforma. [18]

PostgreSQL 8.4

Es un sistema objeto-relacional ya que incluye características de la orientación a objetos como pueden ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. PostgreSQL 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.

Algunas de las principales características que presenta este SGBD son: 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) y cadenas de bits. También permite la creación de tipos propios. Alta concurrencia, mediante un sistema denominado Control de Concurrencia para Múltiples Versiones (MVCC). PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo commit. Incorpora una estructura de datos array y funciones de diversa índole: manejo de fechas, geométricas y orientadas a operaciones con redes. Soporta el uso de índices, reglas y vistas. Incluye herencia entre tablas por lo que a este SGBD se le incluye entre los gestores objeto-relacionales. Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos. [19]

Al finalizar la comparación se llegó a la conclusión de utilizar PostgreSQL como SGBD debido a su condición de ser multiplataforma. Por estar considerado como uno de los gestores de base de datos de código abierto más avanzado. Cuenta con la licencia BSD. Suministra interfaces nativas para el acceso desde múltiples estándares y tecnologías. Soporta además integridad referencial, la cual es utilizada para garantizar la validez de los datos. Tiene herramientas gráficas de diseño y administración de bases de datos. Posee gran escalabilidad y por tanto puede soportar una mayor cantidad de peticiones concurrentes. Además, tiene la capacidad de almacenar procedimientos en la propia base de datos.

1.5.6 Herramienta para el Control de Versiones

En el proceso de desarrollo de software actual, las aplicaciones requieren un gran número de desarrolladores trabajando de una forma concurrente, en una colección variada de archivos durante un largo período de tiempo. Lo que se hace necesario llevar un control estricto de los cambios que se van realizando en los diferentes componentes del software. Es necesario que los cambios hechos por estos desarrolladores sean rastreados con el fin de conocer el responsable de cada cambio.

Una buena herramienta para el control de las versiones ayuda a proteger la integridad de los datos, manteniendo una historia de revisión. 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, control y la combinación de versiones diferentes de un proyecto con el paso del tiempo.

Subversion y CVS

Subversion es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS, el cual posee varias deficiencias. Subversion es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como svn. Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo.

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. 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 comience. Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores.

En Subversion SVN, se sigue la historia de los archivos y directorios a través de copias y renombrados, las modificaciones (incluyendo cambios a varios archivos) son atómicas. Además, se envían sólo las diferencias en ambas direcciones, se maneja eficientemente archivos binarios. Subversion SVN permite selectivamente el bloqueo de archivos y cuando se usa integrado a Apache permite utilizar todas las opciones que este servidor provee al autentificar archivos.

Para tener una idea más general de las diferentes tecnologías y herramientas analizadas anteriormente, estas se pueden ver en: "*Documentos Complementarios (Matriz tecnológica de Capas y Niveles)".*

1.6 Metodología de desarrollo

Las metodologías de desarrollo de software especifican cómo se debe dividir un proyecto en etapas, qué tareas se llevan a cabo en cada etapa, qué salidas se producen y cuándo se deben producir. Además, las metodologías especifican qué restricciones se aplican, qué herramientas se van a usar y cómo se

gestionará y controlará un proyecto. Constituyen un conjunto de actividades necesarias para transformar los requisitos de los usuarios en un sistema software.

La metodología de desarrollo de software se divide en dos grandes grupos: metodologías pesadas y metodologías ágiles o ligeras. Algunas de las metodologías pesadas más empleadas en el mundo de la producción de software son: *Rational Unified Process* (RUP) y *Microsoft Solution Framework* (MSF). Por su parte las metodologías ágiles más utilizadas son: eXtreme Programming (XP), Scrum, Adaptive Software Development (ASD) y Familia de Metodologías Crystal entre otras.

Microsoft Solution Framework (MSF): Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo, dejando en un segundo plano las elecciones tecnológicas. Esta metodología se compone de varios modelos, encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto, estos modelos son: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el Modelo de Aplicación. [20]

Extreme Programming (XP): La metodología XP está centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP está guiada por una rápida programación y se basa en realimentación continua entre el cliente y el equipo de desarrollo. Se basa además en la comunicación fluida entre todos los participantes, en la reutilización de código así como en la realización de pruebas a los principales procesos. La metodología XP se define como adecuada para proyectos de corto plazo con requisitos imprecisos y muy cambiantes, donde existe un alto riesgo técnico.

Rational Unified Process (RUP) Proceso Unificado de Desarrollo: RUP es un proceso para el desarrollo de un proyecto de un software, que define claramente *quién, cómo, cuándo y qué* debe hacerse en el proyecto. El mismo presenta 3 características fundamentales; centrado en la arquitectura, iterativo e incremental y dirigidos por casos de uso. RUP toma en cuenta las mejores prácticas en el modelo de desarrollo de software. En particular, desarrollo de software en forma iterativa, manejo de requerimientos, utiliza arquitectura basada en componentes, modela el software visualmente (modela con UML), verifica la calidad del software y controla los cambios. RUP, divide en 4 fases el desarrollo del software: fase de

Inicio con el objetivo de determinar la visión del proyecto, fase de Elaboración con el objetivo de determinar la arquitectura óptima, fase de Construcción con el objetivo de obtener la capacidad operacional inicial y la fase de Transición con el objetivo de obtener el release del producto.

Esta metodología 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. [4]

En sentido general se puede expresar que: la Metodología RUP es más adaptable para proyectos de largo plazo. XP se recomienda para proyectos de corto plazo. Sin embargo, MSF se adapta a proyectos de cualquier dimensión y de cualquier tecnología.

Después de analizadas algunas de las metodologías utilizadas en el mundo de la producción de software, cada una con sus particularidades, para elaborar el sistema que nos ocupa se seleccionó la metodología RUP pues es la más adecuada, dada la inmensidad del sistema y la complejidad del mismo. Además, la metodología RUP está bien documentada, ejemplificada y el equipo de desarrollo se encuentra bien capacitado en dicha metodología.

1.6.1 Responsabilidades del arquitecto de software

El arquitecto constituye un rol crítico en los proyectos, se enfoca en la calidad de servicio y lidera el proceso de definición y la implementación de la arquitectura. Reutiliza arquitecturas exitosas y frameworks. Este debe poder comunicarse con todos los implicados en el proceso de desarrollo de software, tener excelentes habilidades de diseño, tecnología y un conocimiento de las mejores prácticas de ingeniería del software. El arquitecto adapta el sistema a las restricciones, establece una estructuración correcta del sistema utilizando un conjunto de estrategias, herramientas y patrones de diseño. El principal reto del arquitecto de software es asegurar el éxito del proyecto diseñando las bases de la aplicación.

RUP establece que arquitecto es un rol en un proyecto de desarrollo de software el cual es responsable de liderar el proceso de arquitectura, producir los artefactos necesarios como: documento de descripción de arquitectura, modelos y prototipos de arquitectura.

También se conoce que el arquitecto visualiza el comportamiento del sistema y crea los planos del mismo, define la forma en la cual los elementos del sistema trabajan en conjunto y es el responsable de integrar los requerimientos no funcionales en el sistema. Además, los arquitectos deben conocer y ayudar a la implementación de la metodología, así como conocer a la perfección los requerimientos y restricciones.

1.6.2 Lenguaje y herramienta para el modelado

Existen notaciones que, a través de un conjunto definido de elementos y formas de representación, permiten establecer la descripción de una Arquitectura de Software, muestra de ello lo es el *Unified Modeling Language (*UML*).* El UML es un sistema de notación que se ha convertido en estándar en el mundo del desarrollo de sistemas.

UML ha obtenido un rol muy importante en el proceso de desarrollo de software. En UML existe soporte para algunos de los conceptos relacionados con la Arquitectura de Software, como los componentes, los paquetes, las librerías y la colaboración. Este lenguaje de modelado permite la descripción de componentes en la Arquitectura de Software en 2 niveles; especificar sólo el nombre del componente o especificar las clases o interfaces que implementan estos. Además UML suministra una notación para la descripción de la proyección de los componentes del software en el hardware, correspondiendo esto a la vista física del modelo 4+1. UML es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software.

El Lenguaje de Descripción de Arquitectura (ADL de sus siglas en inglés) es una notación desarrollada para la descripción de la arquitectura del software. Una de las características esperadas en un ADL, es la capacidad de brindar distintas vistas de una misma arquitectura compleja. Los elementos básicos de los ADLs son los componentes y conectores e incluyen reglas y recomendaciones para arquitecturas bien formadas. Sin embargo, como todos los lenguajes especializados, los ADLs solo pueden ser comprendidos por expertos del lenguaje y son complicados para los especialistas tanto de la aplicación como del dominio. Los criterios decisivos al escoger un lenguaje modelador, si es UML o un ADL, es que debería favorecer la comprensión de la arquitectura por los que la analizan.

Teniendo en cuenta lo planteado, se decidió utilizar el lenguaje de modelado UML, para describir la Arquitectura de Software, pues es el lenguaje propuesto por la metodología de desarrollo a utilizar, es la notación que dominan los diseñadores. Utilizar el mismo lenguaje permite mejor comprensión de lo que se realiza y se gana en tiempo en el aprendizaje de otro lenguaje para la comprensión de la arquitectura.

Existen varios software para el modelado UML, son las denominadas herramientas CASE (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Ordenador). Las herramientas CASE son las aplicaciones de tecnologías informáticas a las actividades, las técnicas y las metodologías propias de desarrollo. Su objetivo es acelerar el proceso para el que han sido diseñadas, automatizar o apoyar una o más fases del ciclo de vida del desarrollo de software.

Las herramientas CASE permiten a los analistas tener más tiempo para el análisis y diseño y minimizar el tiempo para codificar y probar. La principal ventaja de la utilización de estas herramientas es la mejora de la calidad de los desarrollos realizados y el aumento de la productividad. Entre las herramientas CASE para el modelado están las herramientas Enterprise Architect, Visual Paradigm y Rational Rose.

Enterprise Architect: es una herramienta comprensible de diseño y análisis UML, que cubre el desarrollo de software desde la captura de requerimientos a través de las etapas del análisis, modelos de diseño, pruebas y mantenimiento. Es una herramienta de multi-usuarios, diseñada para ayudar a construir software robusto y fácil de mantener. También Enterprise Architect provee trazabilidad completa desde el análisis de requerimientos y los artefactos de diseño, a través de la implementación y el despliegue. Proporciona una generación poderosa de documentos y herramientas de reporte con un editor de plantilla completo. Esta herramienta permite navegar y explorar el modelo de código fuente en el mismo entorno.

Rational Rose: Es una de las más poderosas herramientas de modelado visual para el análisis y diseño de sistemas basados en objetos. Se utiliza para modelar un sistema antes de proceder a construirlo. Da soporte al modelado visual con UML ofreciendo distintas perspectivas del sistema. Ofrece un diseño centrado en casos de uso y enfocado al negocio, hace uso de un lenguaje estándar, común a todo el equipo de desarrollo que facilita la comunicación. Rational Rose presenta un soporte UML incomparable así como desarrollo basado en componentes. Rational Rose no es gratuito, además presenta algunas limitantes que la hacen un poco débil en comparación a otras herramientas que cuentan con las mismas facilidades de modelado, estas debilidades consisten en la dependencia de la plataforma Windows y la integración solo con herramientas que estén en el mismo grupo de software propietario.

Visual Paradigm: provee soporte para la generación de código, ingeniería inversa para Java, se integra con Eclipse, para soportar la fase de implementación en el desarrollo de software. Es portable y posee gran facilidad de uso. Dicha herramienta ofrece un entorno de creación de diagramas para UML 2.0 así

como diseño centrado en casos de uso. Visual Paradigm es muy potente, gratuito, fácil de instalar, utilizar y actualizar. Permite exportación e importación XML, permite generar reportes y documentación en formato HTML, PDF y Microsoft Word. Además, contiene editor de figuras, integración con MS Visio, plug-in, integración IDE con Visual Studio, Eclipse y NetBeans. Entre otras de sus características se incluyen el modelado colaborativo con CVS y Subversion además de la interoperabilidad con modelos UML 2 a través de XML.

Se decidió utilizar Visual Paradigm-UML como herramienta de modelado para la descripción de la arquitectura, por ser una herramienta multiplataforma. Garantiza la calidad del software en todo el ciclo de vida, ya que permite la comunicación entre los desarrolladores mediante un lenguaje común para todos los roles que intervienen en el proceso de desarrollo del software. Además, el equipo de desarrollo goza de vasta experiencia con la herramienta.

1.7 Métodos de evaluación de la Arquitectura de Software

Existen varios métodos que permiten realizar la evaluación de la Arquitectura de Software, a continuación se analizarán algunos de ellos para seleccionar cuál emplear al evaluar la arquitectura propuesta. Se centrará el análisis fundamentalmente en los métodos: Revisiones Activas para Diseño Intermedio (ARID, por sus siglas en inglés *Active Reviews for Intermediate Design*), Método de Análisis de Acuerdos de Arquitectura (ATAM, por sus siglas en inglés *Architecture Tradeoff Analysis Method*) y Método de Análisis para Arquitecturas de Software (SAAM, por sus siglas en inglés *Software Architecture Analysis Method*).

SAAM: Este método está basado en escenarios, permite evaluar una arquitectura o evaluar y comparar varias. Fue creado con el propósito de analizar la modificabilidad de una arquitectura, pero hoy en día es muy útil para evaluar rápidamente atributos de calidad tales como modificabilidad, portabilidad e integrabilidad. SAAM contribuye a analizar las decisiones arquitectónicas que afectan a las interacciones de los atributos de calidad. 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.

ARID: es un método de bajo costo y gran beneficio, el mismo es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. Es una combinación entre ADR y ATAM. En

ADR, los involucrados en la evaluación reciben documentación detallada y llenan sus cuestionarios de forma independiente. En ATAM se evalúa toda la arquitectura. El método ARID recoge de ADR, la fidelidad de las respuestas que se obtiene de los involucrados en el desarrollo y del ATAM la idea de que sean los involucrados con el sistema los encargados de generar los escenarios. El método ARID consta de nueve pasos recogidos en dos fases: Actividades previas y Revisión.

ATAM: este método está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM. Revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros. El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. Estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas. [21]

ATAM comprende nueve pasos, agrupados en cuatro fases (Presentación, Investigación y Análisis, Pruebas y Reporte), tiene un equipo de evaluación y un conjunto de salidas mejor definidas y no presenta ninguna restricción con respecto a la característica de calidad a evaluar. Este método evalúa con más profundidad, cuestiones referentes a la arquitectura, como son: los atributos de calidad.

Para la evaluación de la arquitectura definida se decidió utilizar el método de evaluación ATAM, por ser uno de los más acordes a las necesidades del sistema, uno de los más documentados y además con mejores explicaciones en los casos de estudio.

1.8 Conclusiones

Luego de haber estudiado detalladamente los temas relacionados con la Arquitectura de Software, se pudo observar que se cumplieron con algunas de las principales tareas de la investigación. Fueron finalmente seleccionados los patrones arquitectónicos Arquitectura en Capa y MVC, tecnologías y herramientas como la plataforma JEE, frameworks principales como Dalas, Spring e Hibernate, como IDE de desarrollo Eclipse Galileo, SGBD PostgreSQL así como el servidor web Apache Tomcat. Fue seleccionada también la metodología de desarrollo RUP y el lenguaje de modelado que propone la misma, UML. Además quedó seleccionado el método de evaluación de la arquitectura a utilizar siendo este ATAM, y los atributos de calidad ISO/IEC 9126 adaptados a la arquitectura de software que serán empleados para una correcta evaluación de la arquitectura.

Capítulo 2: Descripción de la arquitectura

Teniendo en cuenta la metodología seleccionada, el lenguaje de modelado y las diferentes tecnologías y herramientas presentes en la Fundamentación Teórica, en el presente capítulo encontrará por cual Arquitectura de Software será guiado el desarrollo del sistema. Hallará la descripción de la Arquitectura de Software. Encontrará como representar esta descripción de la arquitectura a través de las vistas de la arquitectura. Además, podrá encontrar decisiones importantes referentes a los estilos de codificación que deben seguir los desarrolladores y los patrones de diseño propuestos para elevar la calidad del software.

2.1 Arquitectura seleccionada

La arquitectura definida para el desarrollo del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito está basada en una arquitectura de tres capas lógicas fundamentales: Capa de Presentación, Capa de Lógica de Negocio, y Capa de Acceso a Datos. Esta arquitectura se eligió siguiendo como objetivo la separación de responsabilidades en cada una de las capas y lograr una mayor reutilización, escalabilidad y facilidad para el desarrollo del sistema. Dichas capas están bien delimitadas una de la otra, una capa superior interactúa con la inferior mediante interfaces que definen las funcionalidades que la misma debe brindar.

Entre los patrones de diseño seleccionados está el de Fachada, debido a que con el uso del mismo se consigue ocultar la complejidad de algunos componentes y aumenta la capacidad del sistema para evolucionar fácilmente, conllevando el fortalecimiento de su modificabilidad. Otro patrón de diseño lo es la Inyección de Dependencia, el cual constituye una de las potencialidades del framework Spring, donde las dependencias con respecto a los servicios son explicitas y no están en el código, debido a que el servicio se identifica y localiza por medio de mecanismos no programáticos, externos al código (por un archivo XML), ganando con esto facilidad de prueba, bajo acoplamiento y mantenimiento.

Además, se utilizará el patrón DAO para la transparencia en el acceso a los datos, el mismo permitiría la migración más fácil a un gestor de base de datos diferente, en caso de ser necesario para un futuro, 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 del negocio y la centralización de todos los accesos a datos en una copia independiente.

Los patrones seleccionados establecen un Bajo Acoplamiento en el sistema, ya que los componentes no se afectan por cambios en otros, son fáciles de reutilizar y entender por separado. El grado del acoplamiento no puede verse separado de otros patrones como la Alta Cohesión, la cual mejora la claridad y facilidad con que se entiende el diseño, soporta mayor capacidad de reutilización y simplifica el mantenimiento y las mejoras de funcionalidad.

2.2 Representación arquitectónica

El presente documento representa la Arquitectura de Software como el modelo 4+1 de Philippe Kruchten, vinculado a la metodología de desarrollo RUP ya mencionada anteriormente, que define una serie de vistas. Para el desarrollo del presente trabajo se representarán las siguientes:

- **Vista de casos de uso:** Muestra los casos de uso que describan alguna funcionalidad importante y crítica, o que impliquen algún requisito importante que deba desarrollarse pronto dentro del ciclo de vida del software.
- **Vista lógica:** Muestra la estructura estática del sistema.
- **Vista de implementación:** Muestra la estructura en modelos del código del sistema.
- **Vista de despliegue:** Muestra el despliegue de la aplicación en la red de computadoras.

Estas vistas usan como lenguaje de modelado a UML y son representadas mediante los modelos de Visual Paradigm.

2.3 Objetivos y restricciones arquitectónicas

Dentro de los objetivos que persigue la Arquitectura de Software está que el sistema debe ser mantenible, lo que significa que debe ser fácilmente analizable, modificable y corregible. Y dentro de las restricciones del sistema que tienen un impacto significativo en la arquitectura se pueden mencionar los siguientes:

- Se deben recopilar los datos de las distintas Coordinaciones Regionales que se encuentran distribuidas geográficamente por todo el país.
- Los datos con los que trabajan las Coordinaciones Regionales son de vital importancia y de carácter estratégico para la prevención del delito del país venezolano, por lo cual es necesario garantizar su seguridad.
- Deberá existir un servidor central en la DGPD en el cual se van a almacenar todos los datos recopilados.

- La cantidad de información que es manipulada es bastante grande.

Los requerimientos no funcionales son las características que hacen al producto atractivo, usable, rápido o confiable. Los requisitos no funcionales que se describen a continuación son los requisitos que el sistema debe cumplir y constituyen la base que debe sustentar la arquitectura.

RNF1 – Requerimientos de Software

Servidores

1. Sistema Operativo: Red Hat Enterprise Linux 4.0
2. Máquina Virtual de Java: JRE 1.6.
3. Gestor de Base de datos: PostgreSQL 8.4
4. Servidor web: Apache Tomcat 6.0.20.

PC cliente

1. Navegadores: Mozilla Firefox 3.5, Internet Explorer 6 o superior.

RNF2 - Requerimientos de Hardware

Servidores para la DGPD

Tanto el servidor de Base de Datos (PostgreSQL), el servidor de ficheros así como el servidor de aplicaciones, deberán contar como mínimo con:

- RAM: 1GB.
- Procesador: P IV 1.6 GHz.
- HDD: 1 GB.
- NIC: 1x GB.

Servidores para las Coordinaciones Regionales

El servidor de cada Coordinación Regional debe poseer como mínimo:

- Procesador: P IV 1.6 GHZ.
- Memoria: 512 MB DDR2.
- HDD: 1 GB.
- NIC: 10/100/1000 bit Ethernet controller.

Requisitos de hardware para PC cliente.

- Intel PIII 700 MHz.
- RAM: 64 MB.
- Adaptador de red LAN.
- HDD: 200 MB.

RNF3 - Restricciones en el diseño y la implementación

- El lenguaje de programación a ser usado para la implementación es Java.
- El gestor de base de datos es PostgreSQL.
- Los componentes del sistema deben presentar el principio de alta cohesión y bajo acoplamiento.

RNF4 - Requerimientos de apariencia o interfaz externa

- Se deben utilizar imágenes y colores identificados con el negocio del sistema. La interfaz externa debe estar diseñada para verse en cualquier resolución igual o superior a 1024x768.

RNF5 - Requerimientos de Seguridad

- La autenticación será la primera acción del usuario en el sistema y consistirá en proveer un nombre de usuario único y una contraseña que debe ser de conocimiento exclusivo de la persona que se autentica.
- Se debe garantizar que la información sensible sólo pueda ser vista por los usuarios con el nivel de acceso adecuado y que las funcionalidades del sistema se muestren según el usuario que esté activo.

RNF6 - Requerimientos de Usabilidad

- El sistema debe permitir a los usuarios un acceso fácil y rápido, contando con un menú que satisfaga las necesidades de los usuarios pues el sistema podrá ser usado por cualquier persona que posea conocimientos básicos en el manejo de una computadora y del ambiente web.

RNF7 - Requerimientos de Soporte

Una vez terminado el desarrollo del software se deben tomar decisiones con motivo de asistir a los clientes del software y para lograr un mejoramiento progresivo y evolución en el tiempo, por lo que se plantean como requerimientos de soporte:

- Reinstalación de aplicaciones ante fallos.
- Instalación de nuevas versiones o actualizaciones del sistema.
- Solución de fallos de configuración de la tecnología asociada al proyecto.

RNF8 - Requerimientos Legales

- La licencia para la comercialización del producto se emitirá por la Dirección de Servicios Legales de la Infraestructura Productiva, teniéndose en cuenta los componentes que se utilizaron para el desarrollo del sistema.
- Todo software a utilizar para el desarrollo del sistema debe estar bajo el principio de migración a software libre al que pretende llegar la República Bolivariana de Venezuela.

RNF9 - Requerimientos de Disponibilidad

- El Sistema de Gestión de Información debe estar disponible para su utilización las 24 horas del día, durante los siete días de la semana, con el menor tiempo posible de recuperación ante fallos.

2.4 Vistas arquitectónicas

Siguiendo la metodología RUP, se desarrolla la descripción de la arquitectura mediante las 4+1 vista. La esencia de las vistas de la arquitectura es la simplificación o abstracción de los modelos, de los cuales se destacan los detalles más significativos.

2.4.1 Vista de Casos de Uso

La vista en cuestión representa los casos de uso significativos para la arquitectura ya que describen alguna funcionalidad importante y crítica o que implementan algún requisito importante que debe desarrollarse pronto dentro del ciclo de vida del producto.

La vista de casos de uso del sistema queda constituida como se muestra en la siguiente figura:

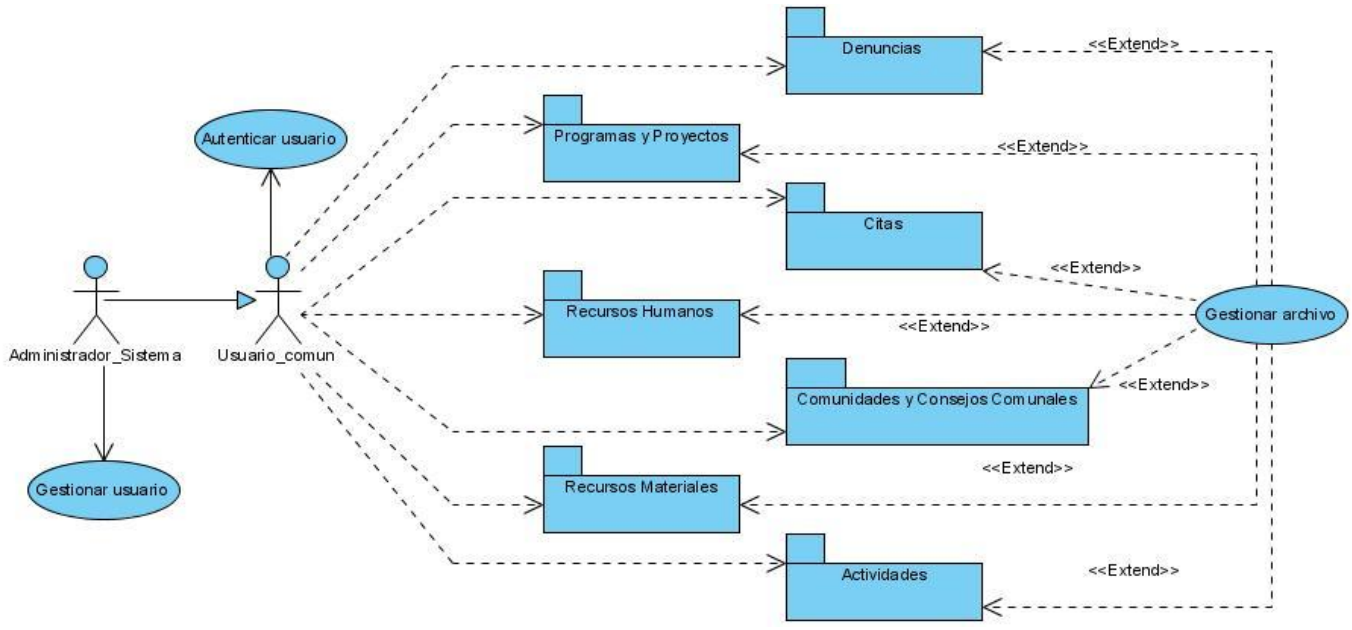

Figura 2. Vista de Casos de Uso del sistema.

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

Caso de uso Gestionar archivo: El caso de uso se inicia cuando el usuario indica subir un archivo o eliminar un archivo adjunto. El sistema da la posibilidad de realizar una de las opciones anteriores. El usuario sube o elimina archivos terminando de esta manera el caso de uso.

Precondición: El caso de uso es utilizado solo por los casos de uso: Gestionar denuncia, Gestionar proyecto, Gestionar programa, Gestionar línea estratégica, Gestionar caso (escenario Remitir caso), Gestionar consulta (escenario Actualizar consulta), Gestionar trabajador, Gestionar control asistencia, Gestionar comunidad (escenario Actualizar comunidad), Gestionar consejo comunal (escenarios Actualizar consejos comunales y Visualizar consejos comunales), Gestionar entrada de bienes muebles, Gestionar entrada de materiales (escenario registrar entrada materiales), Gestionar salida de materiales y Gestionar actividad.

Paquete Denuncias: Paquete donde se agruparon todos los casos de uso arquitectónicamente significativos relacionados con las denuncias. Los casos de uso son inicializados por los actores que tienen los roles de Supervisora, Secretaria, Coordinador Regional (CR), Mirador y Equipo Técnico Profesional (ETP). En este paquete se encuentran como casos de uso arquitectónicamente significativos: Gestionar Denuncia, Gestionar listado denuncia y Generar reporte denuncia.

Paquete Programas y Proyectos: Paquete donde se agruparon todos los casos de uso arquitectónicamente significativos relacionados con los programas y los proyectos. Los casos de uso son inicializados por los actores que tienen los roles de Supervisora y CR. En este paquete se encuentran como casos de uso arquitectónicamente significativos: Gestionar proyecto, Gestionar programa, Gestionar listado proyecto, Gestionar listado programa y Gestionar línea estratégica.

Paquete Citas: Paquete donde se agruparon todos los casos de uso arquitectónicamente significativos relacionados con las citas. Los casos de uso son inicializados por los actores que tienen los roles de

Secretaria, CR y ETP. En este paquete se encuentran como casos de uso arquitectónicamente significativos: Gestionar cita, Visualizar cita, Gestionar listado cita, Gestionar caso, Gestionar consulta y Actualizar expediente ciudadano.

Paquete Recursos Humanos: Paquete donde se agruparon todos los casos de uso arquitectónicamente significativos relacionados con los recursos humanos. Los casos de uso son inicializados por los actores que tienen los roles de Supervisora, Secretaria, CR y Administrador_RRHH. En este paquete se encuentran como casos de uso arquitectónicamente significativos: Gestionar trabajador, Gestionar listado trabajadores y Gestionar control asistencia.

Paquete Comunidades y Consejos Comunales: Paquete donde se agruparon todos los casos de uso arquitectónicamente significativos relacionados con las comunidades y los consejos comunales. Los casos de uso son inicializados por los actores que tienen los roles de Supervisora, Secretaria, CR y ETP. En este paquete se encuentran como casos de uso arquitectónicamente significativos: Gestionar comunidad, Gestionar consejo comunal, Gestionar listados consejos comunales y Gestionar listado comunidad.

Paquete Recursos Materiales: Paquete donde se agruparon todos los casos de uso arquitectónicamente significativos relacionados con los recursos materiales y los bienes muebles. Los casos de uso son inicializados por los actores que tienen los roles de Supervisora, CR y Administrador_RM. En este paquete se encuentran como casos de uso arquitectónicamente significativos: Gestionar entrada de bienes muebles, Gestionar entrada de materiales, Gestionar salida de materiales, Gestionar listado de materiales y Gestionar listado de bienes muebles.

Paquete Actividades: Paquete donde se agruparon todos los casos de uso arquitectónicamente significativos relacionados con las actividades. Los casos de uso son inicializados por los actores que tienen los roles de Supervisora, Secretaria, CR y ETP. En este paquete se encuentran como casos de uso arquitectónicamente significativos: Gestionar actividad y Gestionar listado actividades.

2.4.2 Vista Lógica

La vista lógica permite observar cómo está diseñada la funcionalidad en el interior del sistema. Esta vista describe las clases más importantes, su organización en paquetes de servicio y subsistemas, y la organización de estos subsistemas en capas. Se describen los paquetes de forma abstracta, así como las relaciones que existen entre ellos.

Los principales paquetes por lo que está conformada la aplicación web son:

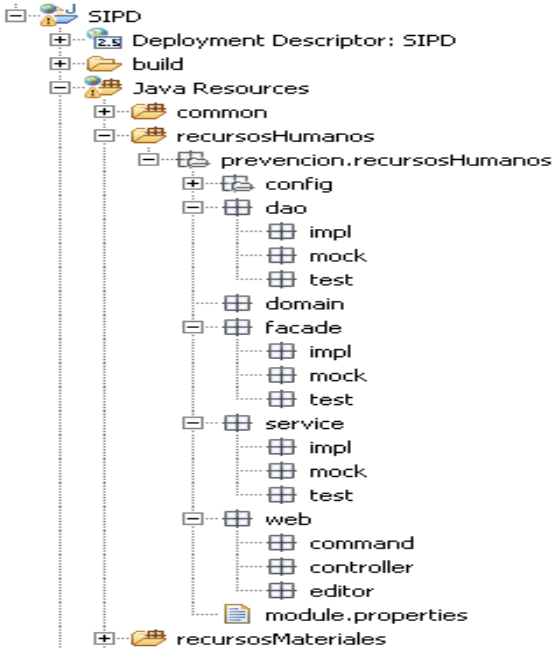

Figura 3: Principales paquetes de la aplicación.

SIPD: contiene las clases y formularios para la aplicación Web agrupados en nueve paquetes para cada uno de los nueve módulos (Recursos Humanos, Recursos Materiales, Programas y proyectos, Actividades, Citas, Denuncias, Comunidades y Consejos Comunales, Administración) y un módulo común que contendrá todas las funcionalidades comunes para todos los módulos.

SIPD. <modulo>: representa a un módulo determinado, cuyo nombre reemplaza la palabra “<modulo>”. Está compuesto por una serie de paquetes como son las vistas, controladores, dao, domain, service, dominio, facade, entre otros. Entre todos dan respuesta tanto a la lógica de negocio como a la persistencia de los datos.

config: se encuentran los ficheros de configuración propuestos por DALAS, rhumanos-bussines.xml, rhumanos-dataaccess.xml y rhumanos-servlet.xml, entre otros.

dao: Se encontrarán las interfaces DAOs.

dao.impl: Se encontrarán las implementaciones de las interfaces DAOs.

dao.test: Se encontrarán las clases de prueba utilizadas para realizar las pruebas de unidad a las implementaciones de los DAO.

domain: Se encontrarán todas las entidades persistentes o no del dominio, pertenecientes al módulo.

facade: Se encontrarán las interfaces de las fachadas de negocio.

facade.impl: Se encontrarán las implementaciones de clases fachadas del negocio.

facade.test: Se encontrarán las implementaciones de prueba de clases fachadas del negocio.

facade.mock: Se encontrarán las implementaciones falsas de clases fachadas del negocio.

service: Se encontrarán las interfaces que representan las operaciones de negocio que se van a exponer o consumir como servicio.

service.Impl: Se encontrarán las implementaciones de las clases de servicio.

service.test: Se encontrarán las implementaciones de prueba de las clases de servicio.

service.mock: Se encontrarán las implementaciones falsas de las clases de servicio.

web.controller: Se encontrarán los controladores de la capa de presentación.

web.command: Se encontrarán las clases utilizadas para guardar datos procedentes de las peticiones web (Command).

web.editor: Se encontrarán los PropertyEditors[1].

El sistema estará distribuido en tres capas principales:

Capa de presentación: encargada de la presentación de la información y del manejo de los aspectos relacionados con la lógica de la presentación de la aplicación. Además de la validación de los datos de la entrada y de los formatos de los datos de salida.

Capa de lógica de negocio: encargada de contener la implementación de las tareas y reglas del negocio.

Capa de acceso a datos: encargada de todo lo relacionado con la persistencia de los objetos que se manejan en el negocio y necesiten ser almacenados, además de la actualización y recuperación de estos.

Servicios de Fachada.

El uso de este servicio es con el objetivo de optimizar la lógica de negocio y para ocultar la complejidad de determinadas clases (Patrón Fachada) a aquellas que solicitan los servicios. Como se observa en la siguiente figura, las clases que se agrupan en el paquete **SIPD.<modulo>.facade** sirven de fachada entre las clases de los paquetes **SIPD.<modulo>.web.controller** y **SIPD.<modulo>.servicios.impl.**

[1] *PropertyEditor:* En Java es utilizado para convertir un bean estructurado en forma de cadena de texto a un objeto de su correspondiente clase.

Figura 4: Vista Lógica - Patrón Fachada en la Aplicación Web.

Bajo Acoplamiento e Inyección de Dependencia.

Se logró un bajo acoplamiento de clases al utilizar el patrón Inyección de Dependencia. Un conjunto de clases importa determinadas interfaces y no crean directamente instancias de las implementaciones de dichas interfaces. Las instancias de estas interfaces se configuran en un fichero XML logrando un bajo acoplamiento sin necesidad de modificar el código fuente.

En la figura 5 se representa como las clases agrupadas en el paquete **SIPD.<modulo>.web.controller** importan las interfaces contenidas en **SIPD.<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. 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.

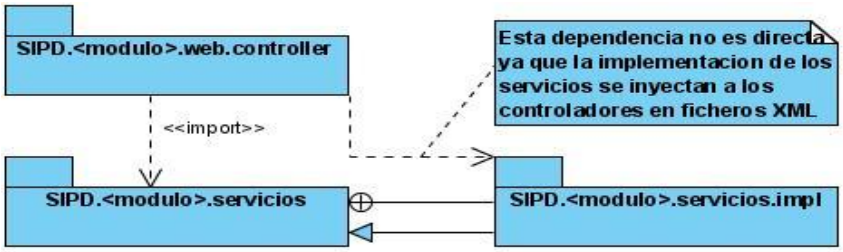

Figura 5: Vista Lógica - Patrones Inyección de dependencia y Bajo Acoplamiento en la Aplicación Web.

Visión general de la arquitectura

A continuación se presenta una vista general de los paquetes de la aplicación, estos responden al patrón en Capas.

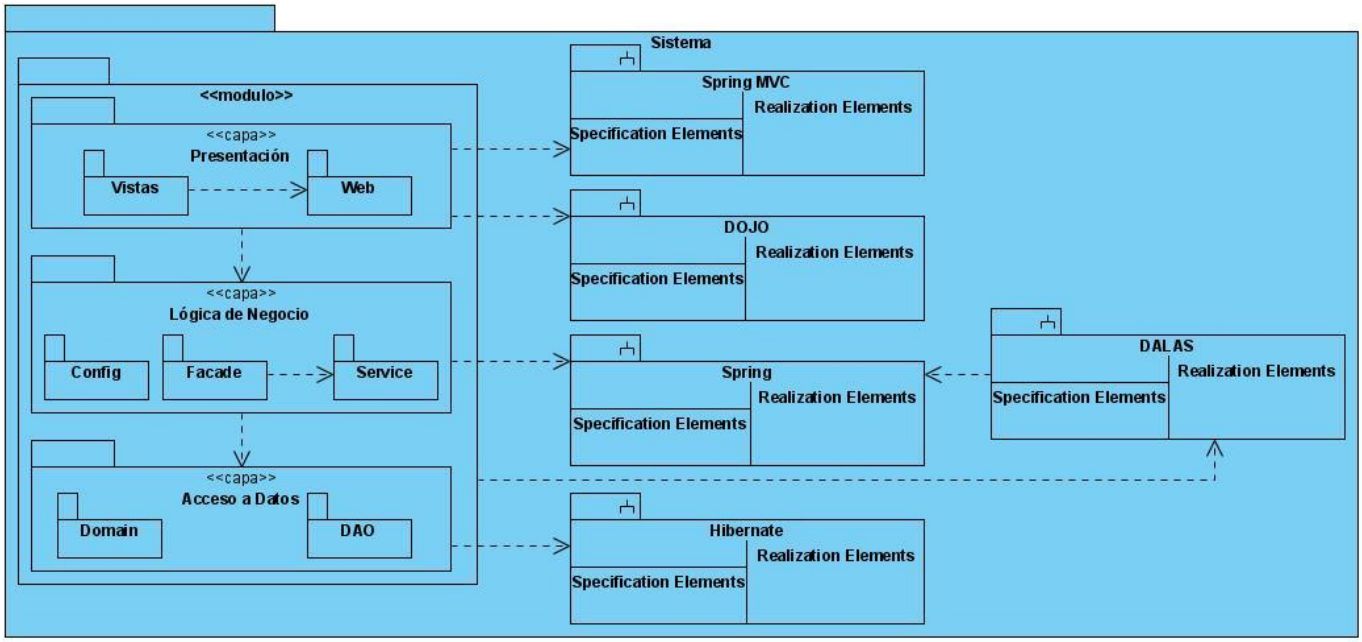

Figura 6: Vista Lógica General.

Representación detallada de los paquetes por capas.

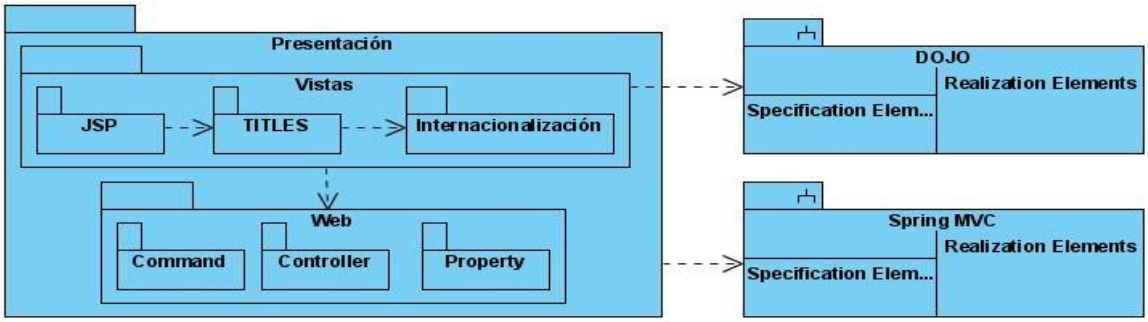

Figura 7: Capa de Presentación.

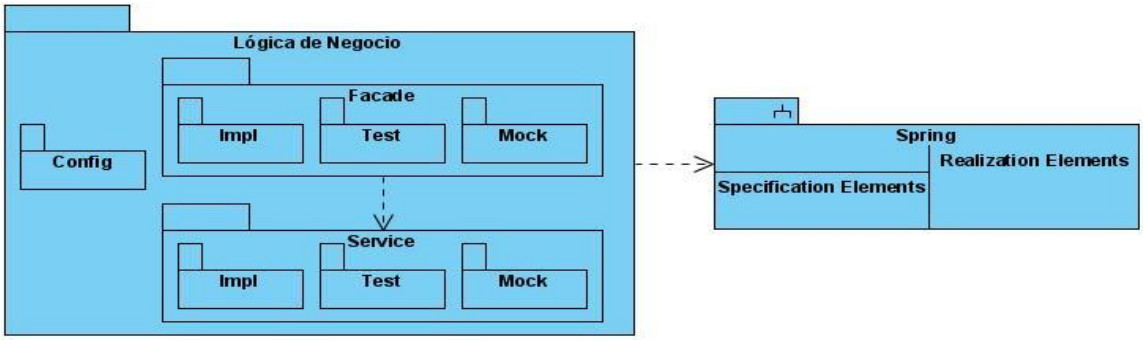

Figura 8: Capa de Lógica de Negocio.

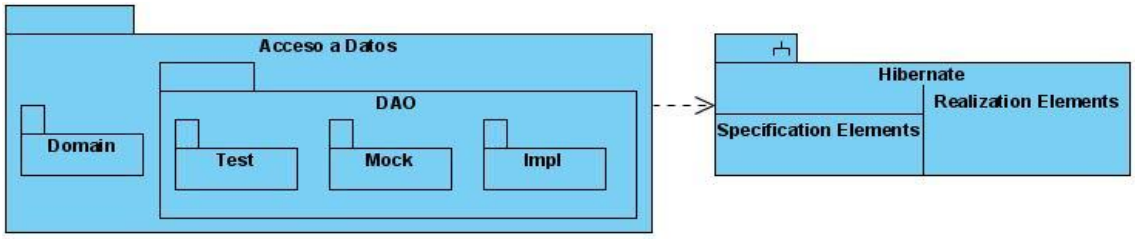

Figura 9: Capa de Acceso a Datos.

2.4.3 Vista de Implementación

La vista de implementación constituye una selección de los aspectos fundamentales del diagrama de componentes del sistema, el cual modela el empaquetado físico del sistema en unidades reutilizables llamadas “componentes” y sus relaciones. Un componente es una unidad física de implementación que encapsula una o más clases del diseño.

Esta vista describe la descomposición del software en capas y subsistemas de implementación. También provee una vista de la trazabilidad de los elementos de diseño de la vista lógica ahora para la implementación. En resumen, esta vista muestra los elementos físicos reales más significativos del sistema.

Anteriormente se ilustran los principales paquetes por lo que está compuesta la aplicación, por lo que seguidamente solo se muestran algunas carpetas importantes dentro del proyecto:

Figura 11: Carpetas importantes del proyecto.

CSS: Contiene todos los ficheros css que definirán los estilos de la aplicación.
Imagenes: Contiene las imágenes necesarias para la aplicación.
JS: Contiene los ficheros java script necesarios para las diferentes funcionalidades del lado del cliente, así como las diferentes clases que componen el framework DOJO.
JSP: contiene las clases .jsp que conforman parte de la Vista.

Lib: contiene todas las librerías que serán utilizadas por la aplicación. Además de todas las librerías que soportan los framework Dalas 0.2 y Spring 3.0.2.

El Sistema (SIPD) está formado por 4 subsistemas compuestos por la aplicación web, la aplicación Reko, la aplicación Servidor de Mensajería (JMS) y la aplicación Servidor FTP, la siguiente figura muestra el diagrama de componentes general de dicho sistema.

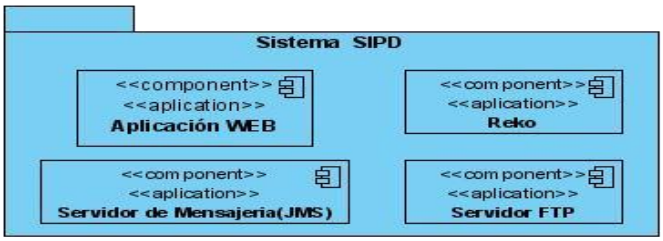

Figura 12: Diagrama de Componente General.

A continuación se desglosa sólo el subsistema Aplicación Web con un mayor grado de detalles con el objetivo de lograr mejor comprensión del mismo.

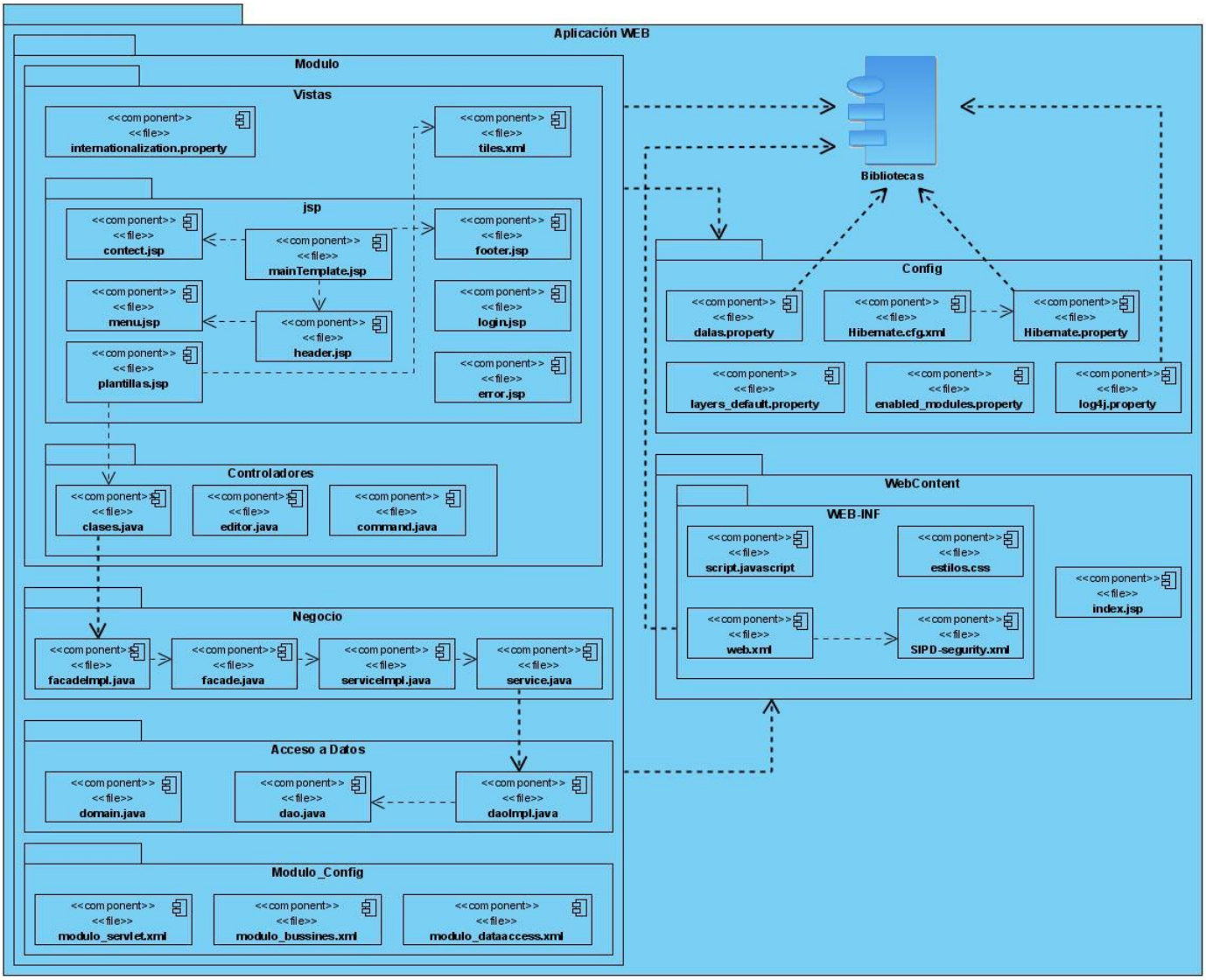

Figura 13: Vista de Implementación General.

La vista se conforma de 4 paquetes que engloban los principales componentes y subsistemas de la aplicación.

El paquete **WebContent** contiene el **index.jsp** que es el punto de entrada a la aplicación. Además, el subpaquete **WEB-INF** que alberga el componente **web.xml**, el cual controla hacia donde serán enrutadas todas las solicitudes. El componente **SIPD-security.xml**, contiene la configuración que permite regular los niveles de acceso a las páginas en dependencia del rol del usuario que interactúa con la aplicación.

El paquete **Config** contiene todas las configuraciones generales de la aplicación, comunes para todos los módulos del sistema, los mismos están definidos por el framework DALAS, entre ellos se encuentra **dalas.property** que contiene las propiedades fundamentales del sistema, así como los componentes que conformarán el mismo**. enabled_modules.property** define los módulos habilitados, **Hibernate.cfg.xml** contiene la configuración necesaria para la conexión a la base de datos y utiliza el fichero **hibernate.property.** También el paquete contiene al **log4j.property** para especificar donde se almacenarán los logs de la aplicación y cuales elementos de la misma serán registrados.

El paquete **Bibliotecas**, contiene las librerías que utiliza la aplicación. Algunas de las principales son: **Spring, DALAS, postgresql-8.3-603.jdbc3 y jasperreport.jar** este último necesario para la generación de reportes.

El paquete **Modulo** contiene a su vez subpaquetes para agrupar los diferentes componentes de cada modulo, estos son: **Vistas**, **Modulo-Config, Acceso a Datos y Negocio.**

Vistas: contiene las clases **jsp** que se desplegarán en dependencia del módulo correspondiente. Este paquete almacena los componentes **internacionalización.properties**, que permite cargar mensajes de texto como base para la internacionalización de la aplicación, o sea, mostrar la misma en otros lenguajes sin necesidad de cambiar el código. El componente **tiles.xml**, que proporciona información sobre cuáles serán las páginas jsp que se cargarán cuando se ejecute ese módulo.

Modulo-Config: Contiene los ficheros de configuración del módulo, **modulo_servlet.xml** establece el controlador frontal para el módulo**. modulo_bussines.xml** contiene las reglas de negocio y el **modulo_dataaccess.xml** configuraciones necesarias para la persistencia de los datos.

Los otros dos paquetes contienen ficheros necesarios para la comunicación entre la vista y el acceso a datos.

2.4.4 Vista de Despliegue

La vista de despliegue describe la configuración física sobre la que será desplegado el software. Esta presenta los nodos computacionales que intervienen en el funcionamiento del sistema, las conexiones entre estos y los protocolos de comunicación, estableciendo posibles configuraciones que se ilustran mediante los diagramas de despliegue.

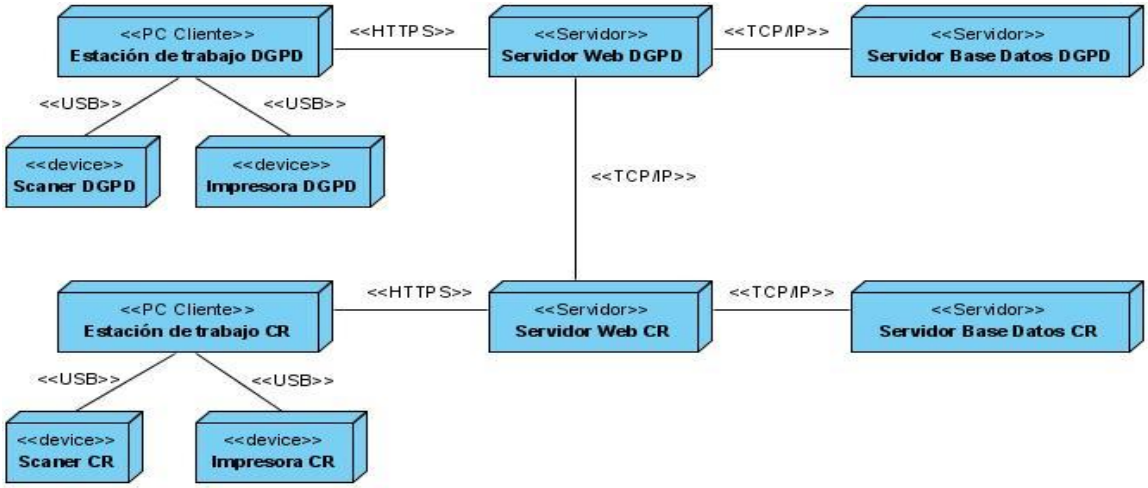

Figura 10: Vista de despliegue

Descripción de Nodos

Estación de trabajo DGPD y CR: PC cliente desde donde se podrá visualizar e interactuar con el Sistema de Gestión de Información y poder transcribir los datos a través de un navegador web que debe estar previamente instalado en la máquina.

Servidor Web DGPD: Servidor que tendrá la aplicación web a la que se conectan los clientes por medio de sus estaciones de trabajo. Este servidor contendrá las medidas de seguridad y protección pertinentes. En el mismo se encuentran además la aplicación FTP que es la encargada de almacenar los ficheros binarios pertenecientes a la réplica de la base de datos y la aplicación de mensajería JMS que es la encargada de la comunicación entre los distintos nodos de réplica. También cuenta con la aplicación de réplica Reko la cual permite la réplica de los datos.

Servidor Base Datos DGPD: Servidor en el que se encuentra el Centro de Datos en el que estarán los datos recopilados de los envíos de las diferentes estaciones de trabajo de la DGPD y todos los datos del sistema.

Servidor Web CR: Servidor que contendrá la aplicación web a la que se conectan los clientes por medio de sus estaciones de trabajo. Este servidor contendrá las medidas de seguridad y protección pertinentes. Además de alojar la aplicación de réplica Reko la cual permite la réplica de los datos.

Servidor Base Datos CR: Servidor en el que se encuentra la Base de Datos local en la que se almacenarán los datos asociados a la Coordinación Regional correspondiente. El servidor de base de datos es PostgreSQL.

Descripción de Dispositivos

Scanner DGPD y CR: Dispositivo que permite escanear algunas de las imágenes que sean de utilidad tanto para la DGPD como para la CR así como aquellos documentos que estén en formato duro.

Impresora DGPD y CR: Dispositivo que permite tanto a la DGPD como a la CR imprimir los reportes e informes generados por el Sistema de Gestión de Información.

Descripción de protocolos de comunicación

HTTPS: Protocolo de comunicación que se describe como HTTP (protocolo estándar básico de la arquitectura Web) con la inclusión de SSL, lo que permite que la información que se mueve a través de los diferentes canales de la red, sea encriptada. Se ha utilizado para la comunicación establecida entre el Servidor Web DGPD y la estación de trabajo DGPD y entre el Servidor Web CR y la estación de trabajo CR.

USB: Protocolo que permite adjuntar los dispositivos periféricos como la impresora y el escáner a las estaciones de trabajo tanto de la DGPD como de la CR.

TCP/IP: Protocolo que representa la manera en la que se realizan las comunicaciones entre el servidor web de la DGPD y el servidor de Bases de Datos de la DGPD así como con el servidor web de la CR, además la comunicación entre el servidor web de la CR y el servidor de Base de Datos de la CR.

Los servidores web de la DGPD y de la CR que utilizan como protocolo de comunicación TCP/IP, para garantizar la seguridad del envío de información entre estos dos servidores hace uso de un mecanismo que presenta la aplicación Reko, el mismo utiliza el protocolo *Secure Socket Layer* (SSL). Este protocolo proporciona sus servicios de seguridad cifrando los datos a través de un cifrado simétrico típicamente el RC4 o IDEA. Además, proporciona autenticación de servidores, integridad de mensajes y autenticación de cliente para conexiones TCP/IP.

2.5 Lineamientos de diseño e implementación

Esta sección contiene las restricciones impuestas por el arquitecto de software para el correcto desarrollo y avance de la solución, tanto desde el punto de vista del diseño, como de la implementación.

2.5.1 Patrones de diseño propuestos

Como se mostró en el capítulo 1, los principales patrones de diseño propuestos fueron: Inyección de Dependencia, Data Access Object (DAO), Fachada, Bajo Acoplamiento y Alta Cohesión.

2.5.2 Estándares de codificación

El arquitecto de software es quien 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 una persona pueda ser comprendido por el resto del equipo. Entre algunas de las decisiones tan importantes que toma el arquitecto está el establecimiento de un estilo de codificación por el cual deban regirse todos los desarrolladores.

Un estilo de codificación es 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 íntegra, sino que, aunque ya existen algunos que son seguidos según el lenguaje de programación en el que se implementa la aplicación, es posible establecer su propio estilo de codificación.

Lo más importante es que el desarrollo de la aplicación se guíe por algún estilo de codificación, pues este permite la comprensión de los programas, haciendo disminuir el número de errores en el desarrollo, 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.

El estilo de codificación adoptado en el presente trabajo se expone en el Anexo 1.

2.6 Conclusiones

Una vez finalizado este capítulo, se han cumplido satisfactoriamente la mayoría de las tareas de la investigación planteadas para la realización del trabajo. Se ha cumplido la más importante en este capítulo, la descripción de la arquitectura, donde se han aplicado favorablemente los conocimientos que se obtuvieron durante el proceso de investigación del capítulo anterior.

Capítulo 3: Evaluación de la arquitectura propuesta

En capítulos anteriores se evidencia la importancia de la Arquitectura de Software para el proceso de desarrollo del software, y se puede ver como el diseño arquitectónico posee varias decisiones con el objetivo de alcanzar no solo la terminación con éxito del software, sino también su factible evolución. Para lograr este objetivo es determinante mantener una rigurosa evaluación del diseño arquitectónico, con el propósito de detectar a tiempo una debilidad presente y tomar las medidas al respecto. Durante este capítulo encontrará el proceso de evaluación de la arquitectura propuesta, mediante el método de evaluación seleccionado. Además, se documentará detalladamente cómo las decisiones del diseño arquitectónico responden a los atributos de calidad y como colaboran entre sí para alcanzarlos.

Una Arquitectura de Software ejerce influencia notable sobre la calidad del sistema que se implementa, de ahí la importancia de evaluarla, para determinar si cumple con los requerimientos de calidad exigidos y estar en condiciones de tomar decisiones acertadas sobre ella.

En la evaluación de una Arquitectura de Software según plantea Bosch [22] se pretende medir propiedades del sistema en base a especificaciones abstractas, como por ejemplo los diseños arquitectónicos. Por ello, la intención es más bien la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos.

En determinadas ocasiones, la evaluación de una Arquitectura de Software 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 [23] propone un esquema general en relación a la evaluación de una arquitectura con respecto a sus atributos de calidad. Kazman afirma 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.

Realizar una evaluación de la arquitectura ayuda a evitar desastres. Esta permite mejorar la visión de los procesos críticos y validar las decisiones de diseño que se tomaron. Permite tomar acciones tempranas.

3.1 Calidad de la arquitectura

Como se planteó en la Fundamentación Teórica, los atributos de calidad que se quieren alcanzar en el diseño arquitectónico son los propuestos en el modelo ISO/IEC 9126 adaptado para arquitecturas de software, para lograr los mismos se tomaron las siguientes decisiones arquitectónicas.

Funcionalidad: Con el uso de la tecnología JEE y el framework Spring que permiten el desarrollo de aplicaciones web de experimentada calidad y seguridad gracias al framework Spring Security de Spring, junto con el estilo arquitectónico en capas, garantizan la habilidad del sistema para cumplir el trabajo para el que fue previsto. Además, es relevante el SGBD PostgreSQL dada la importancia que tiene para la aplicación, la cantidad de datos que se manipulan.

Confiabilidad: La medida de la habilidad del sistema a conservarse activo a lo largo del tiempo se garantiza con el uso del servidor web Apache Tomcat, que es el encargado de restablecer el nivel de desempeño del sistema, también el uso del framework Spring Security de Spring que garantiza el mecanismo de software para manejar las excepciones.

Eficiencia: Dentro de las ventajas que favorecen la decisión arquitectónica de desarrollar una aplicación web está los pocos recursos que la misma demanda de las PC clientes, puesto que realmente la aplicación donde está corriendo es en el servidor web. Además, 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 que el sistema pueda ser objeto de reparaciones y evoluciones de forma rápida y a bajo costo se garantiza con el uso de los patrones de diseño DAO, Inyección de Dependencia, Fachada, Bajo Acoplamiento y Alta cohesión. Los patrones seleccionados para darle soporte a la capacidad de mantenibilidad del sistema son los mismos que favorecen la habilidad de modificabilidad del mismo.

Portabilidad: La habilidad de que el sistema pueda ejecutarse en diferentes ambientes de computación se garantiza 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 JEE, PostgreSQL y Apache Tomcat. Es preciso esclarecer 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 ordenador que tenga instalado un navegador Web de los especificados independientemente del Sistema Operativo en el que se trabaje.

3.2 Aplicación del método de evaluación seleccionado.

En el capítulo 1 quedó plasmado que el método Architecture Tradeoff Analysis Method (ATAM) será el que se utilizará para la evaluación de la arquitectura propuesta. El propósito de ATAM es evaluar las consecuencias de las decisiones arquitectónicas sobre los atributos de calidad necesarios. Es un método que detecta las áreas de riesgos potenciales en la arquitectura de un sistema.

Está integrado por nueve pasos agrupados en cuatro fases, que en general se resumen de la siguiente manera:

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

Seguidamente se muestra 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 que se hayan realizado.

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 muestra el sistema desde la perspectiva del negocio, donde se exponen como principales funciones del sistema:

1. Autenticar usuario.
2. Gestionar usuario.
3. Gestionar denuncia.
4. Gestionar listado de denuncia.
5. Gestionar reporte de denuncia.
6. Gestionar proyecto.
7. Gestionar listado de proyecto.
17. Gestionar trabajador.
18. Gestionar control de asistencia.
19. Gestionar listado de trabajadores.
20. Gestionar comunidad.
21. Gestionar listado de comunidad.
22. Gestionar consejo comunal.
23. Gestionar listado de consejo comunal.

8. Gestionar programa.
9. Gestionar listado de programa.
10. Gestionar línea estratégica.
11. Gestionar cita.
12. Gestionar caso.
13. Gestionar consulta.
14. Actualizar expediente ciudadano.
15. Gestionar listado de citas.
16. Visualizar citas.
24. Gestionar entrada de bien mueble.
25. Gestionar listado de bien mueble.
26. Gestionar entrada de materiales.
27. Gestionar salida de materiales.
28. Gestionar listado de materiales.
29. Gestionar actividad.
30. Gestionar listado de actividad.
31. Gestionar archivo.

Se identifican las restricciones del sistema, arquitectónicamente significativas planteadas en el capítulo 2.

Y como objetivos de los atributos de calidad que dan estructura a la arquitectura:

- Lograr la habilidad del sistema para cumplir el trabajo para el que fue previsto.
- Lograr la habilidad del sistema a conservarse activo a lo largo del tiempo.
- 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.
- Lograr la capacidad de que el sistema en caso de ser sometido a reparaciones y evoluciones, estas sean realizadas de forma rápida y a un bajo costo.
- Lograr la habilidad de efectuar cambios futuros al sistema sin grandes afectaciones a lo que ya se tiene hecho.
- Lograr la habilidad del sistema de que pueda ejecutarse 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.

En este paso, el arquitecto realiza una presentación describiendo la arquitectura en un nivel apropiado de detalle. Como en el capítulo anterior se expuso completamente la descripción de la arquitectura, en este punto no se presenta toda la información referente a la arquitectura propuesta.

Fase 2: Investigación y análisis

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

Es en este paso donde quedan identificadas todas las propuestas arquitectónicas que serán analizadas más tarde, es decir, se especifican las tecnologías que van a ser utilizadas, el patrón arquitectónico y los patrones de diseño. Los mismos no se exponen a continuación ya que se encuentran detallados en la fundamentación teórica.

Paso 5. Generación del Árbol de Utilidad: Los atributos de calidad que engloban la "utilidad" del sistema son obtenidos y especificados en escenarios. Se anotan los estímulos y respuestas, además de establecerse la prioridad entre ellos.

El árbol de utilidad provee un mecanismo que en forma directa y eficiente traduce las pautas del negocio del sistema en escenarios concretos de los atributos de calidad.

Se prioriza el árbol de utilidad en 2 dimensiones: por la importancia que cada escenario tiene para el éxito del sistema y por el grado de dificultad que posee el escenario para ser realizado, según la estimación del arquitecto.

La prioridad del escenario se define en este paso como un par (X, Y) en el cual (X) define el esfuerzo de satisfacer el escenario, y la (Y) indica los riesgos que se corren al excluirlos del árbol de utilidad. Se utilizará la siguiente escala para mayor facilidad: Alta (A), Media (M) y Baja (B).

En el presente paso se obtuvo el siguiente árbol de utilidad:

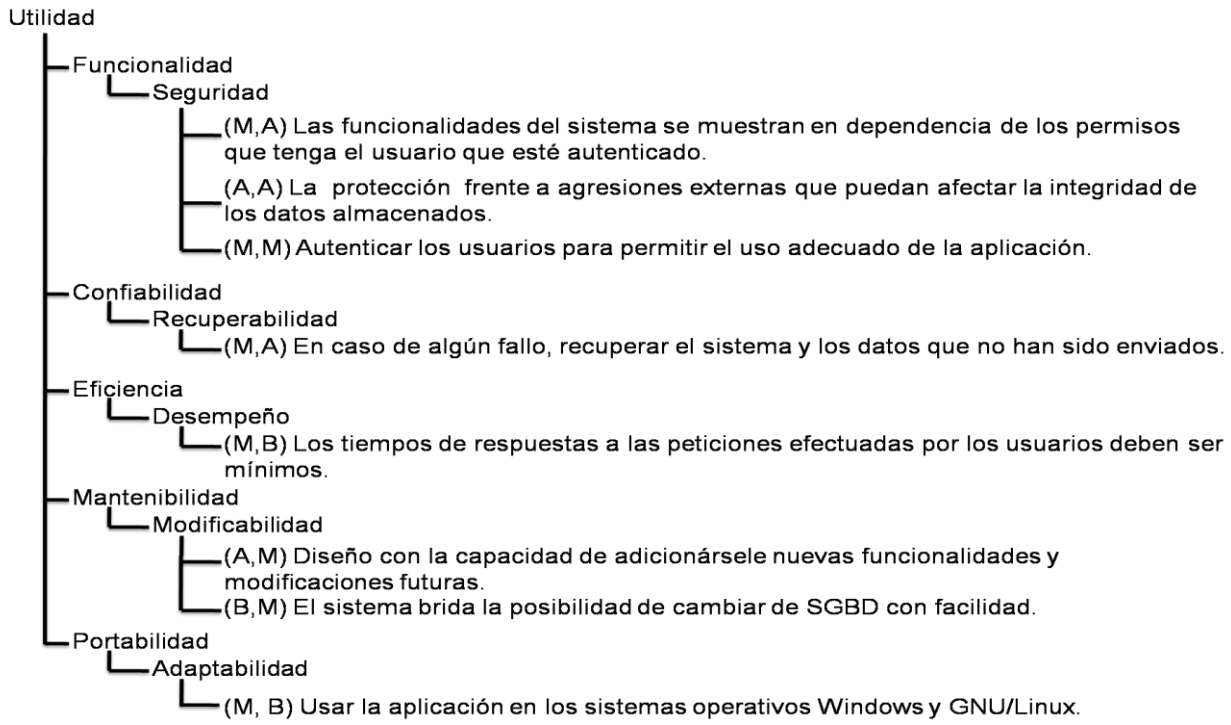

Figura 14: Árbol de utilidad.

Paso 6. Analizar las propuestas arquitectónicas: Son identificados los riesgos arquitectónicos, los no riesgos, los puntos de sensibilidad y los puntos de tradeoff.

Hasta este momento se tienen un conjunto de propuestas arquitectónicas utilizadas en la arquitectura, paso 4 y un conjunto de atributos de calidad priorizado en el paso 5. Por lo que en este paso se mide cuan adecuados son el uno para el otro. En el anexo 2 se encuentran estos escenarios, especificándose de los mismos si presentan riesgos, no riesgos, puntos de sensibilidad o puntos de tradeoff.

Fase 3: Pruebas

Paso 7. Lluvia de ideas y establecimiento de prioridad de escenarios: Un conjunto de escenarios es obtenido por todos los involucrados. Luego este conjunto es priorizado mediante votación.

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

Como resultado de la lluvia de ideas se obtuvo una lista de escenarios priorizados, recogidos en *"Documentos Complementarios (Salidas de método ATAM)",* después de tener este listado se hizo una comparación con los escenarios que están en el árbol de utilidad resultado del paso 5, y al existir algunos nuevos fueron incorporados a este, dando como resultado el árbol de utilidad presente en "*Documentos Complementarios (Salidas de método ATAM)*".

Paso 8. Análisis de los enfoques arquitectónicos: Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Estos escenarios se consideran casos de prueba para confirmar el análisis realizado hasta ahora.

Se realizan las mismas actividades que en el paso 6, mapeando los escenarios recientemente generados con ranking más alto en los artefactos arquitectónicos, obteniéndose entonces la documentación presente en el anexo 2.

Fase 4: Reporte

Paso 9. Presentación de los resultados: Basándose en las informaciones recogidas durante la utilización del método ATAM al realizar la evaluación, se presentan los principales resultados a los involucrados.

Luego de realizados los pasos anteriores, se presentan las principales salidas del método utilizado para evaluar la arquitectura, estos son: el conjunto de escenarios priorizados, el árbol de utilidad actualizado, se encontraron 8 puntos de sensibilidad, 4 puntos de tradeoff, 11 no riesgos y 3 riesgos. Todas estas salidas se pueden encontrar en "*Documentos Complementarios (Salidas de método ATAM)".*

3.3 Conclusiones

Luego de finalizado el presente capítulo, se cumplieron las distintas tareas planteadas para el desarrollo exitoso del sistema. Al evaluar la arquitectura propuesta mediante el método ATAM, se pudo observar como la arquitectura satisface las necesidades del sistema a desarrollar al detectarse menos riesgos que no riesgos y pocos puntos de sensibilidad así como puntos de tradeoff. Dichos riesgos no son tan graves para el sistema pues se cuentan con decisiones arquitectónicas para ayudar a mitigar los mismos.

Conclusiones

Al finalizar el presente trabajo de diploma se obtuvieron las siguientes conclusiones:

1. Se obtuvieron los requerimientos arquitectónicos, los patrones de arquitectura así como las tecnologías y herramientas para dar soporte al desarrollo del sistema, quedando así definida la línea base de la arquitectura necesaria para el sistema.
2. Se describió la arquitectura del sistema a través de las 4 + 1 vistas definidas por RUP.
3. Fue evaluada la arquitectura haciendo uso del método de evaluación “Método de Análisis de Acuerdos de Arquitectura”, arrojando resultados satisfactorios para el sistema.
4. La arquitectura definida cumple con los requisitos de calidad requeridos por el cliente.

Recomendaciones

Después de analizados los resultados obtenidos en el presente trabajo, se recomienda lo siguiente:

1. La arquitectura propuesta teniendo en cuenta los atributos de calidad, puede ser utilizada como referencia para aplicaciones web que hagan uso de la tecnología JEE y framework como Spring.
2. Estudiar las decisiones arquitectónicas que constituyen riegos según el método de evaluación propuesto y valorar su sustitución por otras, que no pongan en peligro el sistema.

Referencias Bibliográficas

1. **Clements, Paul.** *A Survey of Architecture Description Languages.* s.l. : Proceedings of the International Workshop on Software Specification and Design, 1996.
2. **Bass, Len, Clements, Paul and Kazman, Rich.** *Software Architecture in Practice.* s.l. : Addison Wesley, 2003.
3. **IEEE.** *IEEE Std. 1471-2000, IEEE Recommended Practice for Architectural Description of Software Systems.*
4. **Jacobson, Ivar, Booch, Grady and Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* La Habana : Félix Varela, 2004.
5. **Casanovas, Josep.** [En línea] Septiembre 9, 2004. [Citado: Diciembre 14, 2009.] http://www.desarrolloweb.com/articulos/1622.php
6. **Reynoso Billy, Carlos.** [En línea] Marzo 2004. [Citado: Diciembre 14, 2009.] http://www.willydev.net/descargas/prev/IntroArq.pdf
7. **Buschmann, Frank, et al.** *Pattern - Oriented Software Architecture. A System of Patterns.* Inglaterra : John Wiley & Sons, 1996.
8. **Garlan, David and Shaw, Mary.** *An Introduction to Software Architecture.* 1994. CMU-CS-94-166.
9. **Burbeck, Steve.** *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC).* 1992.
10. **Lou Torrijos, Ricard.** Programación en castellano. [En línea] Diciembre 14, 2003. [Citado: Enero 16, 2010.] http://www.programacion.com/java/tutorial/patrones2/8
11. **Bengtsson, PerOlof.** *Design and Evaluation of Software Architecture.* Karlskrona : s.n., 1999. HK-R-RES--99/10.
12. **Pressman, Roger S.** *Ingeniería del Software: Un enfoque práctico*. Cuidad de La Habana: Félix Varela, 2005.
13. *Quality Characteristics for Software Architecture.* **Losavio, Francisca, et al.** 2, 2003, Vol. II.
14. **Kaisler, Stephen H.** *Software Paradigms.* s.l. : John Wiley & Sons, Inc, 2005.
15. **Kruchten, Philippe.** *Rational Software.* 1996.
16. **Szypersky, Clemens.** *Component Software Component Software Beyond Object – Oriented Programming.* s.l. : Addison Wesley, 2002. ISBN 0-201-74572-0.

17. Linea de Codigo. [En línea] Enero 1, 2007. [Citado: Febrero 10, 2010.] http://lineadecodigo.com/2007/01/01/hola-mundo-en-dojo/
18. MySQL. [Citado: Febrero 18, 2010.] http://dev.mysql.com/doc/refman/5.0/es/features.html
19. danielpecos.com. [En línea] Junio 7, 2002. [Citado: Marzo 2, 2010.] http://www.netpecos.org/docs/mysql_postgres/x15.html
20. **Mendoza Sánchez, María A.** [En línea] Junio 7, 2004. [Citado: Marzo 4, 2010.] http://www.willydev.net/Descargas/cualmetodologia.pdf
21. **Camacho, Erika, Cardeso, Fabio and Nuñez, Gabriel.** *Arquitecturas de Software.* 2004.
22. **Bosch, Jan.** *Design & Use of Software Architectures.* s.l. : Addison Wesley, 2000.
23. **Clements, Paul, Kazman, Rick and Klein, Mark.** *Evaluating Software Architectures. Methods and case studies.* s.l. : Addison Wesley, 2001.

Bibliografía

1. Amalgamas. [En línea] Noviembre 9, 2004. [Citado: Enero 21, 2010.] http://homepage.mac.com/imaz/iblog/C612772037/E20050907222635/Media/Algunos%20Tipos%20de%20Arquitecturas.pdf

2. danielpecos.com. [En línea] Junio 7, 2002. [Citado: Marzo 2, 2010.] http://www.netpecos.org/docs/mysql_postgres/x15.html

3. Epidata Consulting. [En línea] Septiembre 29, 2005. [Citado: Enero 20, 2010.] http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=7

4. Linea de Codigo. [En línea] Enero 1, 2007. [Citado: Febrero 10, 2010.] http://lineadecodigo.com/2007/01/01/hola-mundo-en-dojo/

5. mito open course ware. [En línea] Octubre 2, 2001. [Citado: Enero 22, 2010.] http://mit.ocw.universia.net/6.170/6.170/f01/pdf/lecture-12.pdf

6. MySQL. [Citado: Febrero 18, 2010.] http://dev.mysql.com/doc/refman/5.0/es/features.html

7. Seam City. [En línea] Diciembre 16, 2007. [Citado: Febrero 18, 2010.] http://seamcity.madeinxpain.com/archives/comparativa-spring

8. SOFTWARE shop. [Citado: Febrero 2, 2010.] http://www.software-shop.com/in.php?mod=ver_producto&prdID=325#fragment-1

9. Spring source. [En línea] 2010. [Citado: Febrero 16, 2010.] http://www.springsource.org/documentation

10. Tecnología y Synergix. [En línea] Julio 31, 2008. [Citado: Febrero 7, 2010.] http://synergix.wordpress.com/2008/07/31/las-4-mas-1-vistas

11. Write once, share everywhere. [En línea] Marzo 23, 2010. [Citado: Marzo 25, 2010.] http://federicojcdm.wordpress.com/2010/03/23/un-recorrido-por-spring-security-3-0/

12. **Bass, Len, Clements, Paul and Kazman, Rich.** *Software Architecture in Practice.* s.l. : Addison Wesley, 2003.

13. **Bengtsson, PerOlof.** *Design and Evaluation of Software Architecture.* Karlskrona : s.n., 1999. HK-R-RES--99/10.

14. **Blanco González, Victorino.** desarrolloweb. [En línea] Noviembre 3, 2004. [Citado: Febrero 22, 2010.] http://www.desarrolloweb.com/articulos/1692.php

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

16. **Burbeck, Steve.** *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC).* 1992.

17. **Buschmann, Frank, et al.** *Pattern - Oriented Software Architecture. A System of Patterns.* Inglaterra : John Wiley & Sons, 1996.

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

19. **Carrascoso Puebla, Yoan Arlet, Chaviano Gómez, Enrique and Céspedes Vega, Anisleydis.** GestioPolis. [En línea] Mayo 7, 2009. [Citado: Abril 15, 2010.] http://www.gestiopolis.com/administracion-estrategia/procedimiento-para-la-evolucion-de-las-arquitecturas-de-software.htm

20. **Casanovas, Josep.** [En línea] Septiembre 9, 2004. [Citado: Diciembre 14, 2009.] http://www.desarrolloweb.com/articulos/1622.php

21. **Casanovas, Joseph.** alzado.org. [En línea] Julio 31, 2004. [Citado: Diciembre 12, 2009.] http://www.alzado.org/articulo.php?id_art=355

22. **Clements, Paul.** *A Survey of Architecture Description Languages.* s.l. : Proceedings of the International Workshop on Software Specification and Design, 1996.

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

24. **De la Morena, Verónica.** Connexions. [En línea] Enero 7, 2009. [Citado: Febrero 2, 2010.] http://cnx.org/content/m17461/latest

25. **Esteban, Eloy A.** Programación en castellano. [En línea] Diciembre 1, 2001. [Citado: Febrero 16, 2010.] http://www.programacion.com/articulo/tomcat__introduccion_134/1

26. **Garlan, David and Shaw, Mary.** *An Introduction to Software Architecture.* 1994. CMU-CS-94-166.

27. **Gracia, Luis Miguel.** un poco de java JavaMania. [En línea] Mayo 7, 2010. [Citado: Mayo 11, 2010.] http://unpocodejava.wordpress.com/2010/05/07/validaciones-hibernate-validator-vs-oval

28. **Gracia Murugarren, Joaquin.** IngenieroSoftware. [En línea] Mayo 27, 2005. [Citado: Enero 18, 2010.] http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php

29. **IEEE.** *IEEE Std. 1471-2000, IEEE Recommended Practice for Architectural Description of Software Systems.*

30. **Jacobson, Ivar, Booch, Grady and Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* La Habana : Félix Varela, 2004.

31. **Kaisler, Stephen H.** *Software Paradigms.* s.l. : John Wiley & Sons, Inc, 2005.

32. **Keith, Mike and Schincariol, Merrick.** *Pro JPA 2 Mastering the Java™ Persistence API.* s.l. : APress®.

33. **King, Gavin, et al.** *Documentación de referencia de Hibernate.*

34. **Kruchten, Philippe.** *Rational Software.* 1996.

35. *Quality Characteristics for Software Architecture.* **Losavio, Francisca, et al.** 2, 2003, Vol. II.

36. **Lou Torrijos, Ricard.** Programación en castellano. [En línea] Diciembre 14, 2003. [Citado: Enero 16, 2010.] http://www.programacion.com/java/tutorial/patrones2/8

37. **Mendoza Sánchez, María A.** [En línea] Junio 7, 2004. [Citado: Marzo 4, 2010.] http://www.willydev.net/Descargas/cualmetodologia.pdf

38. **Pérez García, Alejandro.** autentia real business solutions. [En línea] Junio 14, 2008. [Citado: Mayo 11, 2010.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=hibernateValidator

39. **Pimentel González, Luis Alberto.** UCIForge: Entorno Virtual de Desarrollo Colaborativo. [En línea] Enero 13, 2010. [Citado: Febrero 06, 2010.] https://forge.uci.cu/gf/project/dalas

40. **Pressman, Roger S.** *Ingeniería del Software: Un enfoque práctico.* Ciudad de La Habana : Félix Varela, 2005.

41. **Reynoso Billy, Carlos.** [En línea] Marzo 2004. [Citado: Diciembre 14, 2009.] http://www.willydev.net/descargas/prev/IntroArq.pdf

42. **Rinaldi, Cristian.** Java User Group del Litoral Argentino. [En línea] Diciembre 22, 2009. [Citado: Mayo 11, 2010.] http://www.juglar.org

43. **Szypersky, Clemens.** *Component Software Component Software Beyond Object – Oriented Programming.* s.l. : Addison Wesley, 2002. 0-201-74572-0.

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. Ejemplo: nombre, edad, entre otros.
- Deben empezar con minúscula y en caso de nombres compuestos la siguiente se escribe con mayúscula. Ejemplo: nombreMadre, direccionActual.
- 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.
- Las variables que constituyen valores constantes deben contener todos sus caracteres en mayúscula. Ejemplo: CANTIDADTOTAL.

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.
- Los métodos deben ser agrupados por funcionalidad en lugar de por el alcance o la accesibilidad.

Clases de la capa de Acceso a Datos:

- Las interfaces que representan las operaciones sobre los objetos de acceso a datos, correspondientes al patrón de diseño Data Access Object terminan con la palabra “DAO”. Ejemplo: PersonaDAO.
- Las implementaciones reales de las interfaces DAO comienza con el nombre de la interfaz correspondiente y terminan con la palabra “Impl”. Ejemplo: PersonaDAOImpl.
- Las implementaciones falsas de las interfaces DAO comienzan con el nombre de la interfaz correspondiente y terminan con la palabra “Mock”. Ejemplo: PersonaDAOMock.
- Las clases utilizadas para realizar pruebas a los interfaces DAO comienzan con el nombre de la interfaz correspondiente y terminan con la palabra “Test”. Ejemplo: PersonaDAOTest.

Todas las interfaces que se utilizarán para la persistencia, heredarán de una interfaz DAO genérica que contendrá todos los métodos necesarios. De esta manera, se agilizará el proceso de desarrollo y se logrará un estándar en la implementación por parte del equipo de desarrollo, ejemplo:

```
public interface DenunciaDao extends Dao<Denuncia, Long>{

}
```

Clases de la capa de Negocio:

- Las interfaces que representan las operaciones del negocio terminarán con la palabra “Facade”. Ejemplo: PersonaFacade.
- Las implementaciones de las interfaces de negocio comenzarán con el nombre de la interfaz correspondiente y terminaran con la palabra “Impl”. Ejemplo: PersonaFacadeImpl.
- Las implementaciones falsas de las interfaces de negocio comienzan con el nombre de la interfaz correspondiente y terminan en la palabra “Mock”. Ejemplo: PersonaFacadeMock.
- Las clases utilizadas para probar las funcionalidades del negocio terminan con la palabra “Test”. Ejemplo: PersonaFacadeTest.

Clases de la capa de Presentación:

Capa Web:

- Las clases que manejan el flujo de la capa de presentación, es decir, los controladores, terminarán con la palabra “Controller”.
- Las clases utilizadas para hacer pruebas de unidad a los Controladores comenzarán con el nombre del controlador correspondiente y terminará con la palabra “Test”.
- Las clases utilizadas para guardar datos procedentes de las vistas, utilizando el patrón Command (GRAND) definido en los patrones GoF, tendrán un nombre lógico según los datos que contiene seguido de la palabra “Command”.
- El manejo de las clases controladoras se realizará a través de anotaciones en correspondencia a los estándares de Spring 3.0.2. Ejemplo:

```
@Controller
public class HelloControl{
}
```

Fachadas de Servicios Remotos:

- Las interfaces que representan las operaciones de negocio que se van a exponer o consumir como servicio terminarán en la palabra “Service”. Ejemplo: PersonaService.
- Las implementaciones de las interfaces de servicio que expondrán las funciones de negocio y las implementaciones que se van a consumir de un servicio, comenzarán con el nombre de la interfaz correspondiente y terminarán con la palabra “Impl”. Ejemplo: PersonaServiceImpl.

- Las implementaciones falsas de las interfaces para consumir los servicios comenzarán con el nombre de la interfaz correspondiente y terminarán en la palabra “Mock”. Ejemplo: PersonaServiceMock.

Paquetes

Para cada modulo que conforme el sistema la nomenclatura quedaría de la siguiente forma: prevencion.<Modulo>, que sería el paquete padre, los demás paquetes serán nombrado en correspondía a las funciones de las clases que contendrán. Ejemplo:

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.

Identación:

Un adecuado espaciado en el código lo hace mucho más legible. Esto incluye espaciado vertical y horizontal en las sentencias y el identado. Identación de 8 espacios para cada nivel (equivalente a un Tab) en lugar de 4 espacios.

- Longitud de Línea: Evite las líneas de más de 80 caracteres, ya que no son bien manejados por muchos terminales y herramientas.
- Líneas de ajuste: Cuando una expresión no cabe en una sola línea, romper según estos principios generales:
  - Romper después de una coma.
  - Romper antes de un operador.
  - Alinear la nueva línea con el principio de la expresión, al mismo nivel que la línea anterior.
  - Si las reglas anteriores causan confusión en la comprensión del código, solamente use la Identación de 8 espacios.

Ejemplo:

```
protected ModelAndView onSubmit(HttpServletRequest request,
        HttpServletResponse response, Object command, BindException errors)
        throws Exception {

    Persona persona=(Persona)command;

    personaService.salvar(persona);

    return new ModelAndView("addPersona");
}
```

Anexo 2. Salida del método ATAM: Análisis de propuestas arquitectónicas.

Escenario 1	Las funcionalidades del sistema se muestran en dependencia de los permisos que tenga el usuario que esté autenticado.			
Atributo(s)	Funcionalidad – Seguridad.			
Ambiente	Operación normal.			
Estímulo	Un usuario accede al sistema.			
Respuesta	Muestra página de autenticación para que el usuario introduzca los datos asociados.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de tradeoff	Riesgo	No Riesgo
Utilización del framework Spring				N1
Utilización del framework Spring Security	S1			N2
Explicación	Las decisiones arquitectónicas que aseguran que las funcionalidades del sistema se muestren en dependencia de los permisos que tenga el usuario que este activo, garantizando así que la aplicación se utilice según los privilegios permitidos a cada usuario según su rol, son: Framework Spring: el mismo ofrece la facilidad de implementar la seguridad utilizando para ello el framework Spring Security. Framework Spring 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. Este framework permite definir reglas de acceso basadas en roles o tipos de usuario para los recursos y funcionalidades de la aplicación.			

El resto de las salidas del método ATAM se pueden encontrar en: *"Documentos Complementarios (Salidas del método ATAM)".*

Glosario

ADL: Lenguaje de descripción de arquitectura cuyas siglas se derivan de su nombre en inglés Architecture Descripction Language y su función como su nombre lo indica es describir una Arquitectura de Software.

API (Application Programming Interface): 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: 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.

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

Framework: Se conoce como marco de trabajo y constituye un conjunto de conceptos, metodologías y herramientas de administración y diseño para el desarrollo de forma estandarizada de una aplicación.

JEE: plataforma para desarrollar aplicaciones empresariales distribuidas, utilizando el lenguaje de programación Java.

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

Open Source: Cualidad de algunos software de incluir el código fuente en la distribución del programa. En general se usa para referirse al software libre.

Plugin: Aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica.

POJO (Plain Old Java Object): Clase del lenguaje de programación Java, que no son de algún tipo especial (EJBs, Java Beans) y no cumplen ningún otro rol ni implementan alguna interfaz especial.

Wedget: Objeto cuyo nombre no se sabe o se olvidó. Símbolo gráfico de interfaz que permite interacción entre el usuario y el ordenador (símbolo, Icono y ventana).

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