Universidad de las Ciencias Informáticas
Facultad 2

Título: Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. Módulos: Denuncias, Recursos Humanos y Consejos Comunales.

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

Autores: Eleanet Menéndez García.

Oreste Enrique Riquenes Peña.

Tutores: Ing. Aidacelys López Días

Cotutor: Ing. Dayron Freddy Calderio Hechavarría.

Ciudad de la Habana, junio 2011
"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 _____.

Eleanet Menéndez García
_____
Firma del Autor

Oreste Enrique Riquenes Peña
_____
Firma del Tutor

Ing. Aidacelys López Díaz.
_____
Firma del Tutor

Ing. Dairon Freddy Calderio Hechavarria
_____
Firma del Cotutor

Datos de Contacto

Tutor:

Ing. Aidacelys López Díaz.

Graduado en la Universidad de las Ciencias Informáticas (UCI) como Ingeniero en Ciencias Informáticas en el año 2007. Profesor Instructor.

Email: alopezd@uci.cu

Cotutor:

Ing. Dairon Freddy Calderio Hechavarria

Graduado en la Universidad de las Ciencias Informáticas (UCI) como Ingeniero en Ciencias Informáticas en el año 2009. Profesor en Adiestramiento.

Email: dfcalderio@uci.cu

Consultantes:

AGRADECIMIENTOS

A nuestros tutores, Aidacelys y Dayron por el apoyo que siempre nos brindaron, por los buenos consejos y por el aliento que nos dieron en todo momento.

A nuestros compañeros del proyecto por toda la ayuda y los buenos momentos compartidos trabajando en equipo.

A todos los profes que durante estos cinco años incentivaron nuestra formación profesional.

A nuestros amigos, que nos brindaron su mano siempre.

A todos los que de una forma u otra nos ayudaron en el desarrollo de nuestra tesis.

DEDICATORIA.

Dedico este trabajo de diploma a:

A mi mamita querida: Por ser mi ejemplo y mi inspiración para querer ser mejor cada día, por su apoyo, por su amor incondicional, por darme fuerzas para vencer cualquier obstáculo , por confiar en mí en todo momento y por mostrarme que las cosas cuando se hacen tienen que ser de corazón.

A Mima la mejor abuela del mundo por cuidar de mí, por sus buenos consejos, por su optimismo ante todo en la vida y por guiarme siempre por el mejor camino.

A mi abuelito lindo José: que es mi orgullo, por siempre recordarme que cuando te fijas metas en la vida son posibles de alcanzar no importa cuán lejos las veas solo debes ser perseverante y no dejar de enfocarte en las cosas buenas y positivas.

A mi hermanito Ariel: A quien adoro por sus pequeños detalles, por darme tanto amor y por llenar mi vida de alegrías.

A mi compañero de tesis Riquenes, que es mi amor, mi soporte, por su dedicación, su paciencia, por estar ahí para mi…siempre.

A mis tías Magali y Mercedes, a mi tío Porto, a todos mis primos por su sacrificio, su cariño, su apoyo.

A mi abuela Zenaida y mi papá que siempre los tengo presentes.

A mí cuñada Yilita y mis suegros Yilian y Oreste por enseñarme que nada es imposible de lograr cuando luchas con todas tus fuerzas y pones fe en ello.

A mis amistades que me dan ánimo para salir adelante en todo momento.

Y a toda esa gente que nos ayudó al desarrollo de este trabajo de diploma.

Eleanet Menéndez García

Dedico este trabajo de diploma a:

A mi papá: Por ser mi ejemplo y mi inspiración para querer ser mejor cada día, por su apoyo, por su amor incondicional, por confiar en mí en todo momento y por enseñarme con su ejemplo a querer ser un mejor hombre en la vida.

A mi mamá la mejor madre del mundo por cuidar de mí, por sus buenos consejos, por su amor incondicional, por la fuerza que me da cada día.

A mi hermana querida: De quien me siento muy orgulloso y adoro en la vida y por llenar mi vida de alegrías.

A mis abuelos y toda mi familia por su apoyo incondicional y su confianza y por todo el amor que siempre me han regalado.

A mi compañera de tesis Eleanet, que es mi amor, mi soporte, por su dedicación, su paciencia, por estar ahí para mí…siempre.

A mí suegra Ileana y Mima por darme tanto cariño y tanto apoyo, por dejarme ser parte de esa familia tan linda.

A mis amistades que me dan ánimo para salir adelante en todo momento.

Y a toda esa gente que nos ayudó al desarrollo de este trabajo de diploma.

Oreste Enrique Riquenes Peña.

RESUMEN

En la actualidad se manifiesta un incremento considerable en el desarrollo de aplicaciones web orientadas a la gestión de información, estas entre otras cosas proporcionan un mejor funcionamiento, ayudan a la toma de decisiones y al control de la organización.

A raíz de la necesidad de desarrollar un sistema informático, basado en software libre y con el objetivo de informatizar la recopilación, el procesamiento y almacenamiento de datos en las Coordinaciones Regionales de Prevención del Delito, pertenecientes al Ministerio del Poder Popular para Relaciones Interiores y Justicia de la República Bolivariana de Venezuela, el presente trabajo muestra el desarrollo del Sistema de Gestión de Información en las Coordinaciones Regionales encargado de gestionar toda la información referente a las Denuncias, Recursos Humanos y Consejos Comunales, además permite generar reportes acerca de toda la información gestionada.

Se desarrolló el sistema empleando el estilo arquitectónico tres capas y el mismo fue implementado haciendo uso de los frameworks Spring, Hibernate y Dojo.

PALABRAS CLAVE: CR, DGPD, código abierto, frameworks, Spring, Hibernate y Dojo

ÍNDICE

INTRODUCCIÓN

Por establecimiento de la Constitución de 1909, la República Bolivariana de Venezuela modificó su división política territorial, quedando estructurada en Estados, y estos a su vez en Municipios autónomos. En total, el país cuenta con 335 municipios integrados en 23 Estados y un Distrito Capital, que se fragmentan en Parroquias, que no guardan relación con la Institución Eclesiástica.

Con el triunfo electoral del actual Presidente Hugo Rafael Chávez Frías en diciembre de 1998, se comenzó a conocer en el seno del pueblo venezolano, la diferencia entre Democracia Representativa y Participativa. En el marco constitucional de la democracia participativa y protagónica, surgen las Comunidades como entidades político-administrativas descentralizadas donde se aglutinan las células de autogobierno local llamadas Consejos Comunales, instancias que permiten al pueblo organizado ejercer directamente la gestión de las políticas públicas y proyectos orientados a responder a las necesidades y aspiraciones de las Comunidades.

Partiendo de la nueva estructura organizacional el gobierno atiende, en primer orden, la agenda social; siendo esta última un elemento que impacta positivamente en la dimensión amplia de la seguridad ciudadana y prevención del delito; en tal sentido el Ministerio del Poder Popular para Relaciones Interiores y Justicia (MPPRIJ), atendiendo a su misión institucional de garantizar la seguridad ciudadana, promueve la formulación y puesta en marcha del proyecto Solución Tecnológica Integral para el Perfeccionamiento del Sistema de Prevención del Delito de la República Bolivariana de Venezuela orientado a mejorar el funcionamiento de los procesos internos de todas las direcciones pertenecientes al mismo.

La Dirección General de Prevención del Delito (DGPD) está adscripta al Viceministerio de Seguridad Ciudadana perteneciente al Ministerio del Poder Popular para Relaciones Interiores y Justicia. Por otra parte, tiene como misión el establecimiento, la promoción y la coordinación de políticas, programas y proyectos que atiendan a la prevención de la violencia criminal y no criminal del país con la integración y participación de la institucionalidad y la sociedad civil. Esta institución posee 22 oficinas ubicadas en los estados: Anzoátegui, Aragua, Barinas, Bolívar, Carabobo, Cojedes, Falcón, Guárico, Lara, Mérida, Miranda, Monagas, Portuguesa, Táchira, Trujillo, Yaracuy, Zulia, Delta Amacuro, Nueva Esparta, Sucre, Vargas y en el Distrito Capital, llamadas Coordinaciones Regionales, conformadas por un Coordinador

y un equipo de profesionales y técnicos que se encargan de la atención e implementación de los programas de prevención del delito.

Entre las funciones de la Dirección General de Prevención del Delito (DGPD) y de sus Coordinaciones Regionales se encuentra gestionar la información referente a sus Recursos Humanos. Actualmente en las Coordinaciones Regionales no existe una correcta estandarización y gestión de las informaciones referentes a los mismos. Debido al cúmulo de información que dichas Coordinaciones registran y al volumen de documentación que se envía a la Dirección General de Prevención del Delito (DGPD), se ve afectada la retroalimentación desde las Coordinaciones Regionales hacia la Dirección General de Prevención del Delito (DGPD). Estas deficiencias conllevan a que exista pérdida de información y gastos innecesarios de recursos materiales. A continuación se mencionan algunas informaciones de las Coordinaciones Regionales que se ven afectadas en su gestión e intercambio:

- La información correspondiente a los datos personales y ocupacionales de los trabajadores de las Coordinaciones Regionales de Prevención del Delito.
- Registros asociados a la asistencia diaria, las horas extras, los períodos vacacionales y de reposo por trabajador, información que se envía a la Dirección General de Prevención del Delito, que es sumamente necesaria para el pago del salario mensual de cada trabajador.
- Los datos de los trabajadores de la Coordinación Regional que participan en las actividades.
- Realización de informes y reportes asociados a la información que se maneja de los Recursos Humanos según las necesidades e indicaciones de la Dirección General de Prevención del Delito.

Las Coordinaciones Regionales gestionan en la actualidad la información relacionada a los Consejos Comunales como:

- Los datos generales que son archivados en un expediente del Consejo Comunal.
- Los miembros que conforman los Comités del Consejo Comunal y las personas de enlace que constituyen la conexión entre el funcionario de la Coordinación Regional y el Consejo Comunal.
- Las demandas sociales que van orientadas a las necesidades o inquietudes de la comunidad.

- Las observaciones que son detectadas en las visitas y actividades que realizan los funcionarios a los diferentes Consejos Comunales.
- El registro de los diagnósticos realizados periódicamente ha dicho Consejo Comunal.

Se lleva el control de las Denuncias que llegan por diferentes vías a la Coordinación Regional registrando:

- Los datos específicos de las Denuncias.
- La información referente a las Remisiones de las mismas.

Las Coordinaciones Regionales no son las encargadas de resolver las denuncias solo se encargan de tramitarlas, canalizarlas y gestionar el problema a otro organismo o institución para que sean procesadas.

Los procesos de gestión de información ya mencionados se realizan de forma manual y generan un gran volumen de información, esto provoca gran demora en la generación de informes, en ocasiones ocurre el deterioro, pérdida o duplicidad de la información, gasto innecesario de papel y otros recursos materiales, afectando de esta forma la toma de decisiones, ya que los jefes y el personal especializado no cuentan con la información o estadísticas necesarias en un momento determinado. Todos estos factores dificultan la ejecución de los procesos internos de la organización y la efectividad en el logro de resultados dirigidos al desarrollo de acciones que contribuyan a la prevención de la violencia y el fortalecimiento de la convivencia ciudadana.

Debido a esto en el marco del convenio Cuba-Venezuela los directivos de la DGPD y Albet, contratan la realización de una herramienta informática. Posteriormente se realizó un levantamiento de requisitos, los cuales fueron aceptados, documentados y firmados por estos directivos, quedando pendiente su implementación.

Por todo lo anteriormente planteado se plantea como **Problema Científico:** ¿Cómo garantizar el cumplimiento de los requisitos funcionales asociados a los módulos Denuncias, Recursos Humanos y Consejos Comunales del Sistema de Gestión de Información de las Coordinaciones Regionales?

Se define como **Objeto de Estudio** Gestión de información en las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela, siendo el **Campo de Acción:** Gestión

de Información asociada a las Denuncias, los Recursos Humanos y Consejos Comunales en las Coordinaciones Regionales.

Para dar solución al problema científico declarado, esta investigación tiene como **Objetivo General:** Desarrollar los módulos que gestionen la información referente a las Denuncias, los Recursos Humanos y Consejos Comunales del Sistema de Gestión de Información de las Coordinaciones Regionales.

A partir de este objetivo general se trazaron los siguientes **Objetivos Específicos:**

1. Realizar el diseño de las clases de los módulos: Denuncias, Recursos Humanos y Consejos Comunales.
2. Implementar los componentes correspondientes a los módulos Denuncias, Recursos Humanos y Consejos Comunales.
3. Validar funcionalmente los módulos: Denuncias, Recursos Humanos y Consejos Comunales.

Los que se cumplirán a través de las siguientes **Tareas de la Investigación:**

1. Estudio y descripción de las herramientas y tecnologías para el desarrollo de un Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela.
2. Realización de los Diagramas de Clases de Diseño de los módulos: Denuncias, Recursos Humanos y Consejos Comunales.
3. Realización de los Diagramas de Secuencia de Diseño de los módulos: Denuncias, Recursos Humanos y Consejos Comunales.
4. Realización de los Diagramas de Componentes de los módulos: Denuncias, Recursos Humanos y Consejos Comunales.
5. Implementación de los componentes de los módulos: Denuncias, Recursos Humanos y Consejos Comunales.
6. Realización de pruebas funcionales a los módulos implementados.

El presente trabajo está estructurado en 4 capítulos, a continuación se muestra una breve descripción de cada uno de ellos:

Capítulo 1 Fundamentación Teórica: En este capítulo se argumentan las características principales de la metodología de desarrollo utilizada, lenguajes de programación y otras herramientas en las que se apoya el desarrollo de sistema.

Capítulo 2 Diseño del Sistema: En este capítulo se muestran los diagramas de clases de diseño y los diagramas de secuencia de los módulos a desarrollar: Denuncias, Recursos Humanos y Consejos Comunales. Además se explican los patrones de diseños utilizados.

Capítulo 3 Implementación del Sistema: En este capítulo se muestran los diagramas de componentes que se definieron durante la implementación de los módulos: Denuncias, Recursos Humanos y Consejos Comunales, también se muestran las principales pantallas pertenecientes a estos módulos.

Capítulo 4 Pruebas del Sistema: En este capítulo se diseñan los casos de pruebas de los módulos: Denuncias, Recursos Humanos y Consejos Comunales con el objetivo de verificar si el producto satisface los requerimientos del usuario, tal y como se describe en la especificación de los requerimientos.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA.

1.1 Introducción.

El presente capítulo abordará primeramente los principales conceptos que permitan entender qué es un sistema de gestión de software. También se argumentan las características principales de la metodología de desarrollo a utilizar y otras herramientas en las que se apoyará el desarrollo del Sistema de Gestión de Información de los módulos: Denuncias, Recursos Humanos y Consejos Comunales.

1.2 Sistemas de Gestión de Información.

En la era de la información a medida que la sociedad se hace cada vez más dependiente de ésta, un mayor número de personas tienen a su vez más posibilidad de beneficiarse con su uso y aplicación, convertida ya en una fuerza productiva necesaria para el desarrollo. Ello es expresión de la circunstancia, tan propia de la presente época, de que con el incremento de la información, crece también la necesidad de utilizarlos para satisfacer necesidades culturales, científicas, docentes, sociales, entre otras. En este contexto, debe entenderse que las tecnologías de la información y las telecomunicaciones no son más que un medio para transmitir, gestionar datos y conocimiento. Sin embargo, uno de los principales problemas de la información es su exceso, es necesario invertir mucho tiempo en ella por lo que debe ser gestionada en disímiles ocasiones.

Gestión de la información: La gestión de la información es el proceso de analizar, utilizar, recuperar y almacenar la información que se ha obtenido y registrado, para permitir a los administradores tomar decisiones documentadas. [1]

Sistemas de gestión de información: Puede definirse como un conjunto de componentes interrelacionados que permiten capturar, procesar, almacenar y distribuir información para apoyar la toma de decisiones y el control de una institución, además de ayudar a dichos directivos y personal a analizar problemas, visualizar cuestiones complejas y crear nuevos productos en un ambiente intensivo de información. [2]

1.3 Sistemas de Gestión de Información para la Prevención del Delito.

- Programa Integral de Gestión e Investigación para la Prevención del Delito

El Programa Integral de Gestión e Investigación para la Prevención del Delito se creó con el objetivo de contar con un Sistema de Información socio-delictiva que cuente con los datos necesarios para elaborar diagnósticos y estudios sobre los factores que inciden en la violencia y la delincuencia, a fin de diseñar planes y programas que respondan a la realidad de los Estados y municipios. Este sistema está dirigido por el Gobierno Federal de México. [3]

Los sistemas que hasta aquí hemos visto son más bien de carácter informativo para los Estados y Municipios de la ciudad de México, brindan espacios donde la sociedad en general puede informarse respecto a temas delictivos, realizan encuestas en la población para identificar factores conflictivos y poder así mantener a los ciudadanos al tanto de cómo prevenirlos. Estos sistemas aparte de que se centran en las características y problemas propios de su país, no almacenan toda la información digitalmente, se realizan las consultas, conferencias y demás actividades en la población, luego estos sistemas de información brindan solamente los resultados de los sucesos ocurridos y de cómo poder integrarte a ellos o simplemente reconocerlos.

Por tanto, la novedad que se busca con el Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela, Módulos Denuncias, Recursos Humanos y Consejos Comunales, es que no sea solo un sistema que le permita a la DGPD estar informada respecto a la información que se maneja en las Coordinaciones Regionales , sino que sirva como herramienta de trabajo para los especialistas y trabajadores de las Coordinaciones Regionales, ya que podrán así gestionar toda la información mediante este sistema, eliminando los problemas existentes respecto al manejo de la información, alcanzando de esta manera una mejor organización de todo el trabajo en las Coordinaciones Regionales.

Se considera necesario para realizar un sistema de gestión de información definir: metodologías, herramientas y tecnologías que guíen el desarrollo del mismo.

1.4 Herramientas, Metodologías y Tecnologías propuestas.

Como antecedente se tienen trabajos de diplomas del curso anterior los cuales definieron las herramientas, tecnologías, metodología y la arquitectura que sigue el presente trabajo de diploma,

entre estos trabajos de diplomas se encuentran “Propuesta de arquitectura para el Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela” por los autores Eneysi Osorio y Asdrúbal Torres y “Sistema Informático de Gestión de Información de las Comunidades, los Consejos Comunales, Citas y Denuncias en las Coordinaciones Regionales” por los autores Osmany Cordero y Francisco Montada.

1.4.1 Metodologías de Desarrollo de Software.

Rational Unified Process (RUP).

Proceso Unificado de Desarrollo (RUP) es un proceso de desarrollo de software en el que se asignan tareas y responsabilidades que tiene como objetivo asegurar la producción de software de calidad dentro de plazos y presupuestos predecibles. [4]

- El Proceso de Desarrollo de Software es un marco de trabajo genérico, que puede especializarse para una gran variedad de sistema software, para diferentes áreas de aplicación, tipos de organizaciones, niveles de actitud y tamaños de proyectos. Está basado en componentes, lo cual quiere decir que el sistema software en construcción está formado por componentes software interconectados a través de interfaces bien definidas. Utiliza el **Lenguaje Unificado de Modelado** (Unified Modeling Language, UML) 2.0 para preparar todos los esquemas de un sistema software. **UML** permite la modelación de sistemas con tecnología Orientada a Objetos (POO), es un lenguaje de representación visual, permite combinar diversos elementos gráficos y crear diagramas, describe lo que hará un sistema pero no dice cómo implementarlo.

Fig. 1.1 Un proceso de desarrollo de software

Roles y Artefactos.

RUP indica que al inicio del proyecto se realice una adecuación de cada flujo de trabajo de manera que se produzcan solo los artefactos y se realicen las actividades que tienen un propósito dentro del proyecto. A continuación se muestra un resumen de los trabajadores y artefactos en las disciplinas de mayor interés para la realización del presente trabajo de diploma.

Análisis y Diseño: Los objetivos de la disciplina Análisis y Diseño son transformar los requisitos de software en un diseño del sistema, desarrollar una arquitectura robusta y adaptar el diseño al ambiente de implementación.

Trabajadores

Diseñador

Artefactos

- Diagrama clases de diseño
- Diagrama de Secuencia

Implementación: En esta disciplina se define la organización del sistema en Subsistemas. Su objetivo principal es la implementación del diseño, además de la realización de pruebas unitarias a los componentes y la integración de los resultados del trabajo de implementadores individuales en un sistema ejecutable.

Trabajadores

Desarrollador

Artefactos

- Elementos de implementación
- Artefactos de instalación

Prueba: Las pruebas son actividades con las cuales los sistemas o componentes son ejecutados bajo condiciones o requerimientos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. La prueba de software es un

elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación.

Trabajadores

- Analista de pruebas
- Probador

Artefactos

- Registro de pruebas
- Resultados de prueba

1.4.2 Herramienta de Modelado de Software.

Las **Herramientas CASE** (**C**omputer **A**ided **S**oftware **E**ngineering, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software, reduciendo el coste de las mismas en términos de tiempo y dinero. Estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software.

Visual Paradigm for UML Enterprise Edition 6.4 (VP Suite 3.4).

Visual Paradigm es una herramienta que provee soporte para la generación de código, tiene integración con diversos IDE's (Entornos de Desarrollo Integrados) como NetBeans (de Sun Microsystems), JDeveloper (de Oracle), Eclipse (de IBM), JBuilder (de Borland), así como la posibilidad de realizar la ingeniería inversa para aplicaciones desarrolladas en JAVA, .NET, XML e Hibernate.

Tiene dentro de sus características que es portable y posee gran factibilidad de uso. Utiliza UML como lenguaje de modelado y soporta el ciclo de vida completo de desarrollo de software, ayuda a una rápida construcción de aplicaciones de calidad y a un menor coste [5].

Entre sus ventajas más relevantes se encontraron que permite realizar ingeniería tanto inversa como directa, es una herramienta colaborativa, es decir, soporta múltiples usuarios trabajando sobre el mismo proyecto; genera documentación del proyecto automáticamente en varios formatos tales como web o PDF, además, permite el control de versiones. Igualmente se puede destacar su robustez, usabilidad y portabilidad.

1.4.3 Lenguaje de Programación.

Un lenguaje de programación es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que se pone a disposición del programador para que éste pueda comunicarse con los dispositivos de hardware y software existentes. [6]

Java.

Es un lenguaje de programación de alto nivel desarrollado por Sun Microsystems a principios de los 90. Entre sus principales características se encuentran:

- **Orientado a objetos:** Soporta las características esenciales del paradigma de la programación orientado a objetos: encapsulación, herencia y polimorfismo.

- **Robusto:** Elimina el uso de apuntadores para referenciar áreas de memoria, además, libera al desarrollador de la necesidad de desalojar la memoria que la aplicación ya no usa. También requiere la declaración explícita tanto de los tipos de datos como de los métodos.

- **Multiplataforma:** El mismo código Java que funciona en un sistema operativo, funciona en cualquier otro que tenga instalada la máquina virtual de Java.

- **Multitareas:** Permite la ejecución concurrente de varios procesos ligeros o hilos de ejecución.

La combinación de las características anteriormente mencionadas lo hace único y por eso está siendo adoptado por multitud de fabricantes como herramienta básica para el desarrollo de aplicaciones comerciales y empresariales de gran repercusión.

Java resulta ser un lenguaje sencillo, no es difícil dominarlo si se tiene experiencia programando. Los desarrolladores de este lenguaje recibieron grandes influencias del paradigma orientado a objetos, lo que trae como resultado que tenga un modelo de objetos sencillo y de fácil ampliación. [7]

1.4.4 Entorno de Desarrollo Integrado (IDE).

Un Entorno Integrado de Desarrollo (Integrated Development Enviroment (IDE)) es un programa compuesto por un conjunto de herramientas para un desarrollador que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. [8]

Eclipse 3.4.

Es un entorno de desarrollo integrado (IDE) porque provee herramientas para administrar áreas de trabajo, para construir, lanzar y depurar aplicaciones, para compartir artefactos con un equipo y versionar el código fuente. Es multiplataforma, ya que se ejecuta en gran cantidad de sistemas operativos incluyendo Windows y Linux, es accesible y su diseño le permite ser extendido fácilmente por terceras partes. Emplea módulos para proporcionar toda su funcionalidad, a diferencia de otros entornos donde las funcionalidades están todas incluidas, las necesite el usuario o no. [9]

Eclipse cuenta con un mecanismo de incorporación de plugins, como el Spring IDE, Hibernate y otros eficaces para el desarrollo sobre frameworks.

El IDE para desarrollar el software mediante el lenguaje de programación Java, fue el Eclipse v3.4, por ser de código abierto y su factibilidad de uso. Además su diseño global permite tener las herramientas que necesitan los programadores inmediatamente al alcance de sus manos. Es fácilmente integrable con la herramienta CASE Visual Paradigm y soporta perfectamente la plataforma de desarrollo Java Enterprise Edition (JEE).

1.4.5 Plataforma de Desarrollo.

Plataforma se refiere al ambiente de hardware y software donde un programa se ejecuta, por ejemplo, plataformas como Linux, Solaris, Windows 2003 y MacOS. En este caso se estudia la plataforma Java, ya que en el epígrafe anterior se escogió Java como lenguaje de programación. En casi todos los casos las plataformas son descritas como la combinación del sistema operativo y el hardware. La plataforma Java se diferencia de estas plataformas, ya que es una plataforma sólo de software y se ejecuta sobre las otras plataformas de hardware. [10]

La plataforma Java está compuesta por dos componentes:

- La máquina virtual de Java (JVM)
- El Java API (Application Programming Interface)

Esta plataforma antes era conocida como Plataforma Java 2 e incluye:

- Plataforma Java, Edición Estándar (Java Platform, Standard Edition), o **Java SE** (antes J2SE).
- Plataforma Java, Edición Empresa (Java Platform, Enterprise Edition), o **Java EE** (antes J2EE).

- Plataforma Java, Edición Micro (Java Platform, Micro Edition), o **Java ME** (antes J2ME).

Java Platform, Enterprise Edition (Java EE).

Java EE cuenta con una arquitectura de varios niveles distribuida basándose ampliamente en componentes de software modulares, ejecutándose sobre un servidor de aplicaciones. Java EE también es considerada informalmente como un estándar debido a que los suministradores deben cumplir ciertos requisitos de conformidad para declarar que sus productos son conformes al mismo.

Java EE configura algunas especificaciones únicas para componentes, éstas incluyen: Enterprise JavaBeans, Servlets, JavaServer Pages y varias tecnologías de servicios web. Esto permite al desarrollador crear una aplicación de empresa portable entre plataformas, y a la vez que sea integrable con tecnologías anteriores. Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes en lugar de en tareas de mantenimiento de bajo nivel. [10]

A continuación una breve descripción de los componentes de Java EE:

- **Java Server Pages (JSP):** Es una tecnología orientada a crear páginas web con programación en Java, es decir las JSP permite fragmentos de código Java incrustado en la página web. Por tanto las páginas JSP están compuestas de código HTML/XML mezclado con etiquetas especiales para programar scripts de servidor en sintaxis Java.
- **Java Servlets:** Proporcionan un método para escribir programas del lado del servidor. Su uso común es la generación de páginas web dinámicas.
- **Enterprise Java Beans (EJB):** Un EJB es una clase java que tiene varias características por ejemplo: son distribuidos, usan transacciones, son Multi Hilos, persistentes, entre otras más. Muchas de las características de los EJB son proporcionadas por los distintos servidores de aplicación.

1.4.6 Sistema Gestor de Base de Datos.

Los Sistemas de Gestión de Bases de Datos (en inglés Database Management System, DBMS) son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan

PostgreSQL 8.3.

Es un Sistema de Gestión de Bases de Datos Objeto-Relacional (ORDBMS) basado en el proyecto POSTGRES, de la universidad de Berkeley.

Fue el pionero en muchos de los conceptos existentes en el sistema objeto-relacional actual, incluido, más tarde en otros sistemas de gestión comerciales. PostgreSQL es un sistema objeto-relacional, ya que incluye características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. A pesar de esto, PostgreSQL no es un sistema de gestión de bases de datos puramente orientado a objetos. [11]

Desventajas:

- Consume bastantes recursos y carga con mucha facilidad el sistema.
- Velocidad de respuesta un poco deficiente al gestionar Bases de Datos (BD) relativamente pequeñas, aunque esta misma velocidad la mantiene al gestionar BD realmente grandes. [12]

1.5 Marco de Trabajo o Frameworks

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

En el desarrollo de la arquitectura base del sistema se utiliza el framework de aplicación Spring y el framework de soporte Hibernate, ambos son muy populares por sus múltiples ventajas en el desarrollo de aplicaciones web dentro de la plataforma JEE y son de código abierto al igual que Dojo, utilizado como framework javascript o de presentación para crear interfaces de manera fácil.

1.5.1 Spring 3.0.2.

Spring es un framework de aplicación desarrollado por la compañía Interface 21, para aplicaciones escritas en el lenguaje de programación Java. Sus desarrolladores basados en su experiencia en el desarrollo de aplicaciones J2EE (Java 2 Enterprise Editions), incluyendo EJB (Enterprise JavaBeans), Servlets y JSP (Java Server Pages), lograron combinar dichas herramientas y otras más en un solo paquete, para brindar una estructura más sólida y un mejor soporte para este tipo de aplicaciones. [13]

Es un framework basado en la Inversión de Control (IoC) y en la Programación Orientada a Aspectos (AOP). Se distribuye de forma libre y su código es abierto. Ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar.

Arquitectura de Spring.

Spring es un framework modular que cuenta con una arquitectura dividida en acerca de veinte módulos integrados en capas (Fig.1.1), lo cual permite tomar y ocupar únicamente las partes que interesen para el proyecto y juntarlas con gran libertad. [14]

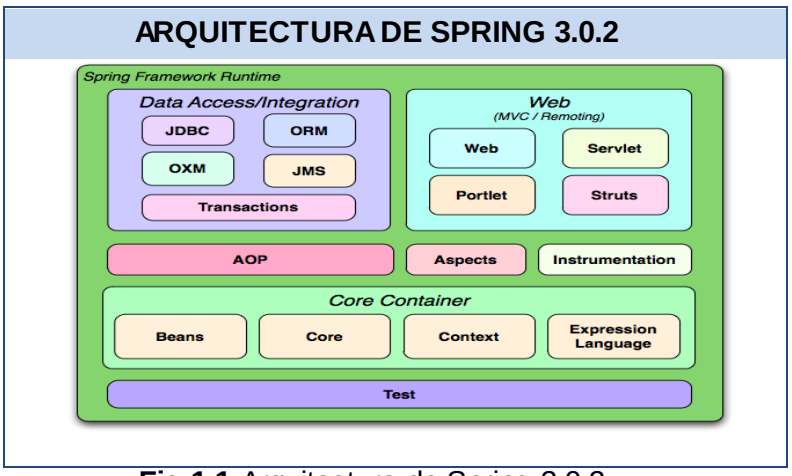

Fig.1.1 Arquitectura de Spring 3.0.2.

Capa Core Container.

Esta parte es la que provee la funcionalidad esencial del framework, está compuesta por el Core, Beans, Context, and Expression Language.

- El **Core y Beans** proveen las partes fundamentales del framework, incluyendo la Inversión de Control (IoC) y las opciones de inyección de dependencias. Bean Factory es una sofisticada implementación del patrón Factory. Esta implementación elimina la necesidad de implementar Singleton y permite desacoplar la configuración de la especificación de dependencias del modelo lógico.

Capa Acceso a Datos/Integración.

Contiene JDBC (Java Database Connectivity), ORM (Object-Relational Mapping), OXM (Mapeo Object/XML), JMS (Java Message Service) y los módulos de transacciones.

- El módulo **JDBC** proporciona una abstracción del modelo JDBC y elimina la necesidad de usar el JDBC básico que proporciona JAVA para acceder a la base de datos y controlar los errores.
- El módulo de **ORM** proporciona una capa de integración para las API's de mapeo entre objetos y el modelo relacional. Incluye integración con Hibernate, Java Persistence API (JPA), Java Data Objects (JDO) e iBatis. Este módulo permite integrar los anteriores frameworks con la funcionalidad y las características que ofrecen Spring.

Capa Web.

La capa Web contiene los módulos Web, Web-Servlet, Web-Struts, y Web-Portlet.

- El módulo **Web-Servlet** contiene el Modelo-Vista-Controlador de Spring (MVC) y la ejecución de aplicaciones web. Spring MVC, establece una clara separación entre las diferentes capas, y se integra con todas las otras características del Spring Framework.

Capa AOP (Aspect-Oriented Programming).

El objetivo del módulo no es proveer la implementación más completa posible, sino una integración cercana entre la implementación de AOP y el contenedor de aplicaciones para resolver los problemas comunes de las aplicaciones empresariales. Por tanto, las funcionalidades de AOP de Spring se usan en conjunción con el contenedor de Spring y puede ser soportado mediante su propio API o mediante la integración con el framework AspectJ, que provee más posibilidades. Sin embargo, el objetivo declarado de ambas soluciones no es competir sino, complementarse entre ellas.

Capa Test.

La capa de prueba apoya el examen de los componentes de Spring con JUnit o TestNG. Proporciona carga constante de ApplicationContexts y el almacenamiento en caché de los contextos. También proporciona objetos mock que se pueden utilizar para probar el código en forma aislada.

Para realizar las pruebas se utiliza **JUnit** integrado al IDE Eclipse en forma de plug-in, lo cual permite que la generación de las plantillas necesarias para la creación de las pruebas de una clase Java se

realice de manera automática, facilitando al programador, enfocarse en la prueba y el resultado esperado, dejando a la herramienta, la creación de las clases que permiten coordinar las pruebas.

A continuación, una representación gráfica del funcionamiento del MVC en Spring (Fig.1.2)

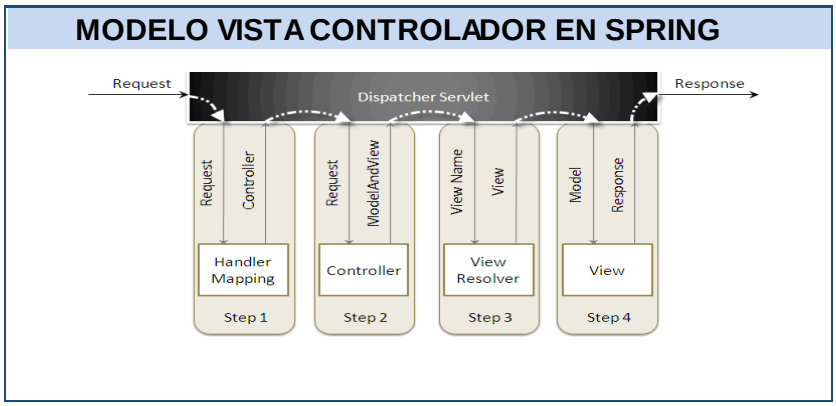

Fig.1.2 MVC en Spring

1.5.2 Framework javascript.

Dojo Toolkit 1.5.

Está compuesto por Widgets que son componentes de código en Javascript pre-empaquetados que pueden ser utilizados para enriquecer sitios web con varias características que trabajan a través de la mayoría de los navegadores, tales como: tabs, tooltips y tablas ordenables.

Dojo cuenta con dojo, dijit y dojox, proporcionan librerías totalmente open source muy bien pensadas y diseñadas. También se puede integrar fácilmente con Spring, framework de aplicación que se utilizará para desarrollar el sistema. [15]

A continuación, una breve descripción de las librerías mencionadas:

- **Dojo:** El núcleo sobre el cual todo lo demás es construido. Incluye alrededor de cincuenta scripts y otros recursos que manejan la normalización del navegador. Modularización del JavaScript, extensiones al JavaScript y al W3C Document Object Model (DOM) API, scripting remoto, Firebug Lite, drag and drop, API para manejo de datos, localización, internacionalización y algunas otras funciones varias.
- **Dijit:** El framework para controles de interfaz de Dojo y cerca de 40 controles o componentes integrados.

- **Dojox:** Extensiones de Dojo. Se incluyen desde el componente grid hasta las librerías para gráficos. Aquí radican algunas librerías sofisticadas y también algunos proyectos que son completamente experimentales.

1.5.3 Framework Hibernate.

Hibernate es un framework ORM para la plataforma Java disponible además para la plataforma .NET con el nombre de NHibernate. Distribuido bajo los términos de la licencia GNU LGPL ha ganado popularidad como herramienta de soporte a la capa de acceso a datos en el desarrollo de aplicaciones empresariales. Provee mapeo objeto-relacional básico, y otras características sofisticadas como caché y caché distribuida.

Como todas las herramientas ORM, Hibernate busca solucionar el problema de la diferencia entre dos modelos ampliamente utilizados para organizar y manipular datos: el orientado a objetos en las aplicaciones y el relacional en las bases de datos. Para lograr esto el desarrollador debe especificar cómo es su modelo de datos. Con esta información Hibernate permite manipular los datos desde las aplicaciones operando sobre objetos con todas las características de la POO (Programación Orientada a Objetos). Hibernate convierte los datos que define Java a los que define SQL (Structured Query Language), siendo transparente esta conversión para el implementador. Genera además las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias. También ofrece un lenguaje de consulta de datos llamado HQL (Hibernate Query Language) y al mismo tiempo API's (Application Programming Interface) para construir las consultas programáticamente. [16]

Una de las ventajas de Hibernate es que posee soporte para múltiples dialectos, es decir, puede mapear diferentes sistemas gestores de bases de datos, entre los que se encuentra Postgre, actualmente unos de los más usados por las múltiples prestaciones que ofrece.

1.6 Reportes.

¿Qué es un reporte?

Un reporte o informe es una manera organizada de mostrar información procedente de una fuente de datos.

1.6.1 JasperReports.

Es una librería de clases 100% Java de código abierto, diseñada para agregar capacidades de reporte a las aplicaciones Java. [17] Además de los datos en texto, JasperReports permite incluir en los reportes imágenes y gráficos, para que los mismos tengan un aspecto profesional.

La tecnología para crear reportes que fue seleccionada fue JasperReport. Es importante señalar que es la que mejor se integra con el framework Spring, constituye una solución concreta y confiable para facilitar la generación, la previsualización y la impresión de los reportes de una manera más eficiente, cubriendo las necesidades del cliente

1.7 Conclusiones.

Luego de haber realizado un estudio sobre las tecnologías y herramientas que cumplen con las tendencias actuales de la programación web y sobretodo orientadas al código abierto:

- Se describió como metodología de desarrollo de software RUP.
- Se describió Visual Paradigm como herramienta CASE para el modelado y UML como lenguaje del mismo.
- Se describió como plataforma de desarrollo J2EE haciendo uso del lenguaje de programación Java y como IDE de desarrollo Eclipse.
- Se describió PostgreSQL como gestor de base de datos.
- Para el desarrollo del sistema fueron descritos los frameworks Spring, Hibernate, Dojo y Jasper Report.

CAPÍTULO 2: DISEÑO DEL SISTEMA

2.1 Introducción

En este capítulo se traducen los requisitos a una especificación que describe cómo implementar el sistema, a través del diseño, enfocado a cómo el sistema cumple sus objetivos teniendo en cuenta los requisitos funcionales y no funcionales, se realizan los diagramas de clases y los diagramas de secuencias más relevantes según los casos de usos y se explica además la arquitectura y los principales patrones de diseño utilizados.

2.2 Propuesta de solución.

Para garantizar el cumplimiento de los requisitos funcionales, se partirá de la realización del diseño de clases con la utilización de la arquitectura definida por el trabajo de diploma: *"Propuesta de arquitectura para el Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela"* por los autores Eneysi Osorio y Asdrúbal Torres [27]. Posteriormente se implementarán las clases diseñadas cumpliendo con estándares de codificación y lógica de validación definida. Con el objetivo de medir el grado en que el software cumple con los requerimientos, se ejecutarán pruebas dirigidas a componentes del software y al sistema en su totalidad. A nivel de unidad las pruebas se enfocarán en un componente que desempeña una función específica, que puede ser probada y que asegure que funcione tal y como lo definen las especificaciones establecidas y a nivel de sistema se aplicarán pruebas funcionales que evalúan las condiciones o capacidades con las que se debe cumplir, así como las pruebas de carga y estrés para verificar el rendimiento del sistema.

2.3 Estilo Arquitectónico Utilizado.

El Estilo Arquitectónico es 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. [18]

¿Qué es un patrón de arquitectura?

Expresan un paradigma fundamental para estructurar u organizar un sistema software. Proporciona un conjunto de subsistemas o módulos predefinidos, con reglas y guías para organizar las relaciones entre ellos. [19]

Para la realización del Sistema de Gestión de Información de las Coordinaciones Regionales se utilizó el estilo arquitectónico: Arquitectura en Capas (tres capas) perteneciente a la familia de estilos arquitectónicos de Llamada y Retorno, basándose en las características y peculiaridades de dicho sistema.

Este estilo constituye una especialización de la arquitectura Cliente-Servidor, donde la carga se divide en tres partes o capas lógicas fundamentales (Fig. 2.1) con un reparto claro de funciones. El desarrollo de cada paquete del Sistema de Gestión de Información de las Coordinaciones Regionales responde a este modelo donde cada capa tiene funcionalidades y objetivos precisos, así su implementación se encuentra desacoplada de la programación de cualquier otra capa y la comunicación con una capa inferior ocurre a través de interfaces.

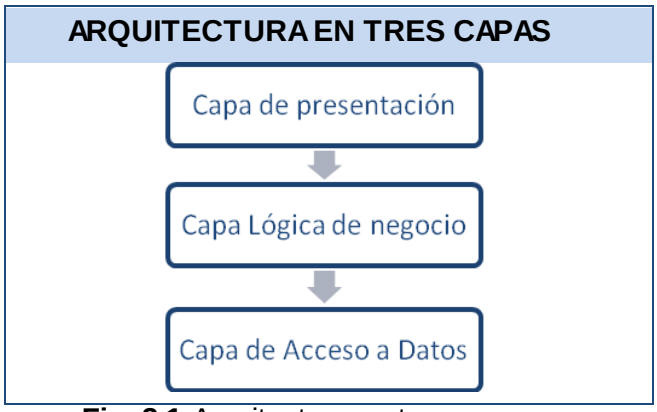

Fig. 2.1 Arquitectura en tres capas

2.2.1 Arquitectura en Capas.

Capa de Acceso a Datos.

Es la responsable de recobrar y persistir información desde y hacia la base de datos y de la comunicación con el gestor de base de datos. Esta capa contiene los objetos que encapsulan la lógica de acceso a datos (DAO) e interfaces brindadas para ser accedida desde la capa de negocio. Las implementaciones de los DAOs extienden de clases de soporte del framework Spring usando el framework ORM Hibernate,

mientras que las interfaces se mantienen independientes de Spring e Hibernate. Se encuentran además en esta capa las clases entidades que representan las clases del dominio.

Capa de Lógica de Negocio.

Esta capa define e implementa las funcionalidades que responden directamente a los requisitos de manera que se conserve la integridad del sistema y de los datos. Está constituida por los servicios y la fachada.

Capa de Presentación.

Define e implementa todo lo relacionado con la interfaz gráfica de usuario. Aquí reside la definición de las peticiones que el usuario puede realizar sobre la aplicación, los controladores que manejan el flujo web y la comunicación con las interfaces de la capa de negocio. Además se encuentran las vistas HTML, XML, PDF, XLS que se muestran al usuario y la implementación del comportamiento dinámico de los documentos HTML a través de Javascript. Las responsabilidades principales de la capa de presentación son: la navegabilidad del sistema, el formateo de los datos de salida, la validación de los datos de entrada y la construcción de la interfaz gráfica de usuario.

Es importante destacar que en esta capa va a estar presente el patrón arquitectónico **Modelo-Vista-Controlador (MVC)** ya que Spring lo utiliza en su funcionamiento como se abordó en el Capítulo 1.

A grandes rasgos el patrón arquitectónico **Modelo-Vista-Controlador (MVC)** funciona de la siguiente manera:

Todas las peticiones son recibidas por el **DispatcherServlet**, este con el **HandlerMapping** identifica al controlador que procesará dicha petición, le pasa el control y obtiene como respuesta un **ModelAndView**, de este toma el nombre de la vista y utilizando el **ViewResolver** encuentra la vista física (archivo JSP), a la que le envía los datos extraídos del **ModelAndView** para que sea dibujada (rendered), resultado de lo cual obtiene el código HTML que es enviado como respuesta al cliente.

DispatcherServlet: Las peticiones de los clientes son recibidas por el DispatcherServlet del Framework Spring, quien es el punto de entrada a la implementación del patrón MVC en Spring. En esencia lo que realiza es pasar el control de las peticiones a los Controladores (**Controller**), esta clase es configurada en el archivo **app_servlet.xml**.

HandlerMapping: Permite mapear las peticiones HTTP, utilizando los URLs o partes de estos, a los controladores que las procesarán; también contiene de forma opcional interceptores que pueden ser invocados antes o después de efectuarse el mapeo. Se tienen varias opciones a utilizar, independientes o combinadas y que son configuradas en archivos XML.

Controller: Al recibir las peticiones HTTP, ejecutan código Java que realiza la lógica del sistema; manipula a los objetos DAO involucrados y decide que Vista usar para el despliegue de los resultados.

ModelAndView: Un objeto de esta clase engloba los datos a mostrar (el modelo, **Model**) en un objeto de la clase **Map** y el nombre de la vista que mostrará dichos datos, este nombre es en general un alias, o sea, no apunta directamente a un archivo físico (JSP).

ViewResolver: Esta clase es utilizada para resolver a partir del nombre de la vista, el alias contenido en el **ModelAndView**, la vista física que será dibujada (rendered) con los datos que le son pasados. Se tienen varias opciones a utilizar, independientes o combinadas y que son configuradas en archivos XML.

View: Este término se refiere las vistas físicas, básicamente a los archivos JSP, PDF, XLS.

2.4 Patrones de Diseños Utilizados.

¿Qué es un patrón de diseño?

Los patrones de diseño son soluciones a problemas específicos, basados en la experiencia y que se ha demostrado que funcionan. Los patrones de diseño no son fáciles de entender, pero una vez entendido su funcionamiento, los diseños serán mucho más flexibles, modulares y reutilizables.

Un patrón de diseño no es más que una solución estándar para un problema común de programación. Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios. Un proyecto o estructura de implementación que logra una finalidad determinada. Un lenguaje de programación de alto nivel. Una manera más práctica de describir ciertos aspectos de la organización de un programa. Conexiones entre componentes de programas. [20]

A continuación se describen los patrones de diseño que permitieron que el Sistema de Gestión de Información de las Coordinaciones Regionales tuviera una mejor organización y un código flexible a la hora de implementar los módulos: Denuncias, Recursos Humanos y Consejos Comunales.

Patrón de diseño Facade (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, en los casos en que se requiere se puede acceder directamente a ellas. El uso de este patrón en el sistema se evidencia en las clases de los paquetes **vnz.mpprij.prevencion.cr.den.facade** las cuales sirven de fachada entre las clases de los paquetes **vnz.mpprij.prevencion.cr.den.web** y **vnz.mpprij.prevencion.cr.den.manager.** El fragmento de diagrama de secuencia que se muestra en la (Fig. 3.2) correspondiente al caso de uso "Registrar Denuncia" es un ejemplo de lo antes expuesto.

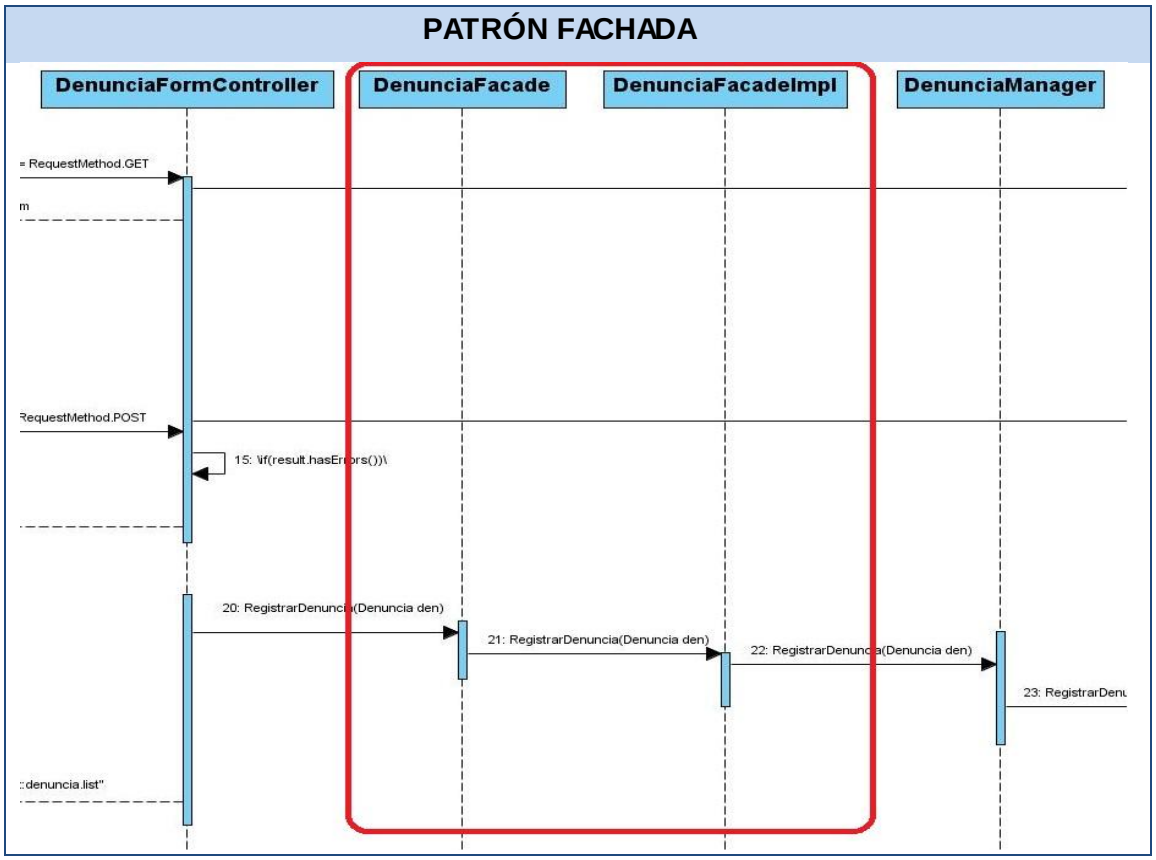

Fig. 2.2 Ejemplo del patrón Fachada.

Controller (Controlador).

Este patrón propone asignar la responsabilidad de controlar el flujo de eventos de un sistema, a clases específicas llamadas controladores. Los controladores no ejecutan las tareas sino que las delegan en otras clases, con las que mantiene un modelo de alta cohesión. En el sistema las clases a las cuales se les asignan estas responsabilidades se encuentran dentro de paquetes con la siguiente estructura vnz.mpprij.prevencion.cr.den.web y dentro de esta están los controladores. A modo de ejemplo se escoge la clase DenunciaFormController.java, esta clase se encarga de controlar la petición de registrar y actualizar una denuncia, manipulando los datos para en dependencia de estos delegarlos a las clases responsables de ejecutar la acción requerida; devolviéndole luego a la vista física la JSP con los datos para presentárselos al usuario. El fragmento de diagrama de secuencia que se muestra en la (Fig. 2.3) correspondiente al caso de uso “Registrar Denuncia” es un ejemplo de lo antes expuesto:

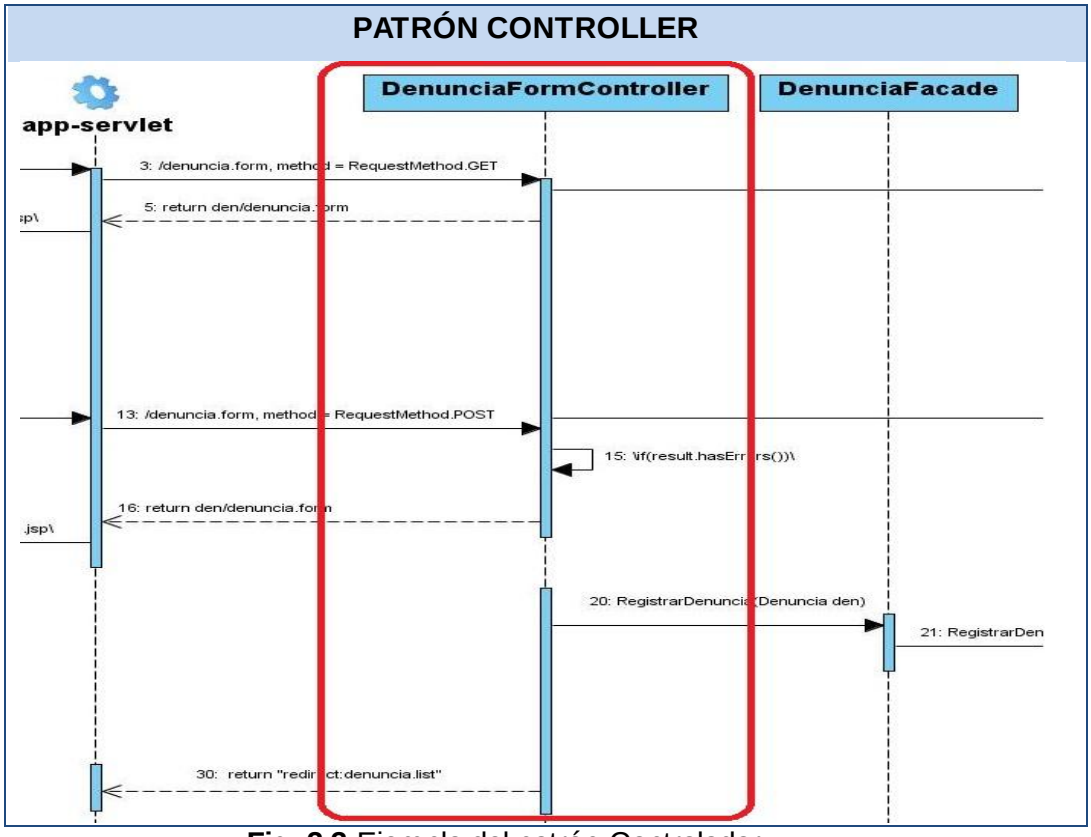

Fig. 2.3 Ejemplo del patrón Controlador

Front-Controller (Controlador Frontal).

El patrón propone utilizar un controlador como el punto inicial de contacto para manejar las peticiones del usuario en una aplicación. El controlador maneja el control de peticiones, incluyendo la invocación de los servicios de seguridad como la autentificación y autorización, la elección de una vista apropiada, el manejo de errores, y el control de la selección de estrategias de creación de contenido. Este patrón es utilizado por Spring MVC a través del DispatcherServlet que en el sistema se encuentra configurado en el **app-servlet.** Un ejemplo de cómo funciona este patrón en el sistema sería el siguiente; si el usuario decide visualizar en el navegador la página con el listado de denuncias, esta petición (la URL denuncia.list.htm) es recibida por el **dispatcher-servlet,** este posee un mecanismo para determinar cuál controlador maneja esta petición (DenunciaMultiActionController en este caso), luego este controlador toma la petición y ejecuta la tarea, devolviendo un ModelAndView al **dispatcher-servlet**, el que se encarga de despachar la petición a la Vista (denuncia.list.jsp)

Otro patrón de diseño utilizado fue la Inyección de Dependencias basada en metadata, el cual constituye una de las potencialidades del framework Spring 3.0.2.

Dependency Injection (Inyección de dependencias).

Es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase quien cree el objeto. Es una forma de Inversión de Control, que está basada en constructores de Java. Este 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 métodos set o bien a través del constructor, pero estos objetos se instancian una vez, se guardan en una factoría y se comparten por todos los usuarios (al menos en el caso de Spring) y evitando tener que andar extendiendo clases.

En el sistema está presente su aplicación ya que a todos los controladores se le inyectan las fachadas correspondientes para su funcionamiento mediante el uso de la anotación @Resource la cual provee de atributos a clases mediante los métodos set, de esta misma manera se soluciona la dependencia entre las fachadas y sus servicios, dependientes estos últimos del DAO Genérico.

Data Access Object (DAO).

Patrón de Diseño Core J2EE que centraliza todo el acceso a datos en una capa independiente, aislándolo del resto de la aplicación. Sus principales beneficios son: que reduce la complejidad de los objetos de

negocio al abstraerlos de la implementación real de la comunicación con la fuente de datos y que permite una migración más fácil de fuente de datos.

El uso de este patrón en el sistema se evidencia en la capa de Acceso a Datos donde se encuentra el DenunciaDao y el DenunciaDaoImpl encargados de realizar la gestión de los datos entre las clases que contienen la lógica de negocio (servicios) y el ORM Hibernate. El fragmento de diagrama de secuencia que se muestra en la (Fig. 2.4) correspondiente al caso de uso “Registrar Denuncia” es un ejemplo de lo antes expuesto.

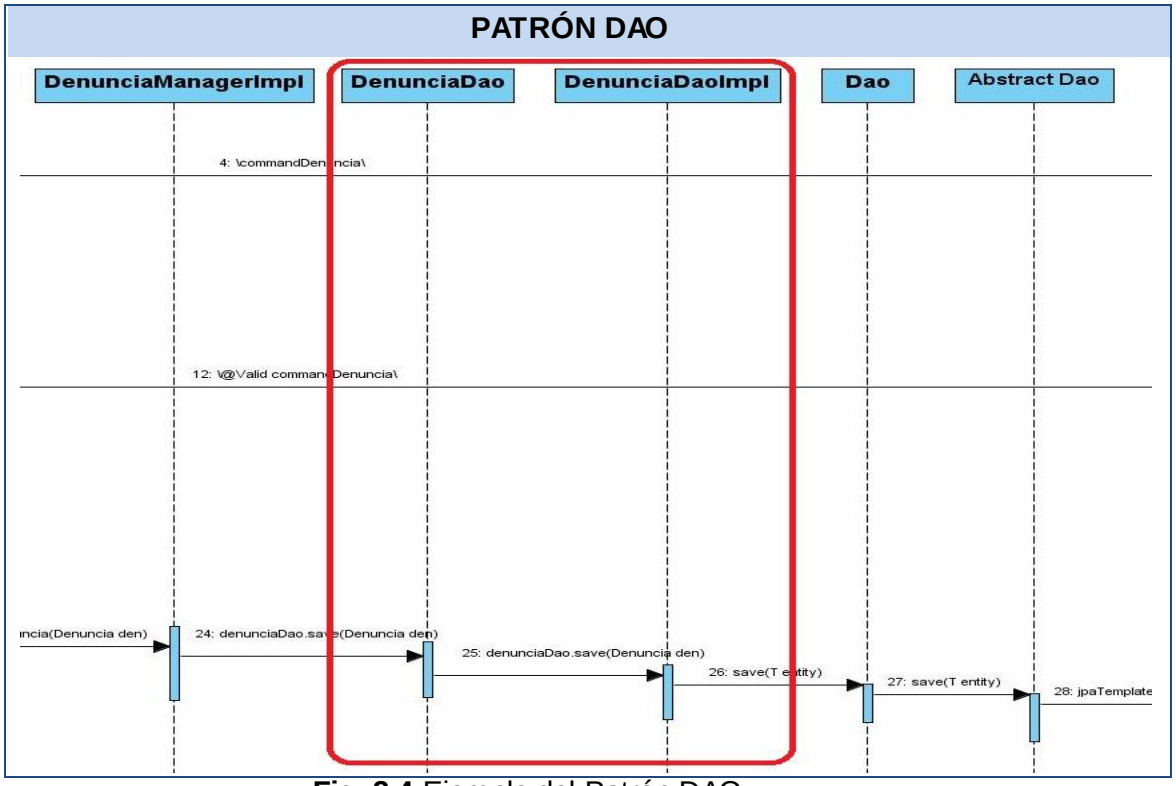

Fig. 2.4 Ejemplo del Patrón DAO.

Alta Cohesión.

La cohesión es la medida de la fuerza que une a las responsabilidades de una clase. Una clase con baja cohesión es aquella que hace muchas cosas no afines o muchas tareas, lo que trae como resultado dificultades para entender, reutilizar y conservarla. Una clase con **alta cohesión** mejora la claridad y la facilidad de su uso, su mantenimiento se simplifica y es fácil de reutilizar. A menudo, se genera un bajo acoplamiento. El sistema fue diseñado en módulos definidos a partir de que en los mismos se manejaran la menor cantidad de entidades posibles de manera tal que las clases pertenecientes a las diferentes

capas se le asignaran las responsabilidades necesarias y bien delimitadas para garantizar una alta cohesión generando por ende **bajo acoplamiento**.

2.5 Diagramas de Clases del Diseño.

Un diagrama de clases del diseño describe gráficamente las especificaciones de las clases del software y de las interfaces de la aplicación. Sirve también para visualizar las relaciones entre las clases que involucran el sistema.

El diseño de las clases partió de los requerimientos funcionales definidos. A continuación se muestra en la (Fig.2.5) el Diagrama de clases del diseño para el caso de uso Gestionar Denuncias perteneciente al Módulo de Denuncias, que abarca los siguientes requisitos agrupados para este caso de uso:

CUS_D.2. Gestionar denuncia.

RF_D.7. Registrar denuncia.

El sistema permitirá registrar los datos de una denuncia realizada en la CR.

RF_D.8. Actualizar denuncia.

El sistema permitirá actualizar los datos de una denuncia realizada en la CR.

Las funcionalidades correspondientes a los restantes casos de uso del módulo Denuncias y los módulos Consejos Comunales y Recursos Humanos pueden ser consultadas en el (Anexo 4). Los diagramas de clases del diseño se encuentran detallados en el documento Modelo de diseño contenido en el Expediente de proyecto.

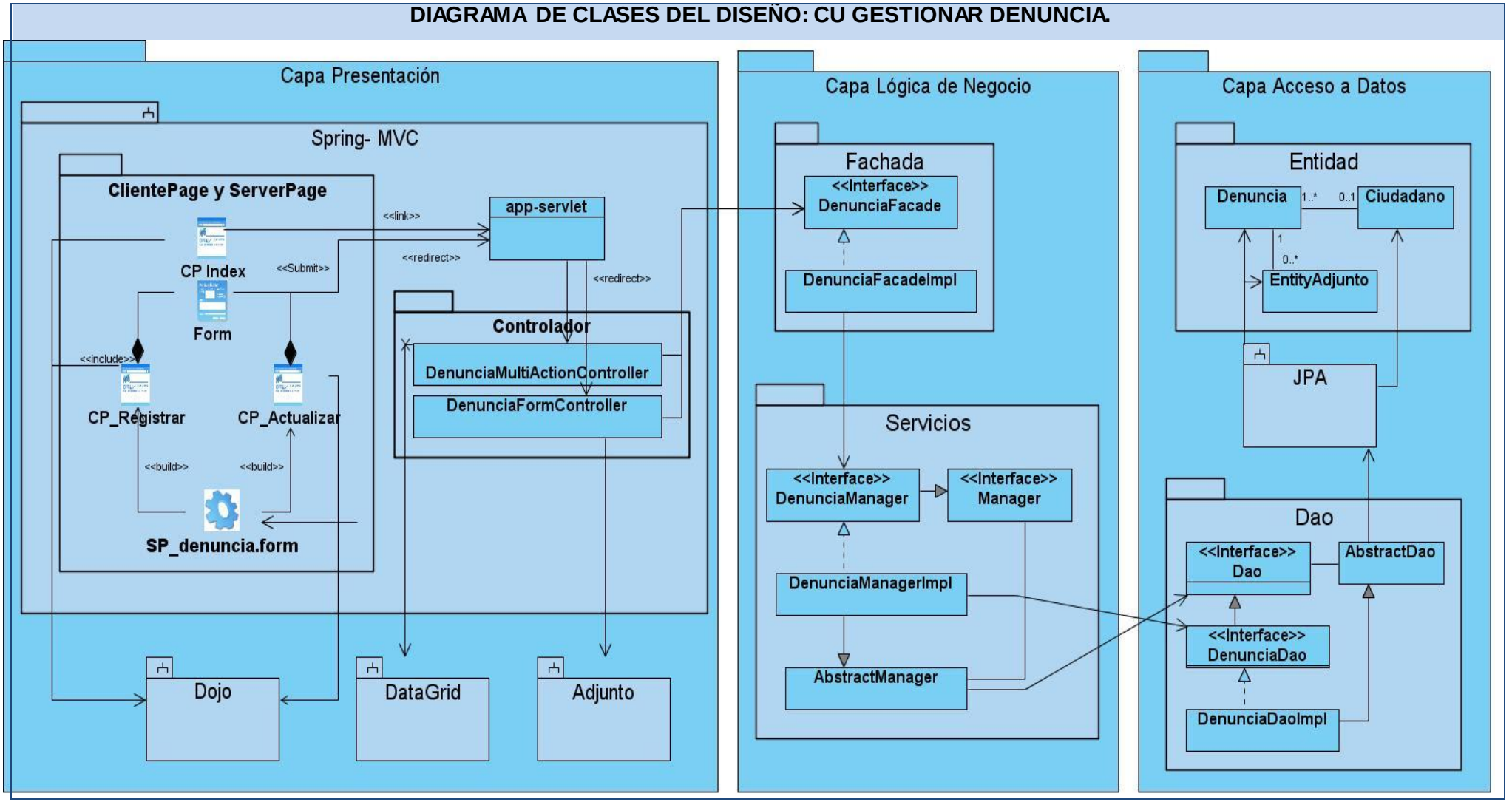

Fig. 2.5 Diagrama de clases del diseño para el CUS Gestionar Denuncia.

En el diagrama de Clases de diseño de la figura 2.5 se evidencia el estilo arquitectónico empleado: Arquitectura en 3 Capas, se observa una vista de la ubicación de las clases por capa y las relaciones entre ellas. Para acceder de la Capa de Presentación a la Capa Lógica de Negocio se establece la comunicación desde los controladores a la interfaz Fachada y para acceder de la Capa Lógica de Negocio a la Capa Acceso a Datos los servicios se comunican con la interfaz Dao.

De forma más detallada en la Capa de Presentación residen las peticiones que el usuario puede realizar sobre la aplicación, estas peticiones son recibidas por el DispatcherServlet y este delega su procesamiento a los Controladores **(Controller)** retornando un ModelAndView que es utilizado para construir las vistas que se muestran al usuario decoradas con el Subsistema Dojo, así los Controladores manejan el flujo web, la comunicación con las interfaces de la Capa Lógica de Negocio y delegan las responsabilidades de las tareas a clases específicas, además utilizan el Subsistema Datagrid, que es un componente para el trabajo con los listados, los filtros y el Subsistema Adjuntos, componente que nos permite subir y descargar adjuntos. Es importante destacar que en esta capa está presente el patrón arquitectónico **Modelo-Vista-Controlador (MVC)** que Spring utiliza en su funcionamiento.

La Capa Lógica de Negocio define e implementa las funcionalidades que responden directamente a los requisitos de manera que se conserve la integridad del sistema y de los datos. Está constituida por el patrón fachada 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, esta además provee una interfaz sencilla que sirve de intermediaria entre el cliente y los servicios.

En la Capa de Acceso a Datos se evidencia el patrón Dao que contiene los objetos que encapsulan la lógica de acceso a datos e interfaces brindadas para ser accedida desde la capa de negocio que utilizan el Subsistema JPA, API que es una especificación de Java que brinda flexibilidad al sistema ya que dicha interfaz es implementada por el framework Hibernate lo que permite cambiar de ORM sin que se vea afectada la Capa de Acceso a Datos. Hibernate posee soporte para múltiples dialectos, es decir, puede mapear diferentes sistemas gestores de bases de datos, lo que permite migrar a otra fuente de datos sin que esto afecte la Lógica de Negocio. Se encuentran además en esta capa las entidades.

2.5.1 Distribución de Clases por capa.

En cada una de las capas se distribuyen las clases fundamentales del sistema, compuestas por atributos y métodos ver Anexo 2

2.6 Diagramas de Secuencia.

Para la realización de los casos de uso del diseño es más factible el empleo de los diagramas de secuencia ya que representan con más claridad el flujo de las acciones que debe realizar el sistema.

Un diagrama de secuencia muestra las interacciones entre objetos ordenados en secuencia temporal; muestra los objetos que se encuentran en el escenario y la secuencia de mensajes intercambiados entre los objetos, para llevar a cabo la funcionalidad descrita por el escenario.

A continuación se muestran los diagramas de secuencia correspondientes a los escenarios: Registrar Denuncia (Fig. 2.9) y Actualizar Denuncia (Fig. 2.10) pertenecientes al Módulo de Denuncias.

DIAGRAMA DE SECUENCIA: REGISTRAR DENUNCIA.

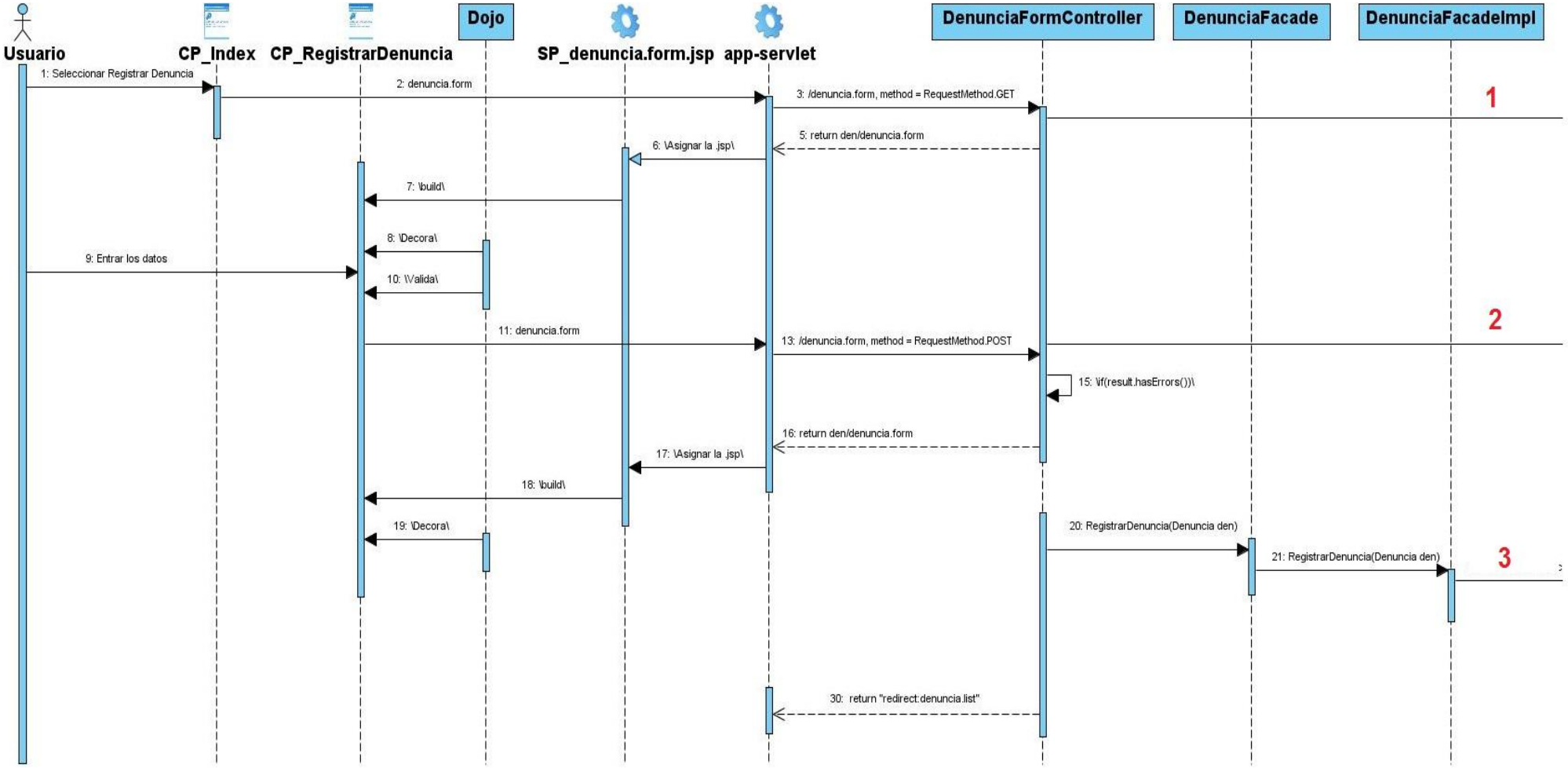

Fig. 2.9 Diagrama de Secuencia del CUS Gestionar Denuncia, escenario Registrar Denuncia. (**PARTE 1)**

DIAGRAMA DE SECUENCIA: REGISTRAR DENUNCIA.

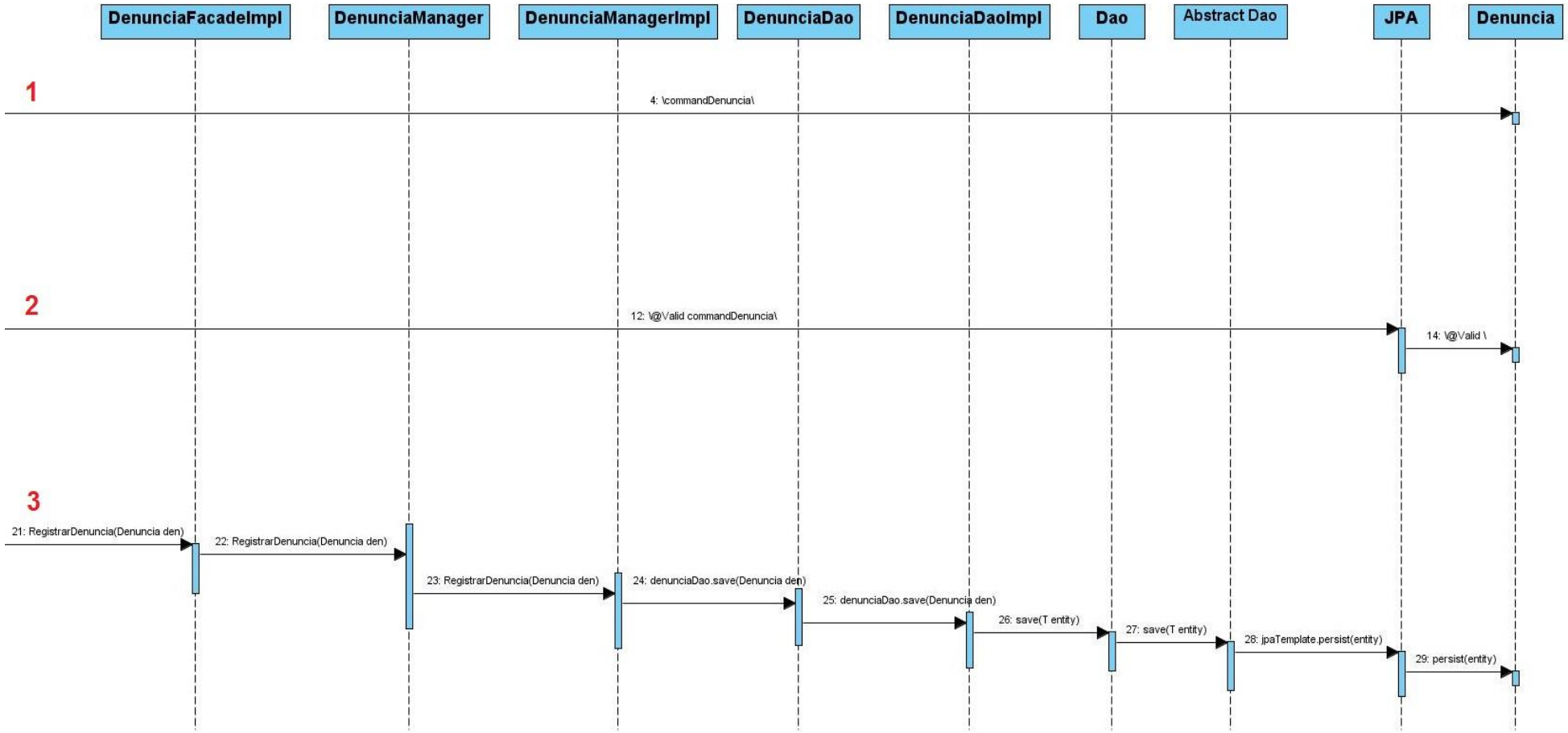

Fig. 2.9 Diagrama de Secuencia del CUS Gestionar Denuncia, escenario Registrar Denuncia. (**PARTE 2)**

DIAGRAMA DE SECUENCIA: ACTUALIZAR DENUNCIA.

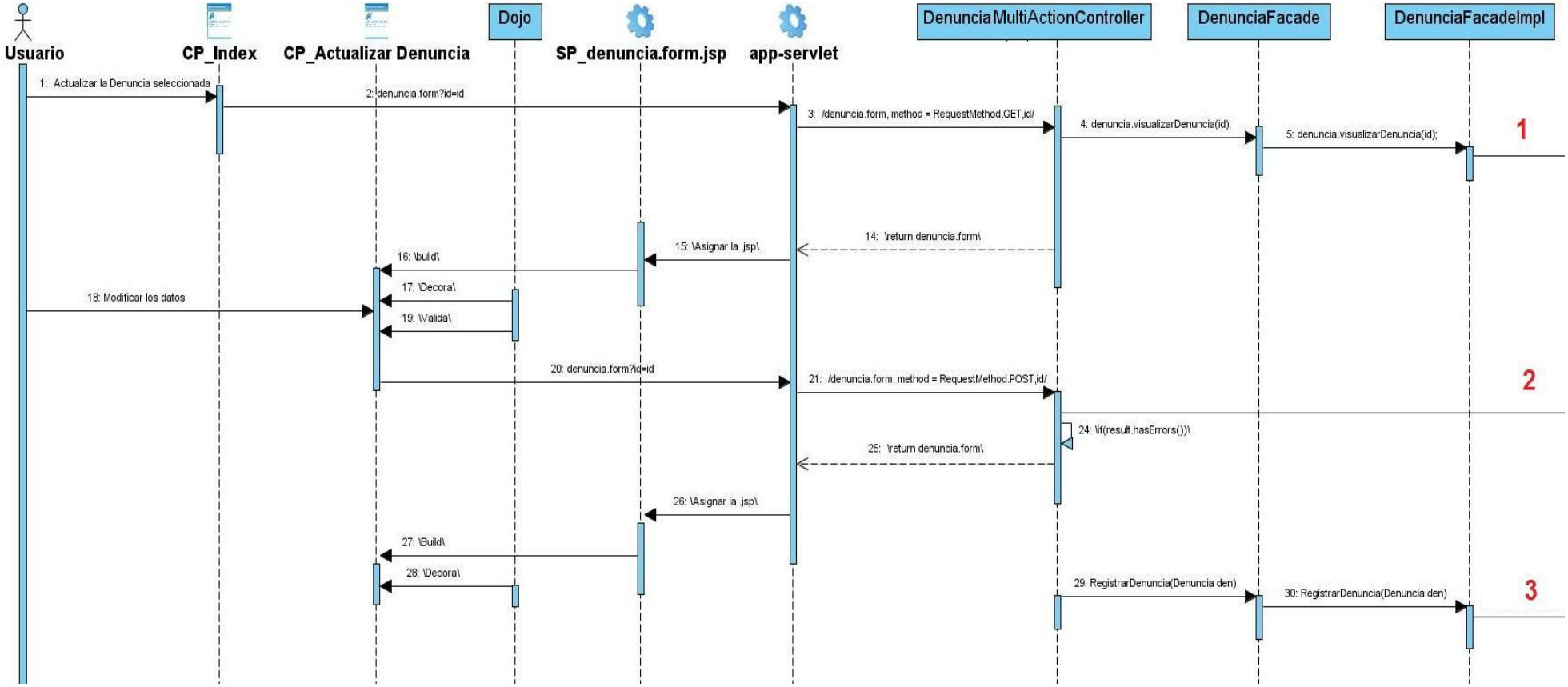

Fig. 2.10 Diagrama de Secuencia del CUS Gestionar Denuncia, sección Actualizar Denuncia. (**PARTE 1**)

DIAGRAMA DE SECUENCIA: ACTUALIZAR DENUNCIA.

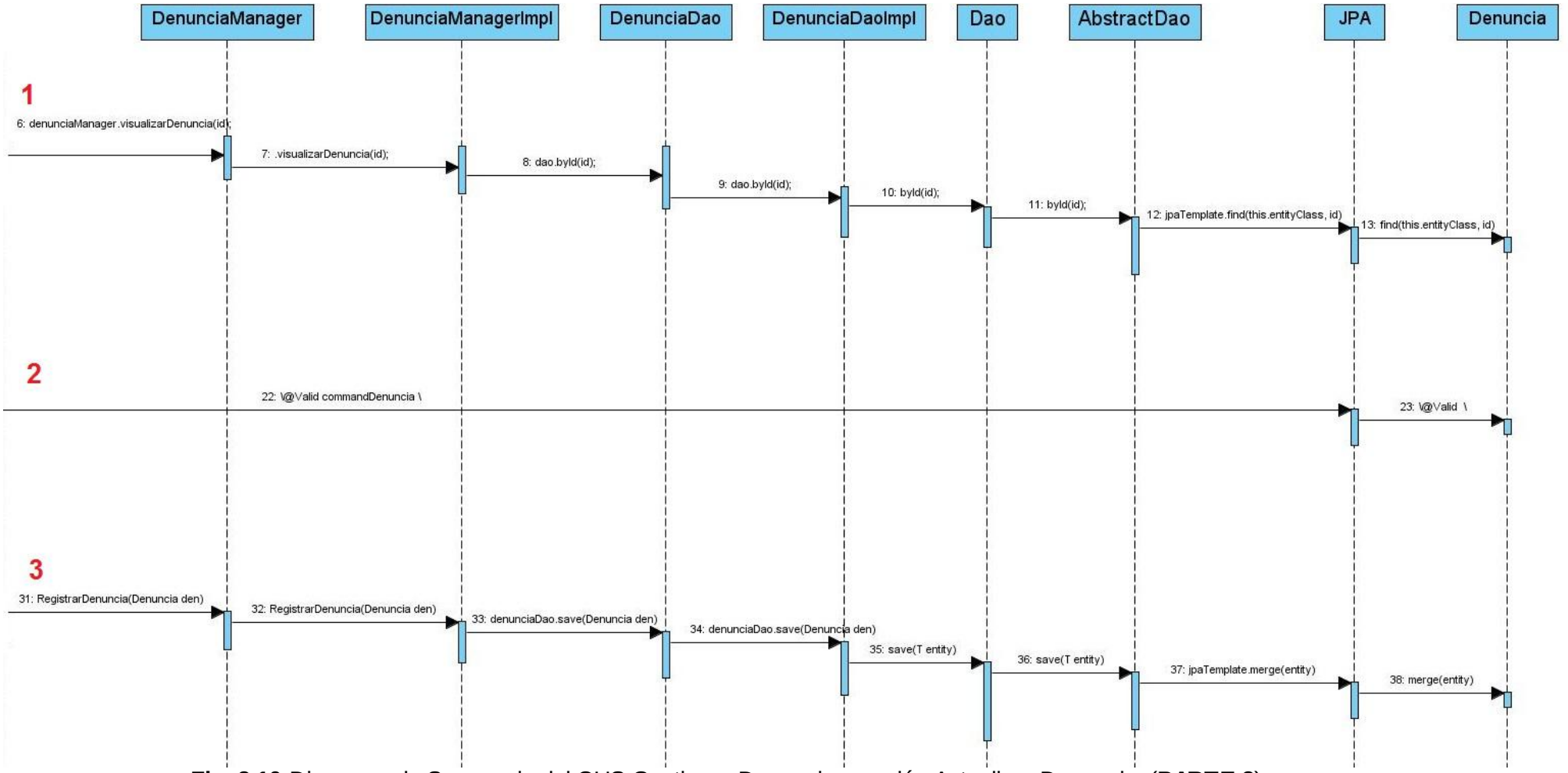

Fig. 2.10 Diagrama de Secuencia del CUS Gestionar Denuncia, sección Actualizar Denuncia. (**PARTE 2**)

2.7 Conclusiones.

En el presente capítulo:

- Se describió el estilo arquitectónico: Arquitectura en Capas (Tres Capas)
- Se describieron los patrones de diseño aplicados.
- Se realizaron los diagramas de clases de diseño a partir de los requerimientos funcionales definidos
- Se realizaron los diagramas de secuencia más relevantes según los escenarios del CUS Gestionar Denuncia.
- Se distribuyeron los atributos y los métodos correspondientes a las clases por cada capa.

CAPÍTULO 3: IMPLEMENTACIÓN DEL SISTEMA

3.1. Introducción.

Durante este capítulo se realiza el flujo de trabajo de implementación, se analizan los principales artefactos como el Modelo de Implementación que incluye componentes, subsistemas de implementación y diagramas de componentes. En este capítulo se comienza a implementar el sistema en términos de componentes mediante las clases del diseño.

3.2. Modelo de Implementación.

El Modelo de implementación representa la composición física de la implementación en términos de subsistemas de implementación y elementos de implementación. Describe como los elementos de diseño se implementan en componentes. Dicho modelo se considera el artefacto más significativo del flujo de trabajo de Implementación, debido a la importancia que tiene para los desarrolladores comprender el funcionamiento del sistema desde el punto de vista de componentes y sus relaciones. Este modelo está conformado por el diagrama de componentes. [21]

3.2.1. Diagrama de Componentes.

Dentro del Modelo de Implementación se encuentran los diagramas de componentes. En este se observan cómo están organizados, estructurados e implementados en términos de componentes los elementos del diseño, además es válido destacar la presencia de nuevos componentes como es el caso de los ficheros javascript y los properties que son ficheros de configuración del componente Datagrid donde se especifican los campos que se desean mostrar en los listados, los filtros y los reportes, en este fichero se especifica además la entidad asociada a dicho componente.

A continuación se muestra el Diagrama de Componentes del CUS Gestionar Denuncias perteneciente al Módulo Denuncias (Fig. 3.1)

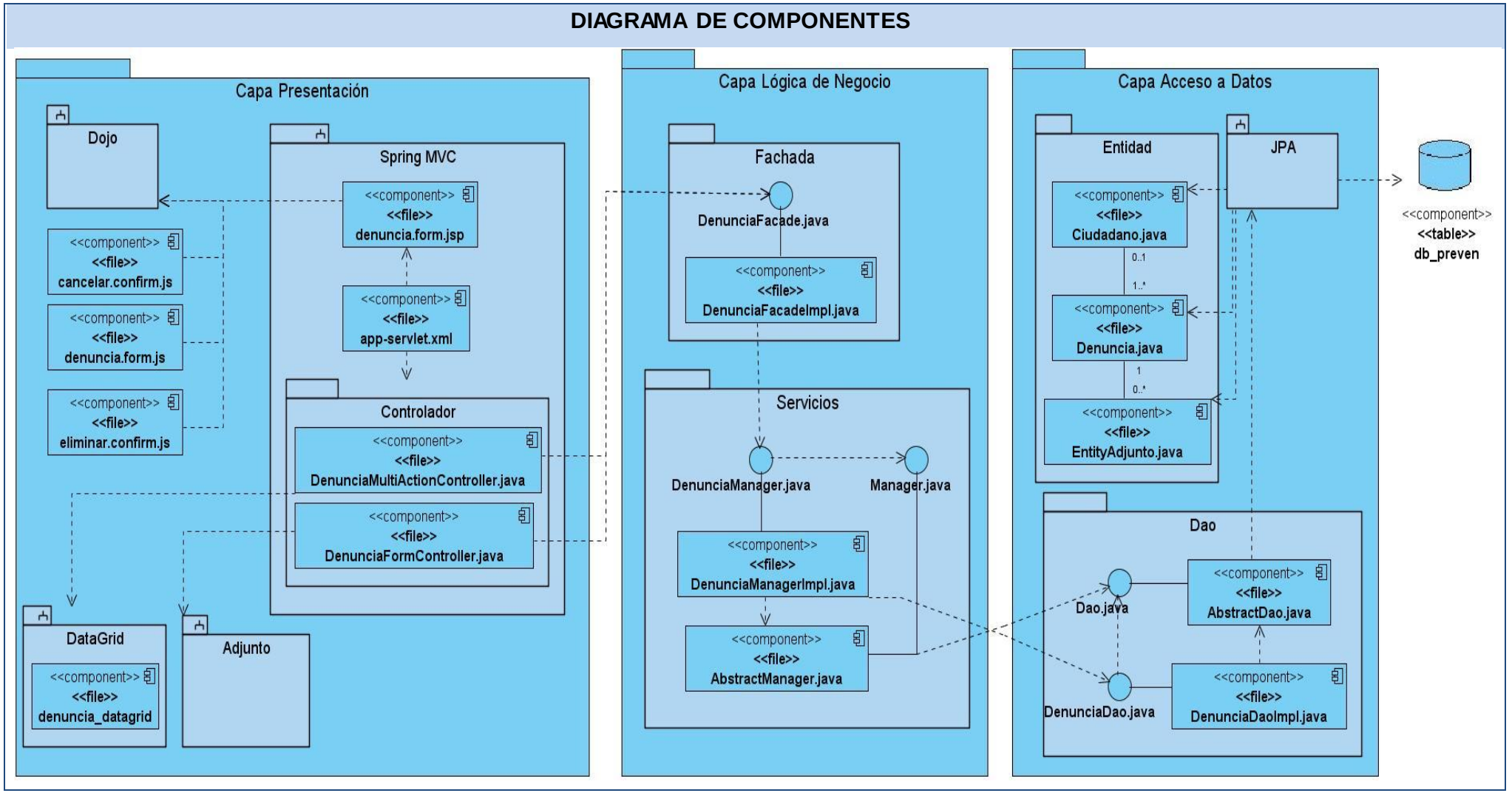

Fig. 3.1 Diagrama de Componentes: CUS Gestionar Denuncias.

3.2.2 Diagrama de Componentes por capas.

A continuación se muestran los componentes referentes al Diagrama de Componentes del CUS Gestionar Denuncia perteneciente al Módulo de Denuncia, distribuidos en cada una de las capas y las relaciones entre ellos :Capa de Presentación (Fig. 3.2), Capa de Lógica de Negocio (Fig. 3.3) y Capa de Acceso a Datos (Fig. 3.4).

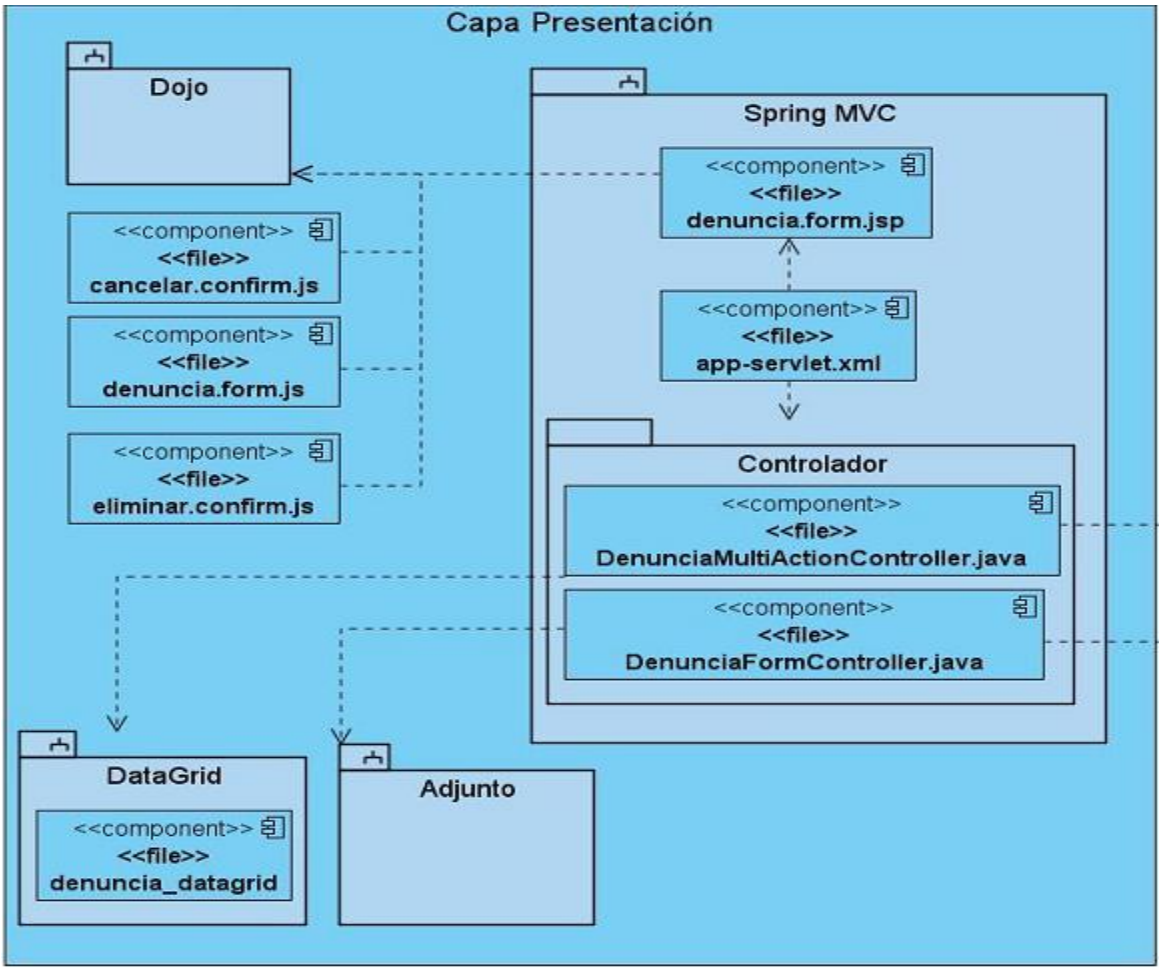

Fig. 3.2 Capa de Presentación (CUS Gestionar Denuncia)

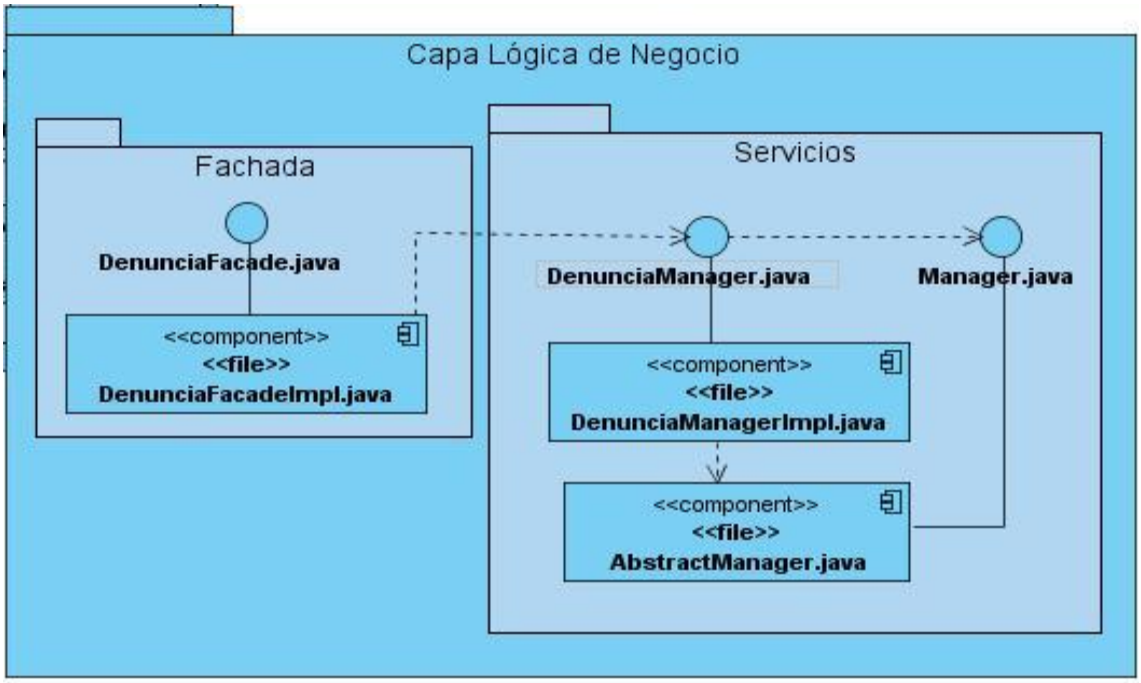

Fig. 3.3 Capa de Lógica de Negocio (CUS Gestionar Denuncia).

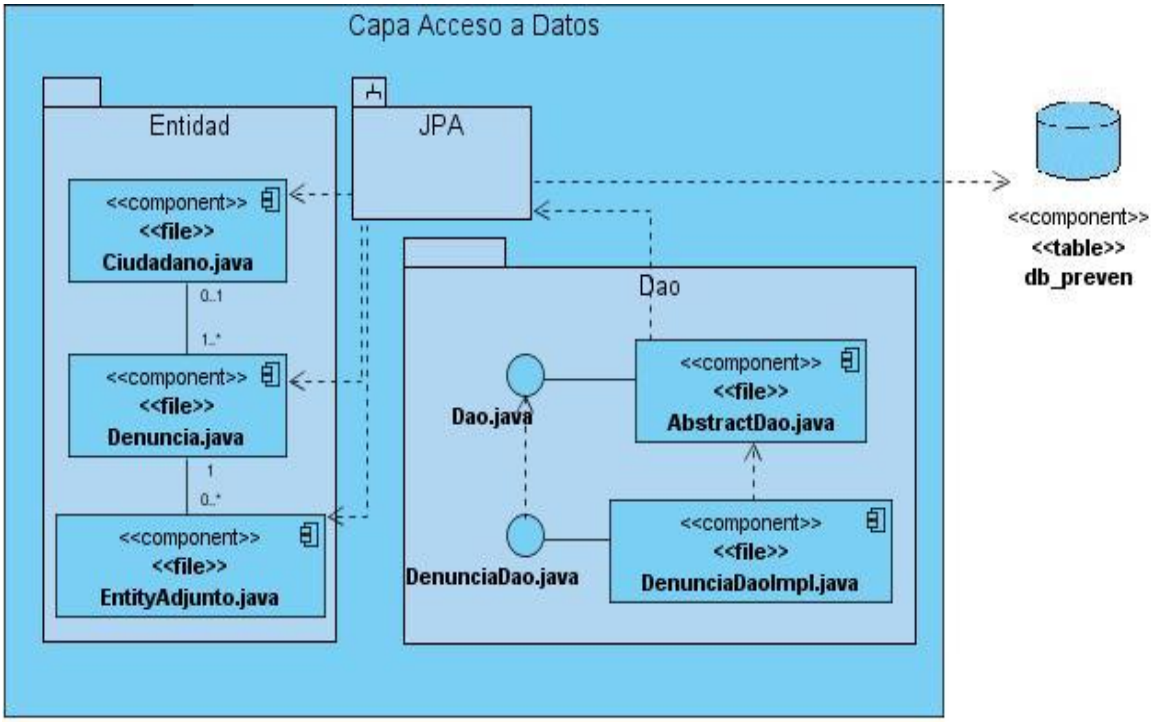

Fig. 3.4 Capa de Acceso a Datos (CUS Gestionar Denuncia).

3.2.3 Distribución de los componentes en el Diagrama de despliegue.

A continuación se muestran los componentes distribuidos en los diferentes nodos definidos en el Diagrama de Despliegue (Ver Anexo 3)

Pc-Cliente:

- Subsistema Dojo
- Clases Javascript
  - ✓ cancelar.confim
  - ✓ Eliminar.confirm
  - ✓ Denuncia.form

Pc-Servidor Web:

- Clase jsp
  - ✓ denuncia.form
- Clase java
  - ✓ DenunciaMultiActionController
  - ✓ DenunciaFormController
  - ✓ DenunciaFacade
  - ✓ DenunciaFacadeImpl
  - ✓ Manager
  - ✓ Abstract Manager
  - ✓ DenunciaManager
  - ✓ DenunciaManagerImpl
  - ✓ Dao
  - ✓ AbstractDao
  - ✓ DenunciaDao
  - ✓ DenunciaDaoImpl
  - ✓ Ciudadano
  - ✓ Denuncia
  - ✓ EntityAdjunto
- Subsistemas

- ✓ Datagrid
- ✓ Adjunto
- ✓ JPA

Servidor de Base de Datos

- db_preven

3.3 Código Fuente.

Para obtener una versión funcional de la aplicación se deben implementar las clases que respondan a las funcionalidades que esta debe cumplir, haciendo uso de los componentes que se han definido, como resultado se obtienen archivos que contienen el código fuente de la aplicación. Estos archivos son un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa. Por tanto, en el código fuente de un programa está descrito por completo su funcionamiento.

3.3.1 Estándares de Codificación.

Un estándar de codificación comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico.

A continuación alguno de los Estilos de Codificación utilizados en la implementación del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela, módulos: Denuncias, Recursos Humanos y Consejos Comunales.

Declaración de variables:

- Codificar cada variable con un nombre que intente describir lo más cercano posible la utilidad que va a tener. Ejemplo: direccion, origen, lugar entre otros.
- Deben empezar con minúscula y en caso de nombres compuestos la siguiente se escribe con mayúscula. Ejemplo: tipoActividad, clasificacionDelito.
- 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. Ejemplo: ciudadano.

Nombre de Métodos:

- El nombre debe describir lo más posible la utilidad que va tener. Ejemplo: RegistrarDenuncia (Denuncia denuncia).
- Deben empezar con mayúscula y en caso de nombres compuestos la siguiente también se escribe con mayúscula. Ejemplo: ActualizarDenuncia (Denuncia denuncia).
- 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 de acceso a datos, correspondientes al patrón de diseño Data Access Object terminan con la palabra "Dao". Ejemplo: ***DenunciaDao.***
- Las implementaciones reales de las interfaces DAO comienza con el nombre de la interfaz correspondiente y terminan con la palabra "Impl". Ejemplo: ***DenunciaDaoImpl***.

Todas las interfaces que se utilizarán para la persistencia heredarán de una interfaz **Dao** genérica, la cual contendrá todos los métodos necesarios. De esta misma forma todas las implementaciones reales de las interfaces heredarán de la implementación real de la interfaz **Dao**, en este caso la clase **AbstractDao**. De esta manera, se agilizará el proceso de desarrollo y se logrará un estándar en la implementación por parte del equipo de desarrollo, ejemplo:

Clases de la capa de Negocio:

- Las interfaces que representan las operaciones del negocio terminarán con la palabra "Facade". Ejemplo: ***DenunciaFacade.***
- Las implementaciones de las interfaces de negocio comenzarán con el nombre de la interfaz correspondiente y terminaran con la palabra "Impl". Ejemplo: ***DenunciaFacadeImpl.***
- Las interfaces que representan las operaciones de negocio que se van a exponer o consumir como servicio terminarán en la palabra "Manager". Ejemplo: ***DenunciaManager.***
- 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: ***DenunciaManagerImpl.***

Todas las interfaces que representan las operaciones de negocio que se van a exponer o consumir como servicio heredarán de una interfaz **Manager** genérica, la cual contendrá todos los métodos necesarios. De esta misma forma todas las implementaciones reales de estas interfaces heredarán de la implementación real de la interfaz **Manager**, en este caso la clase **AbstractManager**. De esta manera, se agilizará el proceso de desarrollo y se logrará un estándar en la implementación por parte del equipo de desarrollo, ejemplo:

Clases de la capa de Presentación:

- Las clases que manejan el flujo de la capa de presentación, es decir, los controladores, terminarán con la palabra "Controller".

Paquetes

Para cada módulo que conforme el sistema la nomenclatura quedaría de la siguiente forma: un paquete central con el nombre que representa al módulo, en este caso se utilizó la abreviatura **den** para hacer referencia al módulo Denuncia, **cc** para hacer referencia al módulo Consejos Comunales y **rh** para hacer referencia al módulo Recursos Humanos.

Organización de los ficheros.

Los ficheros se agruparan por paquetes, cada paquete se corresponderá con cada una de las capas a implementar: La Capa de presentación estará agrupada en la carpeta web, en el caso de la Capa de lógica de negocio, las interfaces estarán agrupadas en las carpetas service y facade, las clases que implementan estas interfaces estarán en las subcarpetas impl. Las clases que representan el dominio estarán en la carpeta domain, en caso de alguna configuración o clases auxiliares se tendrán en la carpeta config, en la carpeta dao estarán la interfaces Dao y las clases que implementan estas interfaces estarán en las subcarpetas impl.

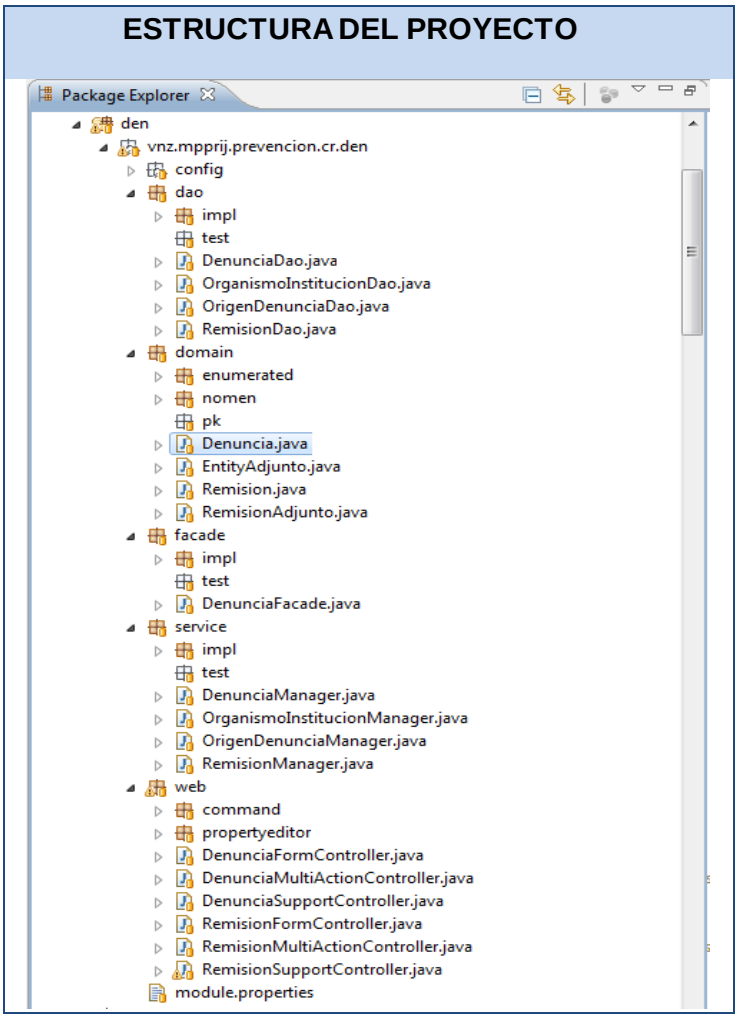

Fig. 3.2 Estructura del proyecto.

3.3.2 Ejemplo de Código Fuente.

A continuación se muestra un fragmento de código donde se ve el uso de la anotación @Controller, dando comportamiento de Controlador a la clase accion_controllers. Para solucionar la dependencia de esta con otras clases se le inyectan mediante la anotación @Resource la instancia DenunciaFacade y mediante la anotación @Autowired la instancia AttachmentEngine.

EJEMPLO DE CÓDIGO FUENTE

DenunciaFormController.java | DenunciaFacade.java | DenunciaFacadeImpl.java | Denuncia.java | DenunciaManager.java | DenunciaMultiActionController.java

```
@Controller
@SessionAttributes("den_denuncia_command")
public class DenunciaFormController{

    @Resource
    private DenunciaFacade facade;

    @Autowired
    private AttachmentEngine attachment;

    @InitBinder
    public void InitBinder(WebDataBinder binder) {

        binder.registerCustomEditor(Parroquia.class, new ParroquiaEditor());
        binder.registerCustomEditor(UsuarioSistema.class, new UsuarioEditor());
        binder.registerCustomEditor(Municipio.class, new MunicipioEditor());
        binder.registerCustomEditor(Sector.class, new SectorEditor());
        binder.registerCustomEditor(Estado.class, new EstadoEditor());
        binder.registerCustomEditor(OrigenDenuncia.class, new OrigenEditor());
        binder.registerCustomEditor(TipoActividad.class,
                new ActividadesEditor());
        // binder.registerCustomEditor(TipoDenuncia.class, new
        // TipoDenunciaEditor());
        binder.registerCustomEditor(ClasificacionDelito.class,
                new ClasificacionDelitoEditor());
        // binder.registerCustomEditor(TipoOrganismoReferido.class, new
        // TipoOrganismoReferidoEditor());
        BinderUtil.registerPkEditor(binder, new Class[] {
                MunicipioPK.class, ParroquiaPK.class,
                SectorPK.class});
    }

    @RequestMapping(value = "/denuncia.form", method = RequestMethod.GET)
    protected final String showForm(Model model,HttpSession session,
            @RequestParam(value = "__id", required = false) Long id)
            throws Exception {

        String title = "Registrar denuncia";

        CommandDenunciaCiudadano command = new CommandDenunciaCiudadano();
```

Fig. 3.3 Ejemplo de Código Fuente: Clase DenunciaForm Controller.

3.4 Validación.

Validar los datos es una tarea común que se produce a través de cualquier aplicación, desde la capa de Presentación hasta la capa de Acceso a Datos. A menudo, la misma lógica de validación se aplica en cada capa, empleándose mucho tiempo en su implementación y propenso a errores. Para evitar la

duplicación de estas validaciones en cada capa, se aplicó la lógica de validación directamente en el dominio, ver Anexo1. Para este proceso se utiliza Hibernate Validator que implementa el API JSR-303 Bean Validation para la plataforma Java, posibilitando una validación declarativa a través de anotaciones que se hacen cumplir en tiempo de ejecución. Spring contiene la anotación @Valid que se integra con Hibernate Validator garantizando la validación del lado del Servidor en cualquiera de las capas. Además, contiene clases para el manejo de los formularios las cuales se pueden comunicar con el dominio anotado, ventaja que fue utilizada para generar todas las validaciones del lado del Cliente de forma dinámica. [22]

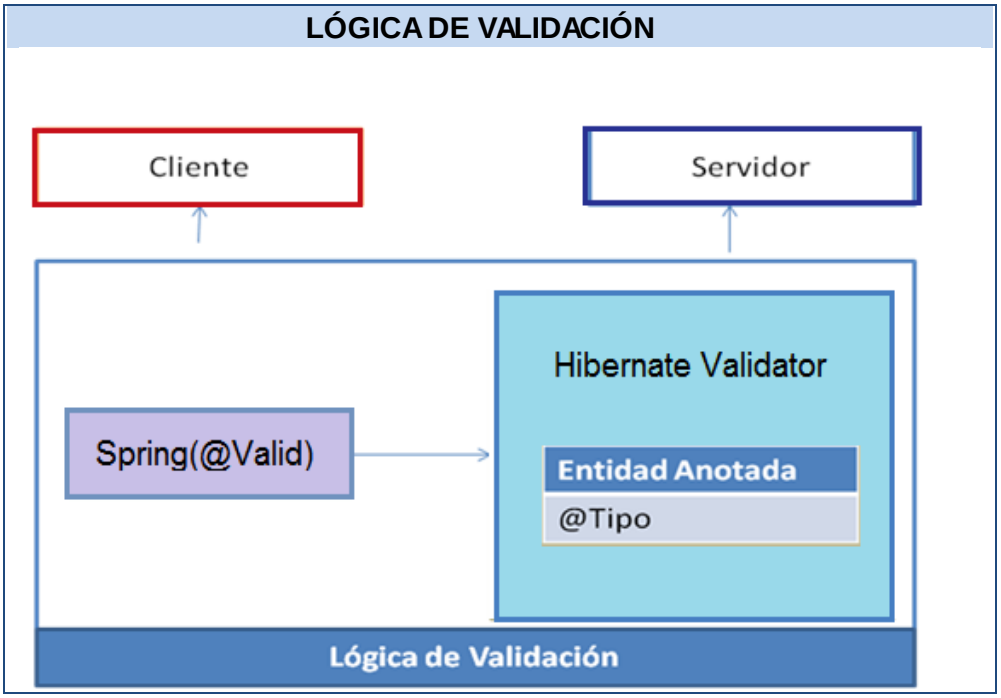

Fig. 3.4 Lógica de validación aplicada en el sistema.

3.5 Interfaces principales de la aplicación.

A continuación se muestran las pantallas del sistema correspondientes al Registrar denuncia (Fig. 3.5) y Listado de denuncias (Fig. 3.6) pertenecientes al módulo Denuncias:

INTERFAZ INSERTAR DENUNCIA

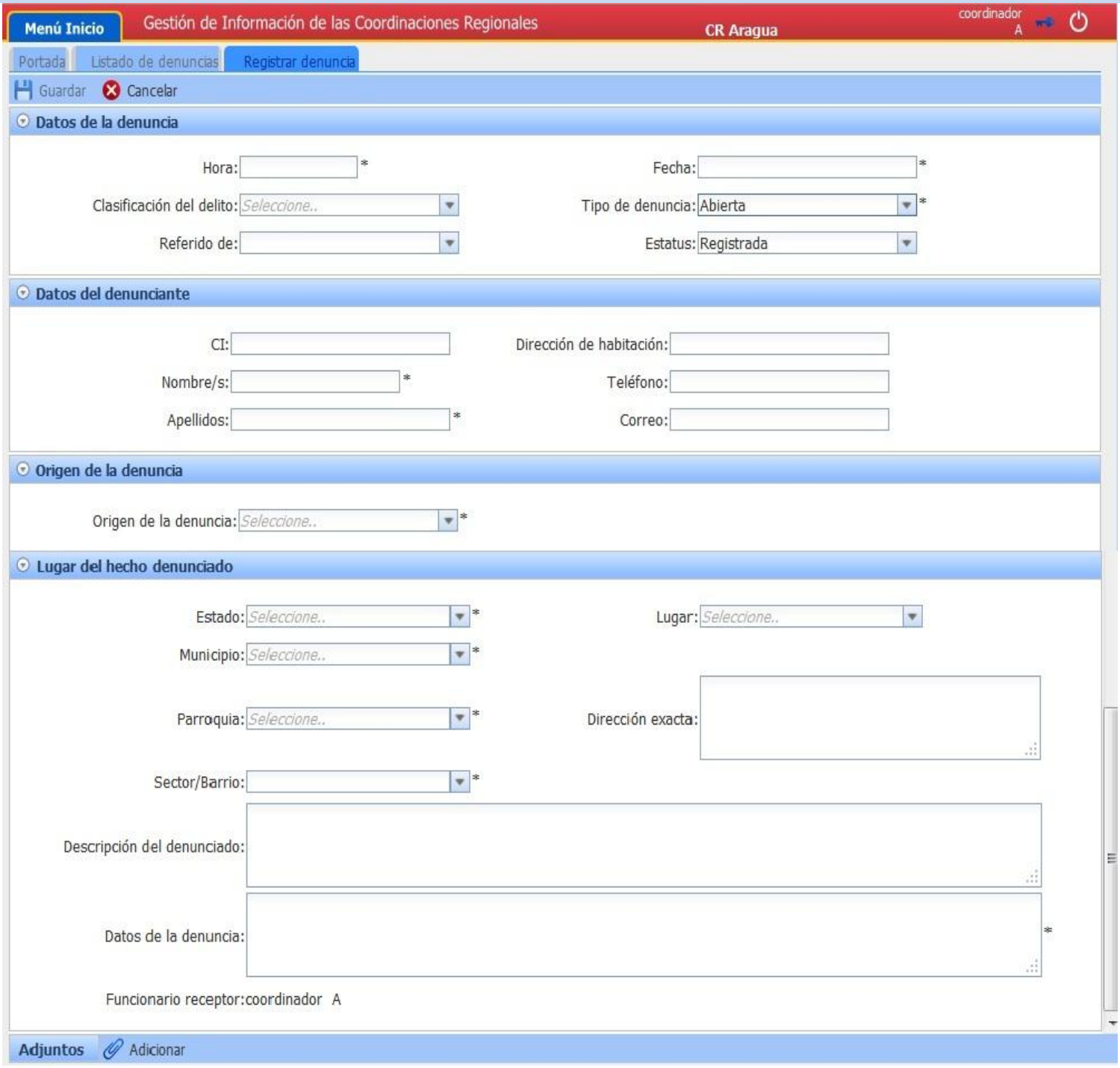

Fig. 3.5 Interfaz Insertar Denuncia.

INTERFAZ LISTADO DE DENUNCIAS

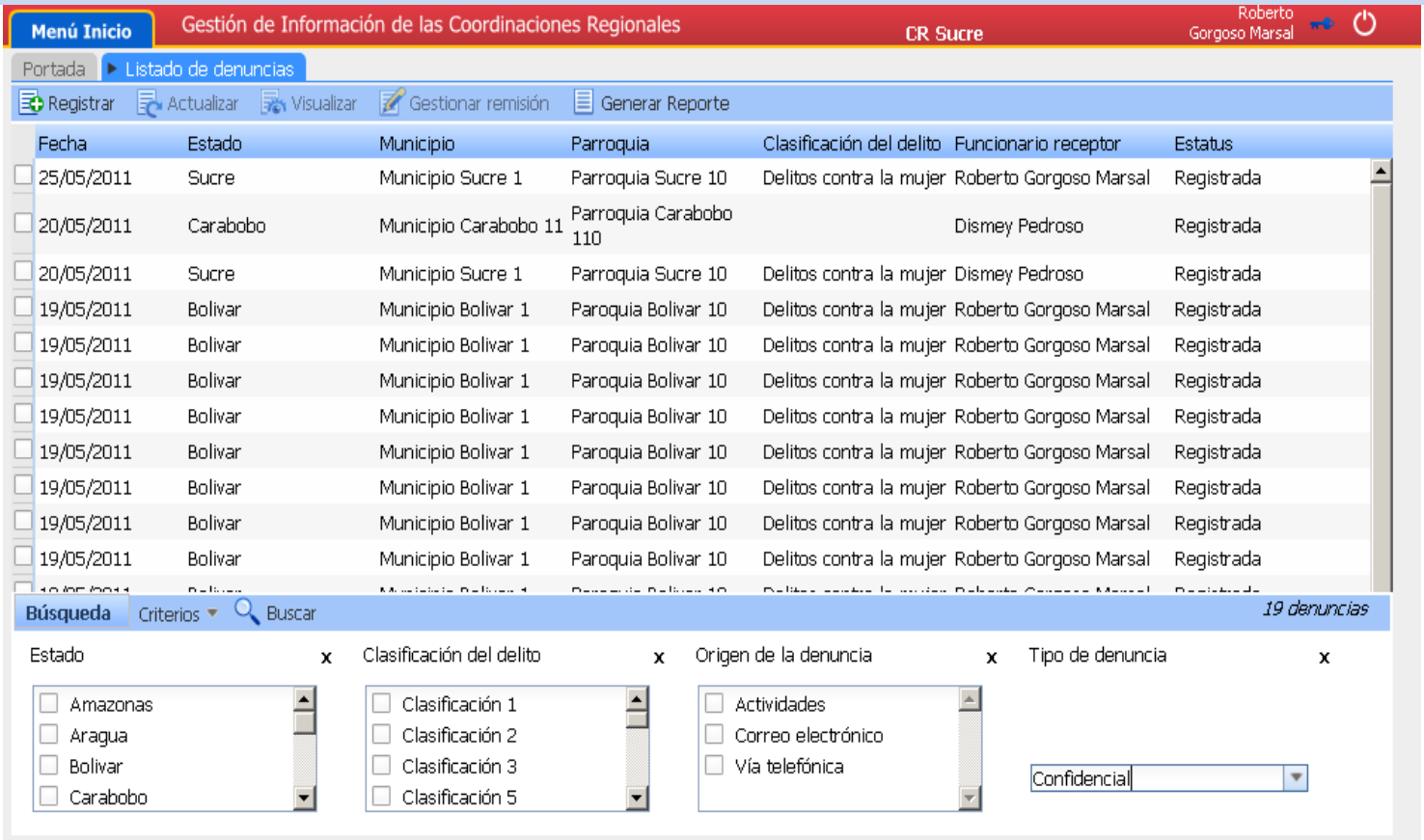

Fig. 3.6 Interfaz Listado de Denuncias

3.6 Conclusiones.

- Esta etapa de desarrollo se caracteriza por resultados ya visibles para los clientes y gratificantes para los desarrolladores, ya que queda implementada la aplicación con las principales funcionalidades que se definieron para la iteración del producto.
- Se elaboraron los diagramas de componentes del CUS Gestionar Denuncia perteneciente al módulo Denuncias.
- Se describió la arquitectura de validación aplicada en el sistema.

CAPÍTULO 4: PRUEBAS DEL SISTEMA

4.1 Introducción.

Durante este capítulo se realiza el flujo de trabajo de prueba para determinar el status de la calidad de un producto software mediante el proceso de pruebas. En este proceso se ejecutan pruebas dirigidas a componentes del software y al sistema de software en su totalidad, con el objetivo de medir el grado en que el software cumple con los requerimientos. Además en las pruebas se usan casos de prueba, especificados de forma estructurada.

4.2 Pruebas de software

Las pruebas de software constituyen un pilar indispensable para evaluar y determinar la calidad de un software. Concretamente se puede definir pruebas de software como:

- El proceso de ejecución de un programa con la intención de descubrir errores previos a la entrega al usuario final.
- Una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones específicas, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente.[23]

4.3. Niveles de Pruebas.

A la hora de evaluar dinámicamente un sistema software se debe comenzar por los componentes más simples y más pequeños e ir avanzando progresivamente hasta probar todo el software en su conjunto. Las pruebas se aplican durante todo el ciclo de desarrollo del software para diferentes objetivos y en distintos niveles de trabajo, dentro de estos se distinguen:

- **Pruebas de Desarrollador**

Es la prueba diseñada e implementada por el equipo de desarrollo. Tradicionalmente estas pruebas han sido consideradas solo para las pruebas de unidad, aunque en la actualidad en algunos casos pueden ejecutar pruebas de integración. Se recomienda que estas pruebas cubran más que las pruebas de unidad.

- **Pruebas al Sistema**

Son usualmente conducidas para asegurar que todos los módulos trabajan como sistema sin error. Es similar a la prueba de integración pero con un alcance mucho más amplio. Las pruebas del sistema examinan qué tan bien el sistema cumple con los requerimientos de la organización y su utilidad, seguridad y desempeño. También se realizan estas pruebas a la documentación del sistema. Concretamente se debe comprobar que:

- Se cumplen los requisitos funcionales establecidos.
- Sea correcto el funcionamiento y rendimiento de las interfaces hardware, software y de usuario.
- Sea apropiada la documentación de usuario.
- Se verifique el rendimiento y respuesta en condiciones límite y de sobrecarga.

- **Pruebas de Unidad**

Se enfocan en un programa o un componente que desempeña una función específica que puede ser probada y que se asegura que funcione tal y como lo define la especificación del programa. Los programadores siempre prueban el código durante el desarrollo, por lo que las pruebas unitarias son realizadas solamente después de que el programador considera que el componente está libre de errores.

4.4.Tipos de Pruebas.

Cada nivel de prueba engloba una técnica de prueba específica según los atributos de calidad que se deseen verificar con las pruebas al software.

Pruebas de Funcionalidad.

Entre las técnicas de pruebas que se realizan en el sistema está la que evalúa la funcionalidad de éste. Se pueden dilucidar distintos tipos de prueba, según los parámetros de evaluación que abarca la técnica:

- **Pruebas funcionales:**

Objetivo: Con el propósito de verificar el cumplimiento de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados.

Caso de prueba: Un caso de prueba se diseña según las funcionalidades descritas en los casos de usos. Este diseño se elabora previo a la realización de las pruebas funcionales de la aplicación. Cada

planilla de caso de prueba recoge la especificación de un caso de uso, dividido en secciones y escenarios, para hacer más fructífera la ejecución de las pruebas.

Los casos de prueba específicos del CUS Gestionar Denuncia se registran en el documento: Diseño de Caso de Prueba del CUS Gestionar denuncia perteneciente al Módulo Denuncia, que se encuentran en el Expediente de proyecto del Sistema Informático de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito. [24]

Método Caja Negra: En el proceso de pruebas en cuestión se hace uso de los casos de prueba de Caja Negra por ser éste el método empleado en la liberación del producto software, de ahí el motivo por el que se diseña un caso de prueba por caso de uso del sistema.

No Conformidades: La plantilla de No Conformidades recoge los errores que son detectados durante la revisión de la documentación del sistema. Se elabora un documento por cada revisión que se haga y se controlan a través de versiones según se vayan eliminando los errores. Además de estar inmerso en la planilla de diseño de casos de prueba, estas No Conformidades se van registrando en un documento aparte para luego enviarlo al equipo de desarrolladores. [25]

En la siguiente tabla se expone un resumen de los resultados de las No Conformidades detectadas por Calisoft durante su revisión y que las clasifica según los tipos de errores de la siguiente manera:

Tipos de Errores
• Funcionalidad **(F)** • Validación **(V)** • Correspondencia con otro artefacto **(CA)** -Aplicación **(A)** • Error de Interfaz **(EI)** • Formato **(FO)** • Redacción **(R)** • Error Técnico **(ET)** • Excepción **(EX)** • Ortografía **(O)**

ITERACIÓN 1

MÓDULO	F	V	CA(A)	EI	FO	R	ET	EX	O
Denuncias	10	-	-	1	1	-	-	-	1
Consejos Comunales	2	1	-	-	-	-	-	-	-
Recursos Humanos	5	5	-	9	-	-	-	2	1

ITERACIÓN 2

MÓDULO	F	V	CA(A)	EI	FO	R	ET	EX	O
Denuncias	2	-	-	-	-	-	-	-	-
Consejos Comunales	2	-	-	-	3	2		2	-
Recursos Humanos	11	3	-	5	-	3	-	1	1

ITERACIÓN 2

MÓDULO	F	V	CA(A)	EI	FO	R	ET	EX	O
Denuncias	1	1	-	2	-	-	-	-	-
Consejos Comunales	1	-	-	-	-	-	-	-	-
Recursos Humanos	-	-	-	-	-	-	-	-	-

Tabla 4.1 No Conformidades detectadas por Calisoft según el tipo de error.

Las NC detectadas reflejadas en la tabla anterior se clasifican según los tipos de errores teniendo en cuenta el impacto en su solución para el equipo de desarrollo.

CLASIFICACIÓN DE NO CONFORMIDADES SEGÚN EQUIPO DE DESARROLLO

Baja (B) -Redacción, Formato, Ortografía

Media (M)- Error de Interfaz, Correspondencia con otro artefacto.

Alta (A)-Validación, Funcionalidad, Error Técnico, Aplicación y Excepción.

MÓDULO	TOTAL DE NO CONFORMIDADES	# DE NO CONFORMIDADES QUE PROCEDEN	# DE NO CONFORMIDADES QUE NO PROCEDEN	# DE NO CONFORMIDADES RESUELTAS
Iteración 1				
Denuncia	14	**(A)-**10 **(M)-**2 **(B)-**1	2	13
Recursos Humanos	23	**(A)-**11 **(M)-**9 **(B)-**1	2	21
Consejos Comunales	4	**(A)-**2 **(B)-**1	1	3
Iteración 2				
Denuncia	4	**(A)-**2 **(B)-**2	0	4
Recursos Humanos	42	**(A)-**14 **(M)-**5 **(B)-**18	5	37
Consejos Comunales	16	**(A)-**4 **(M)-**3 **(B)-**6	3	13
Iteración 3				
Denuncia	7	**(A)-**2 **(M)-**2 **(B)-**3	1	6
Recursos Humanos	-	-	-	-
Consejos	9	**(A)-**1	0	9

Comunales		**(B)-**8		

Tabla 4.2 Resumen de NC del proceso de liberación de Calisoft según el impacto de solución para equipo de desarrollo.

- **Pruebas de control de acceso al sistema:**

  Objetivo: Se realizan para garantizar que los usuarios estén restringidos a funciones específicas o su acceso esté limitado únicamente a los datos que están autorizados a acceder.

En el caso del Módulo Denuncia los permisos de acceso estarían distribuidos de la siguiente manera:

Rol/es	Funcionalidad/es
Coordinador/a Regional. **Equipo Técnico Profesional (ETP).**	-Listado de denuncias. -Listado de remisiones. -Listado general de remisiones. - Imprimir listado de remisiones - Buscar remisión del listado general de remisiones. -Registrar denuncia. -Actualizar denuncia. -Visualizar denuncia. - Adicionar adjunto de denuncias. -Descargar adjuntos de denuncias. - Imprimir denuncia. - Buscar denuncia. - Generar reporte de la denuncia. -Visualizar reporte de la denuncia. - Exportar a Excel. -Gestionar remisión. - Registrar remisión. -Actualizar remisión

	- Visualizar remisión. - Imprimir una remisión.
Director/a Nacional (DN). **Secretaria de la DN.** **Supervisor/a.**	-Descargar adjuntos de denuncias. - Listado de denuncias (de la supervisora y DN) - Imprimir denuncia. - Buscar denuncia. - Buscar denuncia (de la supervisora y DN) - Visualizar reporte de la denuncia. - Exportar a Excel. -Generar reporte de denuncia de la supervisora.

Pruebas exploratorias

Objetivo: Se ejecutan pruebas a ciegas y al azar a la aplicación con el objetivo de encontrar errores y provocar fallos al sistema. Para este tipo de pruebas no se sigue ningún manual de usuario ni documento de especificación de casos de uso o requisitos.

Pruebas de regresión

Objetivo:

-Comprobar por qué ha dejado de funcionar algo que ya funcionaba. El objetivo de las pruebas de regresión es no tener que "volver atrás".

- Asegurar que cuando una No Conformidad encontrada en el sistema ha sido corregida ninguna de las funcionalidades liberadas previamente falla como resultado de las correcciones o que las características nuevamente agregadas no han creado conflicto con las versiones anteriores del software, es una manera de darle seguimiento y tratamiento a las No Conformidades. [25]

Pruebas de Confiabilidad.

La confiabilidad del software se refiere a la precisión con la que una aplicación proporciona, sin errores, los servicios que se establecieron en las especificaciones originales. Se realizan para evaluar el rendimiento y está relacionado también con la consecución de resultados correctos y con el control de la detección de errores y de la recuperación para evitar que se produzcan errores.

- **Prueba de estrés**

  Objetivo: Esta prueba se utiliza normalmente para hacer colapsar la aplicación. Se va doblando el número de usuarios que se agregan a la aplicación y se ejecuta una prueba de carga hasta que se colapsa. Este tipo de prueba se realiza para determinar la solidez de la aplicación en los momentos de carga extrema.

Pruebas de Rendimiento.

Las pruebas de rendimiento o desempeño como también se conoce, son las que se realizan, desde una perspectiva, para determinar cuán rápido realiza una tarea un sistema en condiciones particulares de trabajo.

- **Pruebas de carga**

  Objetivo: Se realiza generalmente para observar el comportamiento de una aplicación bajo una cantidad de peticiones esperadas. Esta carga puede ser el número esperado de usuarios concurrentes utilizando la aplicación y que realizan un número específico de transacciones durante el tiempo que dura la carga.

4.5. Herramientas para automatizar las pruebas.

4.5.1. JUnit.

Para realizar las pruebas unitarias se utilizó el **JUnit** integrado al IDE Eclipse en forma de plug-in, lo cual permite que la generación de las plantillas necesarias para la creación de las pruebas de una clase Java se realice de manera automática, facilitando al programador, enfocarse en la prueba y el resultado esperado, dejando a la herramienta, la creación de las clases que permiten coordinar las pruebas.

Para las Pruebas de Unidad se implementaron clases de pruebas para identificar errores en la Capa de Acceso a Datos y Capa de Lógica de Negocio. A continuación se muestra el resultado satisfactorio de las pruebas realizadas a los métodos *RegistrarDenuncia, ActualizarDenuncia y VisualizarDenuncia* de la clase *DenunciaDao* pertenecientes a la Capa de Acceso a Datos del Módulo Denuncias.

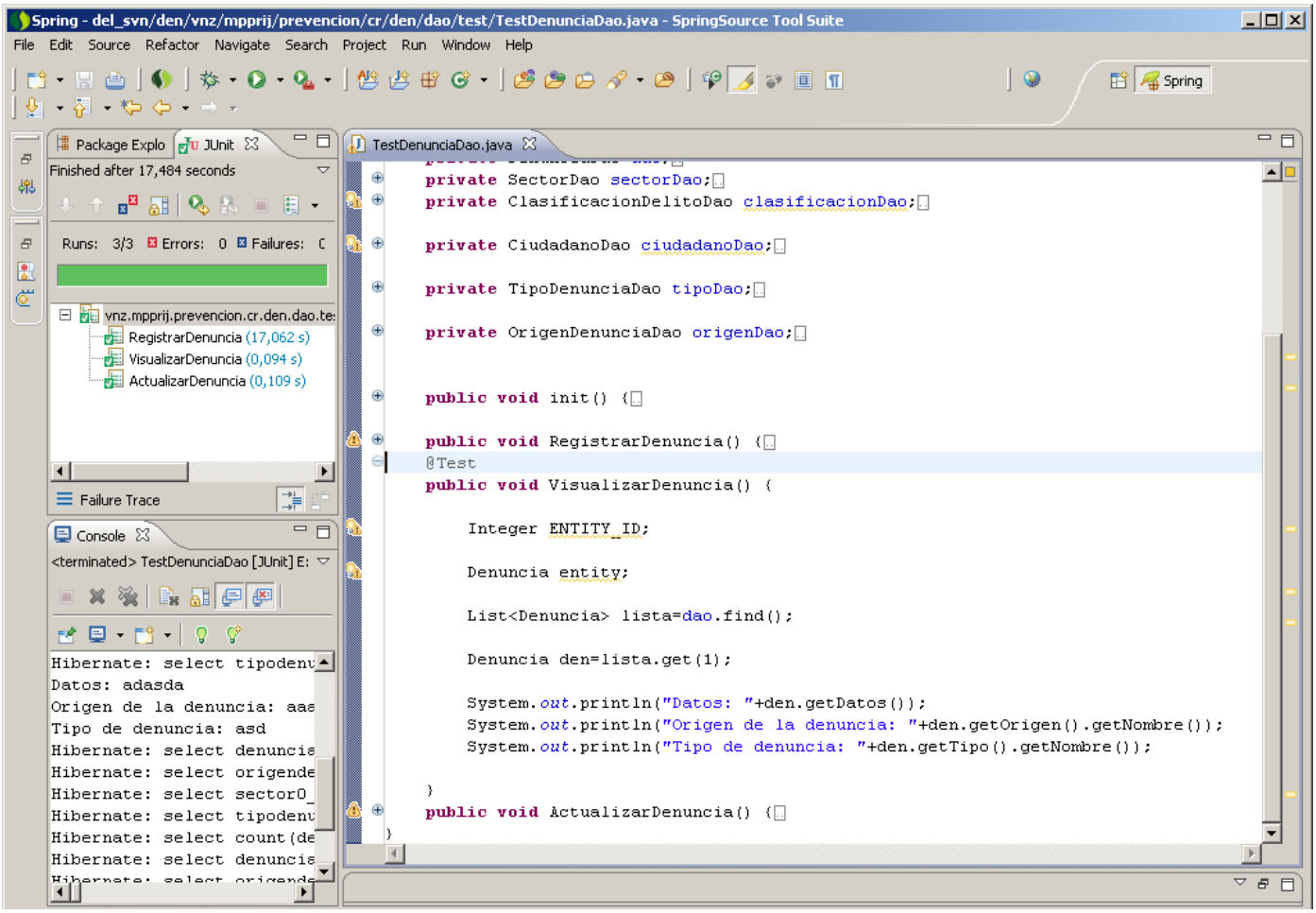

Fig 4.1 Método VisualizarDenuncia, Clase DenunciaDao (Capa Acceso a Datos).

4.5.2 JMeter.

Utilizar **JMeter** en aplicaciones web para la comprobación de los recursos del sistema, supone una mayor efectividad en el proceso y en la fiabilidad de los resultados. Como herramienta de prueba dispone de varios componentes que facilitan la elaboración de los escenarios de prueba con la ventaja de simular para cada uno de esos escenarios miles de usuarios. De esta manera se verifica el rendimiento del sistema mediante las pruebas de Carga y Estrés.

Una ventaja de JMeter es que permite realizar pruebas de carga sobre cada una de las capas que conforman la aplicación a probar. De esta manera, en un momento dado, es sencillo localizar el foco que está generando contención y está afectando los tiempos de respuesta de un caso de uso específico. [26]

4.5.3 Análisis de los resultados de las pruebas de Carga y Estrés con la herramienta JMeter.

Antes de comenzar con el análisis cuantitativo de los índices obtenidos es elemental declarar las condiciones de hardware en que se desarrollaron las pruebas, así como los requerimientos no funcionales de hardware y software que precisan JMeter y el sistema en cuestión.

JMeter	RNF Sistema de Gestión de Información de las CR	Condiciones PC
Máquina Virtual Java 1.6	**SOFTWARE:** **Servidor Web** • Sistema Operativo: Se recomienda Red Hat Enterprise Linux 4.0, aunque pudiera utilizarse otra distribución de Linux para servidores. • Servidor web: Apache Jakarta Tomcat 6.0.20. **PC clientes** Navegador: para un funcionamiento óptimo del Sistema de Gestión se recomienda: Mozilla Firefox 3.5 o superior, Google Chrome 3.0 o superior. **HARDWARE:** **Servidor Web** • Procesador Intel Pentium Xeon, 4 GHz, de 4 Gb de RAM. • 30 GB de HD.	Se emplearon para la ejecución de las pruebas 2 PC con las siguientes características: **SOFTWARE:** • Sistema Operativo:Windorws XP y Linux 10.10 • Servidor web: Apache Tomcat 6.0.20. • Navegador: Mozilla Firefox 4 **HARDWARE:** • Intel(R) Pentium 4 • CPU 3.0 Ghz • 80 GB disco duro • 1GB RAM.

	• NIC: 10/100/1000 bit Ethernet controller. **PC Cliente** • Procesador Intel Pentium IV, 2 GHz, de 256 Mb de RAM • 1 GB de HD disponible para caché del navegador (sin incluir el requerido para el sistema operativo). • Conectividad con el servidor correspondiente.	

Tabla 4.3 Especificación de recursos y condiciones del entorno de pruebas

En la tabla anterior se plasman las condiciones del entorno de pruebas en JMeter, así como los requerimientos no funcionales de hardware y software del sistema, declarados en la plantilla de Especificación de requisitos.

4.5.4 Resultados de la ejecución de las pruebas con la herramienta JMeter.

En el proceso de automatización de las pruebas, se realizaron 6 iteraciones que simulando el acceso concurrente de 30,100 y 150 usuarios respectivamente a la aplicación, con el servidor sobre el sistema operativo Linux y Windows. Se mezclaron las grabaciones de todos los escenarios, con la intención de obtener resultados generales del sistema; de manera que se simulara el acceso de las cantidades mencionadas distribuido en los diferentes escenarios de los casos de uso. De este modo se comprobó el comportamiento del sistema.

Servidor de aplicación sobre Sistema Operativo Windows:

Analizando la iteración para 30 usuarios conectados en un ciclo y un período de subida de 1 segundo, se obtuvieron los siguientes resultados del Informe Agregado:

Elemento	# Muestras	Media	Mediana	Línea de 90%	Min	Max	% Error	Rendimiento	Kb/sec
	2310	17	11	30	1	181	0.0%	1052.9 /sec	10343696.4

Tabla 4.4 Resumen del Informe Agregado obtenido para 30 usuarios.

De la misma forma, pero para un total de 100 usuarios, se obtuvieron los siguientes datos:

Elemento	# Muestras	Media	Mediana	Línea de 90%	Min	Max	% Error	Rendimiento	Kb/sec
	7700	80	45	143	0	2513	0.0%	784.8/ sec	7710416.9

Tabla 4.5 Resumen del Informe Agregado obtenido para 100 usuarios.

Luego de ejecutar las pruebas para simular las peticiones realizadas por 150 usuarios concurrentes, a todos los escenarios propuestos, se recogieron los resultados que a continuación se muestran:

Elemento	# Muestras	Media	Mediana	Línea de 90%	Min	Max	% Error	Rendimiento	Kb/sec
	11550	182	55	426	0	5223	0.03 %	433.1/ sec	4255122.1

Tabla 4.5 Resumen del Informe Agregado obtenido para 150 usuarios.

Servidor de aplicación sobre Sistema Operativo Linux:

Analizando la iteración para 30 usuarios conectados en un ciclo y un período de subida de 1 segundo, se obtuvieron los siguientes resultados del Informe Agregado:

Elemento	# Muestras	Media	Mediana	Línea de 90%	Min	Max	% Error	Rendi miento	Kb/sec
	2310	17	13	29	1	168	0.0%	987.2/ sec	9691730.8

Tabla 4.4 Resumen del Informe Agregado obtenido para 30 usuarios.

De la misma forma, pero para un total de 100 usuarios, se obtuvieron los siguientes datos:

Elemento	# Muestras	Media	Mediana	Línea de 90%	Min	Max	% Error	Rendi miento	Kb/sec
	7700	77	54	136	1	729	0.0%	1036.3 /sec	10174360. 7

Tabla 4.5 Resumen del Informe Agregado obtenido para 100 usuarios.

Luego de ejecutar las pruebas para simular las peticiones realizadas por 150 usuarios concurrentes, a todos los escenarios propuestos, se recogieron los resultados que a continuación se muestran:

Elemento	# Muestras	Media	Mediana	Línea de 90%	Min	Max	% Error	Rendi miento	Kb/sec
	11550	153	61	437	0	1509	0.0%	473.1/ sec	4255122.1

Tabla 4.5 Resumen del Informe Agregado obtenido para 150 usuarios.

Como se observa en los índices previamente expuestos luego de la ejecución de las pruebas, se puede comprobar que, los resultados obtenidos son satisfactorios puesto que el servidor empleado para la realización de dichas pruebas cuenta con características de hardware inferiores a las del servidor en que estará la aplicación. Con el servidor de aplicación montado en Linux a medida que se aumentó la cantidad de usuarios concurrentes se elevó el número de peticiones, lo que se corresponde con un incremento en el tiempo de respuesta. El % de error es de 0 para 30,100 y 150 usuarios, lo que indica

que la mayoría de las páginas se cargaron satisfactoriamente. Para el Sistema Operativo Windows lanzó un error de 0.3% para 150 usuarios, esto no es significativo porque el mínimo de usuarios a interactuar con la aplicación son 30 y se probó para 150 usuarios un # significativamente mayor Se puede concluir que el rendimiento del software sobre este sistema operativo es mejor que sobre Windows.

4.6 6 Resumen del resultado de las pruebas realizadas.

En las pruebas de Unidad se implementaron clases de pruebas para identificar errores en la Capa de Acceso a Datos y Capa de Lógica de Negocio, probando funcionalidades específicas con los métodos *RegistrarDenuncia, ActualizarDenuncia y VisualizarDenuncia* de la clase *DenunciaDao* pertenecientes a la Capa de Acceso a Datos del Módulo Denuncias posibilitando localizar errores tempranamente y corregirlos, haciendo cumplir los objetivos de dichas funcionalidades y asegurando que el código funciona de acuerdo a las especificaciones. En las pruebas funcionales se recogió un número de No Conformidades para cada uno de los módulos implementados que permitieron detectar errores y arreglarlos en el menor plazo posible y asegurar el correcto funcionamiento del sistema, se garantizó con las pruebas de Control de Acceso que solo los roles con permisos accedieron a las funcionalidades especificadas para cada uno de ellos, con las pruebas realizadas con el JMeter se comprobó que el sistema respondió satisfactoriamente bajo un alto número de peticiones y clientes conectados en condiciones de hardware inferiores a las del servidor en que estará la aplicación en su despliegue final. Estas pruebas realizadas han permitido validar el correcto cumplimiento de los requisitos planteados.

4.5 Conclusiones

- Para lograr una mayor calidad y eficacia en el proceso de automatización se llevó a cabo la concepción de las pruebas con el régimen que dictaminan las pautas para su desarrollo y además un buen acondicionamiento del ambiente de las pruebas.
- Se describieron los principales tipos de prueba utilizadas en enfocados a el cumplimiento de los requisitos pactados.
- Para realizar las pruebas unitarias se utilizó la herramienta JUnit.
- Para medir el rendimiento del sistema se ejecutaron pruebas de Carga y Estrés con la herramienta JMeter.

CONCLUSIONES

El desarrollo de los módulos Denuncias, Recursos Humanos y Consejos Comunales del Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela a partir del diseño de clases que cumplen con estándares y patrones, la implementación basada en la utilización de frameworks y la realización de pruebas que garantizan el cumplimiento de las funcionalidades definidas, posibilita a los especialistas de Prevención del Delito a partir de la información gestionada en dichos módulos identificar las zonas donde ocurren delitos con mayor frecuencia, familias expuestas a mayores riesgos y el control de los recursos humanos de las CRs. Esta es una herramienta de gestión que constituye un aporte a la soberanía tecnológica de la República Bolivariana de Venezuela, ya que está desarrollada bajo los nuevos paradigmas de aplicaciones web y bajo tecnologías de desarrollo de software libre.

RECOMENDACIONES

- Continuar con el desarrollo de los componentes realizados en Dojo para los listados que se encuentran en las interfaces de los módulos profundizando en el trabajo con el DataGrid de Dojo.
- Se recomienda el estudio y aplicación de Reflection para facilitar el desarrollo del sistema, disminuyendo la cantidad de código y la reusabilidad del mismo. El API Reflection es una herramienta muy poderosa que nos permite realizar en Java cosas que en otros lenguajes es imposible. Sin embargo y a pesar de su potencial, es un API bastante desconocido, sobre todo para los principiantes en el mundo de Java.

REFERENCIAS BIBLIOGRÁFICAS

1. Lourdes Aja. Gestión de la información, 2009, [En línea] [Citado el: 22 de Enero del 2011] Disponible en: http://www.eubca.edu.uy/materiales/planeamiento_de_servicios_Bibliotecarios/gestión_del_conocimiento_y_gestion_de_la_informacion.pdf
2. Cruz Carballo, Manuel Ivan. " Sistema de información para el apoyo a la toma de decisiones gerenciales" [En línea] [Citado el: 22 de Noviembre del 2010] Disponible en: http://www.monografias.com/trabajos17/sistema-gerencial/sistema-gerencial.shtml
3. Programa Integral de Gestión e Investigación para la Prevención del Delito, Gobierno Federal, México. Disponible en: http://www.presidencia.gob.mx/programas/seguridad/?contenido=35019 [Citado el: 12 de noviembre del 2011]
4. Jacobson, G. Booch y J. Rumbaugh, El Proceso Unificado de Desarrollo de Software. S.l.: PEARSON EDUCACIÓN S.A, 2000. [Citado el: 10 de Noviembre del 2010]
5. Visual Paradigm 2005 [En línea] [Citado el: 28 de Noviembre del 2010] Disponible en: http://www.visual-paradigm.com/product/vpuml/..
6. Definición.org Definición Lenguaje de Programación [En Línea] [Citado el:13 de Noviembre del 2010] Disponible en: http://www.definicion.org/lenguaje-de-programacion
7. Schildt Herbert Java 2 Manual de referencia - Madrid: [s.n.], 2001. [Citado el: 2 de Febrero del 2010]
8. Ciberaula 2009 Ciberaula [En Línea] [Citado el: 5 de Febrero del 2011] Disponible en: http://java.ciberaula.com/articulo/tecnologia_java/.
9. BEATON W. Eclipse Platform Technical Overview [En Línea] [Citado el: 5 de Febrero del 2011] Disponible en: http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html.
10. Perrone Paul J and Krishna J2EE Developer's Handbook [Citado el: 5 de Febrero del 2011] - Indianapolis, Indiana: Sam's Publishing, 2003.
11. Group, P.G.D. PostgreSQL. 2007 [En línea] [Citado el: 10 de Febrero del 2011] Disponible en: http://www.postgresql.org/.
12. Pecos, D. PostGreSQL. 2005 [En línea] [Citado el: 10 de Febrero del 2011] Disponible en: http://www.netpecos.org/docs/mysql_postgres/x15.html.

13. Sánchez Rico, Mario Alfredo, Spring, un framework de aplicación[En línea] [Citado el: 28 de Febrero del 2010] Disponible en:http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/sanchez_r_ma/capitulo3.pdf
14. Spring-Framework-Reference [PDF] [Citado el: 28 de Febrero del 2010]
15. Framework Dojo [En línea] [Citado el: 28 de Febrero del 2011] Disponible en: http://www.dojotoolkit.org/
16. hibernate.org [En línea] [Citado el: 28 de Febrero del 2011] Disponible en: Disponible en: http://www.hibernate.org/
17. jasperforge.org Sitio Web Oficial de JasperReport [En línea] [Citado el: 10 de Febrero del 2011] Disponible en: http://jasperforge.org/website/.
18. Estilos Arquitectónicos[En Línea][Citado el: 12 de Octubre del 2010] Disponible en: http://www.willydev.net/descargas/prev/IntroArq.pdf
19. Evolución y Orientaciones de Patrones [En línea] [Citado el: 15 de Abril del 2010] Disponible en: http://www.monografias.com/trabajos27/evolucion-patrones/evolucion-patrones.shtml?monosearch
20. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. [Citado el: 16 de septiembre del 2010]- *Design Patterns (Elements of Reusable Object-Oriented Software)*.
21. Ayuda extendida de Rational Rose Enterprise Edition 2003, consultado el: 28 de diciembre del 2010.
22. Hibernate Validator JSR 303 Reference Implementation Reference Guide 4.0.1.GA [PDF] [Citado el: 28 de Abril del 2010]
23. Conferencia # 5 Pruebas. Ingeniería del Software II. [Citado el: 05/12/2008] Disponible en: http://teleformacion.uci.cu
24. Diseño de Caso de prueba .Caso de Prueba Gestionar denuncia perteneciente al Módulo Denuncias, Expediente de Proyecto Sistema Informático de Gestión de Información de las Coordinaciones Regionales para la Dirección General de prevención del Delito.
25. No conformidades de los módulos Denuncias, Recursos Humanos y Consejos Comunales, Expediente de Proyecto Sistema Informático de Gestión de Información de las Coordinaciones Regionales para la Dirección General de prevención del Delito.
26. Mejoramiento del Proceso de Pruebas y Corrección de Defectos de Software en un ambiente globalizado. [En línea 2011] [Citado el: 11/01/2011] Disponible en: http://chie.uniandes.edu.co/~gsd/index.php?option=com_content&task=view&id=129&Itemid=183

27. Osorio, Eneisy y Asdrúbal. Propuesta de arquitectura para el Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito. UCI. Ciudad de la Habana, 2010.

BIBLIOGRAFÍA

1. Ayuda extendida de Rational Rose Enterprise Edition 2003, [En línea] [Citado el: 22 de Enero del 2010]
2. Avilas, Diagramas de Secuencia, 2009, [En línea] [Citado el: 22 de Enero del 2010] Disponible en: http://tvdi.det.uvigo.es/~avilas/UML/node42.html
3. BEATON W. Eclipse Platform Technical Overview [En Línea] [Citado el: 5 de Febrero del 2010] Disponible en: http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html.
4. Burbeck Steve Applications Programming in Smalltalk-80(TM): How to use Model-View ontroller (MVC). [Book] [Citado el: 22 de Abril del 2010]
5. Ciberaula 2009 Ciberaula [En Línea] [Citado el: 5 de Febrero del 2010] Disponible en: http://java.ciberaula.com/articulo/tecnologia_java/.
6. C. Walls Breidenbach Spring in Action. [PDF] s.l.: Manning Publications [Book]. – 2005 [Citado el: 20 de Enero del 2010]
7. Definición.org Definición Lenguaje de Programación [En Línea] [Citado el: 28 de Enero del 2010] Disponible en: http://www.definicion.org/lenguaje-de-programacion
8. Danciu Teodor The JasperReports Ultimate Guide Version 1.0 [Book]. - 2004 [Citado el: 20 de Enero del 2010]
9. Estilos Arquitectónicos[En Línea][Citado el: 12 de Febrero del 2010] Disponible en: http://www.willydev.net/descargas/prev/IntroArq.pdf
10. Evolución y Orientaciones de Patrones [En línea] [Citado el: 15 de Abril del 2010] Disponible en: http://www.monografias.com/trabajos27/evolucion-patrones/evolucion-patrones.shtml?monosearch
11. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns (Elements of Reusable Object-Oriented Software) [Book] [Citado el: 16 de Abril del 2010]-
12. Framework Dojo [En línea] [Citado el: 28 de Febrero del 2010] Disponible en: http://www.dojotoolkit.org/
13. Guerrero Luis A. Universidad de Chile. Metodologías De Desarrollo De Software [En línea] [Citado el: 22 de Enero del 2010] Disponible en: http://www.eici.ucm.cl/ Academicos/R Villarroel/ descargas/ ing_sw 1/RUP.pdf

14. Group, P.G.D. PostgreSQL. 2007 [En línea] [Citado el: 10 de Febrero del 2010] Disponible en: http://www.postgresql.org/.
15. hibernate.org [En línea] [Citado el: 28 de Febrero del 2010] Disponible en: Disponible en: http://www.hibernate.org/
16. Hibernate Validator JSR 303 Reference Implementation Reference Guide 4.0.1.GA [PDF] [Citado el: 28 de Abril del 2010]
17. Jacobson, G. Booch y J. Rumbaugh, El Proceso Unificado de Desarrollo de Software. S.l.: PEARSON EDUCACIÓN S.A, 2000. [Citado el: 20 de Enero del 2010]
18. Java.sun.com(1995-2010) [En línea] [Citado el: 28 de Febrero del 2010] Disponible en: http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html
19. jasperforge.org Sitio Web Oficial de JasperReport [En línea] [Citado el: 10 de Febrero del 2010] Disponible en: http://jasperforge.org/website/.
20. Metodologías de Desarrollo de Software [En línea] [Citado el: 22 de Enero del 2010] Disponible:http://www.google.com.cu/search?hl=es&q=rup+y+otras+metodologias+pesadas&btnG=Buscar&meta=&aq=f&oq=
21. Normalización de Bases de Datos[En línea] [Citado el: 20 de Abril del 2010] Disponible en: http://www.mysql-hispano.org/page.php?id=16&pag=1
22. Osorio, Eneisy y Asdrúbal. Propuesta de arquitectura para el Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito. UCI. Ciudad de la Habana, 2010. Capítulos 2 y 3, Tesis.
23. Perrone Paul J and Krishna J2EE Developer's Handbook [Citado el: 5 de Febrero del 2010] - Indianapolis, Indiana: Sam's Publishing, 2003.
24. Pecos, D. PostGreSQL. 2005 [En línea] [Citado el: 10 de Febrero del 2010] Disponible en: http://www.netpecos.org/docs/mysql_postgres/x15.html.
25. Spring-Framework-Reference [PDF] [Citado el: 28 de Febrero del 2010]
26. Schildt Herbert Java 2 Manual de referencia - Madrid: [s.n.], 2001. [Citado el: 2 de Febrero del 2010]
27. Sánchez Rico, Mario Alfredo, Spring, un framework de aplicación[En línea] [Citado el: 28 de Febrero del 2010] Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/sanchez_r_ma/capitulo3.pdf
28. Visual Paradigm 2005 [En línea] [Citado el: 28 de Enero del 2010] Disponible en: http://www.visual-paradigm.com/product/vpuml/..