Universidad de las Ciencias Informáticas
Facultad 6

Título: Sistema Informático de Gestión de Información de las Comunidades, los Consejos Comunales, Citas y Denuncias en las Coordinaciones Regionales.

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

Autores: Francisco Montada Aguilar
Osmany Cordero Domínguez

Tutores: Ing. Linet Lores Sánchez
Ing. René Vega Gorgoso

Ciudad de la Habana, junio 2010
"Año 52 de la Revolución"

DECLARACIÓN DE AUTORÍA

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

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

Francisco Montada Aguilar

_____
Firma del Autor

Osmany Cordero Domínguez

_____
Firma del Tutor

Ing. Linet Lores Sánchez

_____
Firma del Tutor

Ing. René Vega Gorgoso

_____
Firma del Tutor

DATOS DE CONTACTO

Tutores:

Ing. Linet Lores Sánchez

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba

Email: lloress@uci.cu

Ing. René Vega Gorgoso

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba

Email: rvega@uci.cu

Consultantes:

Ing. Aidacelys López Díaz

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba

Email: alopezd@uci.cu

Ing. Andres Ballester Marsal

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba

Email: aballester@uci.cu

AGRADECIMIENTOS

En este papel es imposible plasmar todo los nombres que están ligados indisolublemente a nuestras vidas y no queremos ser injustos dejando de incluir a alguien por cuestiones de tiempo y espacio, por lo tanto, agradecemos a todos los que nos han ayudado a pasar estos años, especialmente a toda la familia de cada uno de nosotros, por contribuir a nuestra formación y por toda la confianza que depositaron en nosotros, y muy en especial a nuestros padres, guía e inspiración en todo momento. No obstante, hay nombres que no pueden dejar de ser escritos y es el caso de:

En primer, lugar a la Revolución y al Comandante en Jefe Fidel Castro Ruz, creador de este magnífico proyecto de la UCI que se convirtió durante 5 años en nuestro hogar y por las incontables experiencias aquí vividas.

De forma especial a Ballester y Liusmila, por dedicarnos tanto tiempo y su asesoría constante.

A nuestros tutores por su abnegada y constante preocupación por este trabajo.

A todos los profesores que tanto empeño pusieron en formarnos ya que somos el resultado del trabajo esmerado de cada uno de ellos.

A nuestros compañeros y amigos que ayudaron de cierta forma al desarrollo de este trabajo.

A todos muchas gracias…

DEDICATORIA

Dedico este trabajo a mis padres por todo su cariño, esfuerzo tremendo, comprensión, por apoyarme y guiarme siempre por el buen camino, por ayudarme a ser todo lo que soy, por todo el amor y la educación que siempre me han ofrecido. A ambos por el ejemplo que me han dado, por ser mis ideales a seguir y mi fuente de inspiración.

A mi hermanito Migue, por ser mi fiel amigo, por ayudarme cuando lo necesito y estar siempre juntos en los momentos malos y buenos, porque sé que nunca me defraudará, porque lo adoro.

A mi hermanita tata que aunque la distancia no le permitió estar aquí hoy, sé que estaría orgullosa de mí, por su apoyo, porque la llevo en el corazón.

A mi sobrinito que lo quiero mucho y me ha inspirado mucho a seguir adelante.

A Oli el amor de mi vida, por darme tanto amor, estar siempre a mi lado, comprenderme, y apoyarme incondicionalmente, por todos los momentos que hemos vivido juntos.

A mi hijito que esta por nacer, que me ha dado fuerzas y razones para seguir adelante.

A mi Abuela y Abuelo, mis tías y tíos, primas y primos que confiaron en mí y brindaron todo su apoyo, en general a toda mi familia.

A Robe y Naty por quererme como a un hijo, darme todo su apoyo y ayudarme tanto.

A Francisco mi mejor amigo, mi hermano, mi compañero de tesis, por sus consejos, por estar siempre a mi lado, saber ser mi hermano, por los momentos que vivimos juntos, te agradezco todo lo que has hecho por mí.

A mis amigos de siempre por estar siempre en las buenas y en las malas.

Osmany Cordero Domínguez

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, el presente trabajo muestra el desarrollo del Sistema Informático para la Gestión de Información en las Coordinaciones Regionales (SIGCR) encargado de gestionar toda la información referente a las comunidades, consejos comunales, citas y denuncias en las Coordinaciones Regionales, además permite generar reporte 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: SGICR, 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. Dicha dirección 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 jefe y un equipo de profesionales y técnicos que se encargan de la atención e implementación de los programas de prevención del delito.

Dichas Coordinaciones Regionales gestionan en la actualidad toda la información relacionada a las comunidades y los consejos comunales, que es guardada en un expediente que contiene la información asociada a datos generales, demandas orientadas a sus necesidades, observaciones y documentación de apoyo a algún tema de interés para la Coordinación Regional. En el caso de las comunidades es necesario conocer los consejos comunales que pertenecen a la comunidad y organizaciones e instituciones que apoyan el funcionamiento de las mismas; de los consejos comunales, se guardan además los datos referentes a las personas de enlace, los comités formados y sus miembros. También es gestionada la información asociada a las citas realizadas por la Coordinación Regional a los ciudadanos de la comunidad que soliciten atención. En estas citas se registran datos personales de los ciudadanos, resultados de la cita, así como las remisiones y otros datos de utilidad para los especialistas. Es de interés de las Coordinaciones Regionales que se lleve el control de todas las denuncias que surgen en la comunidad y en los consejos comunales.

Debido a que los procesos de gestión de información ya mencionados se realizan de forma manual y generan un gran volumen de información, 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, demora a la hora de tomar decisiones ya que los jefes y el personal especializado no cuentan con la información o estadísticas necesarias que aseguren el éxito. 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, el fortalecimiento de la convivencia ciudadana y la paz en aras de mejorar la calidad de vida en las comunidades venezolanas, en consonancia con el nuevo modelo del país.

Por todo lo anteriormente planteado se propone como **problema científico:** ¿Cómo mejorar el proceso de gestión de información de las Comunidades, los Consejos Comunales, Citas y Denuncias, en las Coordinaciones Regionales de la Dirección General de Prevención del Delito de la República Bolivariana de Venezuela?

Se define como **objeto de estudio:** Proceso de Gestión de información en las Coordinaciones Regionales de la Dirección General de Prevención del Delito de la República Bolivariana de Venezuela, siendo el **campo de acción:** El Proceso de Gestión de la Información de las Comunidades, los

Consejos Comunales, Citas y Denuncias en las Coordinaciones Regionales de la Dirección General de Prevención del Delito de la República Bolivariana de Venezuela.

Para dar solución al problema científico declarado, esta investigación tiene como **objetivo general:** Desarrollar un Sistema Informático para la gestión de información de las Comunidades, los Consejos Comunales, Citas y Denuncias en las Coordinaciones Regionales de la Dirección General de Prevención del Delito de la República Bolivariana de Venezuela.

A partir de este objetivo general se trazaron los siguientes **objetivos específicos:**

1. Identificar las funcionalidades que debe cumplir el sistema informático.
2. Diseñar las funcionalidades especificadas para el sistema informático.
3. Implementar las funcionalidades del sistema informático.
4. Realizar pruebas al sistema informático.

Los que se cumplirán a través de las siguientes **tareas de la investigación:**

1. Definición de los requisitos funcionales y no funcionales del sistema informático.
2. Investigación de las herramientas y tecnologías basadas en código abierto.
3. Descripción de los casos de uso del sistema.
4. Confección del diagrama de clases del diseño.
5. Confección de los diagramas de interacción del diseño.
6. Diseño de la base de datos del sistema informático.
7. Realización de los diagramas de componentes del sistema informático.
8. Implementación de los componentes del sistema informático.
9. Realización de las pruebas a nivel de desarrollador del sistema informático.

Capítulo 1 Fundamentación Teórica: Se argumentan las características principales de la metodología de desarrollo utilizada y otras herramientas en las que se apoya el desarrollo del sistema.

Capítulo 2 Características del Sistema: En este capítulo se realiza el diagrama del modelo conceptual propuesto y se definen los requerimientos a implementar. Además se realiza el diagrama de casos de uso del sistema y la descripción de los mismos.

Capítulo 3 Diseño del Sistema: En este capítulo se desarrolla el diagrama de clases del diseño así como la realización de los diagramas de secuencia y el diseño de la base de datos. Además se hace

una breve descripción del estilo arquitectónico y de los patrones de diseños empleados para el desarrollo del sistema.

Capítulo 4 Implementación del Sistema: En este capítulo se muestran los diagramas de componentes que se definieron durante la implementación del sistema, se describen implementaciones relevantes y algunas pantallas del sistema informático. Además se detallan las pruebas realizadas sobre la aplicación para comprobar sus funcionalidades.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA.

1.1 Introducción.

El presente capítulo abordará los principales conceptos que permitan entender qué es un sistema de gestión de software. También se argumentarán las características principales de la metodología de desarrollo que se utilizará y otras herramientas en las que se apoyará el desarrollo del sistema.

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 ella, 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 utilizarla 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 y 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: 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 intenso de información. [2]

A partir de los elementos anteriores se procede a identificar una metodología que guíe el desarrollo del sistema.

1.3 Metodología de Desarrollo de Software.

Una metodología de desarrollo de software es un conjunto de procedimientos que permiten producir y mantener un software, definiendo una serie de pasos a seguir para obtener un software de calidad. Las

metodologías se desarrollan con el objetivo de dar solución a los problemas existentes en la producción de software, que cada vez son más complejos. Estas abarcan procedimientos, técnicas, documentación y herramientas que se utilizan en la creación de un software [3]. Sus procesos se descomponen a nivel de tareas o actividades elementales, donde cada tarea está identificada por un procedimiento que define la forma de llevarla a cabo.

1.3.1 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 sistemas de 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 de software en construcción está formado por componentes de software interconectados a través de interfaces bien definidas. Utiliza el Lenguaje Unificado de Modelado (Unified Modeling Language, UML) para preparar todos los esquemas de un sistema de software. UML es un lenguaje que permite la modelación de sistemas con tecnología orientada a objetos

Fig. 1.1 Un proceso de desarrollo de software

Los verdaderos aspectos definitorios del RUP que lo convierten en único, se resumen en tres frases claves: dirigido por caso de uso, centrado en la arquitectura, iterativo e incremental. [3]

- **Dirigido por casos de uso**: Los casos de uso reflejan lo que los usuarios futuros necesitan, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo.
- **Centrado en la arquitectura:** La arquitectura es una vista del diseño completo con las características más importantes. Estas no sólo incluyen las necesidades del usuario, sino otros

aspectos técnicos como el hardware, el sistema operativo, el sistema de gestión de base de datos y los protocolos de red con los que debe coexistir el sistema. En otras palabras, la arquitectura representa la forma del sistema, la cual va madurando en su iteración con los casos de usos hasta llegar a un equilibrio entre funcionalidad y características técnicas.

- **Iterativo e incremental:** La alta complejidad de los sistemas actuales hace que sea factible dividir el proceso de desarrollo en varios mini-proyectos, se les denomina iteración y pueden o no representar un incremento en el grado de terminación del producto completo. Una iteración involucra actividades de todos los flujos de trabajo, unos más que otros. Las iteraciones hacen referencia a pasos en los flujos de trabajo y los incrementos, al crecimiento del producto. Estas permiten mantener la motivación del equipo pues pueden ver avances claros a corto plazo y que el desarrollo pueda adaptarse a los cambios en los requisitos.

La metodología RUP divide en 4 fases el desarrollo del software:

- **Inicio:** El objetivo es determinar la visión del proyecto.
- **Elaboración:** El objetivo es determinar la arquitectura óptima.
- **Construcción:** El objetivo es obtener la capacidad operacional inicial.
- **Transición:** El objetivo es obtener una versión del proyecto.

Para desarrollar la propuesta que presenta este trabajo se ha decidido utilizar como metodología RUP ya captura varias de las mejores prácticas en el desarrollo moderno del software, en una forma que es aplicable para un amplio rango de proyectos y organizaciones. Es una guía de cómo utilizar UML y le proporciona a cada miembro de un equipo, fácil acceso a una base de conocimientos con guías, plantillas y herramientas para todas las actividades críticas de desarrollo. Propone la generación de los artefactos necesarios para una buena documentación y el entendimiento entre los miembros del equipo y los usuarios finales. Además distribuye las actividades según la fase de desarrollo y establece roles para la ejecución de las mismas.

1.4 Roles y Artefactos.

Un rol define el comportamiento y las responsabilidades de un individuo, es una definición abstracta de un conjunto de actividades realizadas y de artefactos obtenidos. Los roles son realizados típicamente por un individuo, o conjunto de individuos, trabajando juntos en equipo. Un miembro del equipo de

proyecto cumple normalmente muchos roles. Los roles no son individuos; en lugar de ello, describen cómo los individuos se comportan en el negocio y qué responsabilidades tienen estos individuos. [5]

La metodología RUP define un conjunto de roles, pero sólo se desarrollarán los roles de Analista, Diseñador, Diseñador de base de datos, Arquitecto y Desarrollador pertenecientes a los grupos de analistas y desarrolladores definidos por dicha metodología.

Analista: Agrupa los roles que están involucrados fundamentalmente en la captura y gestión de los requisitos, que puede estar representado por una o varias personas entre los que se encuentran: Analista del proceso de negocio, Analista del sistema y Especificador de casos de uso. Los artefactos a realizar desempeñando el rol de **Analista** en el desarrollo de esta investigación se describen a continuación:

- **Modelo de dominio:** Captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan las "cosas" que existen o los eventos que suceden en el entorno en el que trabaja el sistema. [6]
- **Modelo de caso de uso:** Es un modelo del sistema que contiene actores, casos de uso y sus relaciones.
- **Especificaciones de requerimientos del software:** Es la captura de los requerimientos del software para el sistema o una parte del mismo.
- **Glosario:** Es un documento que define los términos comunes que se utilizan para describir el proyecto.

Diseñador: Es el responsable del diseño de una parte del sistema que incluye: las restricciones de los requisitos, la arquitectura y el proceso de desarrollo del proyecto. A continuación se describen sólo los dos artefactos a realizar desempeñado el rol de **Diseñador** en el desarrollo del sistema:

- **Diagrama clases del diseño:** Es una descripción de un grupo de objetos que comparten las mismas responsabilidades, relaciones, operaciones, atributos y semánticas. A diferencia del Modelo del dominio, un Diagrama de clases de diseño muestra definiciones de entidades de software más que conceptos del mundo real.
- **Realizaciones de casos de usos:** Son reseñas textuales del caso de uso. Normalmente, tienen el formato de una nota o un documento relacionado de alguna manera con el caso de uso, y explica los procesos o actividades que tienen lugar en el caso de uso.

Diseñador de base de datos: Es el responsable de la definición de los detalles del diseño de la base de datos. Dicho rol tiene que asegurarse de que los datos persistentes son almacenados consistente y eficientemente, definir el comportamiento que debe ser implementado en la base de datos. El artefacto a realizar desempeñando el rol de **Diseñador de base de datos** en el desarrollo de esta investigación se describe a continuación:

- **Modelo de datos:** Describe la representación lógica y física de los datos persistentes utilizados por la aplicación.

Arquitecto de software: Es el responsable de la arquitectura de software, que incluye las decisiones técnicas claves que restringen el diseño global y la implementación para el proyecto. A continuación se describen sólo los artefactos que desarrollará el **Arquitecto de software** en el desarrollo del sistema:

- **Modelo de diseño:** Es un modelo de objetos que describe la realización de casos de uso, y sirve como una abstracción del modelo de implementación y el código fuente.
- **Descripción de la arquitectura (vista de implementación):** Describe la descomposición del software en capas y subsistemas de implementación.

Desarrollador: Es responsable de desarrollar y de probar componentes de acuerdo con los estándares adoptados por el proyecto, para la integración en subsistemas más grandes. Cuando los componentes de prueba, tales como drivers o partes se deben crear para apoyar la prueba, el desarrollador es también responsable de desarrollar y de probar los componentes de prueba y los subsistemas correspondientes. A continuación se describen sólo los artefactos que se generarán por el **Desarrollador** en esta investigación:

- **Elementos de implementación:** Son las partes físicas que componen la implementación, incluyendo archivos y directorios. Además los archivos de código del software (binario o ejecutable), archivos de datos y documentación.
- **Artefactos de instalación:** Se refieren al software y a las instrucciones documentadas requeridas para instalar el producto.

Para el modelado de los artefactos descritos anteriormente se procede a seleccionar una herramienta que permita su diseño.

1.5 Herramienta para el Modelado.

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, en tareas como: el proceso de realizar un diseño del proyecto, cálculo de costes, implementación automática de parte del código con el diseño dado, compilación automática, documentación o detección de errores, entre otras. [3]

1.5.1 Visual Paradigm.

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 realizarse la ingeniería inversa para aplicaciones realizadas 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 [7].

Entre sus ventajas más relevantes se encuentran: la realización de la 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.

Teniendo en cuenta las características antes mencionadas del Visual Paradigm, el colectivo de autores decide utilizarla como herramienta de modelado, especialmente por su buena integración con los IDE para el desarrollo de Java, ser multiplataforma y la ventaja de que presenta una interfaz de usuario de fácil uso. Además permite realizar los diagramas y artefactos que se generan durante el desarrollo del software, y realizar un control de las versiones durante todo el ciclo de trabajo.

1.6 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 pueda comunicarse con los dispositivos de hardware y software existentes. [8]

1.6.1 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 características anteriormente mencionadas lo hace único y 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. [9]

Teniendo en cuenta las características antes mencionadas de Java, el colectivo de autores decide utilizarlo como lenguaje de programación, especialmente por ser multiplataforma, sumamente flexible, permite la creación de programas modulares y de códigos reutilizables. Además en la actualidad es muy utilizado, ya que permite el desarrollo de programas seguros y de alto rendimiento.

1.7 Entorno de Desarrollo Integrado (IDE).

Un Entorno Integrado de Desarrollo en inglés 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. [10]

1.7.1 Eclipse.

Eclipse surge con la idea de crear una plataforma de desarrollo común para confeccionar aplicaciones, basadas en el lenguaje de programación Java. Es un entorno de desarrollo integrado (IDE) porque provee herramientas para administrar áreas de trabajo (en inglés workspaces), 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 ya que su diseño le permite ser extendido fácilmente por terceras partes. Emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad, a diferencia de otros entornos donde las funcionalidades están todas incluidas, las necesite el usuario o no. [11]

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

El IDE que fue seleccionado por el colectivo de autores para desarrollar el software mediante el lenguaje de programación Java, fue el Eclipse, en su versión 3.5.0, por ser de código abierto y su factibilidad de uso. Además de considerarse más estable y rápido; 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).

1.8 Plataforma de Desarrollo.

La plataforma no es más que el 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 epígrafes anteriores se escogieron Java como lenguaje de programación y Eclipse como IDE de desarrollo. 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. [12]

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).

1.8.1 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, estas 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. [12]

Debido a que las Coordinaciones Regionales están ubicadas en diferentes estados, se hace necesario un sistema orientado a la web [13]; por este motivo el colectivo de autores decide utilizar la especificación **Java EE,** la cual proporciona muchas ventajas para entornos web y es sin dudas una gran opción para el desarrollo de aplicaciones empresariales. Sobre esta plataforma se han desarrollado múltiples frameworks que facilitan la programación de aplicaciones, ya que encapsulan operaciones complejas en instrucciones sencillas.

1.9 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, con el propósito de ayudar a desarrollar y unir los diferentes componentes de un proyecto.

En el desarrollo de la arquitectura base del sistema se utilizó 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.

1.9.1 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. [14]

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, soluciones muy bien documentadas y fáciles de usar.
La simplificación del desarrollo de aplicaciones y de sus respectivas pruebas es una de las claves de su éxito. [15]

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. [16]

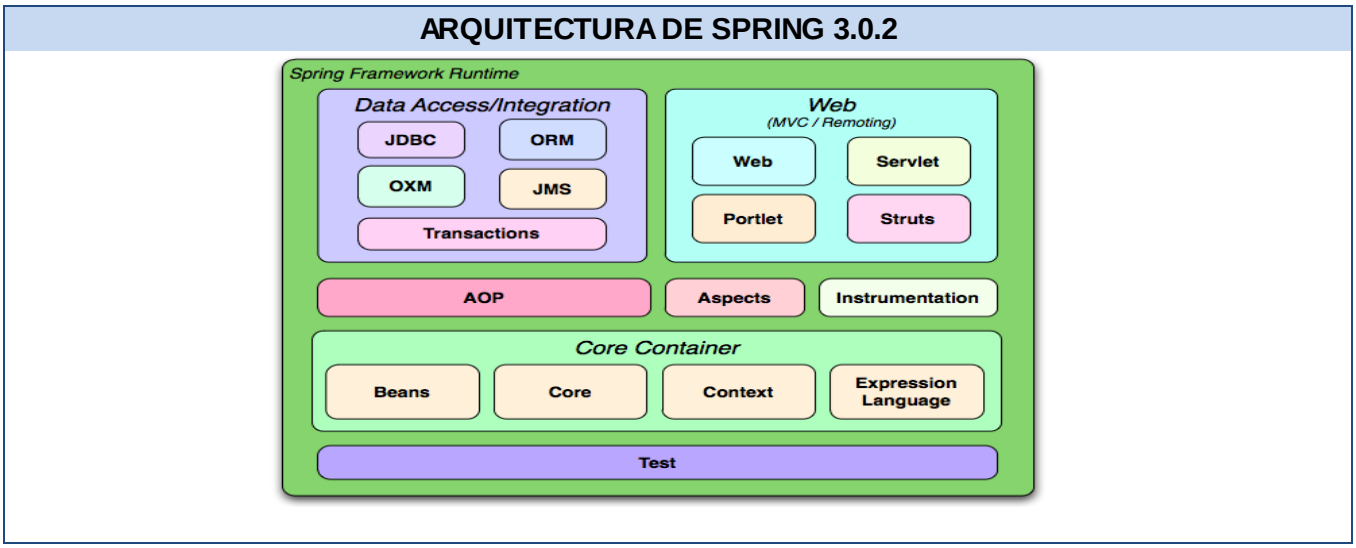


Fig. 1.1 Arquitectura de Spring 3.0.2.

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 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.

A continuación, una representación gráfica del funcionamiento del MVC en Spring (Fig.3.2)

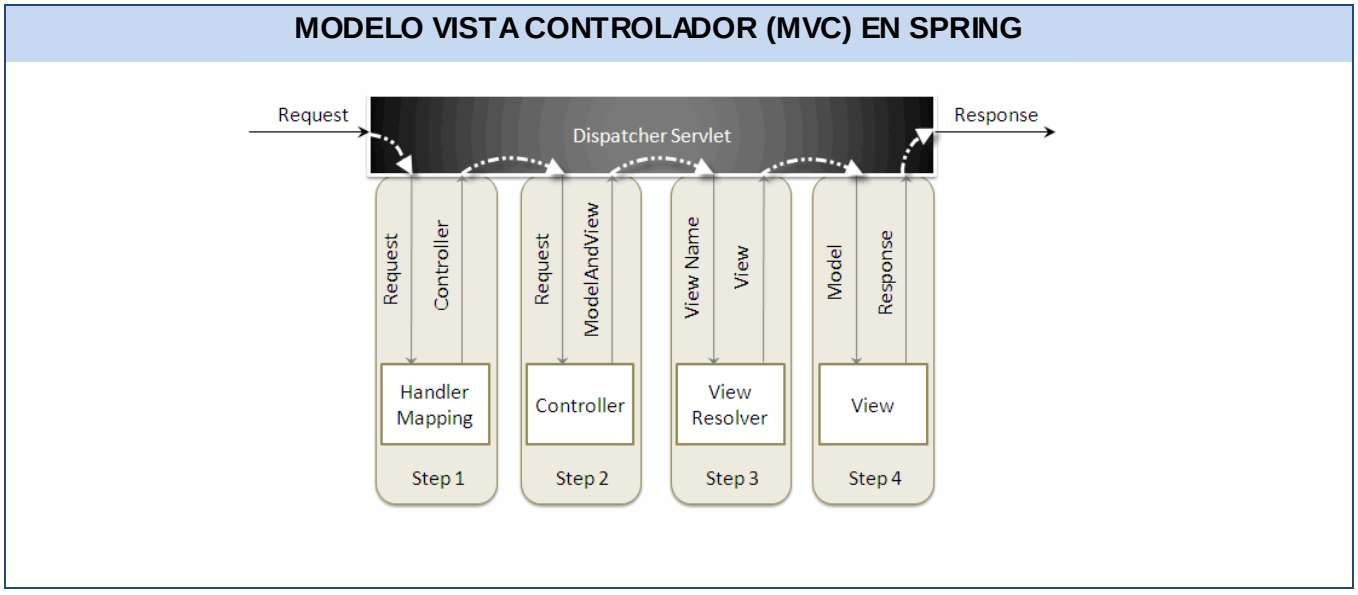


Fig. 1.2 MVC en Spring

1.9.2 Framework javascript.

Dojo Toolkit 1.4.1.

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, los mismos 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. [17]

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.

1.9.3 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. [18]

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.

1.10 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.

1.10.1 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. [19]

Desventajas:

- Consume gran cantidad de 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. [20]

Por todo lo anteriormente expuesto se decide utilizar el Sistema Gestor de Bases de Datos (SGBD) PostgreSQL ya que es de código abierto bajo licencia BSD, es decir, sus potencialidades están en

constante perfeccionamiento, permitiendo su uso y distribución sin costo; además continuamente se elimina y mejoran cualquier hueco de seguridad que pueda aparecer, debido a que sus usuarios pueden acceder a su código y modificarlo a sus necesidades. En el caso que se esté trabajando con sistemas que manejen información de mucha relevancia y sensibles, la integridad referencial de los datos es muy buena y se pueden crear funciones complejas para validar dichos datos. Es también utilizado porque es un SGBD potente y multiplataforma. En el caso del presente trabajo se determinó usar la herramienta EMS SQL Manager for PostgreSQL (SQL Manager 2007 for PostgreSQL 4.3.0.1).

1.10.2 EMS SQL Manager for PostgreSQL.

Es una aplicación de alto desempeño para la administración y desarrollo de PostgreSQL Database Server. El programa trabaja con cualquier versión de PostgreSQL hasta la 8.4 y soporta todas las últimas características de PostgreSQL, incluyendo espacios de tablas, nombres de argumentos en funciones. El programa ofrece muchas herramientas poderosas para usuarios experimentados, como un diseñador visual de bases de datos, constructor visual de consultas y un editor BLOB (**B**inary **L**arge **OB**ject). Su interfaz gráfica es sumamente atractiva e incluye un modo guiado de trabajo.

Es muy importante que los sistemas de gestión de información puedan generar reportes a partir de los datos persistidos en la base de datos, ya que de esta manera los usuarios pueden hacer un análisis más sofisticado y práctico de la información incluso en el momento de tomar decisiones. La generación de reportes sin dudas ayuda a disponer de la información de una manera más rápida y cuenta con medios más flexibles para distribuirla.

1.11 Tecnología para crear Reportes.

¿Qué es un reporte?

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

1.11.1 JasperReports.

Es una librería de clases 100% Java de código abierto, diseñada para agregar capacidades de reporte a las aplicaciones Java. [21] 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 por el colectivo de autores 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.

1.12 Conclusiones.

Luego de haber realizado un estudio sobre las tecnologías y herramientas que cumplieran con las tendencias actuales de la programación web y sobretodo orientadas al código abierto, se definió como plataforma la JEE, haciendo uso del lenguaje de programación Java. Además fueron seleccionados los framework Spring, Hibernate, Dojo y como IDE de desarrollo Eclipse. También se definió RUP como la metodología a utilizar, Visual Paradigm como herramienta CASE para el modelado, como gestor de base de datos PostgreSQL y por último Jasper Report para la generación de reportes; todas estas tecnologías con el fin de crear un sistema que cumpla con la calidad requerida.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

2.1 Introducción.

En este capítulo se describe la propuesta de solución del sistema según el problema descrito. Para ello se muestran los procesos del negocio mediante el modelo del dominio. Además se brinda una concepción general del sistema propuesto y se particulariza en las funcionalidades y características que tendrá el sistema a desarrollar. Se presenta el diagrama de casos de uso del sistema con las correspondientes especificaciones y se describen los patrones de caso de usos empleados.

2.2 Breve descripción del negocio.

En la actualidad las Coordinaciones Regionales gestionan toda la información relacionada a las comunidades y los consejos comunales, que es guardada en un expediente que contiene la información asociada a los datos generales, demandas orientadas a sus necesidades, observaciones y documentación de apoyo a algún tema de interés para la Coordinación Regional. En el caso de las comunidades, el expediente también abarca la información de todos los consejos comunales que pertenecen a la comunidad y las organizaciones e instituciones que apoyan el funcionamiento de las mismas. En el caso de los consejos comunales el expediente contiene la información referente a las personas de enlace y miembros.

En las Coordinaciones Regionales también es gestionada la información asociada a las citas realizadas por la Coordinación Regional a los ciudadanos que soliciten atención. En estas citas se registran los datos personales de los ciudadanos, además cuando el especialista atiende al ciudadano, el mismo conforma el expediente del ciudadano que va a contener los datos de todos los casos por los que ha sido atendido con anterioridad. Cada caso contiene información sobre las consultas que ha recibido y las remisiones que se le han hecho. Es de interés de las Coordinaciones Regionales llevar el control de todas las denuncias que surgen en la comunidad y en los consejos comunales.

El sistema a desarrollar a través del presente trabajo constituirá un software para la gestión de información de los consejos comunales, comunidades, citas y las denuncias en las Coordinaciones Regionales.

Modelado del negocio.

RUP define en su primera fase de desarrollo la realización del modelo de negocio, con el objetivo de comprender el entorno del cliente y detectar las mejoras potenciales en los procesos de la

organización. Cuando no es posible identificar claramente los procesos del negocio, RUP propone realizar un modelo de dominio, que es un subconjunto del modelo de negocio.

2.3.1 Modelo de Dominio.

El estudio del negocio realizado evidencia que los procesos que se llevan a cabo en las Coordinaciones Regionales, no están bien definidos, por ello se procede a la realización del modelo de dominio.

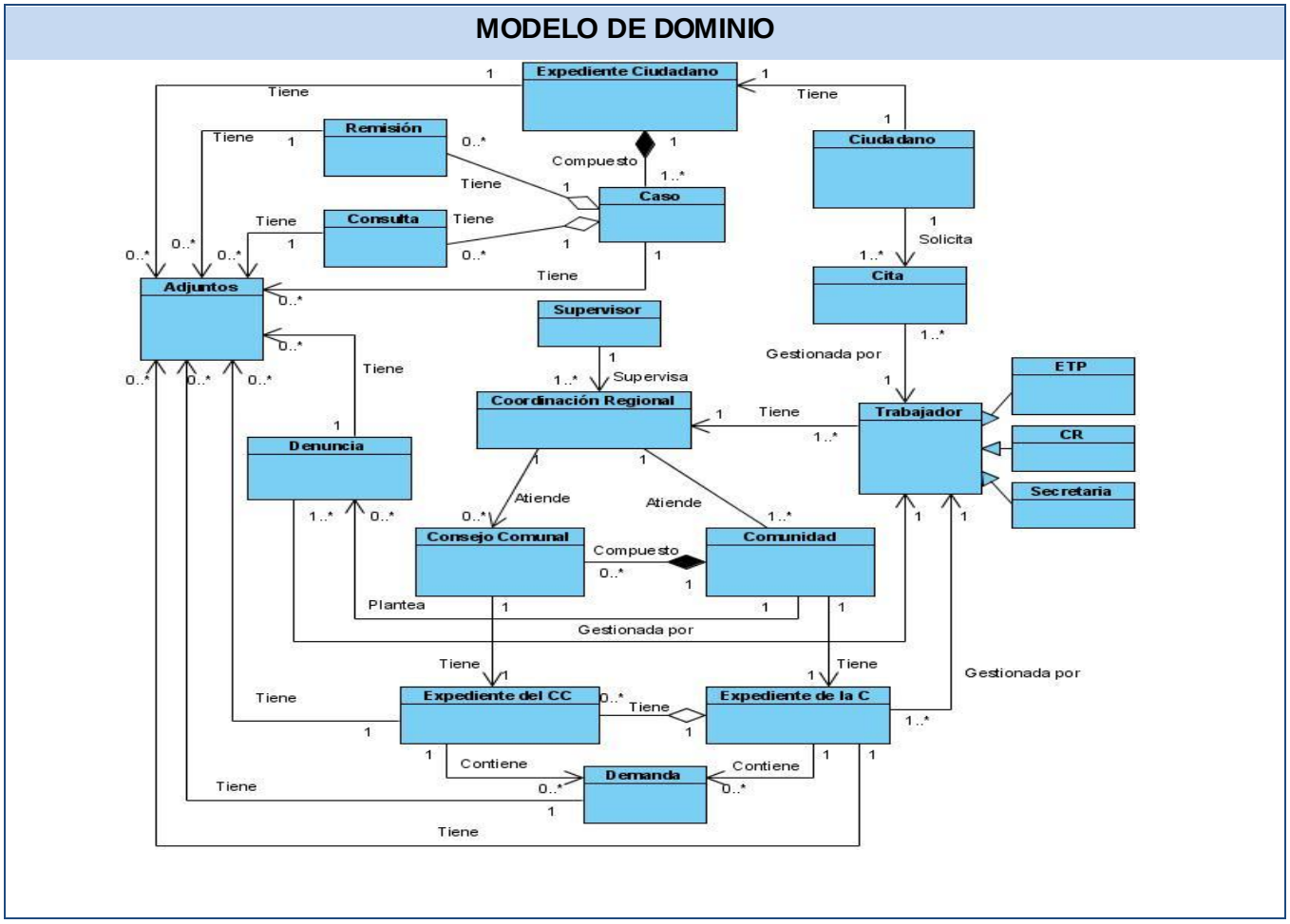


Fig. 2.1 Modelo de Dominio**.**

2.3.2 Descripciones de los conceptos del Modelo de Dominio.

Clase Coordinación Regional.

Es una oficina adscrita a la Dirección General de Prevención del Delito, que se encarga de gestionar toda la información referente a las comunidades, los consejos comunales, las denuncias y las citas.

Clase Comunidad.

Son entidades político-administrativas descentralizadas donde se aglutinan las células de autogobierno locales llamadas Consejos Comunales.

Clase Expediente de la C.

Contiene datos referentes a la comunidad como los datos generales de la misma, las organizaciones e instituciones que apoyan su funcionamiento, las observaciones hechas por la Coordinación Regional en la comunidad, las demandas que tiene la comunidad para satisfacer sus necesidades, los consejos comunales que conforman dicha comunidad.

Clase Consejo Comunal.

Son instancias de participación, articulación e integración entre las diversas organizaciones comunitarias, grupos sociales, ciudadanos y ciudadanas, que permiten al pueblo ejercer directamente la gestión de las políticas públicas y proyectos orientados a responder las necesidades y aspiraciones de las comunidades en la construcción de una sociedad de equidad y justicia social.

Clase Expediente del CC.

Contiene datos referentes al consejo comunal como los datos generales del consejo comunal, las personas enlaces, los miembros que representa a cada comité, las observaciones hechas por la Coordinación Regional en el consejo comunal, las demandas que tiene el consejo para satisfacer sus necesidades.

Clase Demanda.

Son solicitudes que realizan los consejos comunales y la comunidad a la Coordinación Regional, con el objetivo de satisfacer sus necesidades.

Clase Denuncia.

Son declaraciones que realizan los ciudadanos pertenecientes a los consejos comunales y las comunidades en la Coordinación Regional, acerca de situaciones o problemas que están afectando a la comunidad o al consejo comunal o a ambos.

Clase Adjunto.

Es un documento, una imagen, un video, un sonido, que contiene información importante que sirve de constancia o para conocimiento.

Clase Supervisor.

Es un trabajador perteneciente a la Dirección General de Prevención del Delito, encargado de supervisar el trabajo que se realiza en una o más Coordinaciones Regionales.

Clase Trabajador.

Es una generalización que contiene los roles de ETP, CR, Secretaria.

Clase ETP.

Es un equipo que está conformado por técnicos y profesionales vinculados directamente a la Coordinación Regional que realizan una serie de procesos de interés, son los encargados de llevar a cabo el trabajo especializado (Psicológicos, Sociales y de Orientación) en la Coordinación Regional.

Clase CR.

El Coordinador Regional es el máximo responsable de la Coordinación Regional. Además es el encargado de realizar el manejo de información referente a los consejos comunales, comunidades y denuncias, al igual que el ETP.

Clase Secretaria.

Es la secretaria del Coordinador Regional por lo que puede realizar algunas de las funciones que realiza el mismo.

Clase Ciudadano.

Es la persona que reside en el radio de acción de un consejo comunal o una comunidad de la Coordinación Regional.

Clase Cita.

Es el turno que le es dado a un ciudadano que viene por algún problema a la Coordinación Regional con el objetivo de ser atendido por el personal especializado.

Clase Expediente Ciudadano.

Es un documento que se le realiza al ciudadano una vez que es atendido por un especialista, el mismo contiene los datos generales del ciudadano, los casos por los cuales ha venido, la información sobre las consultas a que ha sido sometido y las remisiones.

Clase Caso

Es una clasificación de motivos que aquejan a los ciudadanos definidos por los especialistas de la Coordinación Regional.

Clase Consulta.

Es el tiempo que emplea un especialista en atender a algún ciudadano que presenta un caso en específico.

Clase Remisión.

Es cuando el especialista que está atendiendo al ciudadano por un caso específico lo remite a una institución especializada o a otro especialista.

2.3 Modelación del sistema.

Requisitos.

El objetivo de la captura de requisitos es guiar el desarrollo de software hacia el sistema correcto, definiendo objetivos generales concretos de manera tal que tanto el negocio como los actores se beneficien.

Se define requerimiento como condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo. [5]

2.4.1 Requisitos Funcionales.

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir. [5]

El sistema a desarrollar a través de la realización del presente trabajo debe cumplir los siguientes requisitos, los mismos están agrupados en casos de usos:

CUS Gestionar Listado Denuncias.

RF1. Visualizar listado de denuncias.

RF2. Generar reportes de denuncias.

RF3. Exportar los resultados del reporte de denuncia a formato PDF, Excel.

RF4. Buscar denuncias.

CUS Visualizar Denuncia.

RF5. Visualizar denuncia.

RF6. Imprimir denuncia.

CUS Gestionar Denuncia.

RF7. Registrar denuncia.

RF8. Actualizar denuncia.

RF9. Eliminar denuncia.

CUS Gestionar Adjunto.

RF10. Adjuntar archivo.
RF11. Visualizar archivo.
RF12. Eliminar archivo.
RF13. Visualizar listado de adjuntos.

CUS Gestionar Listado de Demandas de CR.

RF14. Visualizar listado de demandas de la CR.
RF15. Generar reportes de demandas de la CR.
RF16. Exportar los resultados del reporte de demandas a formato PDF, Excel.
RF17. Buscar demanda.

CUS Gestionar Demandas.

RF18. Visualizar listado de demandas de la C.
RF19. Visualizar listado de demandas del CC.
RF20. Registrar demanda.
RF21. Actualizar demanda.
RF22. Eliminar demanda.

CUS Gestionar Observaciones.

RF23. Visualizar listado de observaciones de la C.
RF24. Visualizar listado de observaciones del CC.
RF25. Registrar observación.
RF26. Actualizar observación.
RF27. Eliminar observación.

CUS Gestionar Listado de los CC.

RF28. Visualizar listado de Consejos Comunales.
RF29. Generar reportes de los Consejos Comunales.
RF30. Exportar los resultados del reporte de los consejos comunales a formato PDF, Excel.
RF31. Buscar Consejo Comunal.

CUS Visualizar Expediente del CC.

RF13. Visualizar listado de adjuntos.
RF19. Visualizar listado de demandas del CC.
RF24. Visualizar listado de observaciones del CC.
RF32. Visualizar expediente del Consejo Comunal.
RF33. Imprimir expediente del Consejo Comunal.
RF37. Visualizar listado de personas de enlace.
RF41. Visualizar listado de miembros del CC.

CUS Gestionar Consejo Comunal.

RF34. Visualizar listado de los CC de la Comunidad.
RF35. Registrar Consejo Comunal.
RF36. Actualizar Consejo Comunal.

CUS Gestionar Persona Enlace.

RF37. Visualizar listado de personas de enlace.
RF38. Registrar persona de enlace.
RF39. Actualizar persona de enlace.
RF40. Eliminar persona de enlace.

CUS Gestionar Miembros del CC.

RF41. Visualizar listado de miembros del CC.
RF42. Registrar miembro del CC.

RF43. Actualizar miembro del CC.

RF44. Eliminar miembro del CC.

CUS Gestionar Listado de Comunidades.

RF45. Visualizar listado de Comunidades.

RF46. Generar reportes de Comunidades.

RF47. Exportar los resultados del reporte de comunidades a formato PDF, Excel.

RF48. Buscar Comunidad.

CUS Visualizar Expediente de la C.

RF13. Visualizar listado de adjuntos.

RF18. Visualizar listado de demandas de la C.

RF23. Visualizar listado de observaciones de la C.

RF34. Visualizar listado de los CC de la Comunidad.

RF49. Visualizar expediente de la Comunidad.

RF50. Imprimir expediente de la Comunidad.

RF53. Visualizar listado de instituciones u organizaciones.

CUS Gestionar Comunidades.

RF51. Registrar Comunidad.

RF52. Actualizar Comunidad.

CUS Visualizar Cita.

RF61. Visualizar Cita.

RF62. Imprimir Cita.

CUS Gestionar Instituciones.

RF53. Visualizar listado de instituciones u organizaciones.

RF54. Registrar instituciones u organizaciones.

RF55. Actualizar instituciones u organizaciones.

RF56. Eliminar instituciones u organizaciones.

CUS Gestionar Listado Citas.

RF57. Visualizar listado de Citas.

RF58. Generar reportes de Citas.

RF59. Exportar los resultados del reporte de citas del especialista a formato PDF, Excel.

RF60. Buscar Citas.

CUS Gestionar Citas.

RF63. Registrar Cita.

RF63, 1 Buscar Ciudadano.

RF64. Actualizar Cita.

RF65. Eliminar Cita.

CUS Gestionar Listado Citas del Especialista.

RF66. Visualizar listado de Citas.

RF67. Generar reportes de citas del Especialista.

RF68. Exportar los resultados del reporte de citas del especialista a formato PDF, Excel.

RF69. Buscar citas del Especialista.

CUS Gestionar Expediente Ciudadano.

RF70. Registrar expediente del Ciudadano.

RF71. Actualizar expediente del Ciudadano.

CUS Gestionar Listado de Ciudadanos Atendidos.

RF72. Visualizar listado de Ciudadanos atendidos por Especialista.

RF73. Generar reportes de Ciudadanos atendidos por Especialista.

RF74. Exportar los resultados del reporte de ciudadanos atendidos por el especialista a formato PDF, Excel.

RF75. Buscar Ciudadanos atendidos por Especialista.

CUS Gestionar Casos.

RF76. Visualizar listado de casos.

RF77. Registrar caso.

RF78. Actualizar caso.

RF79. Eliminar caso.

RF80. Visualizar caso.

RF81. Visualizar listado de consultas.

RF89. Visualizar listado de remisiones de un caso.

CUS Gestionar Consulta.

RF81. Visualizar listado de consultas.

RF82. Registrar consulta.

RF83. Actualizar consulta.

RF84. Eliminar consulta.

CUS Gestionar Listado de Remisiones Especialista.

RF85. Visualizar listado de remisiones hechas por el Especialista.

RF86. Generar reportes de las remisiones hechas por el Especialista.

RF87. Exportar los resultados del reporte de las remisiones hechas por el especialista a formato PDF, Excel.

RF88. Buscar remisiones hechas por el Especialista.

CUS Gestionar Remisión.

RF89. Visualizar listado de remisiones de un caso.

RF90. Registrar remisión.

RF91. Actualizar remisión.

RF92. Imprimir remisión.

RF93. Visualizar remisión.

RF94. Eliminar remisión.

CUS Gestionar Usuarios.

RF95. Registrar Usuario.

RF96. Actualizar Usuario.

RF97. Eliminar Usuario.

RF98. Visualizar listado de Usuarios.

CUS Autenticar.

RF99. Autenticar Usuario.

RF100. Cambiar Contraseña

CUS Gestionar Rol.

RF101. Registrar Rol.

RF102. Actualizar Rol.

RF103. Eliminar Rol.

RF104. Visualizar listado de Roles.

CUS Gestionar Estado.

RF105. Registrar Estado.

RF106. Actualizar Estado.

RF107. Eliminar Estado.

RF108. Visualizar listado de Estados.

CUS Gestionar Municipio.

RF109. Registrar Municipio.

RF110. Actualizar Municipio.

RF111. Eliminar Municipio.

RF112. Visualizar listado de Municipios.

CUS Gestionar Parroquia.

RF113. Registrar Parroquia.

RF114. Actualizar Parroquia.

RF115. Eliminar Parroquia.

RF116. Visualizar listado de Parroquias.

CUS Gestionar Sector.

RF117. Registrar Sector.

RF118. Actualizar Sector.

RF119. Eliminar Sector.

RF120. Visualizar listado de Sectores.

CUS Gestionar Área.

RF121. Registrar Área.

RF122. Actualizar Área.

RF123. Eliminar Área.

RF124. Visualizar listado de Áreas.

CUS Gestionar Servicio.

RF125. Registrar Servicio.

RF126. Actualizar Servicio.

RF127. Eliminar Servicio.

RF128. Visualizar listado de Servicios.

CUS Gestionar Comité.

RF129. Registrar Comité.

RF130. Actualizar Comité.

RF131. Eliminar Comité.

RF132. Visualizar listado de Comités.

CUS Gestionar Tipo.

RF133. Registrar Tipo.

RF134. Actualizar Tipo.

RF135. Eliminar Tipo.

RF136. Visualizar listado de Tipos.

CUS Gestionar Fase.

RF137. Registrar Fase.

RF138. Actualizar Fase.

RF139. Eliminar Fase.

RF140. Visualizar listado de Fases.

2.4.2 Requisitos No Funcionales.

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. [5]

Los requisitos no funcionales del sistema se detallan a continuación:

RNF1 - Requerimientos de Apariencia o interfaz externa.

Se deben utilizar imágenes y colores identificados con el negocio.

RNF2 - Requerimientos de Usabilidad.

Los usuarios deben tener conocimientos básicos de informática por lo que el sistema debe permitir a los usuarios un acceso fácil y rápido, contando con un menú que satisfaga las necesidades de los mismos.

RNF3 - Requerimientos de soporte.

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

- Reinstalación de la aplicación ante fallos.
- Instalación de nuevas versiones o actualizaciones de la aplicación.
- Solución de fallos de configuración de la tecnología asociada a la aplicación.

RNF4 - Requerimientos de Seguridad.

- **Confidencialidad.**
  La autenticación será la primera acción del usuario en el sistema y consistirá en proveer un nombre de usuario único y una contraseña que debe ser de conocimiento exclusivo de la persona que se autentica.
- **Integridad.**
  Se debe garantizar que la información sensible sólo pueda ser vista por los usuarios con el nivel de acceso adecuado y que las funcionalidades del sistema se muestren de acuerdo al usuario que esté activo.
- **Disponibilidad.**
  El sistema debe estar disponible para su utilización las 24 horas del día, durante los siete días de la semana, con el menor tiempo posible de recuperación ante fallos.

RNF5 - Requerimientos Legales.

La licencia para la comercialización del producto se emitirá por la Dirección de Servicios Legales de la Infraestructura Productiva, para la misma se tendrán en cuenta los componentes que se utilizaron para el desarrollo de la aplicación.

RNF6 - Requerimientos de software

Cliente

- Debe tener instalado un navegador web. Se recomienda Mozilla Firefox 3.5 o superior, o Google Chrome 4.0.
- Debe tener instalado Sistema Operativo: Windows XP o superior, o GNU Linux.

Servidor

- Se debe instalar TOMCAT como servidor web y PostgreSQL como gestor de base de datos.
- Debe estar instalado el Java Runtime Environment (JRE) versión 1.6 o superior.
- Debe estar instalado Sistema Operativo: Windows XP o superior, o GNU Linux.

RNF7 - Requerimientos de Hardware.

Para el funcionamiento del componente se requiere de máquinas con los siguientes requisitos:

- Para las PC clientes: procesador Pentium o superior, 256 Mb de RAM y al menos 10Gb de capacidad del disco duro.
- El servidor de aplicaciones debe tener las siguientes características: capacidad de disco duro superior a 80 GB, microprocesador Pentium IV superior a 2.0 GHz y como mínimo 1.0 GB de RAM.
- El servidor de base de datos debe tener las siguientes características: capacidad de disco duro superior a 180 GB, microprocesador Pentium IV superior a 2.0 GHz y como mínimo 1.0 GB de RAM.

RNF8- Requerimientos de restricciones en el diseño y la implementación.

- Los componentes del sistema deben desarrollarse siguiendo el principio de alta cohesión y bajo acoplamiento.
- La lógica de presentación constituirá una capa independiente de la lógica de negocio, centrando su función en la interfaz de usuario y validaciones de los datos de entrada.

2.4 Modelo de Casos de Uso del Sistema.

El Diagrama General de Casos de Uso describe la división del sistema en paquetes, así como la interacción de los actores con los casos de uso.

A continuación se presenta el Diagrama de Casos de Uso del Sistema de Gestión de Información para las Coordinaciones Regionales (SGICR) de Prevención del Delito.

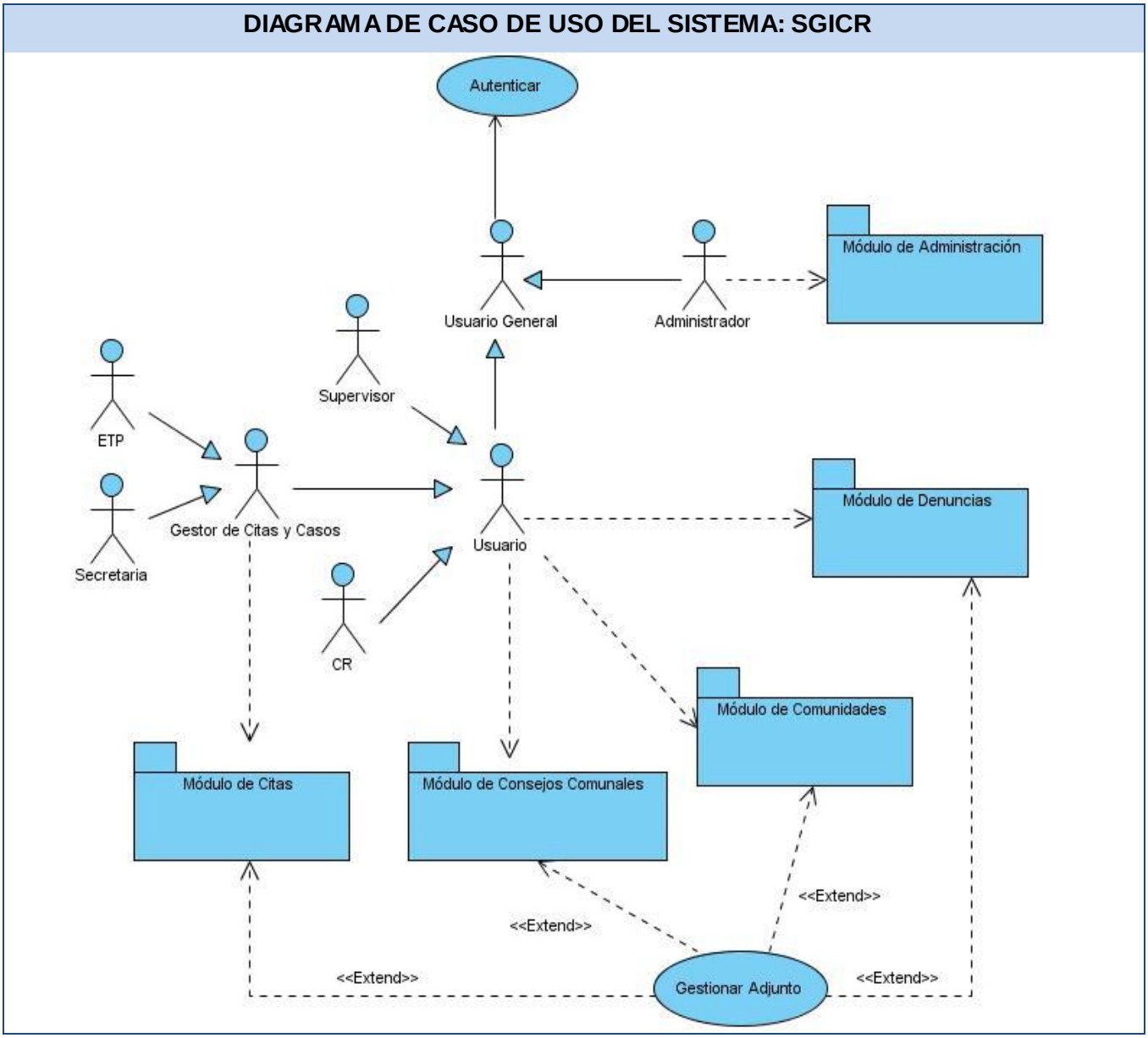


Fig. 2.2 Diagrama de Casos de Uso del Sistema: SGICR.

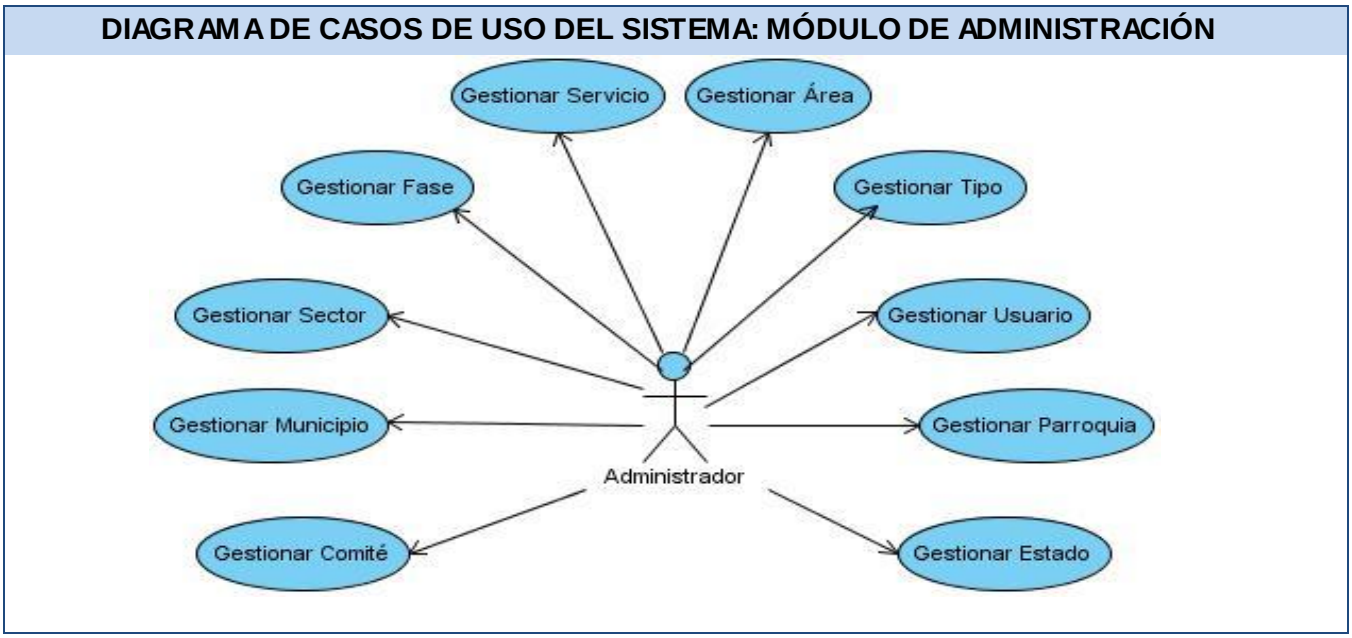


Fig. 2.3 Diagrama de Casos de Uso del Sistema: Módulo de Administración.

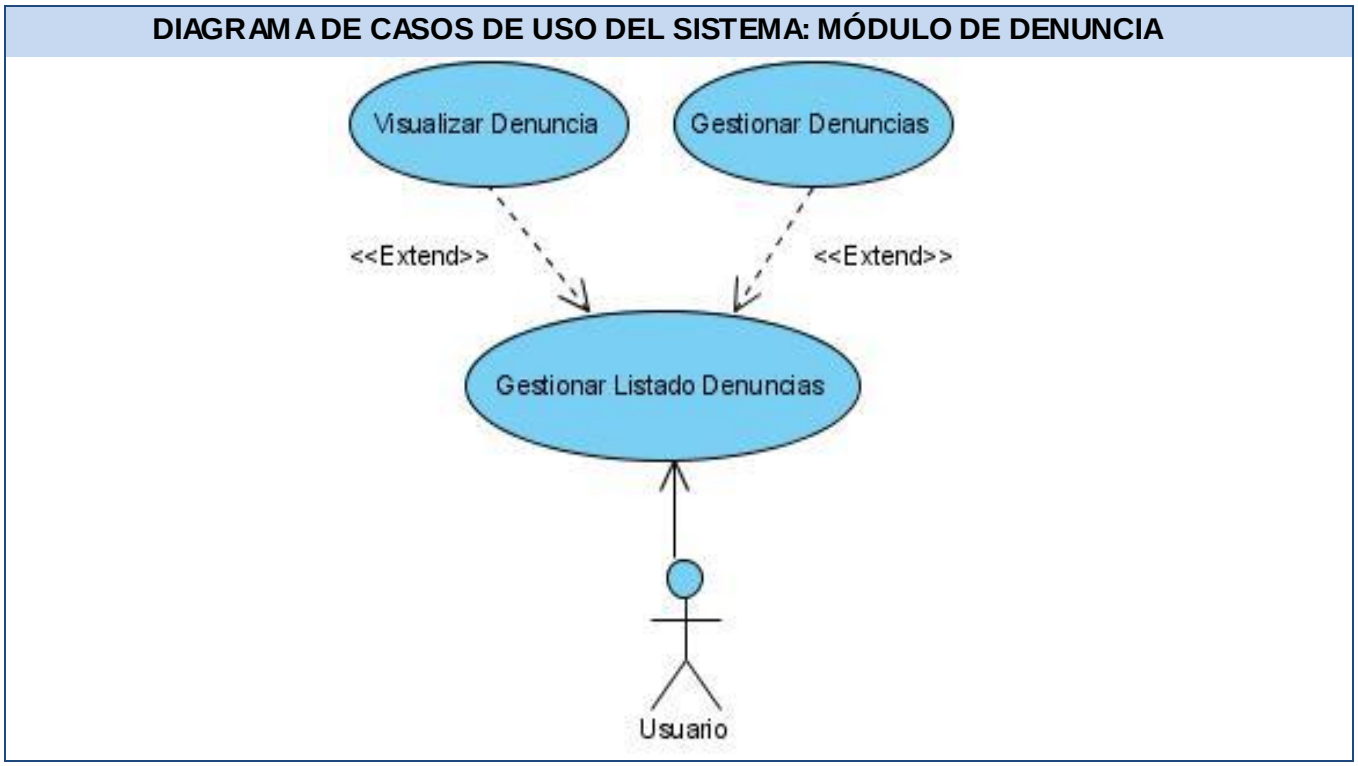


Fig. 2.4 Diagrama de Casos de Uso del Sistema: Módulo de Denuncias.

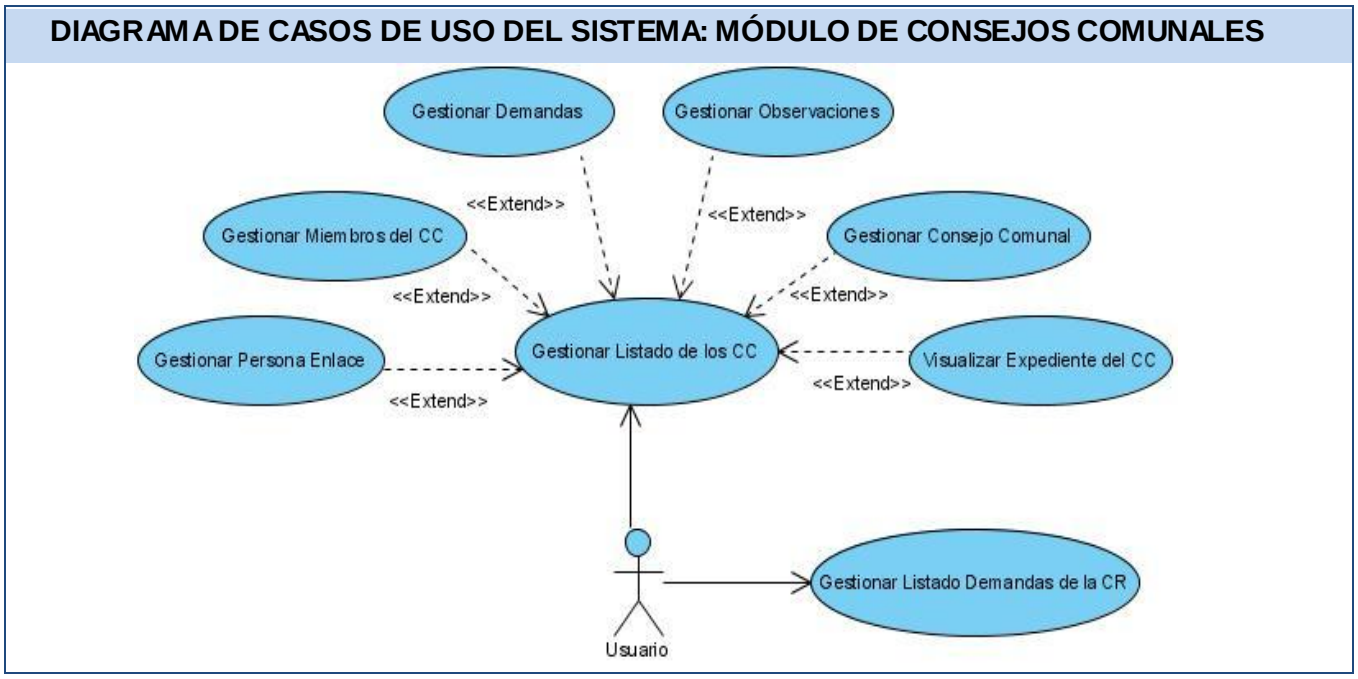


Fig. 2.5 Diagrama de Casos de Uso del Sistema: Módulo de Consejos Comunales.

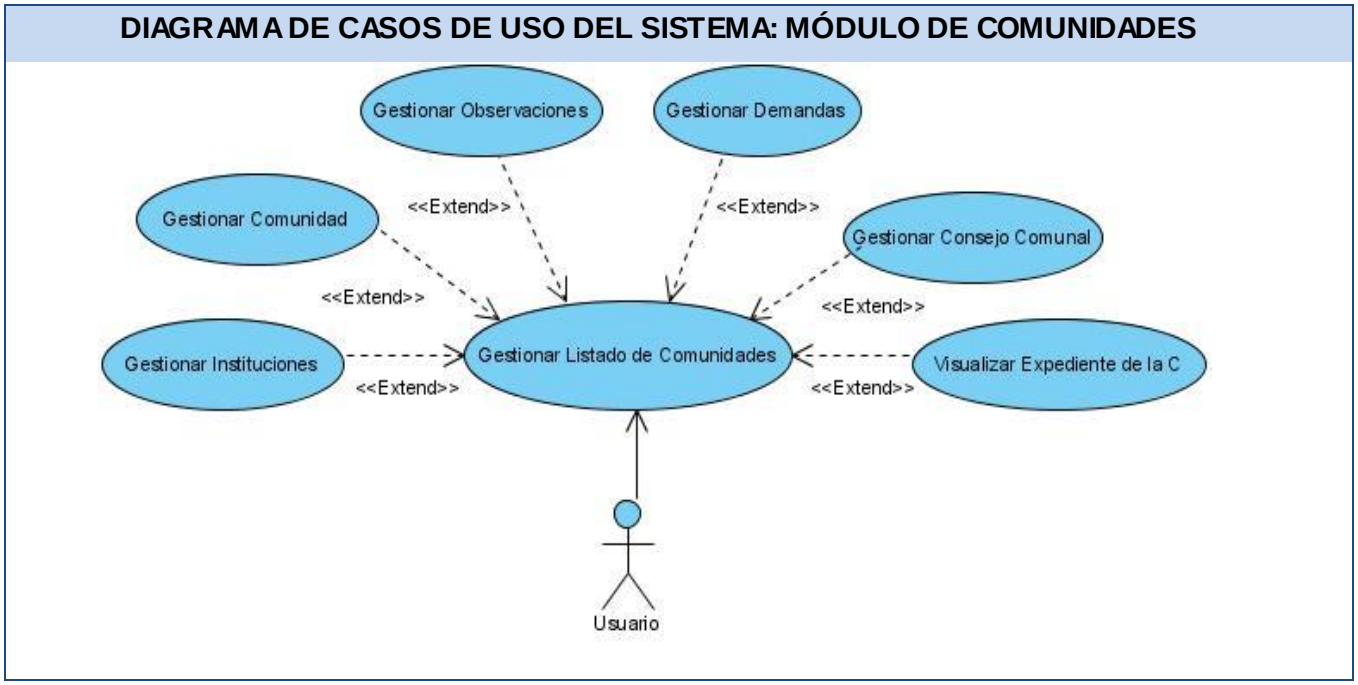


Fig. 2.6 Diagrama de Casos de Uso del Sistema: Módulo de Comunidades.

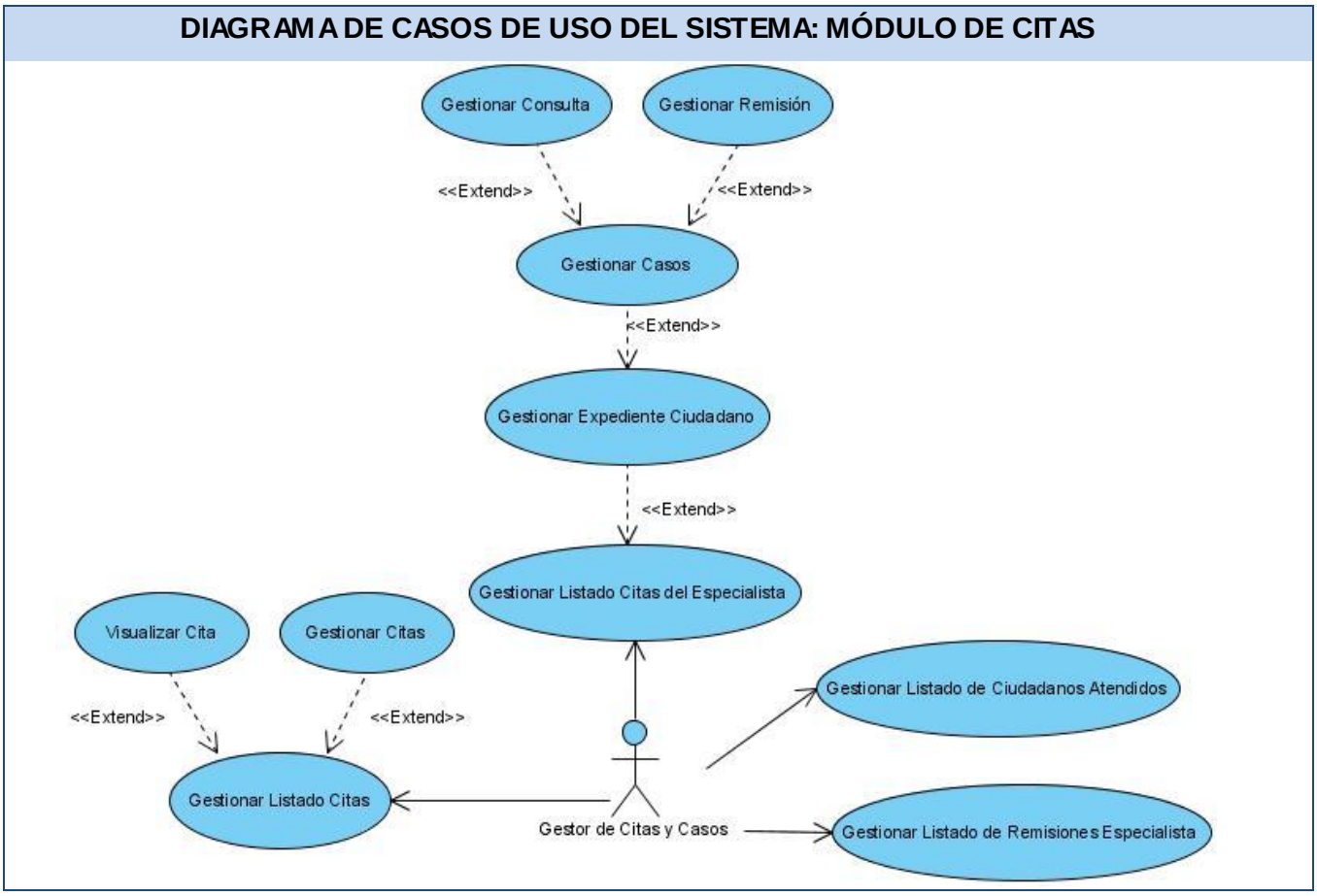


Fig. 2.7 Diagrama de Casos de Uso del Sistema: Módulo de Citas.

2.5 Patrones de Caso de Uso utilizados.

- **Múltiples actores**

Roles comunes:

Puede suceder que los dos actores jueguen el mismo rol sobre el CU. Este rol es representado por otro actor, heredado por los actores que comparten este rol. Es aplicable cuando, desde el punto de vista del caso de uso, sólo exista una entidad externa interactuando con cada una de las instancias del caso de uso.

A continuación se presenta un ejemplo donde se evidencia dicho patrón. En este caso los actores “Supervisor” y “CR”, ambos son usuarios del sistema por lo que para acceder al mismo deben ejecutar el caso de uso “Autenticar Usuario”.

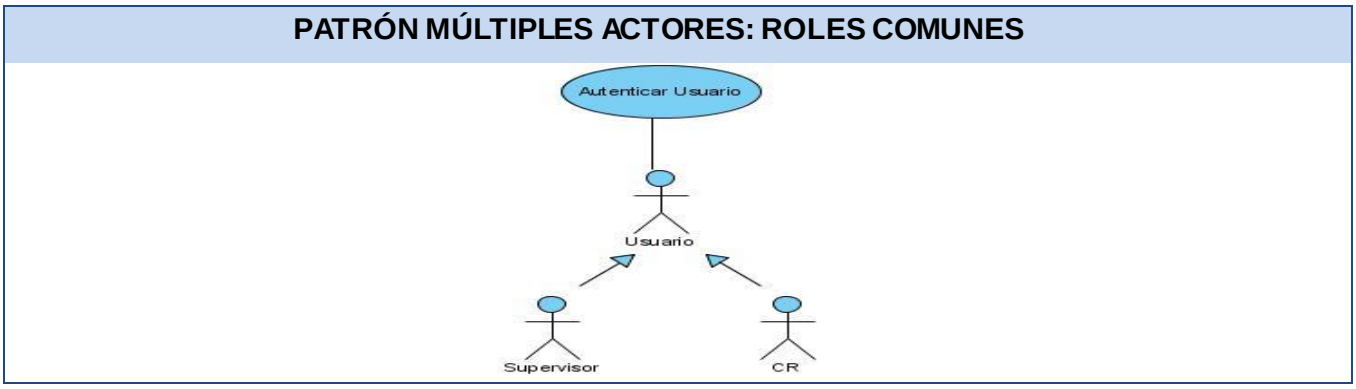


Fig. 2.8 Ejemplo del Patrón Múltiples Actores: Roles comunes.

- **Concordancia**

Adición:

En el caso de este patrón alternativo, la subsecuencia común de casos de uso, extiende los casos de uso compartiendo la subsecuencia de acciones. Los otros casos de uso modelan el flujo que será expandido con la subsecuencia. Este patrón es preferible usarlo cuando otros casos de uso se encuentran propiamente completos, o sea, que no requieren de una subsecuencia común de acciones para modelar los usos completos del sistema.

A continuación se presenta un ejemplo donde se evidencia dicho patrón. En este caso existe un caso de uso base llamado “Gestionar Listado de Denuncias”, el mismo extiende a los casos de usos “Visualizar Denuncia” y “Gestionar Denuncias”. Es decir, que el caso de uso “Gestionar Listado de Denuncias” posee las funcionalidades que presentan los casos de usos “Gestionar Denuncias” y “Visualizar Denuncia”

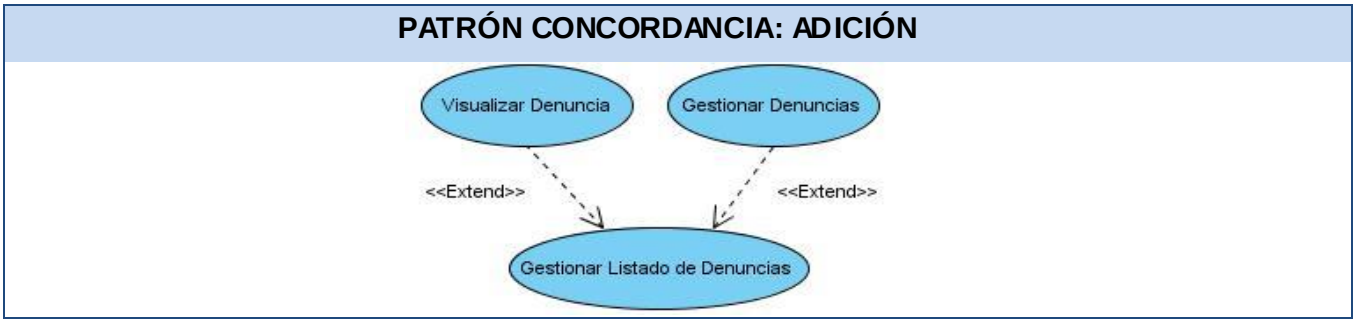


Fig. 2.9 Ejemplo del Patrón Concordancia: Adición.

- **CRUD (Creating, Reading, Updating, Deleting)**

Este patrón se basa en la fusión de casos de uso simples para formar una unidad conceptual.
Este patrón consta de un caso de uso, llamado Información CRUD o Gestionar información, modela todas las operaciones que pueden ser realizadas sobre una parte de la información de un tipo específico, tales como creación, lectura, actualización y eliminación. Suele ser utilizado cuando todos los flujos contribuyen al mismo valor del negocio, y los mismos a su vez son cortos y simples.

A continuación se presenta un ejemplo donde se evidencia dicho patrón. En este caso existe un actor llamado "Usuario" que ejecuta el caso de uso "Gestionar Denuncias". Este caso de uso agrupa un conjunto de funcionalidades asociadas a un mismo tipo de información, como: insertar, eliminar y actualizar Denuncia.

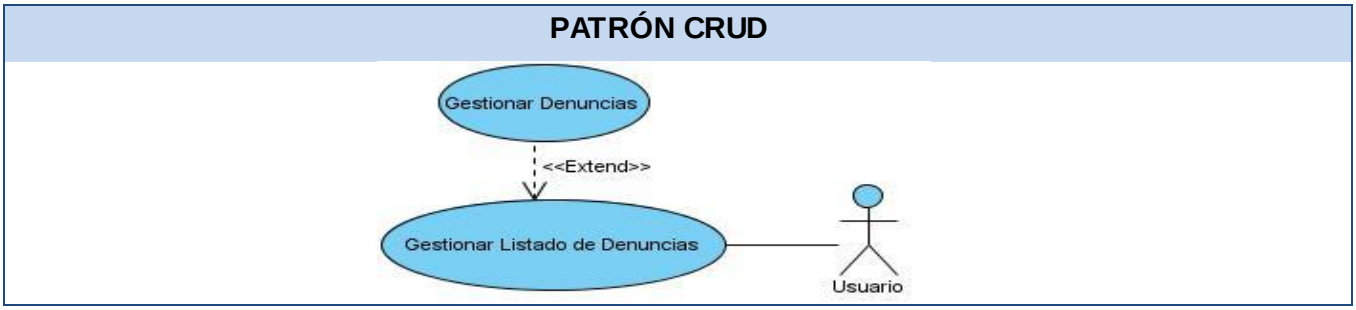


Fig. 2.10 Ejemplo del Patrón CRUD.

2.6 Especificación de los Casos de Uso del Sistema.

A continuación se muestra la descripción del caso de uso Gestionar Denuncias, el mismo es un caso de uso crítico dentro del Módulo de Denuncia. Para ver las descripciones textuales de los restantes casos de usos remitirse al Anexo1 en la versión digital de este documento.

CUS- Gestionar Denuncia.

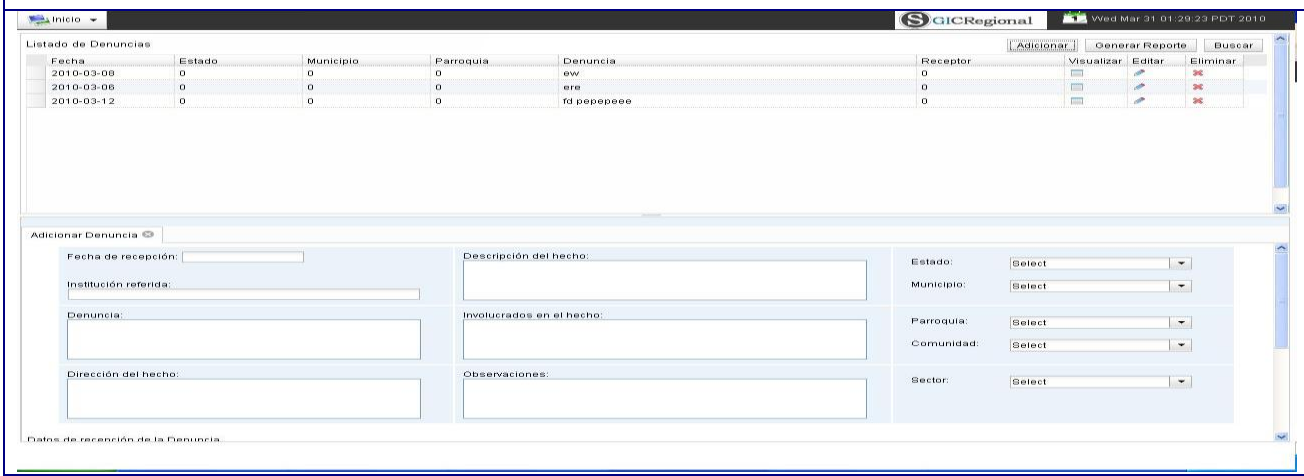
Nombre del CU	Gestionar Denuncia
Actores	Usuario
Propósito	Gestionar toda la información referente a las denuncias.
Resumen	El caso de uso se inicia cuando el usuario decide gestionar denuncias,

	el sistema brinda la posibilidad de registrar una nueva denuncia, actualizar o eliminar una denuncia seleccionada del listado de denuncias de la Coordinación Regional.
Referencias	RF7, RF8, RF9
Precondiciones	El usuario debe estar autenticado con el rol CR ó ETP.
Poscondiciones	Queda actualizado el listado de denuncias.

Curso Normal de los Eventos

Acciones del Actor	**Respuesta del Sistema**
1. El usuario decide Gestionar denuncias	1.1. El sistema brinda la posibilidad de Registrar una nueva denuncia, Actualizar y Eliminar una denuncia seleccionada del listado.
2. El usuario selecciona una de estas opciones.	2.1. El sistema ejecuta la opción seleccionada: ➢ Si decide registrar una nueva denuncia, ir a la sección "**Registrar denuncia**". ➢ Si decide actualizar la denuncia seleccionada, ir a la sección "**Actualizar denuncia**". ➢ Si decide eliminar la denuncia seleccionada, ir a la sección "**Eliminar denuncia**".

Sección "Registrar denuncia"



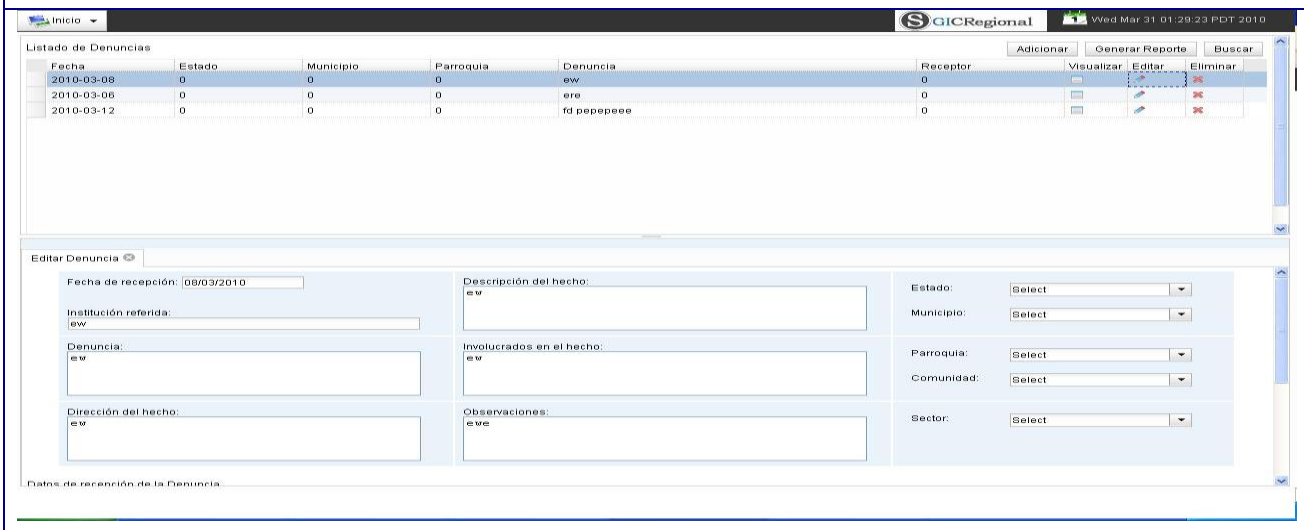
Acciones del Actor	**Respuesta del Sistema**
	1. El sistema muestra un formulario con los datos

	de denuncia que el usuario debe llenar, brinda la posibilidad de registrarlos y de adjuntar algún dato de interés.
2. El usuario decide adjuntar archivos.	2.1. El sistema va al caso de uso "**Gestionar adjunto**", sección "**Adjuntar archivo**".
3. El usuario introduce los datos y escoge la opción "Registrar".	3.1. El sistema verifica que no hayan introducido datos inconsistentes y además que todos los campos obligatorios estén llenos.
	3.2. El sistema registra la denuncia, se actualiza el listado de denuncias y finaliza el caso de uso.

Curso Alternativo de los Eventos

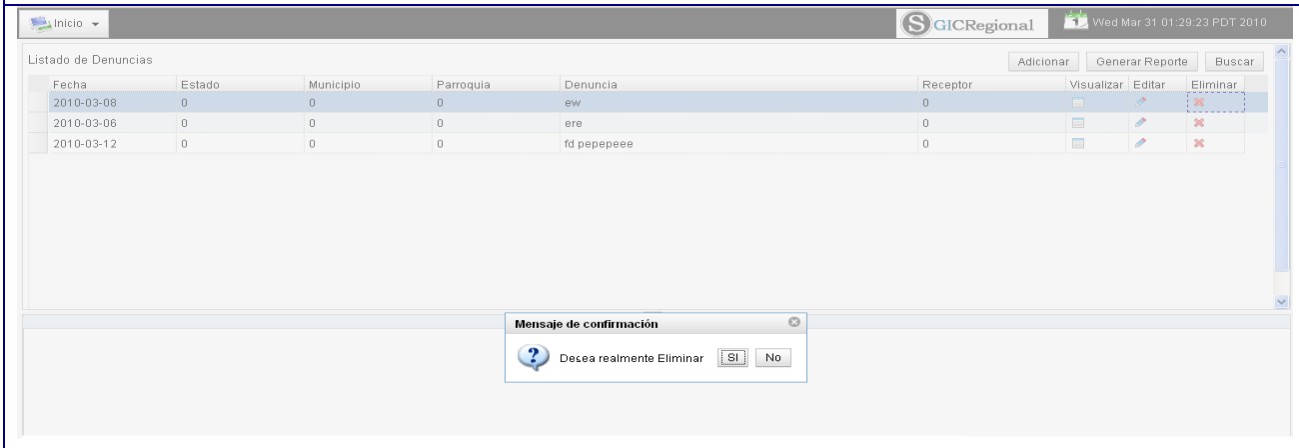
Acciones del Actor	**Respuesta del Sistema**
2. El usuario decide no adjuntar archivos.	2.1. El sistema no va al caso de uso "**Gestionar adjunto**", sección "**Adjuntar archivo**".
3. El usuario selecciona la opción "Cancelar" y finaliza el caso de uso.	3.1. Si el sistema verifica que hay datos inconsistentes y además que no todos los campos obligatorios están llenos; emite un mensaje de error y señala los datos incorrectos.

Sección "Actualizar denuncia"



Acciones del Actor	**Respuesta del Sistema**

1. El usuario selecciona una denuncia del listado de denuncias y escoge la opción "Actualizar".	1. El sistema muestra un formulario con los datos de la denuncia a actualizar. Da la opción de "Actualizar" y de "Cancelar".
2. El usuario decide actualizar los adjuntos.	2.1. El sistema va al caso de uso "**Gestionar adjunto**".
3. El usuario cambia los datos que desea actualizar y selecciona la opción "Actualizar".	3.1. El sistema verifica que no hayan introducido datos inconsistentes, ni hayan campos obligatorios vacíos.
	3.2. El sistema actualiza la denuncia y finaliza el caso de uso.
Curso Alternativo de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
2. El usuario no decide actualizar los adjuntos.	2.1. El sistema no va al caso de uso "**Gestionar adjunto**".
3. El selecciona la opción "Cancelar" y finaliza el caso de uso.	3.1. Si el sistema verifica que hay datos inconsistentes y además que no todos los campos obligatorios están llenos; emite un mensaje de error.
Sección "Eliminar denuncia"	



Acciones del Actor	**Respuesta del Sistema**
1. El usuario selecciona una denuncia del	1.1. El sistema muestra un mensaje de

listado de denuncias y escoge la opción "Eliminar".	confirmación y brinda las opciones de "Si" o "No".
2. El usuario escoge la opción "Si".	2.1. El sistema elimina la denuncia seleccionada y finaliza el caso de uso.
Curso Alternativo de los Eventos	
Acciones del Actor	**Respuesta del Sistema**
2. El usuario escoge la opción "No".	2.1. El sistema no elimina la denuncia seleccionada y finaliza el caso de uso.
Prioridad:	Crítico

Tabla. 2.1 CUS Gestionar Denuncias.

2.7 Conclusiones.

En el presente capítulo, además de hacerse una breve descripción del negocio del sistema a desarrollar, fueron abordados algunos de los artefactos propuestos por RUP para el desarrollo del software tales como: el modelo de dominio, definición de los requisitos funcionales y no funcionales, diagrama de casos de uso del sistema y la descripción textual de los mismos. También se ejemplificaron los patrones de casos de uso presentes en los diagramas de CUS.

CAPÍTULO 3: DISEÑO DEL SISTEMA

3.1 Introducción.

En este capítulo se traducen los requisitos a una especificación que describe como implementar el sistema a través del diseño, enfocado en 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 definidos en el capítulo anterior. También se muestra el modelo entidad-relación (MER) y se explica además la arquitectura y los principales patrones de diseño utilizados.

3.2 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. [22]

¿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. [23]

Para la realización del SGICR el colectivo de autores decidió utilizar 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. [13]

Este estilo constituye una especialización de la arquitectura Cliente-Servidor, donde la carga se divide en tres partes o capas lógicas fundamentales (Fig. 3.1) con un reparto claro de funciones. El desarrollo de cada paquete del SGICR 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.

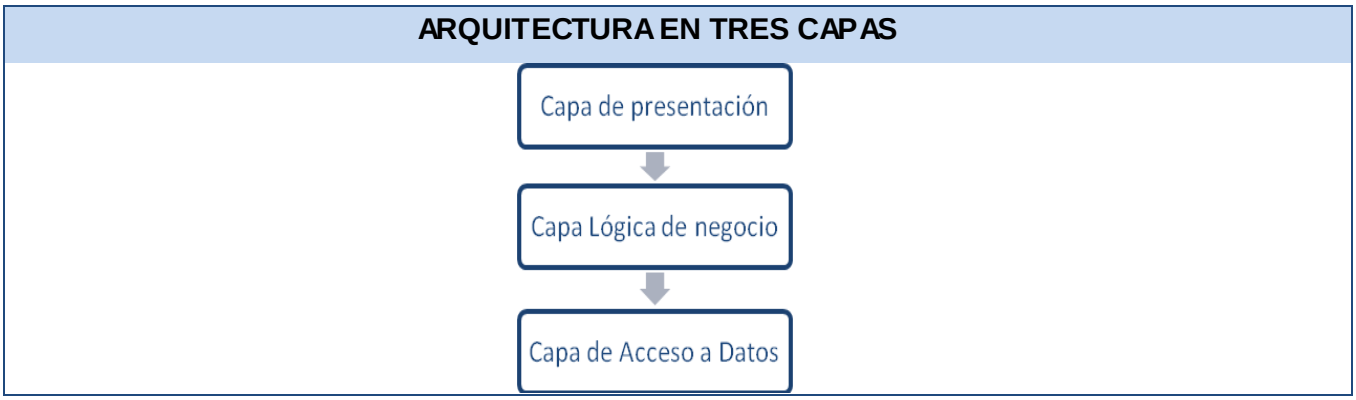


Fig. 3.1 Arquitectura en tres capas

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 para el uso de este patrón usando el framework ORM Hibernate, mientras que las interfaces se mantienen independientes de Spring e Hibernate. Se encuentran además en esta capa las clases del dominio que representan las clases entidades.

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.

3.3 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. [24]

Entre los patrones de diseño seleccionados está el de Fachada, debido a que con el uso del mismo se logra ocultar la complejidad de algunos componentes y aumentar la capacidad del sistema para evolucionar fácilmente, es decir, fortalece su modificabilidad.
Patrón de diseño Fachada: Este patrón sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Utilizado para reducir la dependencia entre clases, ya que ofrece un punto de acceso al resto de las clases, si estas cambian o se sustituyen por otras solo hay que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones clientes. Fachada no oculta las clases sino que ofrece una forma más sencilla de acceder a ellas, 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 **src.<modulo>.facade** las cuales sirven de fachada entre las clases de los paquetes **src.<modulo>.web.controller** y **src.<modulo>.service.** El fragmento de diagrama de secuencia que se muestra en la (Fig. 3.2) correspondiente al caso de uso “Añadir Denuncia” es un ejemplo de lo antes expuesto.

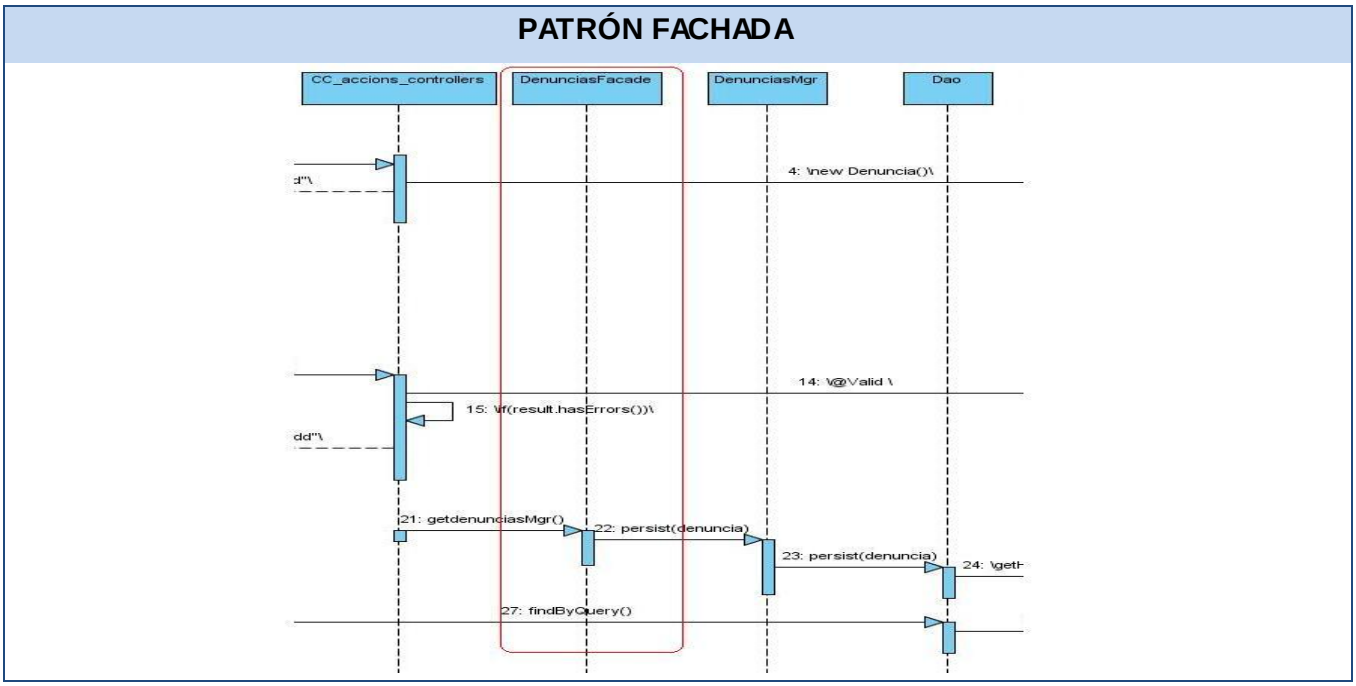


Fig. 3.2 Ejemplo del patrón Fachada.

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.

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 les 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.

También se utilizó **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 un DAO Genérico encargado 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. 3.4) correspondiente al caso de uso “Añadir Denuncia” es un ejemplo de lo antes expuesto.

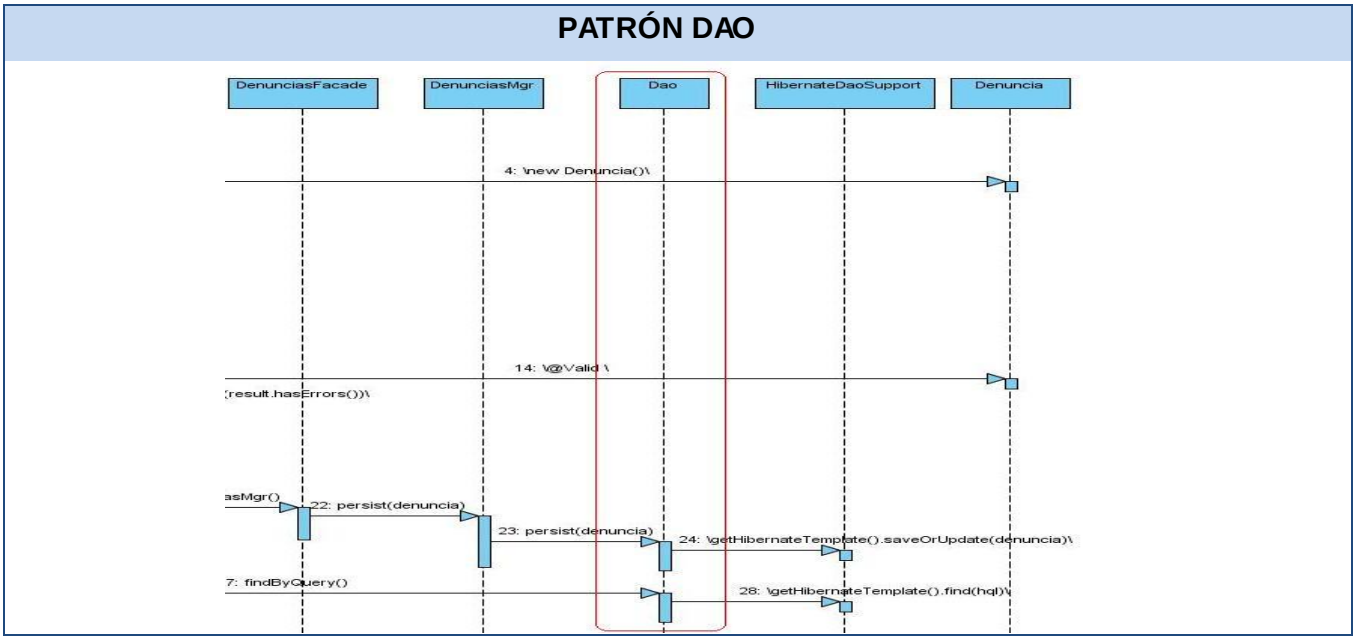


Fig. 3.3 Ejemplo del Patrón DAO

Polimorfismo: ¿Quién, cuándo el comportamiento varía según el tipo?

Cuando varía el tipo (clase) de alternativas o comportamientos relacionados, asignar la responsabilidad del comportamiento mediante operaciones polimórficas a los tipos en que varía el comportamiento. Es fácil agregar las futuras extensiones que requieren las variaciones imprevistas.

El uso de este patrón en el sistema se evidencia en el componente Gridtable el cual contiene para el manejo de los filtros de todos los listados una jerarquía de clases (Fig. 3.5), que en dependencia del tipo de dato configura las variables, restricciones y la vista asociada.

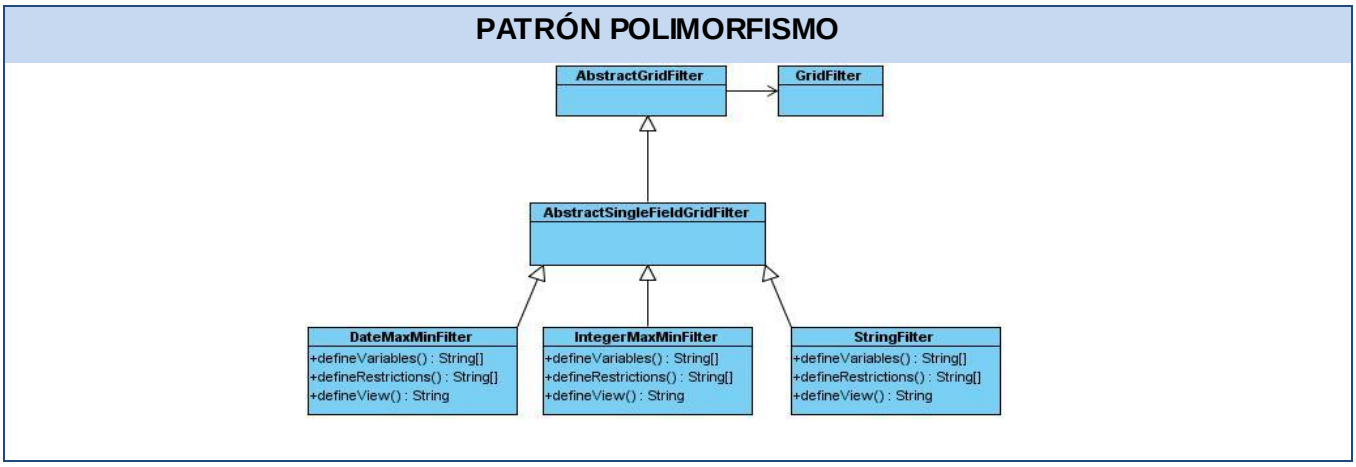


Fig. 3.4 Ejemplo del patrón Polimorfismo.

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. Son delicadas y las afectan constantemente los cambios. 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**.

3.4 Diagramas de Clases del Diseño.

Los diagramas de clases son ampliamente utilizados en el modelado de sistemas orientados a objetos, empleándose para representar las relaciones que se establecen entre las clases. Un diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces de la aplicación. Sirve también para visualizar las relaciones entre las clases que involucran el sistema.

A continuación se muestra en la (Fig. 3.5) el Diagrama de clases del diseño para el caso de uso Gestionar Denuncia perteneciente al Módulo de Denuncias, dicho diagrama muestra una vista detallada de la ubicación de las clases por capas.

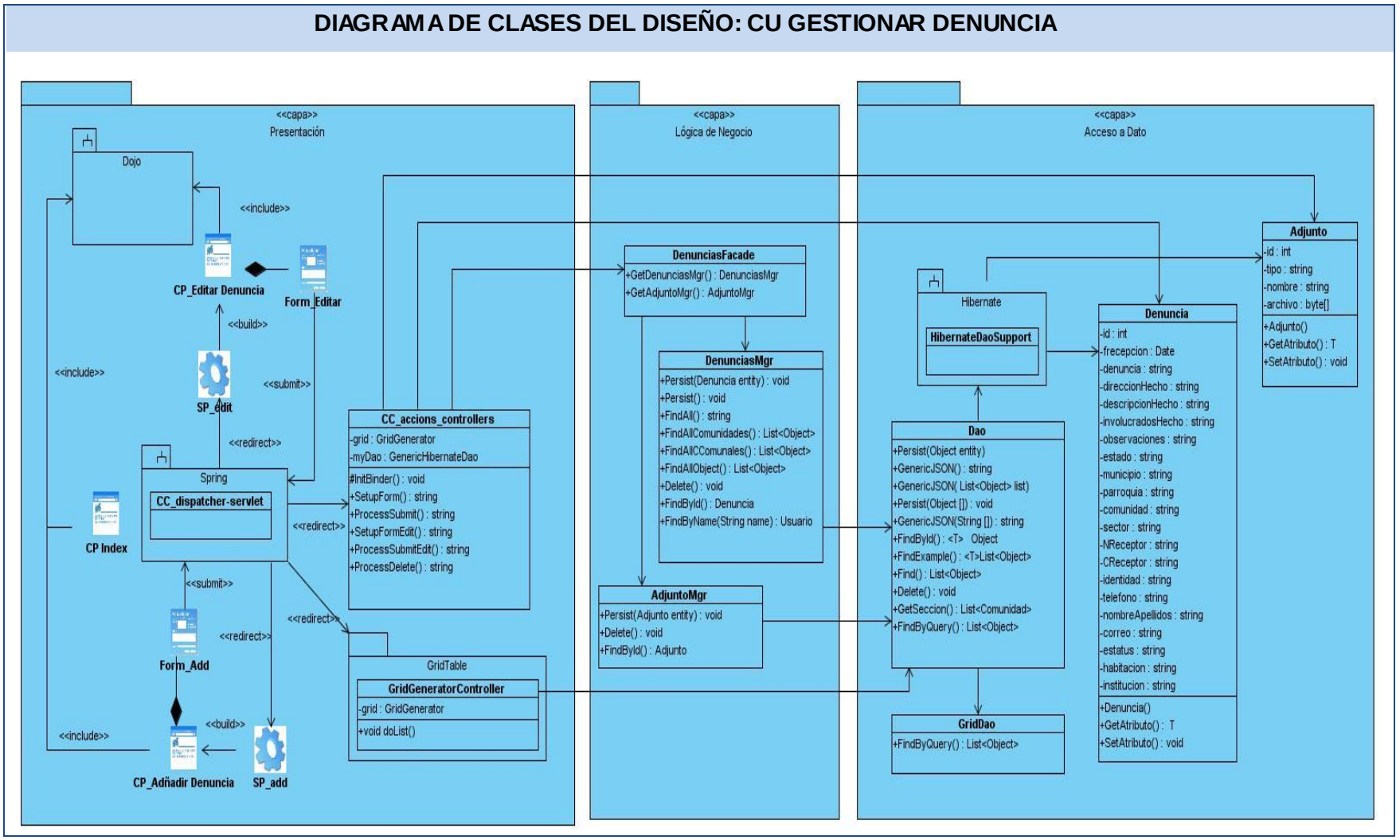


Fig. 3.5 Diagrama de clases del diseño para el CU Gestionar Denuncia.

3.5 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: Añadir Denuncia (Fig. 3.6), Editar Denuncia (Fig. 3.7) y Eliminar Denuncia (Fig. 3.8), todos del CUS Gestionar Denuncias.

3.6 Modelo de Datos.

El Modelo Entidad Relación es un tipo de diagrama para modelado de base de datos. Su objetivo es representar relaciones que existen en la vida real entendiendo su semántica.

Normalización.

Se puede entender la normalización como una serie de reglas que sirven para ayudar a los diseñadores de bases de datos a desarrollar un esquema que minimice los problemas de lógica. Cada regla está basada en la que le antecede. [25]

Es importante destacar además que la base de datos del sistema se encuentra normalizada hasta la tercera forma normal ya que en ella se incluye la eliminación de todos los grupos repetidos (1ra FN), se asegura que todas las columnas que no son llave sean completamente dependientes de la llave primaria (2FN), y se elimina cualquier dependencia transitiva, es decir, se eliminan las dependencias entre las columnas que no son llave y que a su vez son dependientes de otras columnas que tampoco son llave (3FN). A continuación, se muestra el diagrama Entidad-Relación de la base de datos del SGICR (Fig. 3.9).

3.7 Conclusiones.

En el presente capítulo se hizo uso del estilo arquitectónico en capas (tres capas), también se aplicaron varios patrones de diseño, se realizaron diagramas de clases del diseño y los diagramas de secuencia de los diferentes escenarios de cada caso de uso. Además se obtuvo el modelo de datos; quedando de esta forma estructurado el sistema.

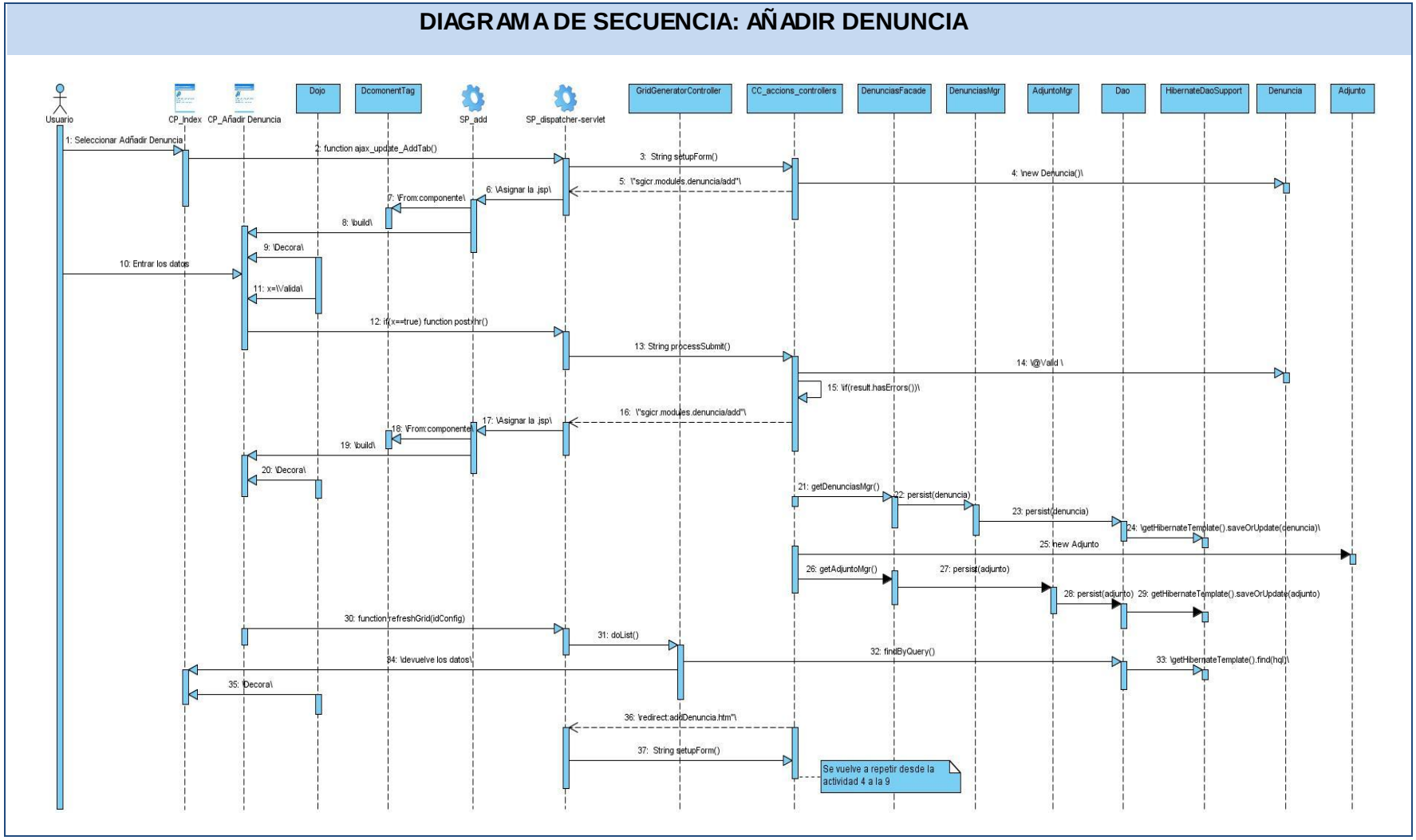


Fig. 3.6 Diagrama de Secuencia del CU Gestionar Denuncia, escenario Añadir Denuncia.

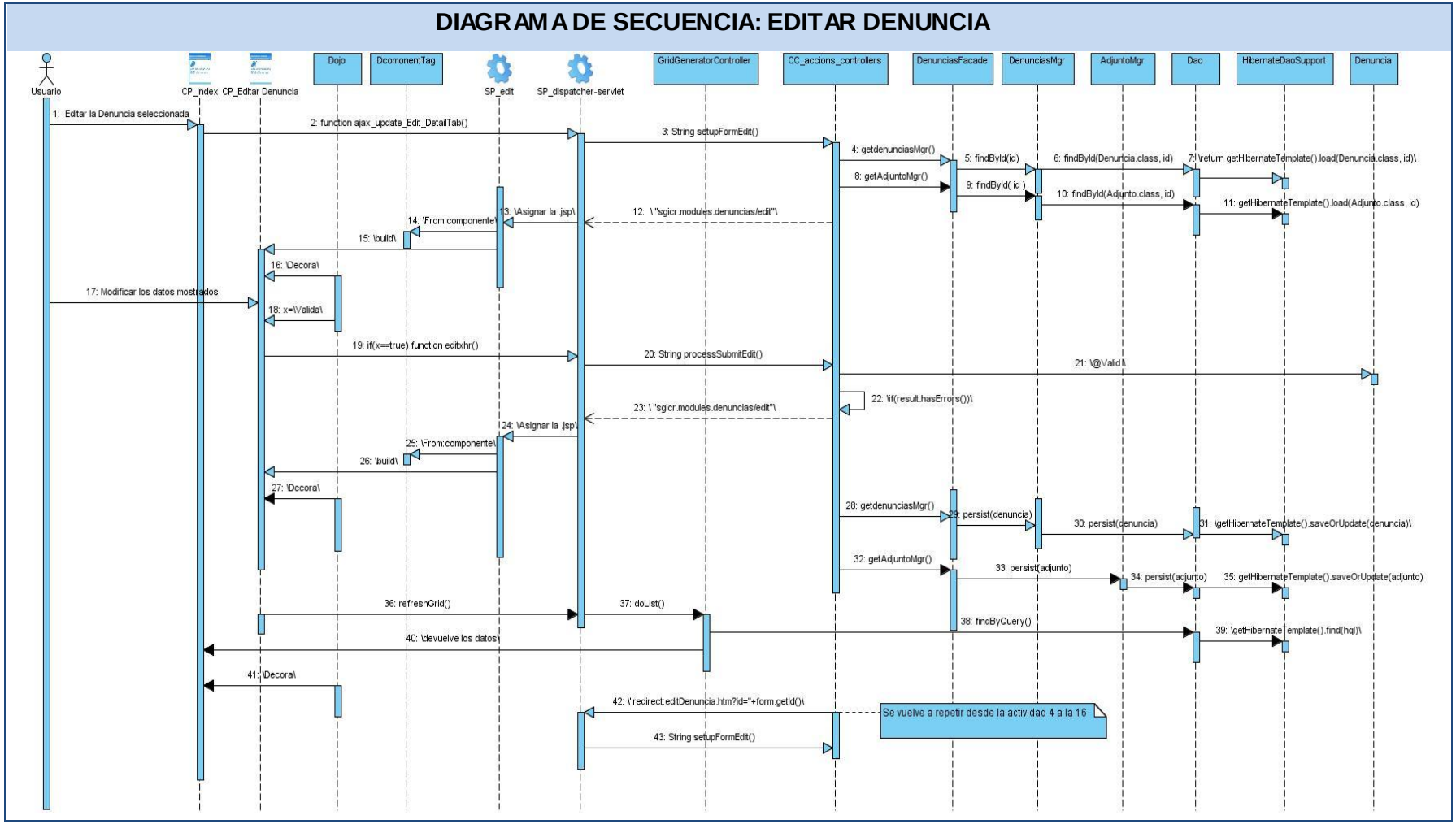


Fig. 3.7 Diagrama de Secuencia del CU Gestionar Denuncia, escenario Editar Denuncia.

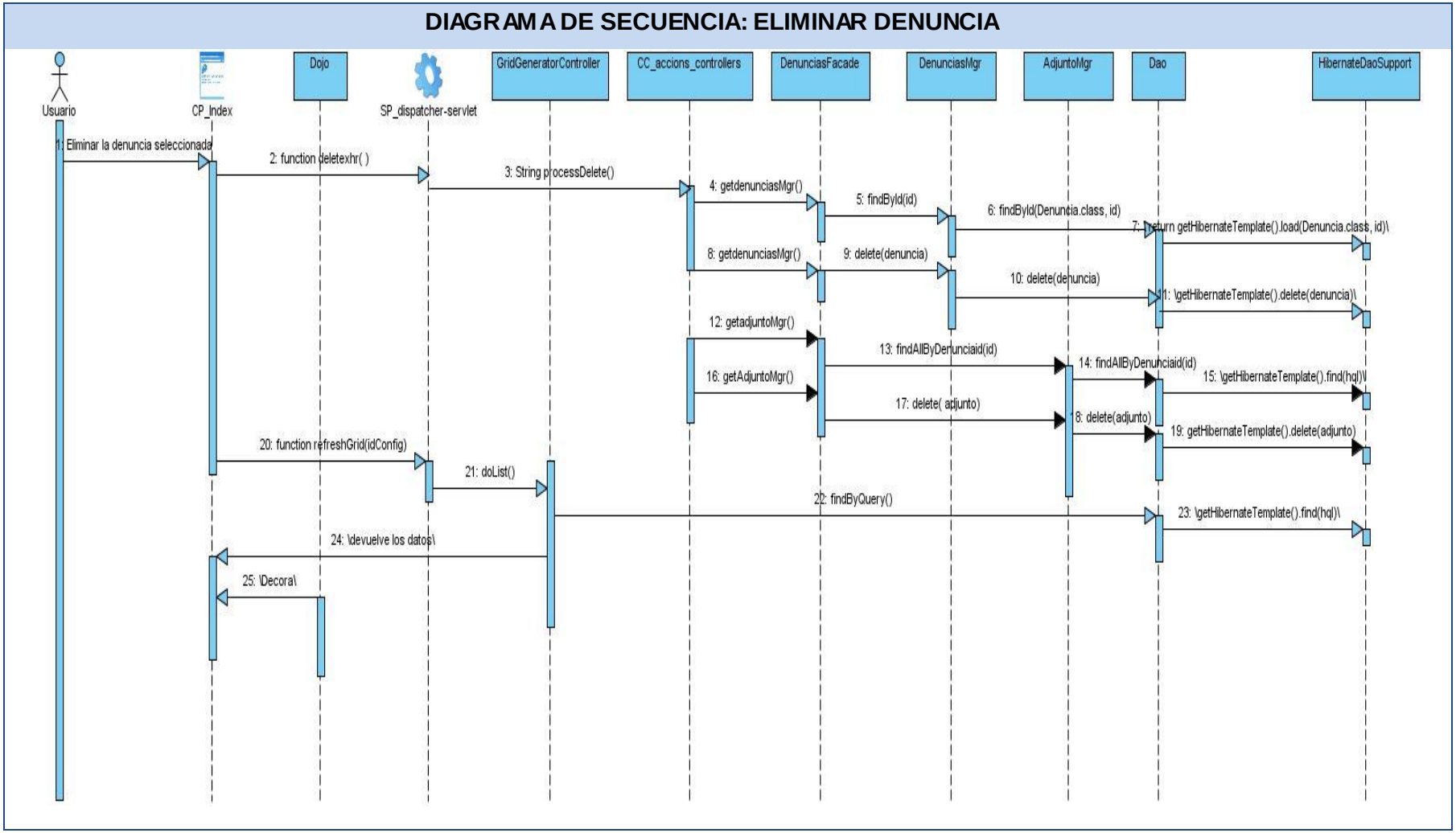


Fig. 3.8 Diagrama de Secuencia del CU Gestionar Denuncia, escenario Eliminar Denuncia.

Fig. 3.9 Diagrama de Entidad Relación.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA

4.1. 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. Además cuando se haya terminado el software se realizarán pruebas a nivel de desarrollador.

4.2. 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. [3]

4.2.1. Diagrama de Componentes.

Dentro del Modelo de Implementación se encuentran los diagramas de componentes. Un componente es la parte modular de un sistema, desplegable y reemplazable que encapsula implementación y un conjunto de interfaces que proporcionan la realización de los mismos. El diagrama de componentes describe cómo se organizan los componentes de acuerdo con los mecanismos de estructuración, modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados.

A continuación se muestra el Diagrama de Componente del CU Gestionar Denuncia perteneciente al Módulo de Denuncia (Fig. 4.1).

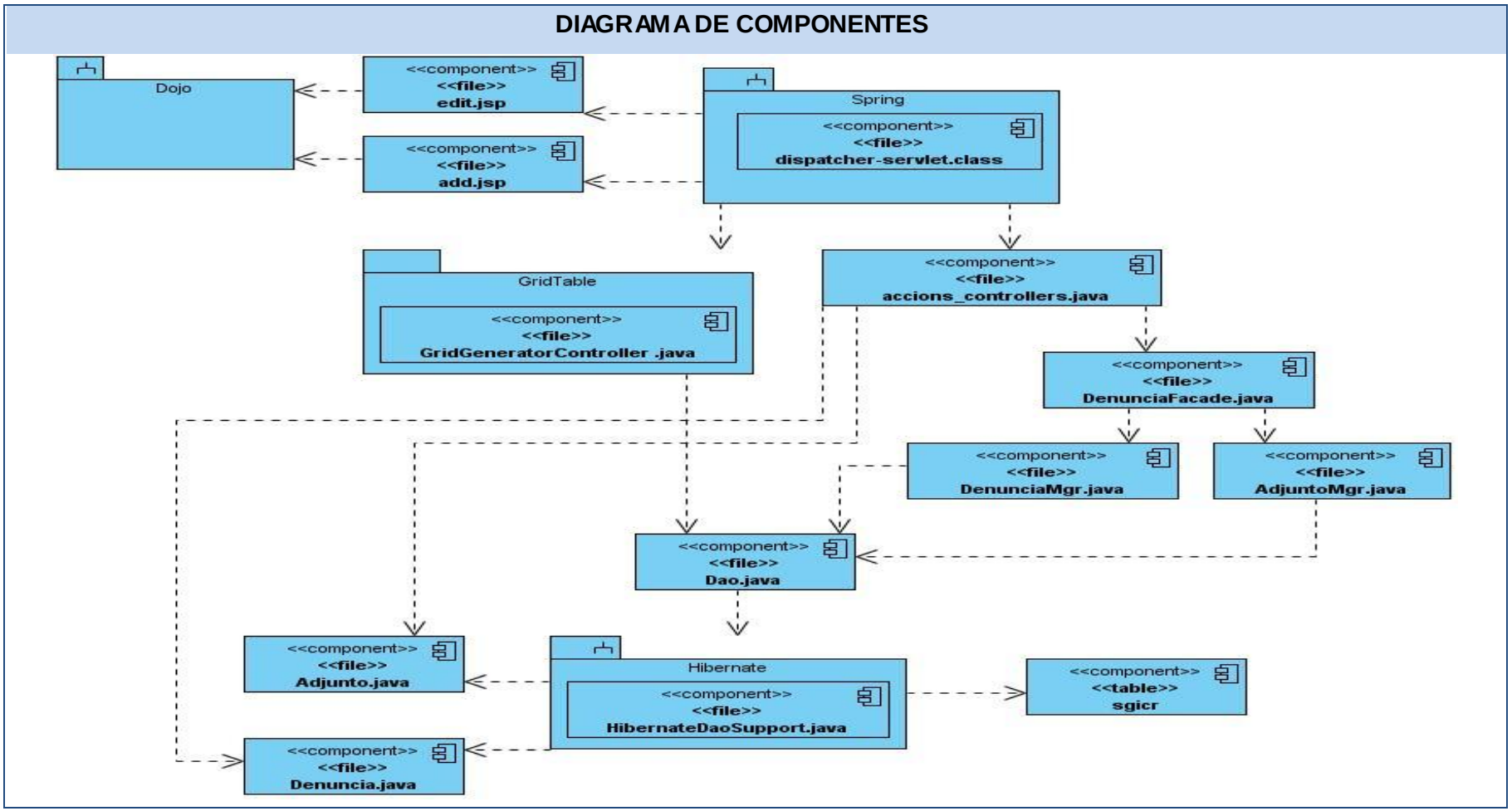


Fig. 4.1 Diagrama de Componentes: CU Gestionar Denuncias.

4.3. Código Fuente.

Para obtener una versión funcional de la aplicación se deben implementar 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.

Estas instrucciones son escritas en un lenguaje de programación que consiste en un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

4.3.1. 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, el mismo debe tender siempre a lo práctico.

Nombres de ficheros.

Los nombres de las clases anotadas con @Controller deben terminar en _controllers, las anotadas con @Service deben terminar en MgrImpl y la interfaz que implementa en Mgr, los @Component deben finalizar en facadeImpl y su interfaz en facade.

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 capeta 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, y si es necesario realizar una operación que no esté definida en el Dao Genérico se creará un Dao que herede de este y el mismo se tendrá en la carpeta dao.

ESTRUCTURA DEL PROYECTO

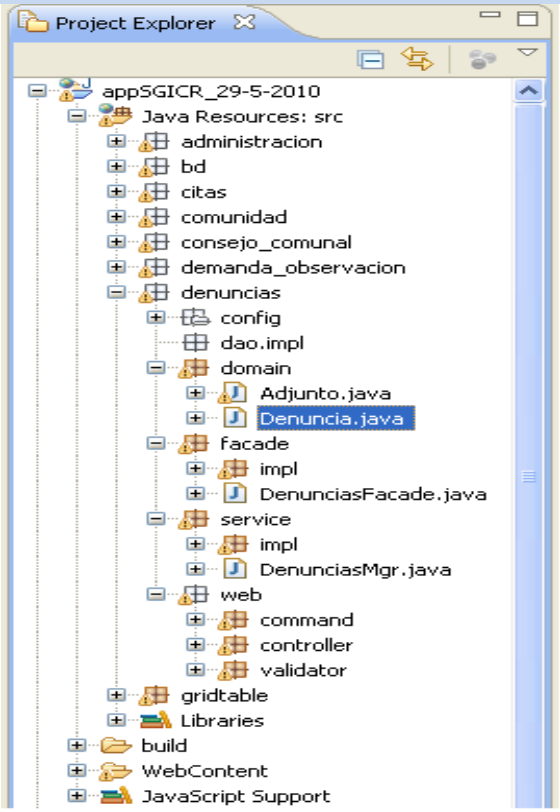


Fig. 4.2 Estructura del proyecto.

4.3.2. 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 las instancias, DenunciaFacade, GridGenerator y GeneriHibernateDao.

En el método FilterGrig () se ve el uso de dos anotaciones @PreAuthorize, que indica el acceso al método y @RequesMapping, que en su cuerpo, value, representa la url que debe ser invocada para que él se ejecute y el method, GET en este caso.

EJEMPLO DE CÓDIGO FUENTE

gridGenerator.js | *accions_controllers | add.jsp | index.jsp | GridFilter.java | DenunciasMgr.java | »12

```
package denuncias.web.controller;

import gridtable.GridGenerator;

@Controller
public class accions_controllers {

	@Resource
	private DenunciasFacade denunciasFacade;

	@Resource
	private GridGenerator grid;

	@Resource
	GenericHibernateDao myDao;

	@PreAuthorize("hasRole('ROLE_ETP,ROLE_CR')")
	@RequestMapping( value="/filterGrid.htm" , method = RequestMethod.GET)
	public String FilterGrig(ModelMap model,
	                         HttpServletRequest request) throws Exception {
	    gridtable.domain.GridConfiguration mgt = grid.getConfiguration("denuncia");
	    model.addAttribute("gridConfiguration", mgt);
	    return "sgicr.modules.denuncias/filterGrid";
	}
```

Fig. 4.3 Ejemplo de Código Fuente: Método FilterGrig ().

4.4. Validación.

Validar los datos es una tarea común que se realiza en todas las aplicaciones, 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. [26]

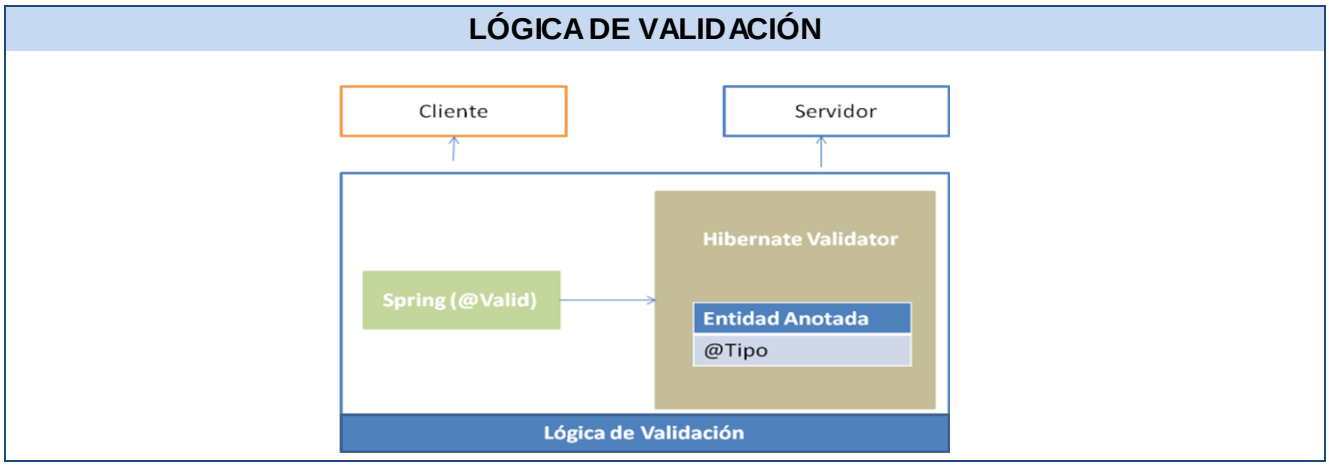


Fig. 4.4 Lógica de validación aplicada en el sistema.

4.5. Pruebas.

En el presente trabajo, los autores se limitan a realizar pruebas unitarias a nivel de desarrollador, teniendo en cuenta, que las mismas deben ser automatizables, completas, reutilizables e independientes.

Para realizar las pruebas se utilizó 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.

Con el uso de JUnit se evaluó el comportamiento de los métodos de cada Controlador, ya que los mismos son de gran interés, por ser el primer filtro por donde pasan las peticiones web. Después de haber pasado por varias iteraciones, se detectó que la clase que presentó mayor cantidad de errores, debido a su complejidad, fue **accions_controllers**, perteneciente al **Módulo de Citas**, aunque la mayoría de los errores fueron problemas con el mal tratamiento de excepciones.

Los resultados demuestran, puesto que en el desarrollo del sistema se utilizó Spring 3.0.2, que reduce significativamente la cantidad y complejidad de líneas de código, debido al uso de las anotaciones, así como, la reducción en la creación de clases, proporcionó la generación de una menor cantidad de errores. Por lo que se concluye que los resultados fueron satisfactorios.

A continuación, se muestra una gráfica con los resultados de las pruebas después de haber pasado por varias iteraciones.

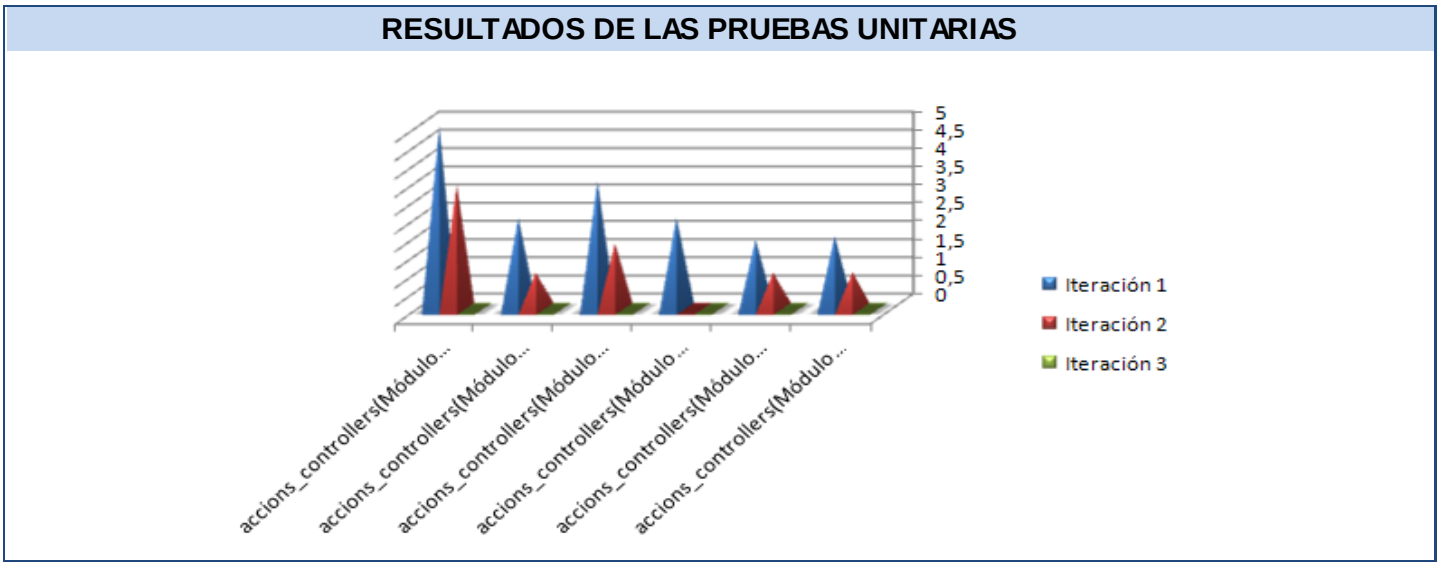


Fig. 4.5 Resultado de las iteraciones de las pruebas unitarias.

4.6. Interfaces principales de la aplicación.

A continuación se muestran algunas pantallas del sistema:

Fig. 4.6 Interfaz Añadir Denuncia.

Fig. 4.7 Interfaz Editar Denuncia.

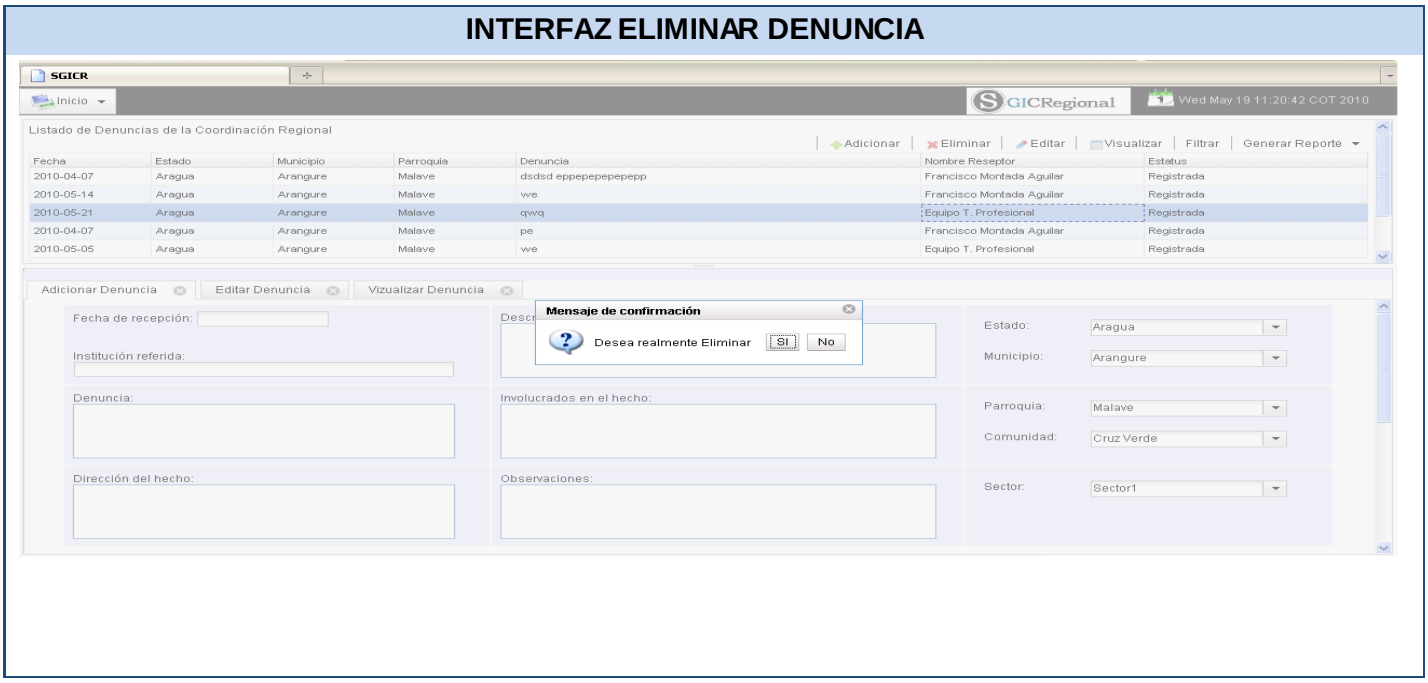


Fig. 4.8 Interfaz Eliminar Denuncia.

4.7. 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 funcionalidades que se definieron y debía tener el producto.

En este capítulo se elaboraron los diagramas de componentes, teniéndose confeccionada completamente la propuesta que trae este trabajo. Además se brindó una breve descripción de algunos métodos implementados para obtener un mejor entendimiento de los mismos. Se describió la arquitectura de validación aplicada en el sistema.

CONCLUSIONES

Una vez culminado el trabajo es posible afirmar que se les dio cumplimiento a los objetivos trazados para el mismo.

1. Se realizó el diseño de las clases, cumpliendo estándares y programación orientada a objetos, el cual sirvió de base para la posterior implementación.
2. El sistema fue implementado utilizando herramientas, lenguajes y tecnologías, en su mayoría distribuidas bajo licencia de software libre en correspondencia con las políticas de la Universidad y del país.
3. Es importante señalar las ventajas que trae como resultado la arquitectura que propone el framework Spring y su integración con los distintos framework usados para la implementación del sistema: esta integración permitió la simplificación del diseño de las clases que encapsula el sistema, garantizando por ende la alta cohesión y el bajo acoplamiento del mismo.
4. Finalmente, las pruebas realizadas para validar las funcionalidades que fueron implementadas, demuestran que el sistema cumple satisfactoriamente con los requisitos que garantizan su correcto funcionamiento.

RECOMENDACIONES

1. Continuar con el desarrollo del componente GridTable, agregándole mejores formas de configuración basada en estándares y patrones o partiendo de experiencia de otros frameworks. Terminar su desarrollo con el fin de encapsularlo en un .jar para ser usado en proyectos de la Universidad que utilicen Spring y Dojo como tecnología de desarrollo.

2. Se recomienda profundizar en el estudio de las Librerías de etiquetas o TagLibs de Spring, las cuales proporcionan el desarrollo de componentes más orientados a presentación que ayudan a simplificar las páginas. Facilitan la generación de forma dinámica de código html, siendo esta una ventaja para la reutilización de código y disminución del tiempo desarrollo.

3. Además se recomienda el estudio de JPA (Java Persistence API) ya que en su definición se han combinado ideas y conceptos de los principales frameworks de persistencia como Hibernate, Toplink, JDO y otros. Todos estos cuentan actualmente con una implementación JPA, su uso en el sistema posibilita migrar a otros ORM que implementen su especificación proporcionando una mayor flexibilidad ante el cambio.

REFERENCIAS BIBLIOGRÁFICAS

1. Lourdes Aja. Gestión de la información, 2009, [En línea] [Citado el: 22 de Enero del 2010] 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 Enero del 2010] Disponible en: http://www.monografias.com/trabajos17/sistema-gerencial/sistema-gerencial.shtml
3. 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]
4. 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/inq sw 1/RUP.pdf
5. Ayuda extendida de Rational Rose Enterprise Edition 2003, consultado el: 22 de Enero del 2010.
6. 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=
7. Visual Paradigm 2005 [En línea] [Citado el: 28 de Enero del 2010] Disponible en: http://www.visual-paradigm.com/product/vpuml/..
8. Definición.org Definición Lenguaje de Programación [En Línea] [Citado el: 28 de Enero del 2010] Disponible en: http://www.definicion.org/lenguaje-de-programacion
9. Schildt Herbert Java 2 Manual de referencia - Madrid: [s.n.], 2001. [Citado el: 2 de Febrero del 2010]
10. Ciberaula 2009 Ciberaula [En Línea] [Citado el: 5 de Febrero del 2010] Disponible en: http://java.ciberaula.com/articulo/tecnologia_java/.
11. 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.
12. Perrone Paul J and Krishna J2EE Developer's Handbook [Citado el: 5 de Febrero del 2010] - Indianapolis, Indiana: Sam's Publishing, 2003.
13. 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.

14. 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
15. 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
16. Spring-Framework-Reference [PDF] [Citado el: 28 de Febrero del 2010]
17. Framework Dojo [En línea] [Citado el: 28 de Febrero del 2010] Disponible en: http://www.dojotoolkit.org/
18. hibernate.org [En línea] [Citado el: 28 de Febrero del 2010] Disponible en: Disponible en: http://www.hibernate.org/
19. Group, P.G.D. PostgreSQL. 2007 [En línea] [Citado el: 10 de Febrero del 2010] Disponible en: http://www.postgresql.org/.
20. Pecos, D. PostGreSQL. 2005 [En línea] [Citado el: 10 de Febrero del 2010] Disponible en: http://www.netpecos.org/docs/mysql_postgres/x15.html.
21. jasperforge.org Sitio Web Oficial de JasperReport [En línea] [Citado el: 10 de Febrero del 2010] Disponible en: http://jasperforge.org/website/.
22. Estilos Arquitectónicos[En Línea][Citado el: 12 de Febrero del 2010] Disponible en: http://www.willydev.net/descargas/prev/IntroArq.pdf
23. 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
24. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. [Citado el: 16 de Abril del 2010]- *Design Patterns (Elements of Reusable Object-Oriented Software)*.
25. 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
26. Hibernate Validator JSR 303 Reference Implementation Reference Guide 4.0.1.GA [PDF] [Citado el: 28 de Abril del 2010]

BIBLIOGRAFÍA

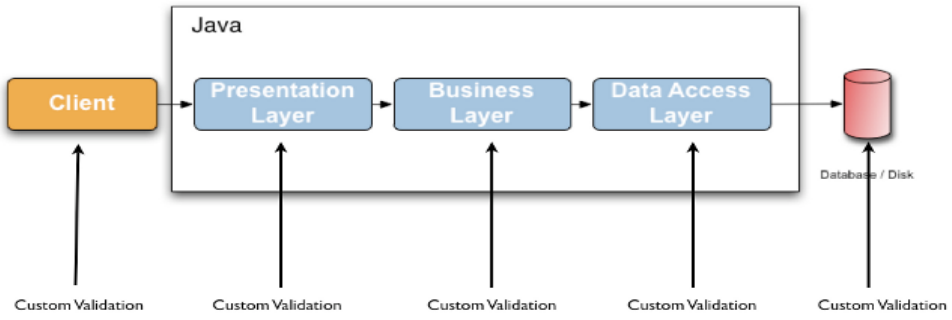
1. Ayuda extendida de Rational Rose Enterprise Edition 2003, [En línea] [Citado el: 22 de Enero del 2010]
2. Avilas**,** Diagramas de Secuencia, 2009, [En línea] [Citado el: 22 de Enero del 2010] Disponible en: http://tvdi.det.uvigo.es/~avilas/UML/node42.html
3. BEATON W. Eclipse Platform Technical Overview [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/ inq_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/..

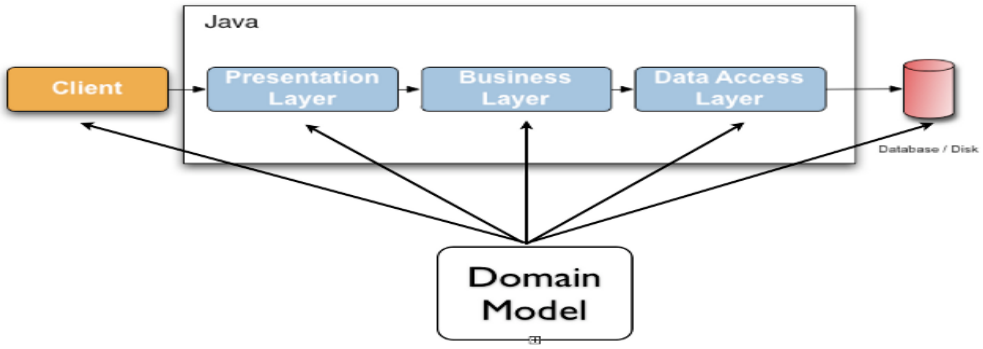
ANEXOS

Anexo 1:

VALIDACIÓN DEFINIDA EN CADA CAPA



VALIDACIÓN DEFINIDA DE FORMA CENTRALIZADA EN EL DOMINIO



Glosario de términos.

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

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

3. **CUS** (Caso de Uso del Sistema) es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software.

4. **Código abierto u Open source:** es el término con el que se conoce al software distribuido y desarrollado libremente.

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

6. **Entidades:** Objetos concretos o abstractos que presentan interés para el sistema.

7. **HTML:** Acrónimo inglés de Hyper Text Markup Language (lenguaje de marcación de hipertexto), es un lenguaje de marcas diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas web. Gracias a Internet y a los navegadores del tipo Explorer o Netscape, el HTML se ha convertido en uno de los formatos más populares que existen para la construcción de documentos.

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

9. **IoC** (Inversion of Control en inglés): es un método de programación en el que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales, en los que la interacción se expresa de forma imperativa haciendo llamadas a procedimientos (procedure calls) o funciones.

10. **Java Script:** Es un lenguaje interpretado, es decir, que no requiere compilación, utilizado principalmente en páginas Web, con una sintaxis semejante a la del lenguaje Java y el lenguaje C. Al contrario que Java, Javascript no es un lenguaje orientado a objetos propiamente dicho, ya que no dispone de herencia, es más bien un lenguaje basado en prototipos, ya que las nuevas clases se generan clonando las clases base (prototipos) y extendiendo su funcionalidad.

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

12. **JPA** (Java Persistence API): proporciona un modelo de persistencia basado en POJO's para mapear bases de datos relacionales en Java.

13. **JSP** (Java Server Pages): es una tecnología orientada a crear páginas web con programación en Java.

14. La API **Java Message Service** (en español **servicio de mensajes Java**), también conocida por sus siglas **JMS**: es un estándar de mensajería que permite a los componentes de aplicaciones basados en la plataforma Java2 crear, enviar, recibir y leer mensajes.

15. **POO:** Programación Orientada a Objeto.

16. **POJO:(Plain Old Java Object):** revalora la simplicidad de las clases Java aportando manejo de transacciones de forma no intrusiva.

17. **SGICR:** Sistema de Gestión de Información para las Coordinaciones Regionales.

18. **SQL:** es un lenguaje formal declarativo, estandarizado ISO, para manipular información en una base de datos.

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