República de Cuba

Universidad de las Ciencias Informáticas

Facultad 2

Sistema de Gestión Policial (SIGEPOL)
"Módulo de Denuncias"

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

Autor: Yusmar Castro Vera.

Tutor: Ing. Yaneisy Cruz Navarro.

"Año del 50 aniversario del triunfo de la Revolución"
Ciudad de la Habana, Cuba. Febrero de 2009.

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____
Yusmar Castro Vera
Autor

_____
Ing. Yaneisy Cruz Navarro
Tutora

Opinión del Tutor del Trabajo de Diploma

Dedicatoria

A mis padres (Omar y Lucia): por ser las personas que siempre están ahí cuando me hace falta, por inculcarme sus ejemplos como seres humanos y como profesionales, por quererme tanto y confiar siempre en mí, los quiero mucho.

A mi Hermano (Omarito): por trasmitirme siempre sus experiencias como hermano mayor y ser mi guía en todas las situaciones complicadas que te presenta la vida, eres el mejor hermano.

A mis abuelos: por hacerme saber que ellos siempre estarán orgullosos de mí.

A mi profe de Física de la secundaria (Berta): por hacer de mí un muchacho interesado por los estudios, capaz de aspirar a una buena carrera, por salvarme como profesional, de veras te agradezco mucho profe.

A mi novia (Dainy): por nunca dejarme caer, por la energía que me transmites, por tu amor hacia mi, eres una chica muy especial.

A todos mis amigos Rolando, Gustavo, Sandy, Oslando, Robert, Karel, Yenier, Víctor en fin a todos aquellos que siempre me han querido y desean que me convierta en un gran profesional.

Agradecimientos

A Yaneisy, mi tutora, por apoyarme en todo, por su gran ayuda y preocupación, por dedicar gran parte de su tiempo para que esta tesis tuviera la mejor calidad posible.
A mis compañeros del proyecto, porque en el tiempo que compartimos tareas aprendí mucho de ellos.

A todos Muchas Gracias.

Resumen

Actualmente la República Bolivariana de Venezuela no cuenta con un sistema de gestión capaz de integrar a todos los entes de seguridad ciudadana. Por esta razón el Ministerio del Poder Popular para Relaciones Interiores y Justicia (MPPRIJ), promueve realizar el Sistema de Gestión Policial (SIGEPOL), que tiene como objetivo capturar la información generada por los órganos policiales a nivel nacional, que sirva para la toma de decisiones y ayude a controlar el delito en la República Bolivariana de Venezuela. Como parte del Sistema de Gestión Policial se define el Módulo de Denuncias, este subsistema permitirá registrar y mostrar las denuncias de los ciudadanos que concurran a sus dependencias policiales y generar reportes sobre las mismas agrupadas bajo diversos criterios, brindando un mecanismo integrado más rápido y eficiente para almacenar y procesar la información de los distintos sucesos delictivos que ocurren a nivel nacional diariamente. El documento que se presenta a continuación recoge los resultados del trabajo investigativo realizado.

Índice

Introducción

El Artículo 1^{ero} del Decreto Con Fuerza De Ley De Coordinación De Seguridad Ciudadana de la República Bolivariana de Venezuela establece que: "El presente Decreto Ley tiene por objeto regular la coordinación entre los Órganos de Seguridad Ciudadana, sus competencias concurrentes, cooperación recíproca y el establecimiento de parámetros en el ámbito de su ejercicio". Al lograr la coordinación de las actividades que realizan los órganos policiales, intercambiar información entre los mismos y definir parámetros que estandaricen el accionar diario de estos órganos, se posibilitaría una mayor organización del trabajo policial a nivel nacional y tomar acertadas decisiones para enfrentar las situaciones delictivas.

El MPPRIJ de la República Bolivariana de Venezuela, el 6 de noviembre del 2001, crea la Ley de Coordinación de Seguridad Ciudadana en la que establece las bases para el proyecto Sistema Nacional de Registro Delictivo Emergencia y Desastres (SINARDED), con la finalidad de integrar y conectar a todos los órganos policiales en un solo repositorio de información; sin embargo este sistema no fue implementado en su totalidad, por lo que todos sus objetivos no fueron cumplidos y en estos momentos es empleado únicamente en la Dirección de Coordinación Policial. Actualmente las áreas policiales no disponen de un proceso unificado a nivel nacional para la actuación policial, existiendo diferencias en el proceso de levantamiento de denuncias, reseña del ciudadano y planificación de operativos policiales.

El proceso de toma y seguimiento de denuncias no lo realizan todas las policías estadales y municipales, ya que el Organismo calificado que cuenta con los recursos para llevar a cabo este proceso es el CICPC[1]. Algunas policías toman denuncias con el fin de llevar estadísticas, pero no todas lo hacen de la misma manera ni solicitan la misma información, unas llevan registros automatizados y la mayoría solo llevan registros manuales.

El MPPRIJ, atendiendo a su misión institucional de garantizar la seguridad ciudadana, promueve la formulación y puesta en marcha del Sistema de Gestión Policial (SIGEPOL), sistema que pretende unificar y compartir la información relacionada con denuncias, reseñas y operativos policiales que se registran en las dependencias de las policías estadales y municipales, así como intercambiar información con el CICPC y servir de fuente de información al CTAISC[2]. Todo esto con la finalidad de que, además de los datos en línea y compartidos, junto con los análisis que puedan hacerse, ayuden a

[1] Cuerpo de Investigaciones Científicas Penales y Criminalística.
[2] Centro de Tratamiento y Análisis de la Información de Seguridad Ciudadana.

controlar el delito en la República Bolivariana de Venezuela y alcanzar la tranquilidad ciudadana del pueblo en el marco del estricto cumplimiento de las regulaciones legales.

SIGEPOL estará constituido por un conjunto de módulos, los cuales son:

- Módulo de Administración.
- Módulo de Denuncias.
- Módulo de Reseña.
- Módulo de Operativos Policiales.
- Módulo de Dependencias.

El Módulo de Denuncias permite gestionar la información referente a las denuncias realizadas en las dependencias policiales. Entre sus principales funcionalidades se encuentran:

- Registrar las denuncias de los ciudadanos que concurran a sus dependencias policiales.
  - Consultar el álbum fotográfico de ciudadanos reseñados permitiendo asociar a la persona natural denunciada una de las fotos mostradas.
  - Identificar visualmente al denunciante por medio de la consulta a la BD de SAIME[3].
  - Registrar los datos del denunciante, del o los denunciados, de los objetos, de los testigos y víctimas.
- Mostrar y consultar información sobre las denuncias realizadas por otras dependencias policiales.
- Generar reportes sobre las denuncias registradas en la dependencia policial agrupadas bajo diversos criterios.

El Módulo de Denuncias surge para dar solución a las situaciones antes mencionadas, por lo que se plantea como problema científico: ¿Cómo desarrollar una aplicación informática para llevar a cabo el proceso de levantamiento de denuncias en todas las dependencias policiales de la República Bolivariana de Venezuela, de forma tal que la información que se recopila esté unificada y pueda ser consultada con mayor rapidez?

El objeto de investigación lo constituye el proceso de levantamiento de denuncias, siendo el campo de acción la informatización del proceso de levantamiento de denuncias en los órganos policiales de la República Bolivariana de Venezuela.

Para darle solución al problema planteado se define como objetivo general desarrollar una aplicación informática para gestionar el proceso de levantamiento de denuncias como parte del Sistema de Gestión Policial. De acuerdo con este planteamiento se trazaron los siguientes objetivos específicos:

[3] Servicio Autónomo de Identidad, Migración y Extranjería.

- Realizar un análisis del estado actual de los procesos asociados al levantamiento de denuncias.
- Realizar el diseño del Módulo de Denuncias a partir de las funcionalidades ya definidas.
- Desarrollar la implementación del Módulo de Denuncias.

Para cumplir con nuestros objetivos y resolver la situación problémica mencionada, nos planteamos un grupo de tareas de investigación:

1. Analizar las circunstancias actuales de la gestión policial en la República Bolivariana de Venezuela.
2. Valorar la metodología de desarrollo de software, plataforma, lenguaje y el conjunto de herramientas de desarrollo definidas para la elaboración de la aplicación.
3. Identificar y seleccionar los patrones de arquitectura y diseño más apropiados para la elaboración del producto de software.
4. Realizar el diseño del software teniendo en cuenta las funcionalidades antes descritas.
5. Definir la organización del código e implementar los elementos de diseño en términos de componentes de implementación.

Con el cumplimiento de estas tareas se pretende obtener un producto de software que sea configurable de modo que pueda funcionar con un número variable de usuarios y/o áreas de trabajo; escalable para garantizar su flexibilidad ante futuros cambios no previstos, en función de las necesidades del cliente; fácil e intuitivo de utilizar; seguro, confiable y haciendo uso de modernas tecnologías de desarrollo que permita el intercambio de información con otros sistemas existentes.

El presente documento está compuesto por 3 capítulos:

En el Capítulo 1 **"Fundamentación Teórica"** se exponen los fundamentos generales que sirven de soporte teórico en la solución del problema. Se analizan las herramientas y lenguajes de programación, para comprobar que son las idóneas para la implementación del Módulo de Denuncias. Además se plantea la metodología a emplear en el desarrollo del mismo.

En el Capítulo 2 **"Diseño del Sistema"** se construye la solución propuesta, se modelan diagramas de clases que representan las funcionalidades del Módulo de Denuncias aplicando los patrones de arquitectura y diseño seleccionados.

En el Capítulo 3 **"Implementación de la solución"** se representa el diagrama de componentes que detalla la forma en que está estructurado el sistema, reflejando la transformación de los elementos del modelo del diseño en términos de componentes, ficheros que contienen código fuente, así como las dependencias entre ellos.

Capítulo 1 Fundamentación Teórica

1.1. Introducción

El objetivo fundamental de este capítulo es realizar la investigación sobre las actividades policiales; particularmente el proceso de levantamiento de denuncias y exponer los fundamentos generales que sirven de soporte teórico en la solución y concepción del problema. Se abordan aspectos de los distintos tipos de software, especificando el tema de los sistemas de gestión, así como elementos que se deben tener en cuenta para su funcionamiento. Se justifica la selección de la metodología, tecnologías, técnicas y herramientas que se emplean en el desarrollo del módulo.

1.2. Seguridad Ciudadana

Se identifica como seguridad ciudadana a la acción integrada que desarrolla un país, con la colaboración de la ciudadanía, destinada a asegurar su convivencia pacífica y la erradicación de la violencia. Del mismo modo, contribuir a la prevención de la comisión de delitos y faltas. (1)

La seguridad es un estado-sensación que se experimenta para manejar el riesgo y el peligro dentro de los límites que pueden dar sentido a desarrollar determinadas acciones, particularmente aquellas vinculadas a las prevenciones y resguardos.

Entre las acciones que suelen realizarse para tributar a la seguridad ciudadana se encuentran el dictado de leyes de seguridad ciudadana, que regulan los métodos y parámetros de actuación de los Órganos de Seguridad Ciudadana ante una situación; la creación de órganos de seguridad para velar el cumplimiento de las regulaciones establecidas; entre otras.

La seguridad ciudadana está orientada a prevenir males sociales como la violencia, las infracciones y el crimen; acciones dañinas que generan peligros para el apropiado desarrollo de las actividades económico-socio-cultural de toda sociedad humana actual.

La inseguridad ciudadana se ha convertido en uno de los grandes desafíos de las sociedades contemporáneas. El impacto del fenómeno sobre la calidad de la vida de los ciudadanos obliga a los gobiernos nacionales y locales y a los sectores organizados de la sociedad, a diseñar esquemas alternativos a los existentes que, siendo en su cometido de disminuir los niveles de inseguridad, no sacrifiquen el avance de la Democracia y el respeto por los Derechos Humanos y las Garantías Ciudadanas. (2)

1.2.1. Seguridad Ciudadana en Venezuela

La seguridad de los ciudadanos debe ser atendida de acuerdo al mandato establecido en el artículo 55 de la Constitución de la República Bolivariana de Venezuela, el cual establece "Toda persona tiene derecho a la protección del Estado, a través de los órganos de seguridad ciudadana regulados por ley, frente a situaciones que constituyan amenazas, vulnerabilidad o riesgos para la integridad física de las personas, sus propiedades, el disfrute de sus derechos y el cumplimiento de sus deberes".

La cantidad de delitos que se cometen en esta nación de América del Sur ha originado gran incertidumbre, teniendo en cuenta la situación política vivida en estos últimos años, generada por una oposición que busca crear caos para tratar de llegar al poder, lo cual ha influido para que el gobierno tome cartas en este asunto con la decisión y firmeza que se requiere para solventar el problema de la inseguridad. (3)

En la población de la República Bolivariana de Venezuela están presentes problemas como el desempleo y la vivienda; pero uno de los más preocupantes es la inseguridad ciudadana. En ese sentido el MPPRIJ conjunto con los Órganos de Seguridad Ciudadana ha tomado medidas para el combate y prevención del delito, así como asegurar una convivencia pacífica en el país. En este marco es que surge la idea y puesta en marcha del Sistema de Gestión Policial con la finalidad de ayudar a coordinar las acciones de los órganos policiales.

En la actual coyuntura política e institucional del país, el ejercicio del derecho a la seguridad ciudadana de la población se encuentra severa y peligrosamente limitado. El incremento registrado en las cifras de criminalidad en el último año, especialmente en las tasas de homicidios y en el número de robos y secuestros, revela una creciente y peligrosa erosión del monopolio de la violencia legítima por parte del Estado Venezolano, que afecta a toda la población, pero muy especialmente a los más pobres y discriminados.

1.2.2. Marco Legal o Seguridad Ciudadana según la Ley

Para establecer la normativa jurídica que sirve de base para la creación del Sistema de Gestión Policial del MPPRIJ de la República Bolivariana de Venezuela se tomarán como referencia:

- Constitución de la República Bolivariana de Venezuela.

La Constitución de la República Bolivariana de Venezuela, publicada en Gaceta Oficial Extraordinaria No. 5 453 de la República Bolivariana de Venezuela en Caracas, viernes 24 de Marzo de 2000, expresa:

Artículo 9: Cuando resulte inminente el desbordamiento de la capacidad de respuesta del órgano actuante para controlar la situación, debido a su magnitud o complejidad de la misma, asumirá la

responsabilidad de la coordinación y el manejo de ésta, el Órgano de Seguridad Ciudadana que disponga de los medios y la capacidad de respuesta para ello.

Artículo 18. El Consejo de Seguridad Ciudadana tendrá por objeto el estudio, formulación y evaluación de las políticas nacionales en materia de seguridad ciudadana.

Artículo 36: La coordinación nacional de seguridad ciudadana y las coordinaciones regionales de seguridad ciudadana, como administradores de las bases de datos de sus respectivas localidades, procesarán la data e información suministrada por los integrantes del sistema y generarán los reportes de información relacionados con el comportamiento de la acción delictiva, emergencias y situaciones de desastres.

Artículo 322: Establece la organización de los Órganos de Seguridad Ciudadana, como medio para garantizar la protección de los ciudadanos y sus hogares en el disfrute de los derechos fundamentales, incorpora la creación de instituciones e instrumentos legales que permitan abordar integral y eficazmente la problemática de la inseguridad ciudadana.

Los Órganos de Seguridad Ciudadana son:

- La Policía Nacional.
- Las Policías de cada Estado.
- Las Policías de cada Municipio.
- El cuerpo de investigaciones científicas, penales y criminalísticas.
- El cuerpo de bomberos y administración de emergencias de carácter civil.
- La organización de protección civil y administración de desastre.

Corresponde a la Policía Nacional atender las situaciones con implicaciones internacionales, incluyendo delitos con proceso ejecutivo fraccionado entre varios países y con implicaciones que trascienden a más de un estado, entre otras.

Las policías estadales y municipales comparten las mismas funciones, según el ámbito territorial y nivel de complejidad, intensidad de intervención y especialidad de la situación a ser controlada. Deberán actuar de inmediato en la atención temprana del conflicto o situación de que se trate, independientemente de su complejidad, extensión o repercusión, al tiempo que deberán informar y requerir la participación de los cuerpos policiales más próximos en orden ascendente cuando la situación rebase sus posibilidades.

La investigación penal como manifestación de la seguridad ciudadana soporta las acciones, técnicas, medios y procedimientos para investigar la comisión de delitos e identificación de los autores, como actividad de auxilio al Ministerio Público en la investigación penal.

Estos Órganos de Seguridad Ciudadana tienen entre sus deberes comunes organizar y desarrollar sistemas informáticos, comunicacionales, administrativos y de cualquier otra naturaleza que permitan optimizar la coordinación entre los distintos órganos de seguridad ciudadana.

1.3. Gestión Policial en Venezuela

La actividad policial, parte de la administración del Estado, puede concebirse como "la disposición de una fuerza organizada para el mantenimiento del orden público mediante la vigilancia (aspecto preventivo) y la aprehensión de los infractores a los fines de imposición de una sanción (aspecto represivo), sanción a cargo de la propia instancia policial, de otras dependencias administrativas o de una instancia jurisdiccional". (4)

Uno de los objetivos principales para garantizar la seguridad ciudadana en la República Bolivariana de Venezuela es fortalecer las políticas, estrategias, lineamientos y directrices que conlleven a la automatización del control de gestión de las acciones policiales, a fin de proveer a los diferentes entes y/o unidades del MPPRIJ de tecnología de información y comunicación que permitan lograr el cumplimiento de sus tareas sobre una plataforma tecnológica de avanzada. Además de promover y ejecutar las políticas del Estado en materia de investigación del fenómeno delictivo, mantener actualizado el registro de la población penal, llevar las estadísticas del ramo de prisiones, patronatos o presos libertados. (5)

Actualmente en la República Bolivariana de Venezuela, cada Órgano de Seguridad Ciudadana controla y almacena la información referente a los ciudadanos que habitan en su jurisdicción, así como también la información acerca de los posibles hechos delictivos cometidos en su área de acuerdo a sus distintas clasificaciones. Además las dependencias policiales no cuentan con un proceso único de actuación policial ya que los procesos de levantamiento de denuncias, reseña del ciudadano y planificación de operativos policiales no se realizan de igual manera.

Por lo tanto, las limitantes de cada Órgano de Seguridad Ciudadana en el manejo de la información, trae como consecuencia que no exista un repositorio de datos central que permita mantener un control a nivel nacional de los registros delictivos y denuncias que el mismo pudiera generar, dificultando de esta manera la obtención de una visión general de la situación de seguridad ciudadana desde la óptica policial o las situaciones de riesgos en el país, y mermando en gran medida la manera en que se trata o atienden las acciones para la seguridad y prevención de delitos oportunamente, asimismo afecta de manera definitiva el establecimiento de políticas adecuadas para la prevención de delitos y faltas.

1.3.1. Propuesta del Proceso Automatizado de Levantamiento de Denuncias

La propuesta del proceso para el levantamiento de denuncias en la República Bolivariana de Venezuela será de la siguiente manera:

1. El Analista de Denuncia busca y selecciona el caso policial al que se le va a asociar la denuncia. Un caso policial se creará a partir de la ocurrencia de un hecho delictivo o de una falta. Si el caso policial no está aún registrado, puede proceder a registrar este caso por primera vez en su dependencia a fin de crear un expediente para el mismo. Sobre el expediente se podrán registrar en cualquier momento posterior, nuevas denuncias.
2. De las denuncias se almacenan datos básicos como el origen, la fecha, la hora, el número de la denuncia y una descripción del hecho que se denuncia.
3. Para registrar los datos de los entrevistados, denunciados y objetos la aplicación permitirá:
   - Identificar visualmente al ciudadano entrevistado por medio de la consulta a la base de datos de SAIME, habiéndose proporcionado por el Analista de Denuncia los datos básicos referentes a la persona, entre los que podemos mencionar nombres y apellidos y número de cédula. Si la identificación con la base de datos de SAIME no resultara posible por no existir conexión en ese momento, el ciudadano entrevistado se anexará a la denuncia como “No identificado”.
   - Los denunciados pueden clasificarse en personas naturales y jurídicas.
   - Sobre la(s) persona(s) natural(es) mencionar la utilización de un mecanismo para una posible identificación de la(s) misma(s) a través de un álbum de fotos de ciudadanos reseñados que le permitirá al sistema una mayor integración de la información que se registra por otros módulos, lo que facilitará en gran medida las investigaciones que se generen por las denuncias formuladas.
   - De la persona jurídica se toman los datos referentes a la identificación y ubicación de la entidad.
   - Se registran la(s) presunta(s) falta(s) o delitos(s) cometidos por el o los denunciado(s), así como los objetos involucrados en la denuncia.
4. Cuando culmine el registro de la denuncia el sistema genera un reporte con toda la información recopilada referente a la misma, para su posterior impresión.

1.4. Conceptos asociados al dominio del problema

1.4.1. Software

El software es un conjunto de programas de cómputo y procedimientos necesarios para hacer posible la realización de una tarea específica, en contraposición a los componentes físicos del sistema, es decir, es el soporte lógico a todos los componentes intangibles de un ordenador o computadora. Está formado por una serie de instrucciones y datos, que permiten aprovechar todos los recursos que el ordenador tiene, de manera que pueda resolver gran cantidad de problemas. Una computadora en sí, es sólo un conglomerado de componentes electrónicos; el software le da vida al computador, haciendo que sus componentes funcionen de forma ordenada.

El software es un conjunto de instrucciones detalladas que controlan la operación de un sistema computacional. Adopta varias formas en distintos momentos de su ciclo de vida:

- Código fuente: escrito por programadores, contiene el conjunto de instrucciones destinadas a la computadora.
- Código objeto: resultado del uso de un compilador sobre el código fuente. Consiste en una traducción de este último.
- Código ejecutable: resultado de enlazar uno o varios fragmentos de código objeto. El código ejecutable es directamente inteligible por la computadora. (6)

Entre las principales funciones de un software se encuentran:

- Administrar los recursos del ordenador.
- Proporcionar las herramientas para optimizar estos recursos.
- Actuar como intermediario entre el usuario y la información almacenada.

Entre los tipos de software más utilizados se encuentran el software de gestión.

1.4.2. Software de Gestión

La gestión, en el amplio mundo de la informática se entiende por el sistema encargado del control de los repositorios de información, de los grupos de usuarios, y de los procesos de soporte para otros subsistemas.

El software de gestión se encarga de definir y controlar los flujos de trabajo que son utilizados por otros subsistemas, y de la definición de parámetros para el funcionamiento del sistema.

La gestión de la información puede definirse como el conjunto de actividades realizadas con el fin de controlar, almacenar, y posteriormente, recuperar adecuadamente la información producida, recibida o retenida por cualquier organización en el desarrollo de sus actividades. Un sistema de gestión puede ayudar a centrar, organizar y sistematizar los procesos para la gestión y mejora. (7)

1.4.3. Aplicaciones

Aplicación es el término que se utiliza para designar un programa que se ejecuta en la computadora. Para evaluar si una aplicación está realmente bien construida no solo basta con que realice su tarea correctamente, sino también que sea fácil de utilizar por el usuario. Es decir, que el usuario se pueda relacionar con ella de forma rápida y comprensible. Para esto la aplicación dispone de un diseño el cual se llama Interfaz de usuario o Conexión de usuario, actualmente casi todo el esfuerzo de quienes diseñan aplicaciones está orientado a lograr una interfaz lo más amistosa e intuitiva posible. Sin embargo, es difícil establecer categorías genéricas significativas para las aplicaciones del software.

Una aplicación web es un sistema informático que los usuarios utilizan accediendo a un servidor web a través de Internet o de una intranet. Las aplicaciones web son populares debido a la practicidad del navegador web como cliente ligero. Otra razón de su popularidad es la facilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software en miles de potenciales clientes. Es importante mencionar que una página Web puede contener elementos que permiten una comunicación activa entre el usuario y la información lo cual permite que el usuario acceda a ella de modo interactivo, gracias a que la página responderá a cada una de sus acciones. (8)

1.5. Lenguaje Unificado de Modelado (UML)

El Lenguaje Unificado de Modelado (UML) es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software.

Está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Debido a que el UML es un lenguaje, cuenta con reglas para combinar tales elementos, permite la modelación de sistemas con tecnología orientada a objetos. Los diagramas son entes importantes de UML, cuya finalidad es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. Un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementarlo. El modelo gráfico de UML tiene un vocabulario en el que se identifican: elementos, relaciones y diagramas.

1.6. Metodologías de Desarrollo de Software

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo software. Las metodologías de desarrollo de software contribuyen a mejorar la calidad y a realizar un software en el tiempo esperado y con el coste estimado.

Van indicando paso a paso todas las actividades a realizar para lograr el producto informático deseado, proponen qué personas deben participar en el desarrollo de las actividades y qué papel deben de tener. Además detallan la información que se debe producir como resultado de una actividad y la información necesaria para comenzarla. (9)

Una metodología puede seguir uno o varios modelos de ciclo de vida, es decir, el ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto pero no cómo hacerlo. La metodología indica cómo hay que obtener los distintos productos parciales y finales. Algunos de los aspectos positivos de las metodologías son:

- Son iterativas e incrementales.
- Fácil de dividir el sistema en varios subsistemas independientes.
- Se fomenta la reutilización de componentes.

1.6.1. Metodología RUP

Para controlar, planificar, organizar y guiar el desarrollo del Módulo de Denuncias se decidió utilizar como metodología de desarrollo del software RUP (Proceso Unificado de Desarrollo de Software). RUP es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyectos.

RUP divide el proceso de desarrollo en ciclos, teniendo un producto funcional al final de cada ciclo, cada ciclo se divide en fases que finalizan con un hito donde se debe tomar una decisión importante, las cuatro fases que incluye RUP son:

- **Inicio**, el objetivo en esta etapa es determinar la visión del proyecto.
- **Elaboración,** en esta etapa el objetivo es determinar la arquitectura óptima.
- **Construcción,** en esta etapa el objetivo es obtener la capacidad operacional inicial.
- **Transición,** el objetivo es llegar a obtener el release del proyecto.

Vale mencionar que el ciclo de vida que se desarrolla por cada iteración, es llevado bajo dos disciplinas:

Disciplina de Desarrollo

- Modelamiento del Negocio: Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
- Requerimientos: Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.

- Análisis y Diseño: Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.
- Implementación: Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de las capas de la aplicación.
- Pruebas: Busca los defectos a lo largo del ciclo de vida.

Disciplina de Soporte

- Administración de Configuración y Cambios: Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a utilización/actualización concurrente de elementos, control de versiones, etc.
- Administración del proyecto: Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.
- Ambiente: Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.
- Instalación: Produce release (versión terminada y estable de una aplicación informática) del producto y realiza actividades para entregar el software a los usuarios finales.

RUP está basado principalmente en tres características fundamentales:

Dirigido por casos de uso: Un caso de uso es un fragmento de funcionalidad del sistema que expresa necesidades del usuario y produce un resultado de valor para este. Todos los modelos que se obtienen, como resultado de los diferentes flujos de trabajo por los que transita todo el proceso de desarrollo del software, representan la realización de los casos de uso obtenidos cuando se modelan las funcionalidades del sistema.

Centrado en la arquitectura: La arquitectura de un sistema es la organización o estructura de sus partes más relevantes. Muestra una visión del sistema completo, describe los elementos del modelo que son más importantes para su construcción. El modelo de arquitectura se representa a través de vistas en las que se incluyen los diagramas de UML.

Iterativo e Incremental: Para facilitar el trabajo, el proyecto se realiza mediante iteraciones que terminan con un incremento. Cada iteración comprende diferentes disciplinas y de esta resulta una versión de un producto que irá creciendo en cada iteración. (10)

Como RUP es un proceso, en su modelación define como sus principales elementos:

- **Actividades** (“cómo”), Es una tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos.
- **Trabajadores** (“quién”), Definen el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos. Vienen a ser las personas o entes involucrados en cada proceso.
- **Artefactos** (”qué”), Un artefacto puede ser un documento, un modelo, o un elemento de modelo, o sea productos tangibles que son producidos, modificados y usados por las actividades.
- **Flujo de actividades** (“cuándo”), Secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software. (11)

Esta metodología está pensada para proyectos grandes en cuanto a tamaño y duración, al igual que el sistema que se propone; el cual además cuenta con un equipo de desarrollo grande e inestable, pues en su mayoría son estudiantes y profesores que pueden pasar a cumplir otras funciones en cualquier momento, por lo que se necesitará una buena organización y abundante documentación para que el avance del proyecto no se vea afectado y se asegure la continuidad del mismo. RUP no necesita tener al usuario final como parte del equipo de desarrollo, lo cual es un punto de gran importancia teniendo en cuenta que los clientes proceden de otro país. Todas las características mencionadas anteriormente, coinciden con las que propone RUP para que su uso sea realmente eficiente.

1.7. Plataforma de Desarrollo de Software

Una plataforma de desarrollo es el entorno común en el cual se desenvuelve la programación de un grupo definido de aplicaciones.

1.7.1. Plataforma Java

La plataforma Java es un entorno o plataforma de computación creada por la Sun Microsystems[4], capaz de ejecutar aplicaciones desarrolladas con el uso del lenguaje de programación Java y un

[4] Es una empresa informática de Silicon Valley, fabricante de semiconductores y software. Las siglas SUN se derivan de «Stanford University Network», proyecto que se había creado para interconectar en red las bibliotecas de la Universidad de Stanford.

conjunto de herramientas de desarrollo. En este caso, la plataforma no es un hardware específico o un sistema operativo, sino una máquina virtual encargada de la ejecución, y un conjunto de librerías estándares que ofrecen funcionalidades comunes. (12)

Para el desarrollo de la aplicación se seleccionó la Edición Empresarial: J2EE o Java EE (Java Platform, Enterprise Edition).

Una de las principales razones por la que se seleccionó esta plataforma es porque se cuenta con limitaciones legales por parte del cliente: el decreto Ley No. 3.390 de la Gaceta Oficial de la República Bolivariana de Venezuela establece que la Administración Pública Nacional empleará prioritariamente Software Libre Desarrollado con Estándares Abiertos, en sus Sistemas, Proyectos y Servicios Informáticos, permitiendo el uso de otro tipo de software solo en casos de no ser posible la adquisición de Software Libre Bajo Estándares Abiertos, en tal caso, los órganos y entes de la Administración Pública Nacional deberán solicitar ante el Ministerio de Ciencia y Tecnología, la autorización para adoptar otro tipo de soluciones bajo normas y criterios establecidos por ese Ministerio. (13)

J2EE es independiente del sistema operativo, es una plataforma madura y con una amplia gama de proveedores y documentación. Además el usuario no necesitará de requerimientos de hardware ni de software para acceder al sistema, solo se debe equipar con buen hardware el servidor de aplicaciones.

Java es uno de los lenguajes de programación más elaborados y más utilizados para la creación de software de empresa. La evolución de Java, que ha pasado de ser un medio para desarrollar *applets*[5] para ser ejecutados en navegadores, a un modelo de programación capaz de manejar las aplicaciones de una empresa de hoy en día. (14)

Los componentes principales de la plataforma son la Máquina Virtual de Java (JVM) y el lenguaje de programación Java.

Máquina Virtual de Java (JVM)

El término JVM se refiere a la especificación abstracta de una máquina de software para ejecutar programas Java. Es el núcleo de ese lenguaje de programación, por lo que resulta imposible ejecutar cualquier programa Java sin la ejecución de alguna implantación de la JVM, o sea, el código no se ejecuta directamente sobre un procesador físico, sino sobre un procesador virtual Java. Es la encargada de traducir los bytecode (código resultante de la compilación del código fuente) en las instrucciones nativas, permitiendo la portabilidad de las aplicaciones.

[5] Un *applet* es un componente de *aplicación* que se ejecuta en el contexto de otro programa. Los Java applet son pequeños programas hechos en Java, que se transfieren con las páginas web y que el navegador ejecuta en el espacio de la página.

Lenguaje de Programación Java

Este lenguaje nace bajo la filosofía de la Programación Orientada a Objetos (*OOP*), lo que permite lograr una mayor reutilización del código y una forma más eficiente de producir software, es una técnica centrada principalmente en los datos y la manera de llegar a ellos. Fue creado a partir de C++ y diseñado para integrarse sin mucha complejidad a entornos de red, ideal para ambientes bajo filosofía Cliente-Servidor. Es portable, sus programas pueden ser ejecutados en multiplataformas, esto posibilita hacer más liviana la carga de trabajo de un sistema, debido a que puede distribuirse en diferentes equipos de cómputo sin los problemas de incompatibilidad de estructuras de datos, o sea, se pueden utilizar recursos situados en diferentes ubicaciones geográficas, característica que favorece la modularidad y distribución del mismo como sistema.

Por todas las características anteriormente mencionadas, es que se propone el empleo de esta plataforma que integra todo lo necesario para que el desarrollo de la aplicación en cuestión sea favorable, sin costes elevados y ajustado a cronogramas.

1.8. Herramienta Case

Las herramientas CASE (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 de dinero. Las herramientas CASE permiten organizar y manejar la información de un proyecto informático. Permite que los sistemas (especialmente los complejos, en el caso de SIGEPOL), se tornen más flexibles, más comprensibles y además mejorar la comunicación entre los participantes.

Estas herramientas pueden servir de apoyo 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, compilación automática, documentación o detección de errores entre otras. (15)

1.8.1. Visual Paradigm para UML

Visual Paradigm para UML es una herramienta CASE profesional, fácil de utilizar y que soporta el ciclo de vida completo del desarrollo de software; usa al UML como lenguaje de modelado, característica que ayuda a la construcción de aplicaciones con calidad, de manera intuitiva, a menor coste; además de que facilita la comunicación entre los miembros del equipo de desarrollo, al garantizar el uso de un lenguaje estándar común. (16)

Se decidió utilizar esta herramienta CASE para el modelado del diseño de la aplicación, ya que entre sus principales características y facilidades se encuentran:

- Brinda la posibilidad de crear un conjunto bastante amplio de artefactos utilizados con mucha frecuencia durante la confección de algún diagrama en el desarrollo del software. Todos estos, cumpliendo con el Estándar UML 2.0.
- Disponibilidad en múltiples plataformas (multiplataforma): Microsoft Windows (98, 2000, XP, o Vista), Linux, Mac OS X, Solaris o Java. (17)
- Puede ser utilizado en varios idiomas, sus componentes se encuentran relacionados, por lo que se hace muy fácil la creación de cualquier tipo de diagrama, no se requieren grandes conocimientos para su manejo.
- Brinda un número considerable de estereotipos, lo que permite un mayor entendimiento de los diagramas.
- Generación de código e ingeniería inversa: brinda la posibilidad de generar código a partir de los diagramas, para plataformas como Java, así cómo obtener diagramas a partir del código, lo que facilita en gran medida el entendimiento entre analistas e implementadores, a la vez que es posible obtener el código, partiendo de un diagrama elaborado por el analista, también es posible visualizar diagramas con un alto nivel de detalle, a partir del código implementado por un programador.
- Integración con distintos Ambientes de Desarrollo Integrados (IDE): se integra fácilmente con varios IDEs de desarrollo, entre los que se encuentra Eclipse, el que se utilizará para el desarrollo de la aplicación.
- Generación de documentación: brinda la posibilidad de documentar todo el trabajo sin necesidad de utilizar herramientas externas.

1.9. Patrones arquitectónicos, de diseño

En ingeniería del software, un patrón es una solución ya probada y aplicable a un problema que se presenta una y otra vez en el desarrollo de distintas aplicaciones y en distintos contextos. Es importante destacar que un patrón no es en general una solución en forma de código directamente, sino una descripción de cómo resolver el problema y ante qué circunstancias es aplicable.
Entre los principales patrones que se emplearán en el desarrollo del módulo se encuentran:

1.9.1. De arquitectura

Modelo-Vista-Controlador (MVC):

El patrón Modelo-Vista-Controlador separa el modelamiento del dominio, la presentación, y las acciones basadas en las entradas hechas por el usuario en tres clases fundamentales:

Modelo: Administra y maneja el comportamiento y los datos del dominio de aplicación, da respuestas a peticiones de información sobre el estado de la aplicación (normalmente desde la Vista), y responde con instrucciones de cambio de estado (usualmente desde el controlador) a la vista.

Vista: Gestiona lo relacionado con mostrar la información al usuario.

Controlador: El controlador interpreta los eventos que son lanzados por la entrada estándar del usuario (normalmente mouse y teclado), informando de los mismos al modelo y/o la vista para que se ejecuten los cambios apropiadamente. (18)

Tres Capas (Layers):

El modelo tres capas permite desglosar la aplicación en partes lógicas que favorecen a una mejor organización en el desarrollo de la misma. Este patrón es fácilmente acoplable con el MVC. La arquitectura quedaría separada en lo relacionado con la gestión de interfaz (presentación), negocio y acceso a datos, además de las clases del dominio.

1.9.2. De diseño

Básicos: GRASP (General Responsibility Assignment Software Patterns)

Creador:

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento.

Beneficios:

- Se brinda soporte a un bajo acoplamiento, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Es probable que el acoplamiento no aumente, pues la clase creada tiende a ser visible a la clase creador, debido a las asociaciones actuales que llevaron a elegirla como el parámetro adecuado.

Controlador:

Este patrón ofrece una guía para tomar decisiones apropiadas que generalmente se aceptan. Un defecto frecuente al diseñar controladores consiste en asignarles demasiada responsabilidad.

Normalmente un controlador debería delegar a otros objetos el trabajo que ha de realizarse mientras coordina la actividad.

Beneficios:

- Mayor potencial de los componentes reutilizables. Garantiza que la empresa o los procesos de dominio sean manejados por la capa de los objetos del dominio y no por la de la interfaz. Desde el punto de vista técnico, las responsabilidades del controlador podrían cumplirse en un objeto de interfaz, pero esto supone que el código del programa y la lógica relacionada con la realización de los procesos del dominio puro quedarían incrustados en los objetos interfaz o ventana.
- Reflexionar sobre el estado del caso de uso. A veces es necesario asegurarse de que las operaciones del sistema sigan una secuencia legal o poder razonar sobre el estado actual de la actividad y las operaciones en el caso de uso subyacente.

Experto:

Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen. Nótese que el cumplimiento de una responsabilidad requiere a menudo información distribuida en varias clases de objetos. Ello significa que hay muchos expertos "parciales" que colaboraron en la tarea.

El patrón Experto ofrece una analogía con el mundo real.

Beneficios:

- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento.
- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases "sencillas" y más cohesivas que son más fáciles de comprender y de mantener.

Bajo Acoplamiento:

El Bajo Acoplamiento es un principio que se debe recordar durante las decisiones de diseño, es la meta principal que es preciso tener presente siempre. Es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño. El Bajo Acoplamiento estimula asignar una responsabilidad de

modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento.

El Bajo Acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad. No puede considerarse en forma independiente de otros patrones como Experto o Alta Cohesión, sino que más bien ha de incluirse como uno de los principios del diseño que influyen en la decisión de asignar responsabilidades. (19)

Beneficios:

- No se afectan por cambios de otros componentes.
- Fáciles de entender por separado.
- Fáciles de reutilizar.

Alta Cohesión:

Como el patrón Bajo Acoplamiento, también Alta Cohesión es un principio que se debe tener presente en todas las decisiones de diseño, es la meta principal que ha de buscarse en todo momento. Grady Booch señala que se da una alta cohesión funcional cuando los elementos de un componente "colaboran para producir algún comportamiento bien definido".

El patrón Alta Cohesión presenta semejanzas con el mundo real, ya que si alguien asume demasiadas responsabilidades -sobre todo las que debería delegar-, no será eficiente.

Beneficios:

- Mejoran la claridad y la facilidad con que se entiende el diseño.
- Se simplifican el mantenimiento y las mejoras en funcionalidad.
- A menudo se genera un bajo acoplamiento.
- La ventaja de una gran funcionalidad soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.

Polimorfismo:

Como el patrón Experto, el uso del patrón Polimorfismo está acorde al espíritu del patrón “Lo hago yo mismo”. Es un principio fundamental en que se fundan las estrategias globales, o planes de ataque, al diseñar cómo organizar un sistema que se encargue del trabajo. Un diseño basado en la asignación de responsabilidades mediante el polimorfismo puede ser extendido fácilmente para que realicen nuevas variantes.

Beneficios:

- Es fácil agregar las futuras extensiones que requieren las variaciones imprevistas.

Avanzados: GOF (Gang of Four)

Fábrica Abstracta (Abstract Factory):

Es un patrón creacional en la clasificación de los patrones GOF. Proporciona una interfaz para crear familias de objetos sin especificar su clase de forma concreta.

Consecuencias:

- Se potencia el encapsulamiento, puesto que se aísla a los clientes de las implementaciones.
- Se incrementa la flexibilidad del diseño, resultando fácil cambiar de familia de productos.
- Se refuerza la consistencia (alta cohesión y bajo acoplamiento), puesto que se restringe el uso a productos de una sola familia cada vez.

Fachada (Facade):

Simplifica el acceso a un conjunto de clases o interfaces. Proporciona una interfaz unificada de un subsistema sin ocultar sus interfaces. Permite acceder a elementos del sistema y realizar operaciones más complejas con ellos de forma transparente.

Consecuencias:

- Oculta a los clientes parte de la complejidad de los subsistemas.
- Disminuye el acoplamiento entre subsistemas y cliente.
- Disminuye el acoplamiento (las interfaces de por sí reducen el acoplamiento).
- Ocultación de componentes del subsistema.
- Que el cliente pueda utilizar toda la funcionalidad del sistema.

Instancia única o Solitario (Singleton):

El patrón Singleton garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ésta instancia.

Consecuencias:

- Acceso controlado a la única instancia. Puede tener un control estricto sobre cómo y cuándo acceden los clientes a la instancia.
- Espacio de nombres reducido. El patrón Singleton es una mejora sobre las variables globales.
- Permite el refinamiento de operaciones y la representación. Se puede crear una subclase de Singleton.

- Permite un número variable de instancias. El patrón hace que sea fácil cambiar de opinión y permitir más de una instancia de la clase Singleton.
- Más flexible que las operaciones de clase (static en C#, Java).

1.10. Frameworks

En el desarrollo de software, un marco de trabajo (framework en inglés) es una estructura de soporte definida sobre la cual un proyecto puede ser organizado y desarrollado; puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre aplicaciones para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Para desarrollar una aplicación Web es necesario tener en cuenta que esta filosofía está basada en el estilo arquitectónico Cliente-Servidor, por lo que es necesario definir ambos comportamientos. Por ello se propone, para el lado del cliente, la utilización del siguiente marco de trabajo para el trabajo con javascript.

Dojo

Dojo es un marco de trabajo que contiene APIs[6] y controles (widgets) para facilitar el desarrollo de aplicaciones Web que utilicen el modelo AJAX. Con la utilización de Dojo, se gana en facilidad y agilización del proceso de desarrollo de este tipo de aplicaciones; permite reutilizar código y promueve buenas prácticas de desarrollo; resuelve problemas comunes de compatibilidad entre navegadores. Dojo contiene un sistema de empaquetado inteligente, efectos de Interfaz de Usuario (UI), APIs de arrastrar y soltar (*drag and drop APIs*), APIs de controles (*widget APIs*), abstracción de eventos, almacenamiento de APIs en el cliente, e interacción de APIs con AJAX, y la habilidad de degradar cuando AJAX/JavaScript no es completamente soportado en el cliente. Proporciona una gama más amplia de opciones en una sola biblioteca, que otros frameworks del mismo tipo. (20)

Para darle soporte a toda la gestión de información y manejo de páginas dinámicas que llegan a los clientes, se propone para el lado del servidor la utilización del siguiente marco de trabajo, dadas las potencialidades del mismo que se exponen a continuación.

Spring Framework

Marco de Trabajo de código abierto que facilita la creación y desarrollo de aplicaciones para la plataforma Java. Ofrece muchas libertades a los desarrolladores, además de que cuenta con una

[6] Interfaz de Programación de Aplicaciones.

abundante documentación. Se basa y promueve el empleo de las mejores prácticas de programación aceptadas en la actualidad.

Está compuesto por siete módulos principales que colaboran y brindan una amplia gama de funcionalidades fáciles de emplear, que garantizan una arquitectura robusta para cualquier proyecto que tenga su basamento sobre Spring.

- Módulo núcleo (Core Container and Supporting Utilities): Contiene la Fábrica de Clases (BeanFactory) que es el corazón de toda aplicación sobre Spring. *BeanFactory* aplica la Inversión de Controles (Invertion of Control o IoC), específicamente la Inyección de Dependencias (Dependency Injection), para separar la configuración de la aplicación y sus especificaciones de dependencias entre objetos de la lógica de negocio de la aplicación.
- Módulo AOP (Aspect-Orient Programming): Brinda soporte a la Programación Orientada a Aspectos, con la que es posible manejar y solucionar problemáticas como las auditorías y las transacciones de datos con la Base de Dato o entre capas de abstracción dentro de la aplicación, manteniendo la legibilidad y limpieza en el código que define la lógica de negocio de la aplicación en cuestión.
- Módulos ORM y JDBC abstraction and DAO: Brindan soporte a la gama de funcionalidades y problemáticas comunes en el trabajo con Bases de Datos y abstraen al programador de las acciones básicas de intercambio con la Base de Dato permitiendo que sea posible mantener el código de acceso a datos limpio y simple. Además, ORM implementa mecanismos de enganches para lograr compatibilidad e integración con otros frameworks que soportan el mapeo de objetos relacionales como Hibernate e iBatis.
- Módulo MVC (Model – View – Controller): Brinda soporte al desarrollo de aplicaciones web basando el intercambio de los clientes (dígase navegadores web) con el servidor, en la filosofía que sigue el patrón arquitectónico del mismo nombre. Esta facilidad también emplea la *IoC* para lograr separar limpiamente la lógica de controlador de los objetos de negocio, también permite unir declarativamente, parámetros de una petición a objetos de negocio. Además incorpora un mecanismo de validación de formularios muy útil en la comprobación de los datos provenientes del cliente. (21)

Por todas las características ya mencionadas, es posible afirmar que Spring es un framework muy simple, conveniente y flexible, pero al mismo tiempo muy poderoso; facilita la manipulación de objetos, elimina la necesidad de usar distintos y variados tipos de ficheros de configuración, mejora la práctica de programación y suaviza la curva de aprendizaje favorablemente para el desarrollador; es por ello que se propone como una alternativa viable para el desarrollo de la aplicación en cuestión.

iBatis

Es un marco de trabajo especializado en dar soporte a las operaciones y problemáticas más comunes referentes a las interacciones entre el acceso a datos de una aplicación y la base de datos correspondiente. Resulta un componente de software encargado de traducir entre objetos y registros, basado en capas, situado entre la lógica de negocio y la capa de origen de datos. Compuesto de dos paquetes complementarios pero independientes: IBatis Data Access Objects (DAO), que implementa la capa de abstracción (ibatis-dao.jar) e iBatis SQL MAPS, que implementa la capa de persistencia (ibatis-sqlmap.jar). IBatis Data Access Objects (DAO) oculta los detalles de la capa de persistencia y proporciona un API común para todas las aplicaciones iBatis Data Mapper proporciona un modo simple y flexible de mover los datos entre los objetos Java y la base de datos relacional, se tiene toda la potencia de SQL sin una línea de código *JDBC (Java DataBase Connectivity)*. (22)

Para su uso no requiere de grandes esfuerzos de aprendizaje, consta de buena documentación y es un proyecto *Open Source*. Ofrece un sistema de mapeo directo en ficheros XML. Proporciona mecanismos para el acceso a procedimientos almacenados, con la debida consideración por parte del programador de no implementar lógica de negocio en los mismos.

Dada las condiciones en la que se desarrolla la aplicación, así como las características y facilidades antes mencionadas, se propone este framework de persistencia como una opción viable.

1.11. Herramientas de desarrollo

Entre las principales herramientas de desarrollo que se utilizarán en el proyecto se encuentran:

Eclipse*.*

Eclipse es un ambiente integrado de desarrollo de código abierto basada en Java. Es un desarrollo de IBM cuyo código fuente fue puesto a disposición de los usuarios. En sí mismo Eclipse es un marco y un conjunto de servicios para construir un entorno de desarrollo a partir de componentes conectados (plug-in). Eclipse contiene una serie de perspectivas, cada perspectiva proporciona una serie de funcionalidades para el desarrollo de un tipo específico de tarea. La perspectiva Java combina un conjunto de vistas que permiten ver información útil cuando se está escribiendo código fuente, mientras que la perspectiva de depuración contiene vistas que muestran información útil para la depuración de los programas Java. (23)

La característica clave de Eclipse es la extensibilidad. Eclipse es una gran estructura formada por un núcleo y muchos plug-ins que van conformando la funcionalidad final. La forma en que los plug-ins interactúan es mediante interfaces o puntos de extensión; así, las nuevas aportaciones se integran sin dificultad ni conflictos. Está compuesto de tres subproyectos: la Plataforma Eclipse, la Java

Development Tool y el Plug-in Development Environment. El éxito de la Plataforma Eclipse depende de cómo sea capaz de admitir una amplia gama de herramientas de desarrollo para reproducir lo mejor posible las herramientas existentes en la actualidad. (24)

En el desarrollo de la aplicación se incluyen un conjunto de plug-ins que amplían sus funcionalidades como plataforma de desarrollo de aplicaciones sobre la web. Los plug-ins más importantes son los siguientes.

- Web Tool Platform (WTP): Para soportar todas las funciones necesarias para desarrollar aplicaciones web sobre J2EE.
- Spring IDE: Para facilitar el uso sobre los beans y xml definidos por Spring Framework.
- Subeclipse: Para permitir de una forma mucho más ágil y cómoda el desarrollo colaborativo de software en un equipo de desarrolladores.

1.12. Conclusiones

En este capítulo se realizó un análisis de los principales conceptos que deben conocerse para comprender la solución que se propone. Se describió como transcurre el proceso de levantamiento de denuncias. Se estudiaron las metodologías y patrones de arquitectura y diseño a utilizar a lo largo del desarrollo del sistema propuesto, y se fundamentó la elección de las herramientas y tecnologías a utilizar en la elaboración de la aplicación.

Capítulo 2 Diseño del Sistema

2.1. Introducción

En este capítulo se realiza el diseño del Módulo de Denuncias. Se describe la arquitectura del Sistema de Gestión Policial, se presenta el diagrama de paquetes que se propone para estructurar los elementos del diseño. Se muestran además los diagramas de clases de cada uno de estos paquetes divididos por capas, y la descripción de las principales clases del diseño.

2.2. Descripción de la Arquitectura

Establecer las bases sólidas que soporten la construcción, escalabilidad, correcto funcionamiento y mantenimiento de una aplicación informática, requiere un estudio detallado de los patrones seguidos a lo largo del desarrollo del software, las buenas prácticas de programación, el entorno de desarrollo de la misma, las prestaciones que ha de satisfacer y los requerimientos necesarios para su puesta en marcha. La descripción de la arquitectura define los principales aspectos tomados en cuenta en el diseño de un sistema. La arquitectura de SIGEPOL está organizada desde dos enfoques, uno vertical y otro horizontal.

Enfoque horizontal

El sistema está dividido en varios módulos los cuales responden a un conjunto de funcionalidades y casos de uso específicos del cliente. Todos estos módulos interactúan y comparten datos de interés en dependencia de la funcionalidad de cada uno y siguiendo un estricto régimen de seguridad de la información.

Enfoque Vertical

El desarrollo de cada módulo responde a un modelo multicapas. Este modelo está integrado fundamentalmente por Objetos del Dominio (Domain).y las siguientes capas:

- Capa presentación
- Capa de negocio
- Capa acceso a datos
- Capa de persistencia.

2.2.1. Modelo Basado en Capas

Una capa es una agrupación lógica de un conjunto de clases acopladas entre sí. Desde el punto de vista de la Ingeniería del Software, la división de un sistema en capas facilita el diseño modular (cada capa encapsula un aspecto concreto del sistema) y permite la construcción de sistemas débilmente

acoplados (si se minimiza las dependencias entre capas, resultará más fácil sustituir la implementación de una capa sin afectar al resto del sistema). Además, el uso de capas también posibilita la reutilización y el desarrollo en paralelo. El Módulo de Denuncias posee un conjunto de clases agrupadas jerárquicamente en capas, de acuerdo a la funcionalidad que brinda cada una de ellas, ya sea presentación, de negocio, acceso a datos. Cada capa para completar su funcionamiento delega tareas a la capa que se encuentra por debajo en la jerarquía, brindándole los datos necesarios para las mismas o solicitando de ella la información requerida.

A continuación se muestra por medio de un diagrama los aspectos más importantes de la arquitectura del módulo y posteriormente se explican cada una de sus partes.

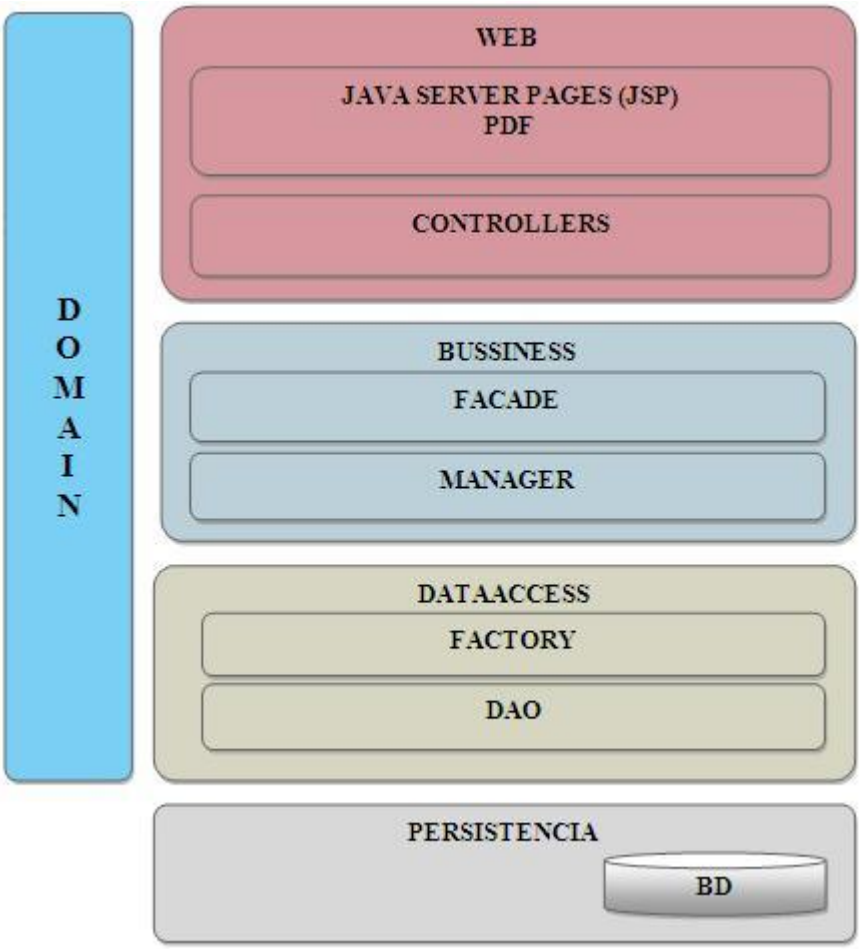


Fig. 2.1. Modelo de la Arquitectura.

Para que exista una mejor comprensión y teniendo en cuenta, que las clases de cada capa para realizar sus responsabilidades se auxilian de las de la capa inferior, se comenzará la explicación por la

capa más baja en la jerarquía, pero para ello se hace necesario conocer la razón por la cual las clases del dominio (DOMAIN), se representa verticalmente en la figura del modelo de la arquitectura.

Dominio (DOMAIN)

El dominio delimita el área del problema que se estudiará, por ejemplo, el proceso levantamiento de denuncias en la República Bolivariana de Venezuela. Está descrito por clases que de manera abstracta representan la información que el sistema debe manipular a través de las capas que posee, dándole en cada una de ellas el uso necesario, ya sea, visualizar, verificar o persistir, garantizando sus propiedades y restricciones.

Dado que las capas se van a encargar de hacer cumplir las restricciones del negocio, el dominio constituye el soporte para la transferencia de datos desde el acceso a datos hasta la capa de presentación y viceversa.

Capa de Persistencia

Esta capa corresponde a los almacenes de datos, contiene todas las estructuras necesarias para poder almacenar la información del dominio que debe persistir y está contenida físicamente en servidor de bases de datos. Es única y común a todos los módulos de SIGEPOL y debe garantizar que el modelo de datos esté al menos, en tercera forma normal.

Capa de acceso a datos

El término de Objeto de Acceso a Datos o en inglés, Data Access Object (DAO), es ampliamente usado en el desarrollo de software. Los DAOs son responsables de hacer persistir los objetos del dominio, constituyen una abstracción entre la capa de persistencia y la capa de negocio, y encapsulan la forma en que ambas se relacionan. Oculta completamente los detalles de implementación de la fuente de datos a sus clientes.

Manteniendo la filosofía de que es mejor programar orientado a interfaces que a clases, en la capa de Acceso a Datos se definen varias interfaces, que son expuestas a la capa de negocio a través de una fábrica, que construye para cada una de sus implementaciones, una instancia única. La fábrica se implementó utilizando el patrón *AbstractFactory.*

Los métodos fundamentales declarados en estas interfaces, se encaminan a guardar, eliminar, buscar y actualizar objetos del dominio en la base de datos.

Capa de Negocio

Las clases del negocio implementan la lógica de negocio de las aplicaciones. En esta capa se separa la lógica de la aplicación de los datos mediante la utilización de entidades del dominio, que carecen de lógica, y son utilizados como objetos que transitan por las diversas capas arquitectónicas.

El Manager agrupa la lógica de negocio de cada una de las entidades que son representativas en el modelo. Sus implementaciones se encargan de controlar el flujo de información con la capa de acceso a datos y el chequeo adecuado de las transacciones del negocio.

La solución de diseño para lograr un bajo acoplamiento está en el empleo del patrón estructural Fachada. En Facade se encuentran las interfaces que brindan a la capa de presentación los métodos necesarios para resolver las funcionalidades especificadas en los casos de uso. Sus implementaciones delegan su labor a los managers especializados en gestionar la información que poseen o necesitan.

Capa de Presentación

La capa de presentación descansa sobre una capa de servicios de negocio. Esto significa que esta capa será fina y no contendrá lógica de negocio, sino simplemente lo concerniente a aspectos de presentación, por ejemplo, el código para manipular las interacciones web.

Spring Web MVC Framework, es una implementación del patrón arquitectónico llamado MVC (Modelo Vista Controlador), el cual se utilizará en esta capa de presentación.

En el módulo MVC que implementa Spring, define al *DispatcherServlet* como el controlador frontal de la aplicación, es el encargado de atender todas las peticiones que se le hagan al servidor. El componente responsable de manipular la petición en el MVC de Spring es un *Controller.*

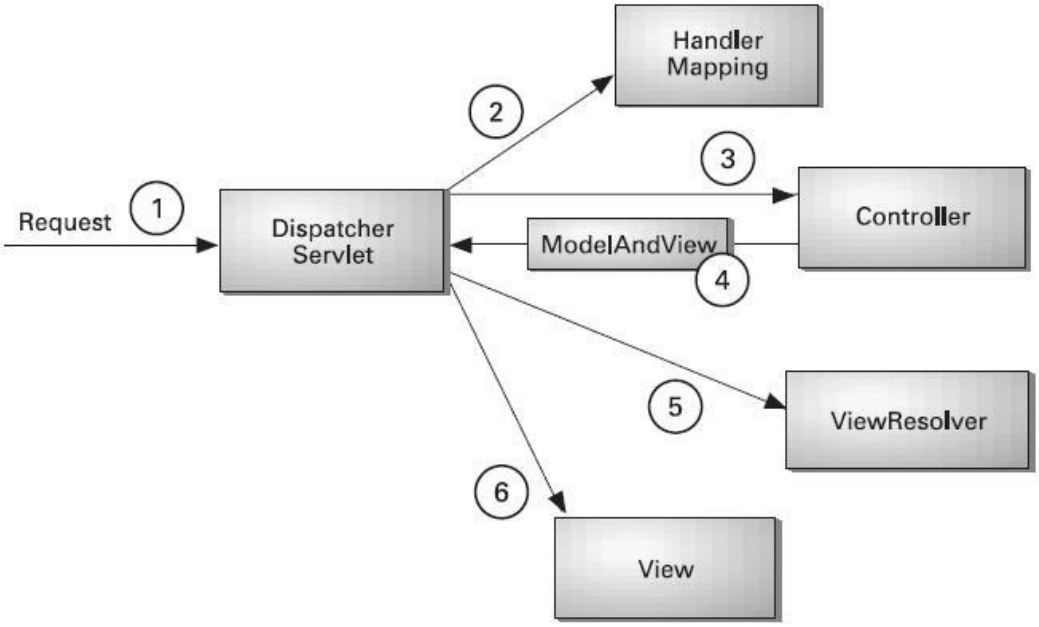


Fig. 2.2. Spring MVC.

Para que el *DispatcherServlet* logre identificar que *Controller* manipulará la petición actual, este se auxilia del o los *HandlerMappings*. Un *HandlerMapping,* usualmente es el encargado de mapear los patrones de URL con el objeto *Controller* asociado. Una vez que el *DispatcherServlet* conoce el objeto *Controller*, le delega la responsabilidad de atender la petición para que realice la lógica de negocio para la que fue diseñado y devuelva un objeto *ModelAndView* al *DispatcherServlet* con el objeto *View* o el nombre lógico del mismo. Si el objeto *ModelAndView* contiene el nombre lógico del *View*, entonces el *DispatcherServlet* consulta al *ViewResolver* para localizar al objeto *View* que visualizará la respuesta (response) al cliente.

En su módulo MVC, Spring implementa una serie de controladores personificados que facilitan la construcción de nuevos controladores.

Dentro de Controllers, se encuentran las clases controladoras de la capa, que son las responsables de procesar las peticiones HTTP, y a partir de las funcionalidades expuestas por las fachadas de la capa de negocio, construir el modelo correspondiente a la nueva vista que se ha de mostrar finalmente al usuario. Los mismos se van a auxiliar de clases validadoras que chequearán que los objetos del dominio posean la información correcta y necesaria para poder realizar el proceso del negocio en que están inmersos.

Modelo: Estos objetos contienen los datos resultantes de la ejecución de la lógica de negocio, los cuales deberían ser mostrados en la respuesta.

Vista: Estos objetos son responsables de mostrar el modelo en la respuesta de la petición. La forma de mostrar el modelo podrá ser de diferentes tipos de vistas, por ejemplo, archivos JSP, HTML, PDF, entre otras. Las vistas no son responsables de modificar los datos o incluso de obtener los datos; estas simplemente sirven para mostrar los datos del modelo que han sido suministrados por un controlador.

2.3. Diagrama General de Clases del Diseño

El diagrama general de clases del diseño representa la interacción entre las clases de las diferentes capas por las que está integrado el Módulo de Denuncias. Se ha utilizado como estereotipo para cada clase el nombre de la capa donde se encuentra ubicada. Las clases que no han sido estereotipadas indican las clases que proporcionan los frameworks utilizados y se representan para mayor comprensión del diagrama.

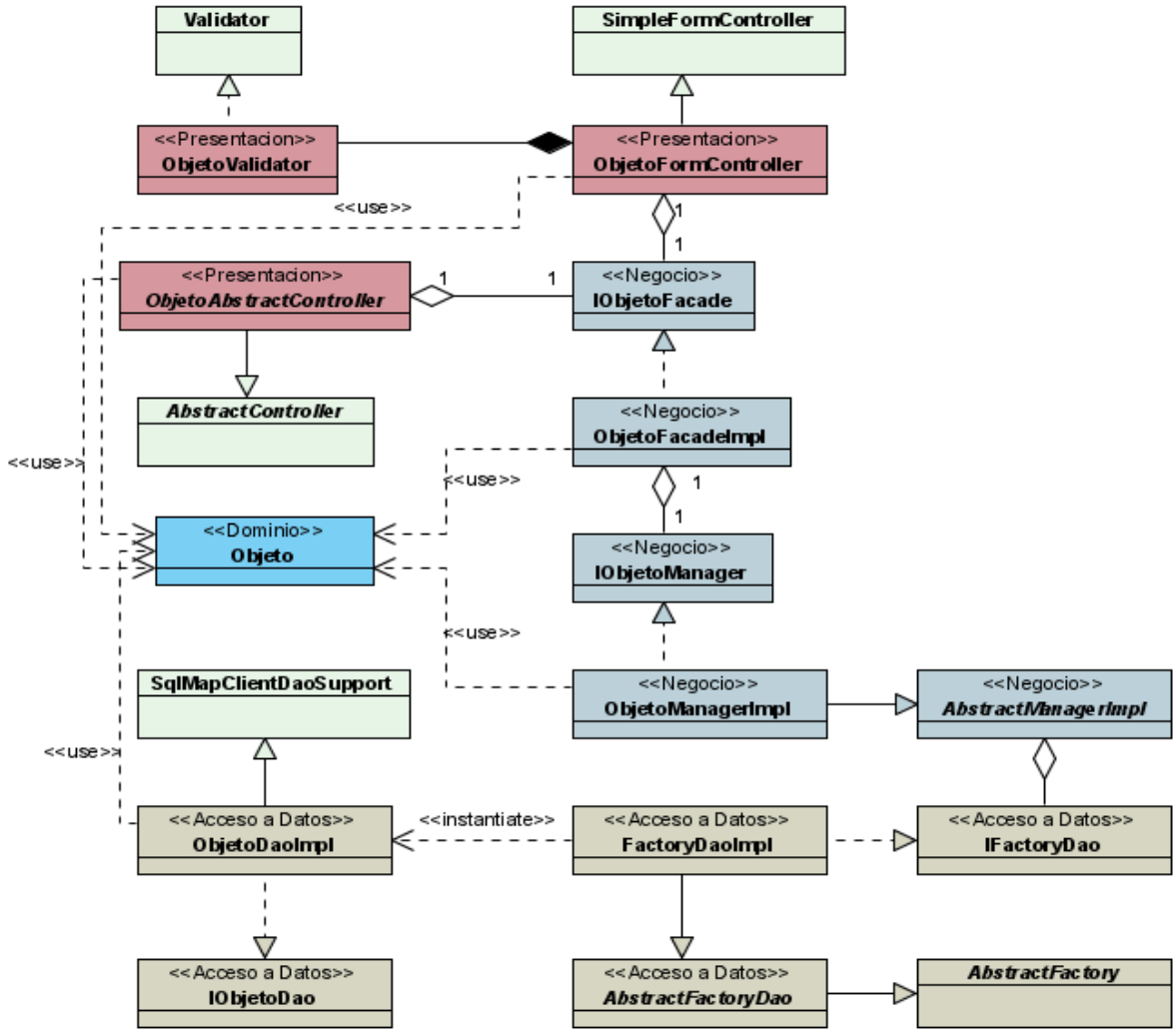


Fig. 2.3. Diagrama General de Clases del Diseño.

2.3.1 Principales Clases Utilizadas de SpringFramework

En el diagrama general de clases del diseño están representadas clases e interfaces del framework utilizado, y es necesario conocer el propósito que tienen, ya que se heredan o implementan sus funcionalidades.

SimpleFormController: Clase implementada en SpringFramework que se especializa en atender las peticiones de un formulario web. Brinda la posibilidad de configurarle la vista del formulario, que se va a utilizar y la vista que debe mostrarse luego de ser enviado (submit). Tiene en cuenta la validación de los datos enviados, para lo cual se le ha de especificar un objeto que implemente la interface Validator del framework. Posee métodos que al ser reimplementados en las clases hijas, facilita enviar al formulario los datos específicos que necesite.

Validator: Es una interface utilizada por SpringFramework para la validación. Puede ser implementada por clases que se especialicen en validar los datos de un tipo de objeto específico.
AbstractController: Clase implementada en el springframework que sirve de base a otros controladores.
SqlMapClientDaoSupport: Clase implementada por SpringFramework, que sirve de base para el trabajo con el framework iBatis, que se utiliza en la capa de acceso a datos.

2.4. Diagrama de Paquetes

Los diagramas de paquetes se usan para reflejar la organización de paquetes y sus elementos, muestran como un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. Un paquete de diseño es una colección de clases, relaciones, realizaciones de casos de usos, diagramas y otros paquetes. Es usado para estructurar el modelo de diseño mediante su división en partes más pequeñas y agrupar elementos relacionados de dicho modelo con propósitos organizacionales.
Para el diseño de este módulo las clases se agruparon por paquetes según su funcionalidad y las entidades que gestionan. Los paquetes definidos son: Gestionar Caso Policial, Gestionar Denuncia, Gestionar Entrevistado, Gestionar Denunciado, Gestionar Objeto, Reportes y Común. Cada paquete contiene cuatro diagramas de clases correspondientes a la Capa de Presentación, Capa de Negocio, Capa de Acceso a Datos y el Dominio. Esta división se hace necesaria para comprender con más precisión cada uno de los elementos que contiene, debido a la complejidad de los diagramas y el gran número de clases contenidos en los mismos.
Se realizará una breve descripción de las principales clases contenidas en el paquete. En la descripción de las clases solo se especificarán los atributos y métodos principales. En cada clase por cada atributo definido existirán los métodos *get* y *set* que aunque no se describan estarán presentes.
Los paquetes especificados no comprenden las funcionalidades definidas para la seguridad del sistema debido a la integración existente con el CAS (Central Autentication Service) de SIGEPOL, que es una aplicación que se encarga de filtrar todas las peticiones antes de ejecutar alguna otra acción. Además controla la apertura y cierre de la sesión para los usuarios que permanezcan inactivos, y garantiza que quien se autentica es un humano.

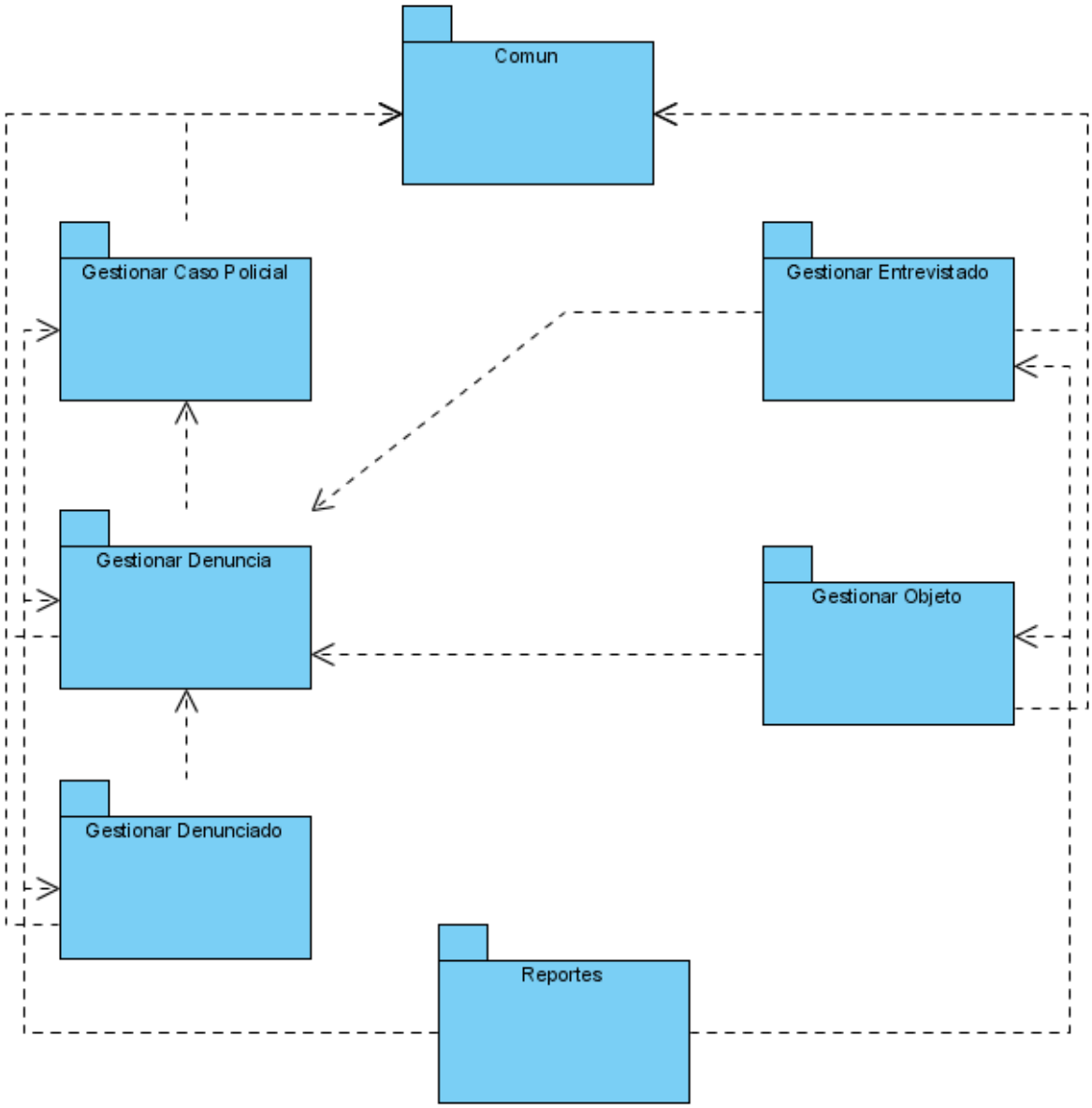


Fig. 2.4. Diagrama de Paquetes del Diseño.

2.5. Paquete Gestionar Caso Policial

El paquete Gestionar Caso Policial permitirá gestionar toda la información acerca de los casos policiales. Un caso policial se creará a partir de la ocurrencia de un hecho delictivo o de una falta y su información se recogerá en un expediente. El expediente asociado a un caso policial será único en cada dependencia. La aplicación permitirá registrar un nuevo caso policial, mostrar el listado de casos

policiales que cumplan con los criterios de búsqueda introducidos por el usuario, así como mostrar los detalles del caso policial que se seleccione.

2.5.1. Diagrama de Clases de la Capa Presentación

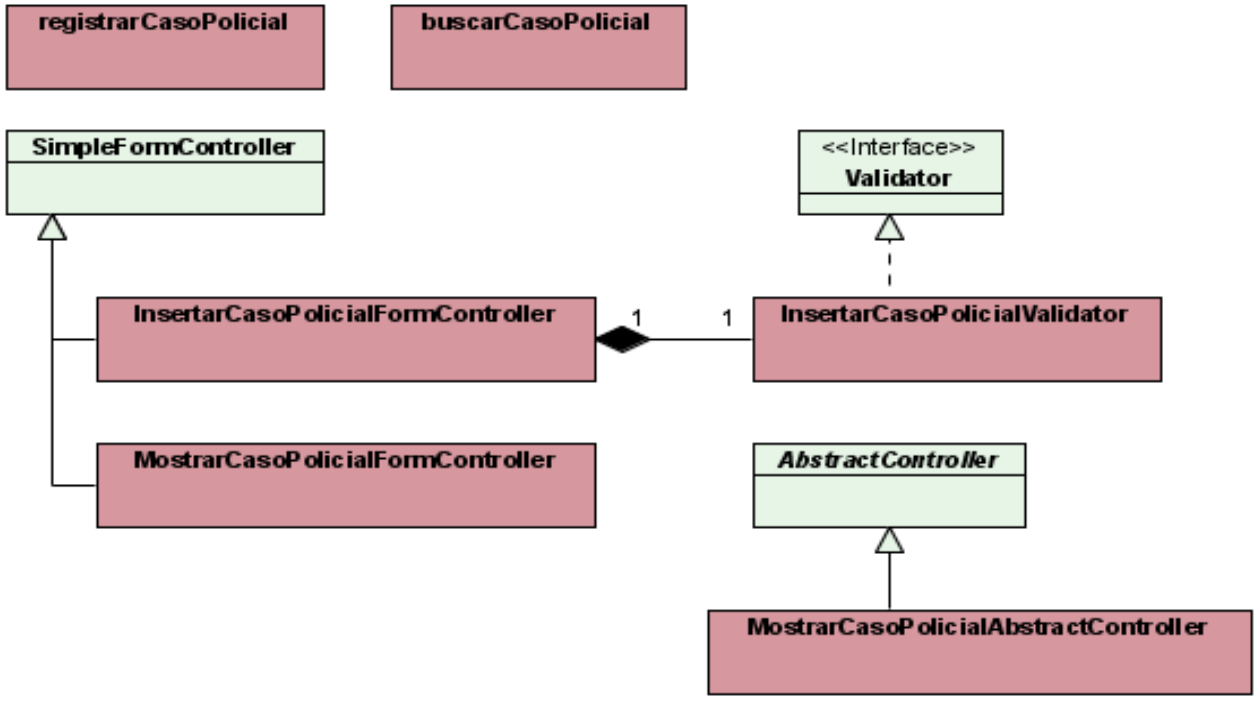


Fig. 2.5. Diagrama de Clases de la Capa Presentación del Paquete Gestionar Caso Policial.

Clase InsertarCasoPolicialFormController

Propósito

Controlador que atiende la petición de registro del Caso Policial.

Atributos

- ICasoPolicialFacade casoPolicialFacade: referencia a la fachada (interface) de caso policial que define las posibles acciones sobre el negocio del mismo.

Métodos

- void initBinder(HttpServletRequest request, ServletRequestDataBinder response): Define las estrategias de conversiones de tipos de datos. Permite registrar los editores para aquellas propiedades del objeto comando (*command*) que no sean String.

- Object formBackingObject(HttpServletRequest request): Retorna por defecto una instancia de la clase comando que tiene configurada, pero se puede sobrescribir para obtener ese objeto con valores de la base de datos.
- Map<String,Object> referenceData(HttpServletRequest request, Object command, Errors errors): Permite suministrar datos que se necesitan en la vista del formulario.
- ModelAndView onSubmit(HttpServletRequest request, HttpServletResponse response, Object command, BindException errors): Si no existen errores (almacenados en *errors*) en la validación del formulario, procesa el mismo y retorna el modelo y/o la vista que se va a mostrar.

2.5.2. Diagrama de Clases de la Capa Negocio

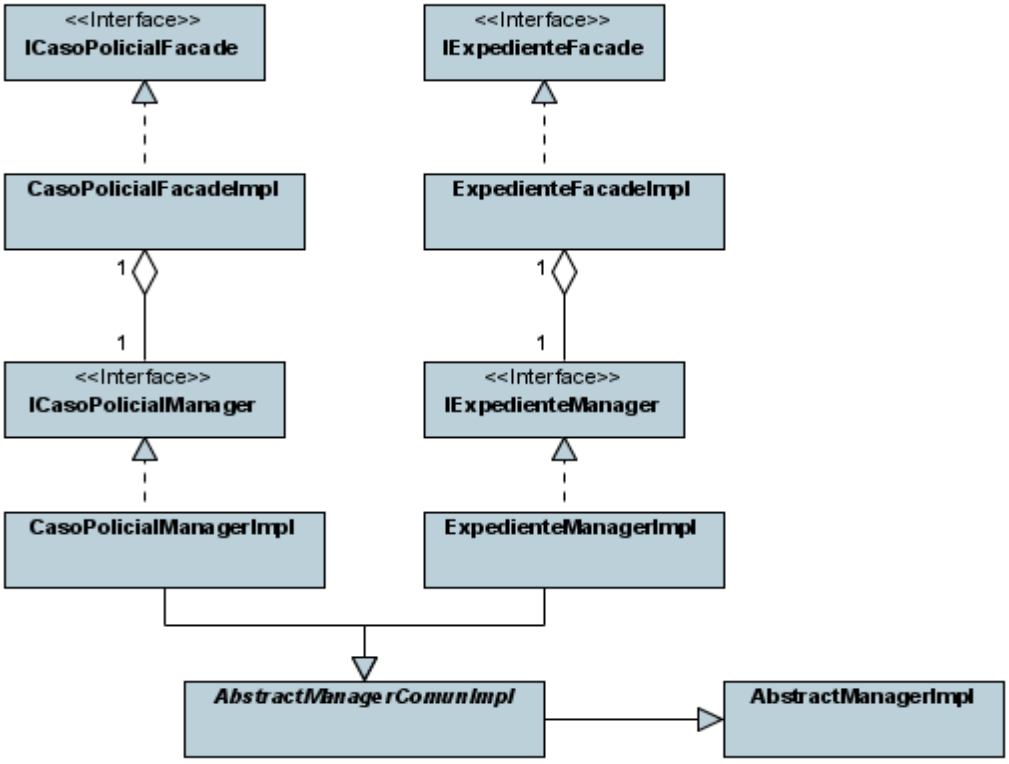


Fig. 2.6. Diagrama de Clases de la Capa Negocio del Paquete Gestionar Caso Policial.

Clase CasoPolicialManagerImpl

Propósito

Clase de servicio con el fin de darle una implementación concreta a la interfaz ICasoPolicialManager que define las posibles acciones sobre el Caso Policial a nivel de lógica de negocio.

Atributos

No aplica

Métodos

- Expediente detallesCasoPolicial(String idCasoPolicial): Dado el identificador de un caso policial devuelve el expediente que contiene todos los detalles del caso policial.
- Map insertar(Expediente expediente): Inserta el expediente que contiene el caso policial y devuelve los datos necesarios para poder asociar las denuncias relacionadas al caso.
- Map<String, Object> listar(Map<String, Object> criterios): Busca el listado de casos policiales dado criterios; devuelve una lista auxiliándose de los métodos de la capa de acceso a datos.

2.5.3. Diagrama de Clases de la Capa Acceso a Datos

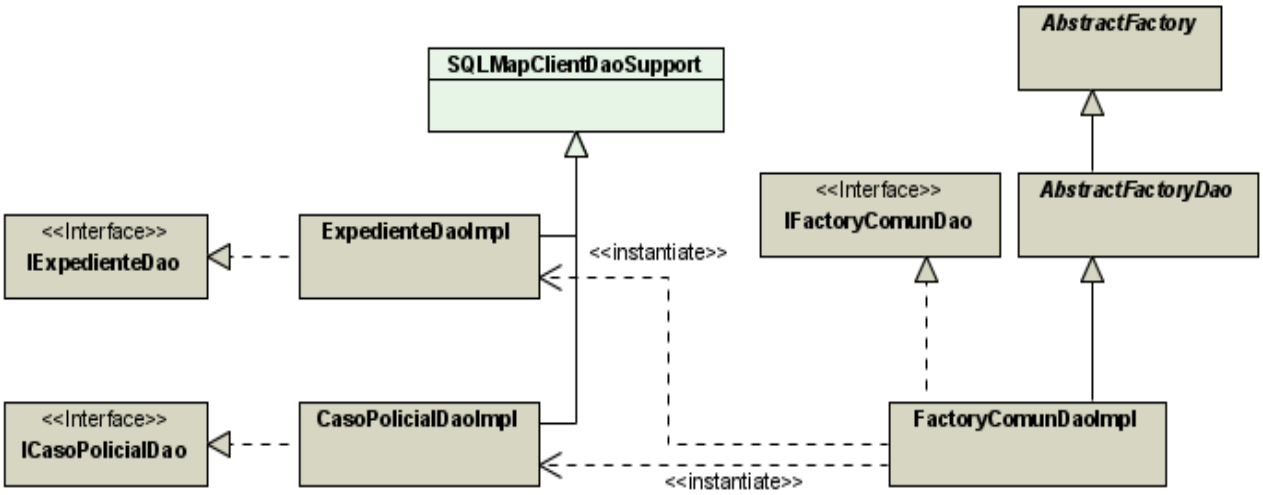


Fig. 2.7. Diagrama de Clases de la Capa Acceso a Datos del Paquete Gestionar Caso Policial.

Clase ExpedienteDaoImpl

Propósito

Implementar los métodos definidos por la interfaz *IExpedienteDao* para el manejo de posibles acciones de un Expediente al nivel de lógica de acceso a datos.

Atributos

No aplica

Métodos

- String existeFalta(String nombreUsuario, int idFalta): Verifica haciendo una consulta a la Base de Datos si en la dependencia a la que pertenece *nombreUsuario*, existe una falta con *idFalta* asociado.

- String insertar(Expediente record): Inserta en la BD un expediente según las especificaciones de la clase de dominio Expediente.
- void actualizarFecha(String idExpediente, Date fechaNueva): Modifica el valor de la fecha de elaboración de un expediente.
- Expediente expedientePorDependencia(String idCasoPolicial, String idDependencia): Devuelve un expediente asociado a un caso policial dado el *idCasoPolicial* y el *idDependencia* referente a la dependencia donde se desea ubicar el expediente.
- void actualizar(Expediente record): Modifica los valores de un expediente, sobrescribiendo los datos del que coincida con el valor del parámetro.

2.5.4. Diagrama de Clases del Dominio

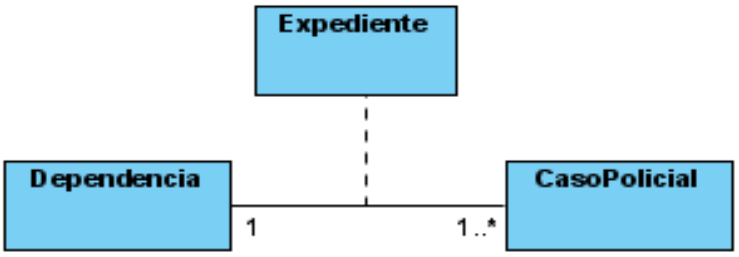


Fig. 2.8. Diagrama de Clases del Dominio del Paquete Gestionar Caso Policial.

Clase CasoPolicial

Propósito

Clase del dominio que define conceptualmente un Caso Policial. Como clase del dominio es accesible desde todas las capas de la aplicación.

Atributos

- Dependencia dependencia: Indica los datos de la dependencia a la cual pertenece el caso policial en cuestión.
- Date fechaInicial: Indica la fecha inicial de ocurrencia del caso policial.
- Date fechaFinal: Indica la fecha final de ocurrencia del caso policial.
- Nomenclador motivo: Indica el motivo del caso policial.
- Nomenclador rangoHora: Indica el rango de hora de ocurrencia del caso policial.
- Dirección dirección: Indica la dirección de ocurrencia del caso policial.

2.6. Paquete Gestionar Denuncia

El paquete Gestionar Denuncia recoge las funcionalidades de la aplicación que van a permitir registrar los datos básicos de una denuncia asociada a un caso policial, mostrar el listado de denuncias que cumplan con los criterios de búsqueda introducidos por el usuario y conocer los detalles de una denuncia seleccionada, así como modificar los datos de la misma.

2.6.1. Diagrama de Clases de la Capa Presentación

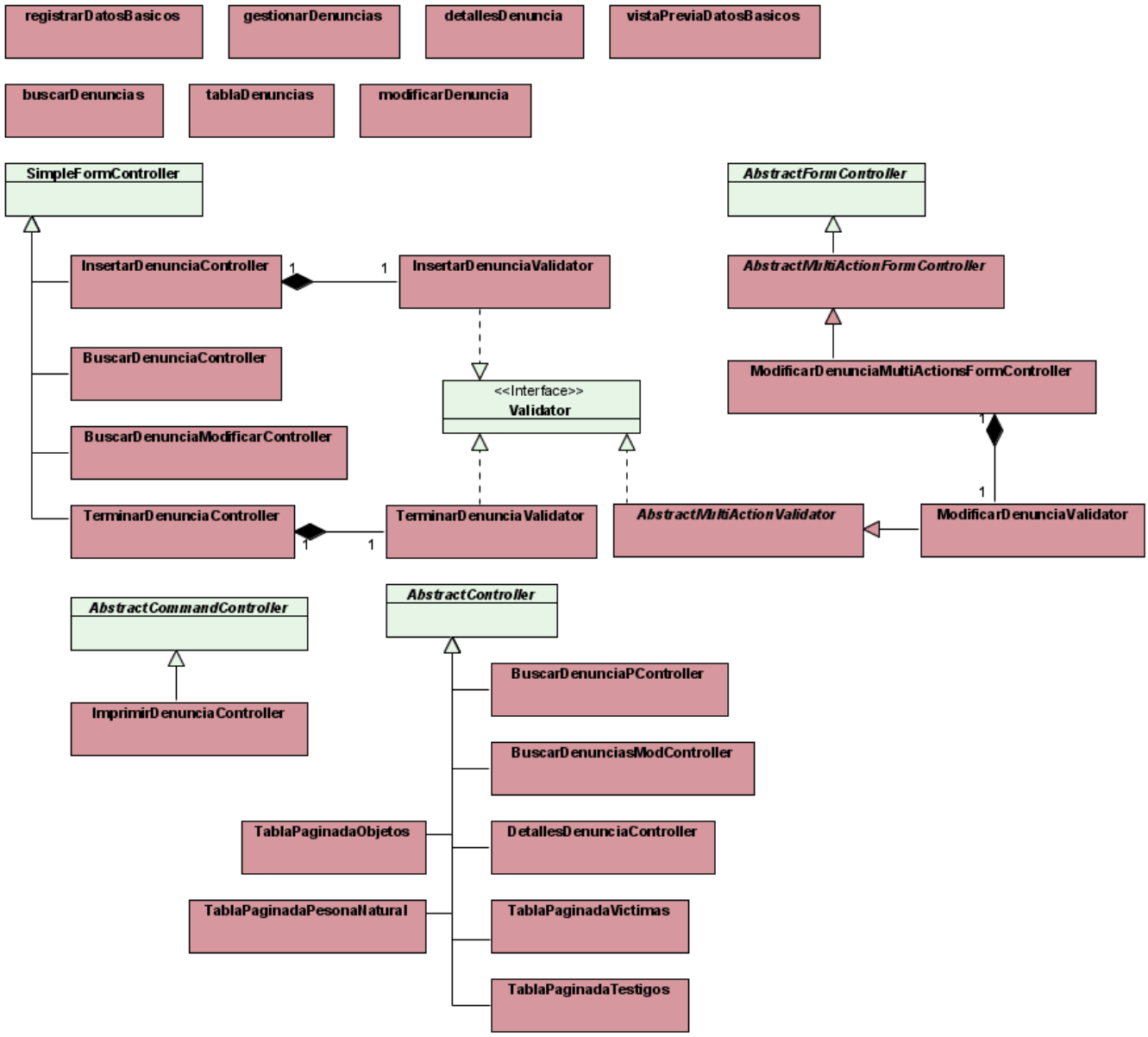


Fig. 2.9. Diagrama de Clases de la Capa Presentación del Paquete Gestionar Denuncia.

Clase TerminarDenunciaController

Propósito

Controlador que atiende la petición de terminar el proceso de levantamiento de denuncia.

Atributos

- IDenunciaFacade denunciaFacade: Referencia a la fachada (interface) de denuncia que define las posibles acciones sobre la misma.
- IEntrevistadoFacade entrevistadoFacade: Referencia a la fachada (interface) de entrevistado que define las posibles acciones sobre la misma.
- IPersonaNaturalFacade personaNaturalFacade: Referencia a la fachada (interface) de persona natural que define las posibles acciones sobre la misma.
- IPersonaJuridicaFacade personaJuridicaFacade: Referencia a la fachada (interface) de persona jurídica que define las posibles acciones sobre la misma.
- IObjetoFacade objetoFacade: Referencia a la fachada (interface) de objeto que define las posibles acciones sobre la misma.

Métodos

- Map<String, Object> onBind(HttpServletRequest request, Object command, BindException errors): Método auxiliar para la unión de parámetros externos a la clase (command).
- private void limpiarFromulario(): Método para inicializar todas las listas que se encuentran en la memoria del servidor.
- Map<String,Object> referenceData(HttpServletRequest request, Object command, Errors errors): Permite suministrar datos que se necesitan en la vista del formulario.
- ModelAndView onSubmit(HttpServletRequest request, HttpServletResponse response, Object command, BindException errors): Si no existen errores (almacenados en *errors*) en la validación del formulario, procesa el mismo y retorna el modelo y/o la vista que se va a mostrar.

Clase TerminarDenunciaValidator

Propósito

Clase que contiene la lógica de validación para los datos básicos de la denuncia.

Atributos

No aplica

Métodos

- boolean supports(Class arg0): Recibe como parámetro la clase cuyos atributos va a validar en el método *validate*, si estos no coinciden lanza una excepción.
- void validate(Object command, Errors errors): Contiene la lógica de validación. Guarda los errores en *errors* para que el *DispatcherServlet* se encargue de redireccionar la petición a la vista inicial o al controlador asociado a la petición.

2.6.2. Diagrama de Clases de la Capa Negocio

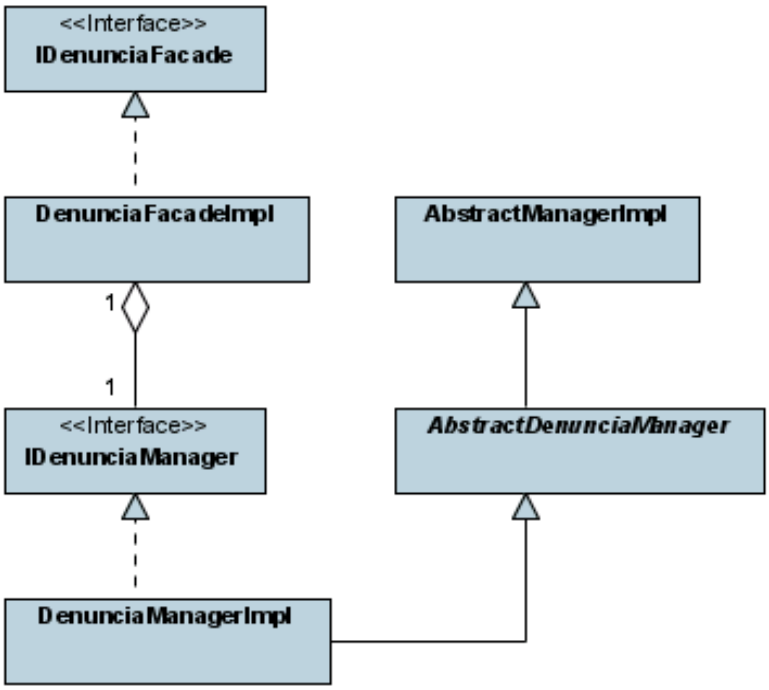


Fig. 2.10. Diagrama de Clases de la Capa Negocio del Paquete Gestionar Denuncia.

Clase DenunciaManagerImpl

Propósito

Clase de servicio con el fin de darle una implementación concreta a la interfaz *IDenunciaManager* que define las posibles acciones sobre la Denuncia a nivel de lógica de negocio.

Atributos

- IDelitoFaltaManager delitoFaltaManager: Instancia de la clase DelitoFaltaManagerImpl para acceder a las acciones que se realizan sobre la clase DelitoFalta.

Métodos

- void insertarDenuncia(Denuncia denuncia): Inserta los datos de una denuncia.
- void modificarDenuncia(Denuncia denuncia): Modifica los datos de una denuncia.

- void obtenerDetallesDenuncia(String idDenuncia): Obtiene los datos referentes a una denuncia a partir de su identificador en la Base de Datos.
- Map<String, Object> obtenerTestigos(String idDenuncia, int registroInicial, int cantidadRegistros): Devuelve la lista de testigos asociados a una denuncia a partir del identificador de la denuncia, la posición del registro inicial y la cantidad de registros.
- Map<String, Object> obtenerVictimas(String idDenuncia, int registroInicial, int cantidadRegistros): Devuelve la lista de víctimas asociadas a una denuncia a partir del identificador de la denuncia, la posición del registro inicial y la cantidad de registros.
- Map obtenerDenunciasPor(Map<String, Object> criterios): Devuelve la lista de denuncias filtradas por los criterios de búsqueda especificados como parámetro.
- List<DelitoFalta> obtenerMotivosDenuncia(String idDenuncia): Devuelve la lista de motivos de la denuncia especificada como parámetro.
- String obtenerCodigoDenuncia(String idDenuncia): Devuelve el código generado por la base de datos a partir de la denuncia especificada como parámetro.
- List<Otro> obtenerObjetos(String idDenuncia): Devuelve la lista de objetos asociados a la denuncia especificada como parámetro.

2.6.3. Diagrama de Clases de la Capa Acceso a Datos

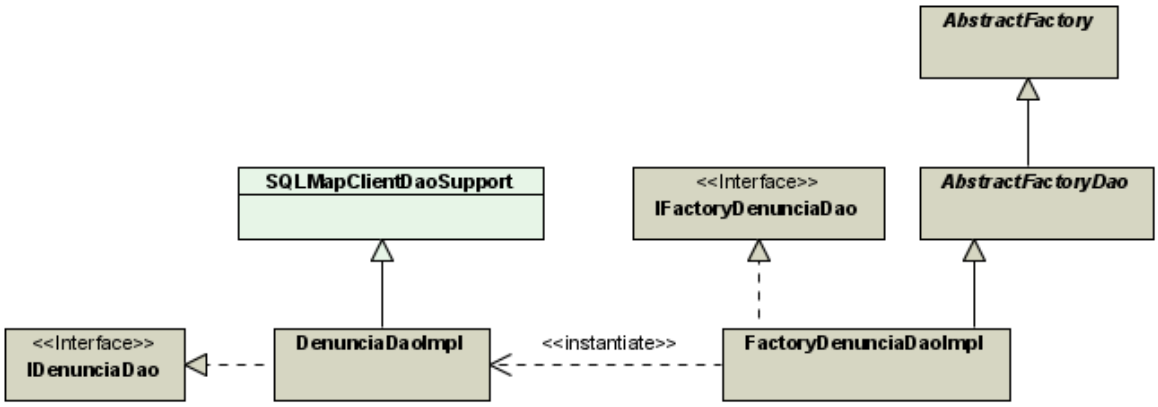


Fig. 2.11. Diagrama de Clases de la Capa Acceso a Datos del Paquete Gestionar Denuncia.

Clase DenunciaDaoImpl

Propósito

Implementar los métodos definidos por la interfaz *IDenunciaDao* para el manejo de posibles acciones de la Denuncia al nivel de lógica de acceso a datos.

Atributos

No aplica

Métodos

- boolean existeDenuncia(String idExpediente, String numeroDenuncia): Verifica realizando una consulta a la Base de Datos, si en el expediente especificado existe una denuncia con el número de Denuncia dado como parámetro.
- String insertar(Denuncia record): Inserta en la BD los datos básicos de la denuncia.
- void asociarDelitoFaltaDenuncia(String idDenuncia, int idDelFalta): Asocia el motivo especificado como parámetro a la denuncia.
- void actualizar(Denuncia record): Modifica en la BD los datos básicos de la denuncia.
- List<DelitoFalta> obtenerMotivosPorDenuncia(String idDenuncia): Devuelve la lista de motivos asociados a la denuncia especificada como parámetro.
- Map<String, Object> obtenerDenunciasPaginadas(Map<String, Object> criterios): Devuelve la lista de denuncias paginadas a través de los criterios especificados.
- Denuncia obtenerDatosDenunciaPorId(String idDenuncia): Obtiene los datos de la denuncia a través del identificador de la denuncia especificado como parámetro.
- PersonaJuridica obtenerEmpresaDenunciada(String idDenuncia): Obtiene la persona jurídica asociada a la denuncia especificada como parámetro.
- Denuncia seleccionarPorId(String idEntidad): Obtiene los datos de la denuncia por el identificador especificado como parámetro.
- Entrevistado obtenerDenunciante(String idDenuncia): Obtiene el denunciante asociado a la denuncia especificada como parámetro.
- Map obtenerTestigos(String idDenuncia, int registroInicial, int cantidadRegistros): Devuelve la lista de testigos asociados a la denuncia especificada como parámetro a través de una consulta paginada.
- Map<String, Object> obtenerVictimas(String idDenuncia, int registroInicial, int cantidadRegistros): Devuelve la lista de testigos asociados a la denuncia especificada como parámetro a través de una consulta paginada.
- void insertarComentario(String idDenuncia, String comentario, int idTipoModificacion): Inserta en la BD la justificación por la cual fue modificada la denuncia.
- List<Entrevistado> obtenerTestigos(String idDenuncia): Devuelve la lista de testigos asociados a la denuncia especificada como parámetro.

- List<Entrevistado> obtenerVictimas(String idDenuncia) : Devuelve la lista de víctimas asociados a la denuncia especificada como parámetro.
- void asociarDenunciaObjeto(String idDenuncia, String idObjeto): Asocia a la denuncia el objeto especificado como parámetro.
- String obtenerCodigoDenuncia(String idDenuncia): Devuelve el código generado en la BD a través de la denuncia especificada como parámetro.
- List<Otro> obtenerObjetos(String idDenuncia): Devuelve la lista de objetos asociados a la denuncia especificada como parámetro.

2.6.4. Diagrama de Clases del Dominio

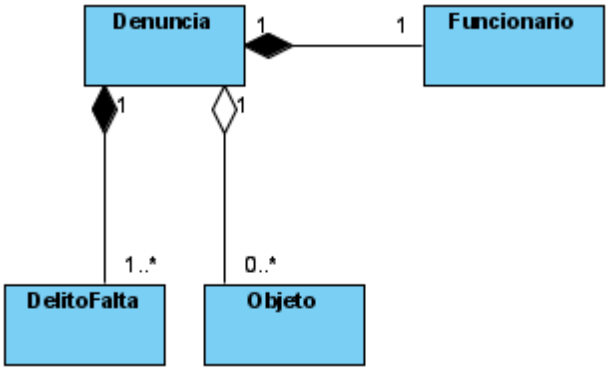


Fig. 2.12. Diagrama de Clases del Dominio del Paquete Gestionar Denuncia.

Clase Denuncia

Propósito

Clase de dominio que representa conceptualmente a una Denuncia.

Atributos

- String idDenuncia: Identificador de la denuncia.
- String numeroDenuncia: Número de la denuncia en la dependencia.
- Date fechaElaboracion: Fecha de registro de la denuncia en la dependencia policial.
- String descripcion: Breve descripción de lo ocurrido en el hecho.
- Funcionario funcionario: Funcionario que toma la denuncia.
- Nomenclador origenDenuncia: Representa los valores nomenclados de los orígenes de la denuncia.
- Expediente expediente: Expediente al cual está asociado la denuncia.

- Boolean esAnonima: Representa de forma lógica si la denuncia en cuestión es anónima o no.
- String causaModificación: Descripción de la causa por la cual fue necesario modificar la denuncia.
- Nomenclador tipoCausaModificación: Representa los valores nomenclados de las causas por las que se modificó la denuncia.
- List<Entrevistado> entrevistados: Entrevistados asociados a la denuncia.
- List<PersonaNatural> personasNaturales: Personas naturales asociados a la denuncia.
- List<Objeto> objetos: Objetos asociados a la denuncia.
- List<DelitoFalta> delitos: Delitos o faltas que se están denunciando.
- PersonaJuridica personaJuridica: Persona jurídica asociada a la denuncia.

2.7. Paquete Gestionar Entrevistado

El paquete Gestionar Entrevistado recoge todas las funcionalidades que le permitirán a la aplicación asociarle a la Denuncia los distintos tipos de Entrevistados que pueden ser: Denunciante, Denunciante-Víctima, Víctima y Testigo. Para lograr una veracidad superior en los datos registrados la aplicación cuenta con un mecanismo que permite identificar al Entrevistado a través de un Servicio Web solicitado a SAIME el cual posibilita llenar de forma automática los datos referentes a la foto del Entrevistado, su(s) nombre(s) y apellido(s), tipo y número de cédula, además se registran la(s) dirección(es) y teléfono(s), profesión u oficio, sexo, nacionalidad, estado civil, grado de instrucción así como los datos referentes al acta de entrevista en la que se recogen datos como el funcionario que toma la declaración, la fecha de confección del acta y la declaración escrita del Entrevistado. Este paquete también contiene funcionalidades para modificar y consultar los datos de los entrevistados ya registrados en la aplicación.

2.7.1. Diagrama de Clases de la Capa Presentación

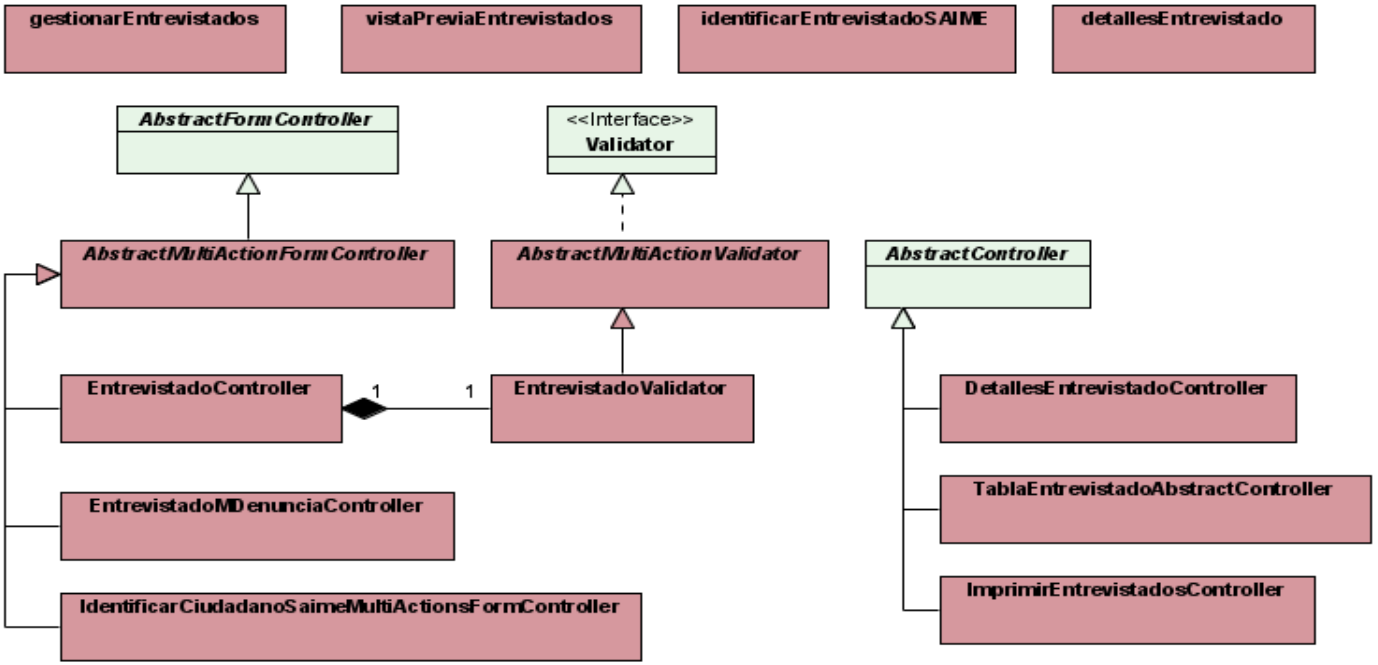


Fig. 2.13. Diagrama de Clases de la Capa Presentación del Paquete Gestionar Entrevistado.

Clase EntrevistadoController

Propósito

Controlador multiacciones que permite insertar, modificar y eliminar los datos de un Entrevistado.

Atributos

- IEntrevistadoFacade entrevistadoFacade: Referencia a la fachada (interface) de Entrevistado que define las posibles acciones sobre la misma.
- IDireccionFacade direccionFacade: Referencia a la fachada (interface) de Dirección que define las posibles acciones sobre la misma.
- ITelefonoFacade telefonoFacade: Referencia a la fachada (interface) de Teléfono que define las posibles acciones sobre la misma.
- Foto foto = new Foto(): Representa la foto del Entrevistado.
- IDenunciaFacade denunciaFacade: Referencia a la fachada (interface) de Denuncia que define las posibles acciones sobre la misma.

Métodos

- Map<String, Object> onBind(HttpServletRequest request, Object command, BindException errors): Método auxiliar para la unión de parámetros externos a la clase (command).
- private void limpiarFromulario(): Método para inicializar todas las listas que se encuentran en la memoria del servidor.
- Map<String,Object> referenceData(HttpServletRequest request, Object command, Errors errors): Permite suministrar datos que se necesitan en la vista del formulario.
- void insertarEntrevistado(HttpServletRequest request, HttpServletResponse response, Object form, BindException errors): Si no existen errores (almacenados en *errors*) en la validación del formulario, inserta el Entrevistado en la lista de entrevistados de la denuncia.
- void modificarEntrevistado(HttpServletRequest request, HttpServletResponse response, Object form, BindException errors): Si no existen errores (almacenados en *errors*) en la validación del formulario, modifica el Entrevistado en la lista de entrevistados de la denuncia.
- void eliminarEntrevistado(HttpServletRequest request, HttpServletResponse response, Object form, BindException errors): Si no existen errores (almacenados en *errors*) en la validación del formulario, elimina el Entrevistado de la lista de entrevistados de la denuncia.

- void identificarEntrevistado(HttpServletRequest request, HttpServletResponse response, Object form, BindException errors): Identifica al Entrevistado a través de SAIME llenándole a la clase Entrevistado los campos: tipo, número de cédula, nombre(s) y apellidos y la foto del mismo.

2.7.2. Diagrama de Clases de la Capa Negocio

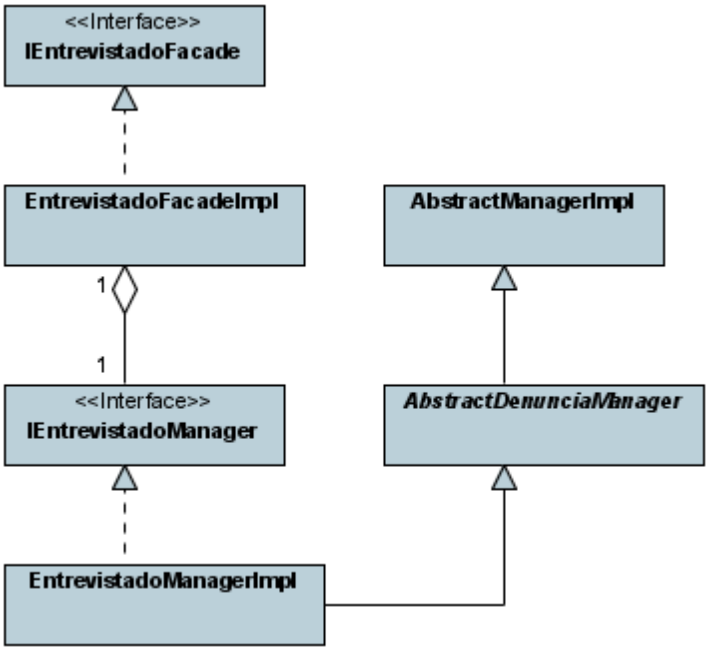


Fig. 2.14. Diagrama de Clases de la Capa Negocio del Paquete Gestionar Entrevistado.

Clase EntrevistadoManagerImpl

Propósito

Clase de servicio con el fin de darle una implementación concreta a la interfaz *IEntrevistadoManager* que define las posibles acciones sobre el Entrevistado a nivel de lógica de negocio.

Atributos

- List<Entrevistado> entrevistados: Lista de entrevistados que aún no ha sido persistente.

Métodos

- void insertarEntrevistado(Entrevistado entrevistado): Inserta en la BD los datos referentes a un Entrevistado especificado como parámetro.
- Map<String, Object> detallesEntrevistado(String idEntrevistado): Devuelve los datos del Entrevistado a partir del identificador especificado como parámetro.

- List<Entrevistado> seleccionarEntrevistados(String idDenuncia): Devuelve la lista de Entrevistados asociados a la denuncia especificada como parámetro.
- Map<String, Object> seleccionarEntrevistadoPaginado(String idDenuncia, int posInicial, int cantidad): Devuelve la lista de Entrevistados asociados a la denuncia especificada como parámetro. Esta consulta se hace de forma paginada.
- void insertarEntrevistadoMemoria(Entrevistado entrevistado): Inserta un Entrevistado en la lista de entrevistados que están en memoria.
- void limpiarListaEntrevistados(): Elimina todos los elementos de la lista de Entrevistados que está en memoria.
- Entrevistado obtenerDatosEntrevistado(String idEntrevistado): Obtiene los detalles del Entrevistado dado su identificador especificado como parámetro.
- List<Entrevistado> obtenerEntrevistadosMemoria(): Devuelve la lista de Entrevistados almacenados en memoria.
- Map<String, Object> obtenerEntrevistadosPaginados(int inicio, int cantidad): Devuelve la lista de Entrevistados que están en memoria de forma paginada.
- boolean existeDenunciante(): Verifica si existe en la lista de Entrevistados al menos un Denunciante.
- void asignarEntrevistados(List<Entrevistado> entrevistados): Asigna la lista de Entrevistados especificada como parámetro a la lista en memoria.

2.7.3. Diagrama de Clases de la Capa Acceso a Datos

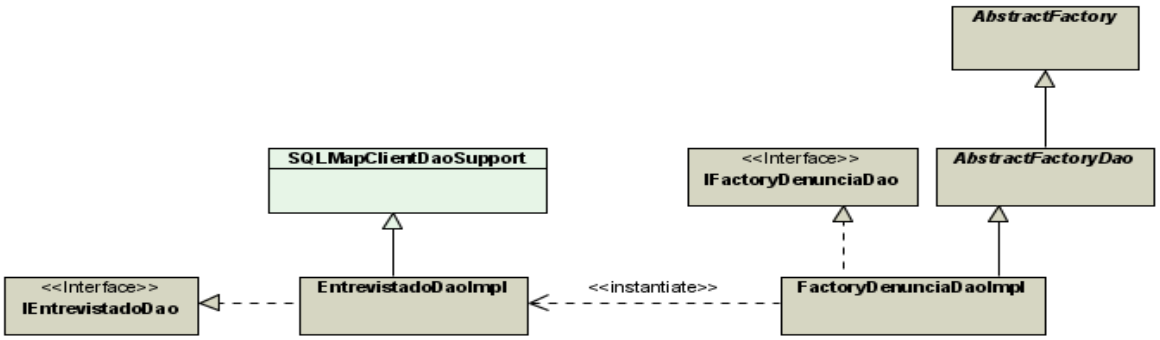


Fig. 2.15. Diagrama de Clases de la Capa Acceso a Datos del Paquete Gestionar Entrevistado.

Clase EntrevistadoDaoImpl

Propósito

Implementar los métodos definidos por la interfaz IEntrevistadoDao para el manejo de posibles acciones del Entrevistado al nivel de lógica de acceso a datos.

Atributos

No aplica

Métodos

- void actualizarActa(Acta acta): Actualiza los datos del acta de entrevista especificada como parámetro.
- void insertarActaEntrevista(Acta acta): Inserta en la BD los datos del acta especificados como parámetro.
- Entrevistado seleccionarPorId(String idEntrevistado): Obtiene los datos del entrevistado especificado como parámetro.
- String insertarEntrevistado(Entrevistado entrevistado, String nombre): Inserta los datos del Entrevistado especificado como parámetro señalándole el tipo de entrevistado almacenado en el parámetro *nombre.*
- void asignarDireccionEntrevistado(String idEntrevistado, String idDireccion, int idTipoDireccion): Asigna una dirección al Entrevistado especificados como parámetros.
- void asignarTelefonoEntrevistado(String idEntrevistado, String idTelefono): Asigna un teléfono al Entrevistado especificados como parámetros.
- List<Direccion> obtenerDireccionesEntrevistado(String idEntrevistado): Devuelve la lista de direcciones de un entrevistado especificado como parámetro.
- List<Telefono> obtenerTelefonosEntrevistado(String idEntrevistado): Devuelve la lista de teléfonos de un Entrevistado especificado como parámetro.
- boolean existeDenunciante(String idDenuncia): Verifica en la BD si la denuncia especificada como parámetro tiene denunciante.
- void actualizar(Entrevistado entrevistado): Actualiza los datos del Entrevistado especificado como parámetro.

- void eliminarDireccionesEntrevistado(String idEntrevistado): Elimina todas las direcciones del Entrevistado especificado como parámetro.
- void eliminarTelefonosEntrevistado(String idEntrevistado): Elimina todos los teléfonos del Entrevistado especificado como parámetro.
- String insertar(Entrevistado record): Inserta el Entrevistado especificado como parámetro.

2.7.4. Diagrama de Clases del Dominio

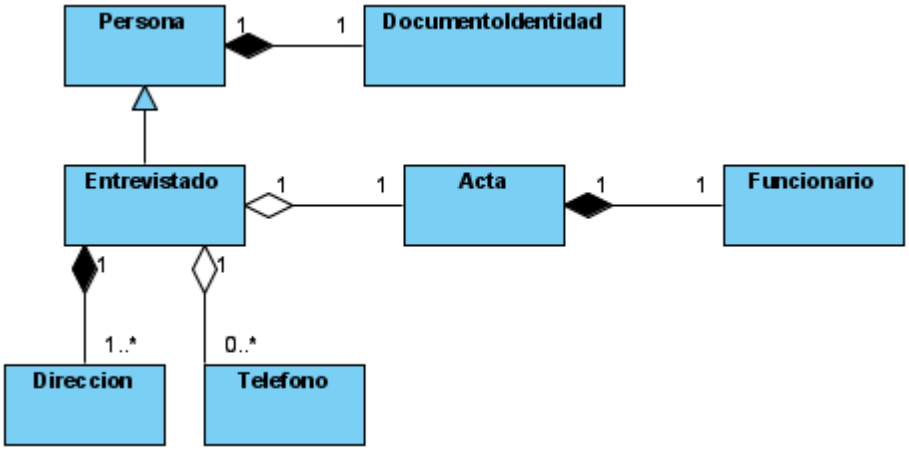


Fig. 2.16. Diagrama de Clases del Dominio del Paquete Gestionar Entrevistado.

Clase Entrevistado

Propósito

Clase de dominio que representa conceptualmente a un Entrevistado.

Atributos

- String idEntrevistado: Identificador del Entrevistado.
- Nomenclador nacionalidad: Representa la información nomenclada de la nacionalidad de los Entrevistados.
- Nomenclador tipoNacionalidad: Representa la información nomenclada de los tipos de nacionalidad.
- Nomenclador estadoCivil: Representa la información nomenclada del estado civil.
- Nomenclador gradoInstruccion: Representa la información nomenclada de los grados de instrucción.

- Nomenclador profesión: Representa la información nomenclada de la profesión del Entrevistado.
- Acta acta: Acta de entrevista del Entrevistado.
- boolean victima: Valor lógico que indica si el Entrevistado es víctima.
- boolean denunciante: Valor lógico que indica si el Entrevistado es denunciante.
- boolean testigo: Valor lógico que indica si el Entrevistado es testigo.
- boolean esIdentificado: Valor lógico que indica si el Entrevistado es identificado o no.
- Foto foto: Foto del Entrevistado.
- List<Direccion> listaDirecciones: Direcciones del entrevistado.
- List<Telefono> listaTelefonos: Teléfonos del entrevistado.

2.8. Paquete Gestionar Denunciado

El paquete Gestionar Denunciado recoge todas las funcionalidades que le permitirán a la aplicación asociarle a la Denuncia los Denunciados que pueden clasificarse en: Persona Natural y Persona Jurídica. Los denunciados de tipo Persona Natural dado que sus características vienen dada por la apreciación del denunciante la aplicación brinda un mecanismo de identificación a través de fotos de ciudadanos reseñados previamente en el sistema el cual permite asociar la foto de dicho ciudadano a la de la persona denunciada, también se le señalan características primarias y secundarias, apodo, si pertenece a alguna banda. El denunciado de tipo Persona Jurídica contiene número de RIF, dirección. Para ambos tipos de denunciados además de los datos indicados se registra los presuntos delitos o faltas por los que se denuncia.

2.8.1. Diagrama de Clases de la Capa Presentación

Fig. 2.17. Diagrama de Clases de la Capa Presentación del Paquete Gestionar Denunciado.

Clase PersonaNaturalController

Propósito

Controlador multi-acciones que permite insertar, modificar y eliminar los datos de un Denunciado tipo Persona Natural.

Atributos

- IPersonaNaturalFacade personaNaturalFacade: Referencia a la fachada (interface) de Persona Natural que define las posibles acciones sobre la misma.
- IDireccionFacade direccionFacade: Referencia a la fachada (interface) de Dirección que define las posibles acciones sobre la misma.
- IDelitoFaltaFacade delitoFaltaFacade: Referencia a la fachada (interface) de Delito o Falta que define las posibles acciones sobre la misma.
- ICaracteristicasPrimariasFacade caracteristicasPrimariasFacade: Referencia a la fachada (interface) de Características Primarias que define las posibles acciones sobre la misma.
- ICaracteristicasSecundariasFacade caracteristicasSecundariasFacade: Referencia a la fachada (interface) de Características Secundarias que define las posibles acciones sobre la misma.
- FotoPersonaNatural foto: Foto que se le asocia de un ciudadano reseñado.
- IDenunciaFacade denunciaFacade: Referencia a la fachada (interface) de Denuncia que define las posibles acciones sobre la misma.

Métodos

- Map<String, Object> onBind(HttpServletRequest request, Object command, BindException errors): Método auxiliar para la unión de parámetros externos a la clase (command).
- private void limpiarFromulario(): Método para inicializar todas las listas que se encuentran en la memoria del servidor.
- Map<String,Object> referenceData(HttpServletRequest request, Object command, Errors errors): Permite suministrar datos que se necesitan en la vista del formulario.
- void insertarPersonaNatural(HttpServletRequest request, HttpServletResponse response, Object form, BindException errors): Si no existen errores (almacenados en *errors*) en la validación del formulario, inserta la Persona Natural en la lista de personas naturales de la denuncia.
- void modificarPersonaNatural(HttpServletRequest request, HttpServletResponse response, Object form, BindException *errors*): Si no existen errores (almacenados en *errors*) en la validación del formulario, modifica la Persona Natural en la lista de personas naturales de la denuncia.

- void eliminarPersonaNatural(HttpServletRequest request, HttpServletResponse response, Object form, BindException errors): Si no existen errores (almacenados en *errors*) en la validación del formulario, elimina la Persona Natural de la lista de personas naturales de la denuncia.
- void registrarCaracteristicasPrimarias(HttpServletRequest request, HttpServletResponse response, Object form, BindException errors): Abre el formulario para la gestión de las caractersticas primarias.
- void registrarCaracteristicasSecundarias(HttpServletRequest request, HttpServletResponse response, Object form, BindException errors): Abre el formulario para la gestión de las caractersticas secundarias.
- void asociarFotoCiudadano(HttpServletRequest request, HttpServletResponse response, Object form, BindException errors): Almacena la foto del ciudadano reseñado seleccionada por el usuario.

2.8.2. Diagrama de Clases de la Capa Negocio

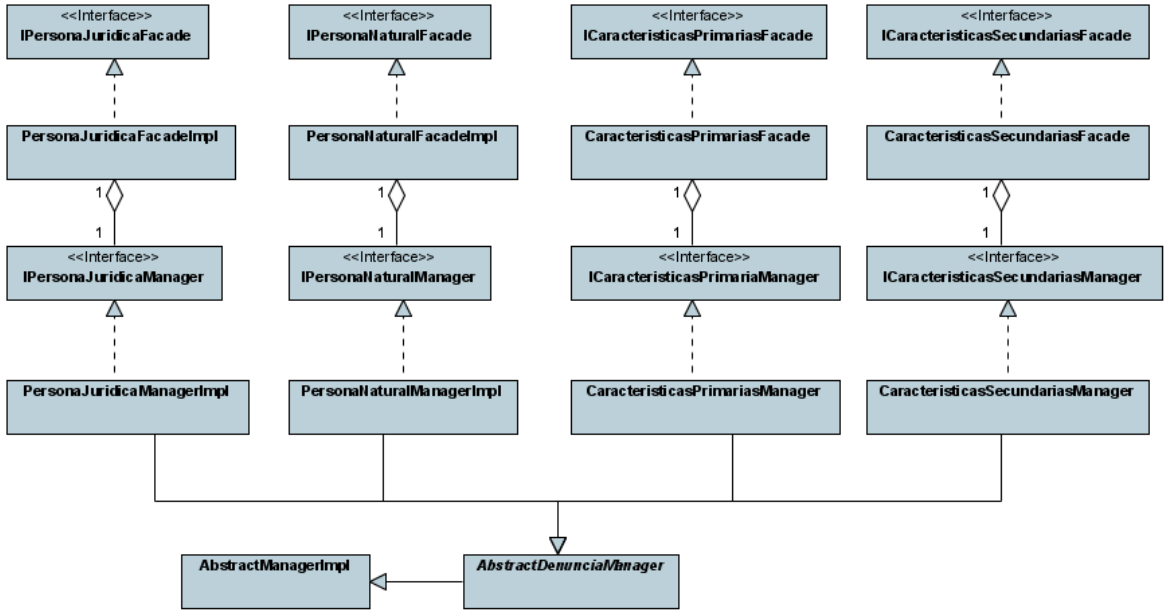


Fig. 2.18. Diagrama de Clases de la Capa Negocio del Paquete Gestionar Denunciado.

Clase PersonaJuridicaManagerImpl

Propósito

Clase de servicio con el fin de darle una implementación concreta a la interfaz *IPersonaJuridicaManager* que define las posibles acciones sobre la Persona Jurídica a nivel de lógica de negocio.

Atributos

- PersonaJuridica personaJuridica: Referencia a la persona jurídica almacenada en memoria.

Métodos

- void insertarPersonaJuridica(PersonaJuridica personaJuridica): Almacena la persona jurídica especificada como parámetro en la memoria.
- PersonaJuridica obtenerPersonaJuridica(): Devuelve la persona jurídica almacenada en memoria.
- void eliminarPersonaJuridica(String idPersonaJuridica): Elimina la persona jurídica especificada por el parámetro *idPersonaJuridica* utilizando el objeto de acceso a datos PersonaJuridicaDaoImpl.
- void insertarPersonaJuridica(PersonaJuridica personaJuridica, String idDenuncia): Asocia a la denuncia representada por el parámetro *idDenuncia* la Persona Jurídica utilizando el objeto de acceso a datos PersonaJuridicaDaoImpl.
- List<DelitoFalta> obtenerDelitosPersonaJuridica(String idPersonaJuridica, String idDenuncia): Devuelve la lista de motivos por los que fue denunciada la persona jurídica especificada como parámetro.
- boolean existePersonaJuridica(String idDenuncia): Verifica si en la denuncia especificada como parámetro existe una Persona Jurídica asociada.

2.8.3. Diagrama de Clases de la Capa Acceso a Datos

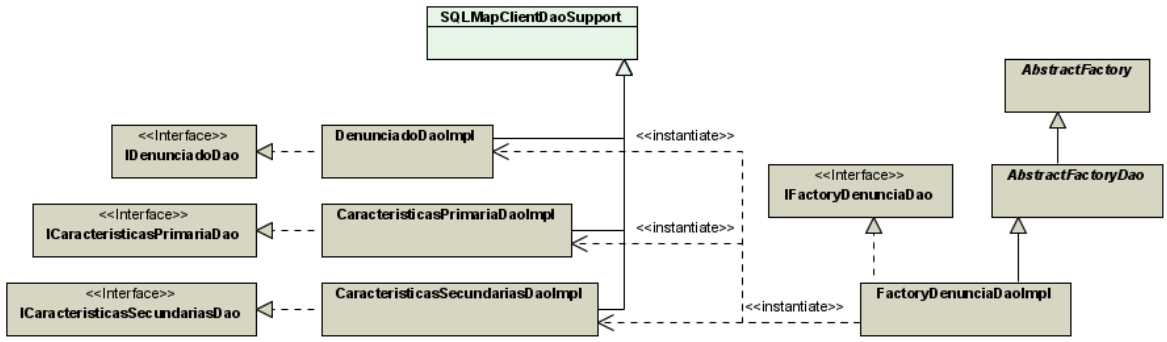


Fig. 2.19. Diagrama de Clases de la Capa Acceso a Datos del Paquete Gestionar Denunciado.

2.8.4. Diagrama de Clases del Dominio

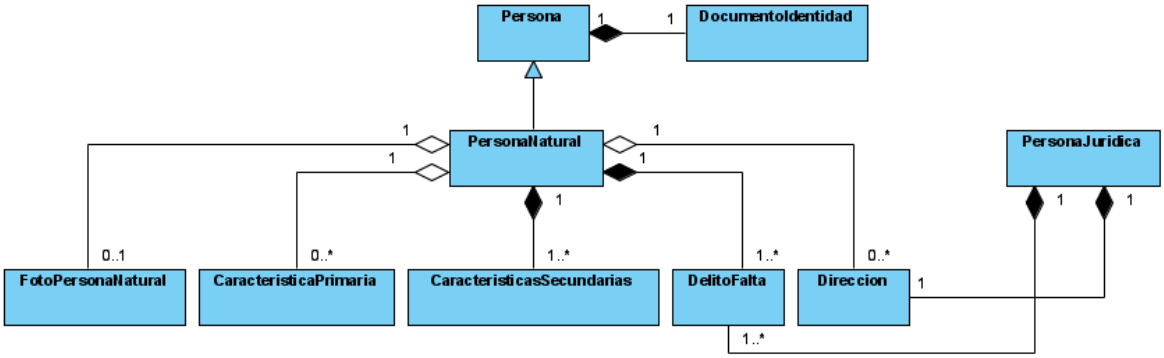


Fig. 2.20. Diagrama de Clases del Dominio del Paquete Gestionar Denunciado.

Clase PersonaNatural

Propósito

Clase de dominio que representa conceptualmente a una Persona Natural.

Atributos

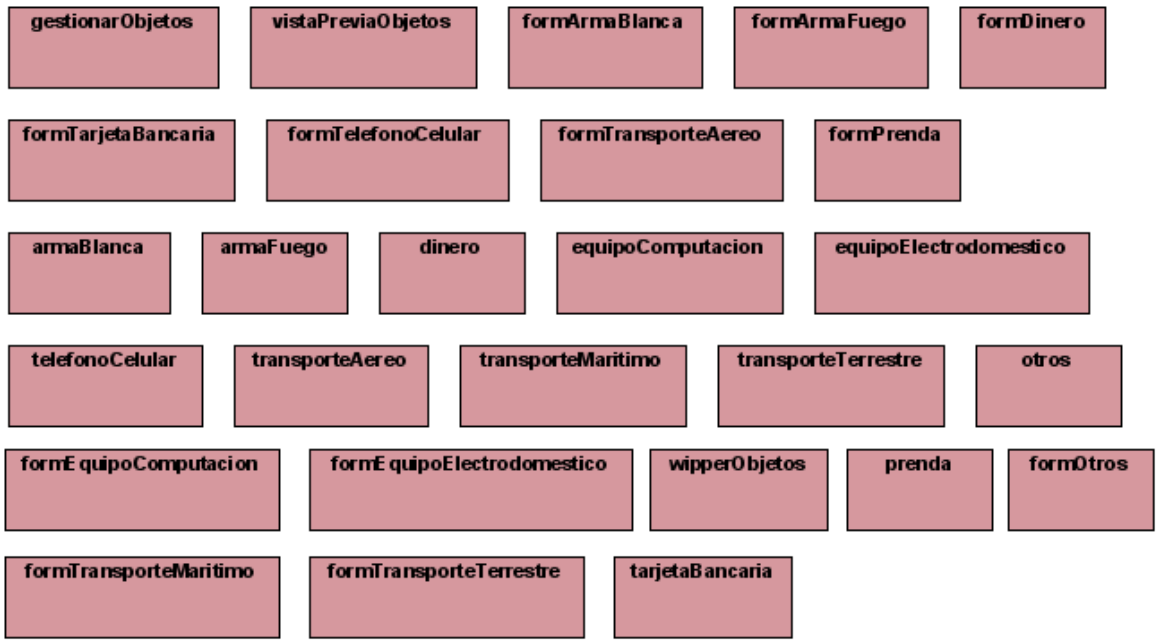
- String apodo: Apodo de la persona Natural.
- String alias: Alias de la persona Natural.
- Integer edadEstimada: Edad estimada de la Persona Natural.
- Nomenclador grupo: Representa el grupo o banda al que está vinculada la Persona Natural.
- String caracteristicaVestimenta: Característica de la vestimenta de la Persona Natural.
- Nomenclador nacionalidad: Representa la información nomenclada de la nacionalidad de la Persona Natural.
- Nomenclador tipoNacionalidad: Representa la información nomenclada del tipo de nacionalidad de la Persona Natural.
- List<CaracteristicaPrimaria> caracteristicasPrimarias: Lista de características primarias de la Persona Natural.
- List<Direccion> listaDirecciones: Lista de direcciones de la Persona Natural.
- List<DelitoFalta> listaDelitoFalta: Lista de delitos o faltas por los que se denuncia a la Persona Natural.
- String observaciones: Observaciones adicionales de las características primarias de la Persona Natural.
- boolean esIdentificado: Valor lógico que representa si la Persona Natural fue identificada o no.
- FotoPersonaNatural foto: Representa la foto de la Persona Natural.

- CaracteristicasSecundarias caractSecundarias: Características secundarias de la Persona Natural.

2.9. Paquete Gestionar Objeto

El paquete Gestionar Objeto recoge todas las funcionalidades que van a permitir insertar los datos de los objetos que estén involucrados en una denuncia, estos datos estarán en función del tipo de objeto que se seleccione, que pueden ser arma de fuego, arma blanca, dinero, equipo de computación, equipo electrodoméstico, prenda, tarjeta bancaria, teléfono celular, transporte aéreo, transporte marítimo, vehículo y otros objetos que no se incluyen dentro de estas clasificaciones.

2.9.1 Diagrama de Clases de la Capa Presentación



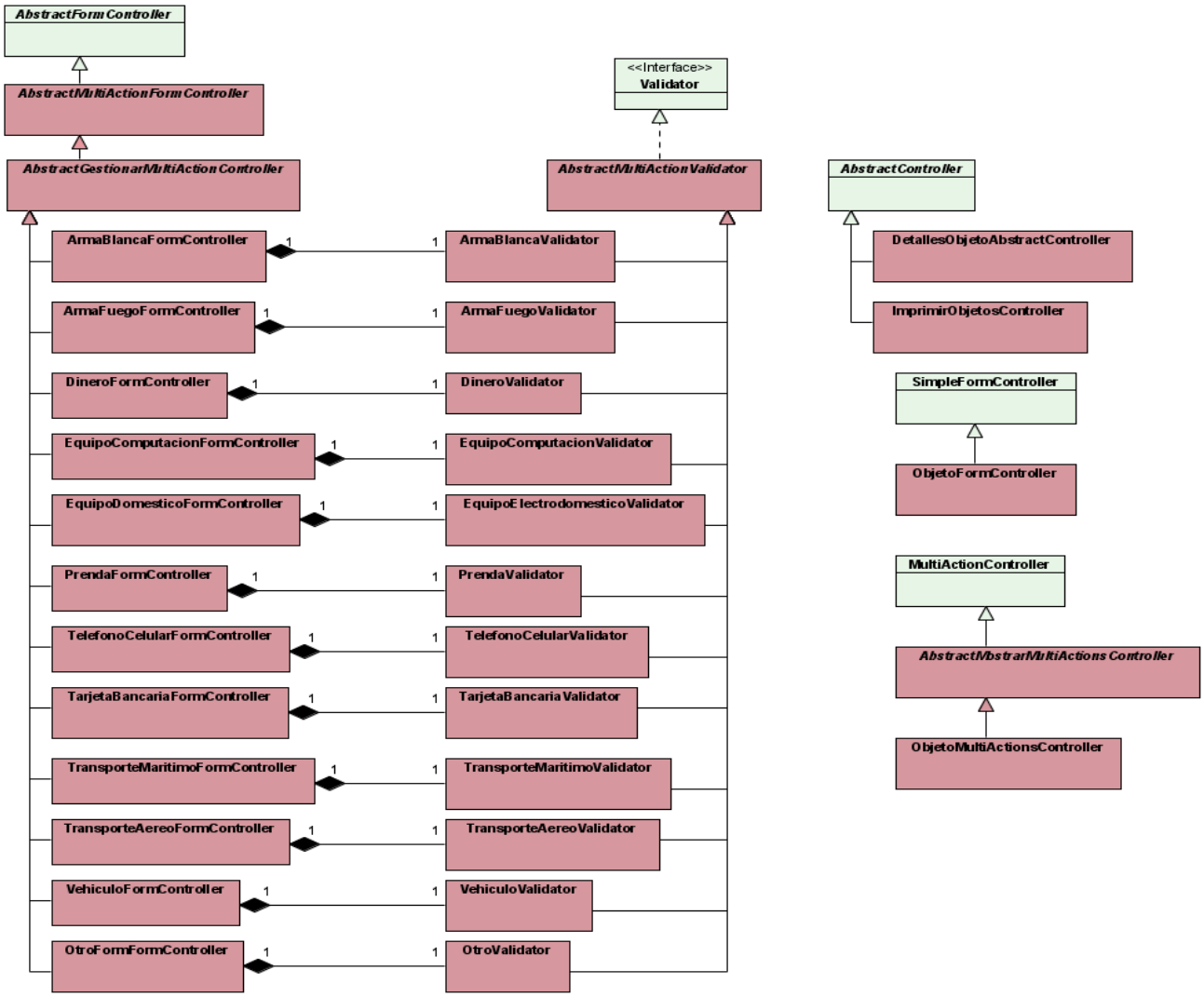


Fig. 2.22. Diagrama de Clases de la Capa Presentación del Paquete Gestionar Objeto.

2.9.2. Diagrama de Clases de la Capa Negocio

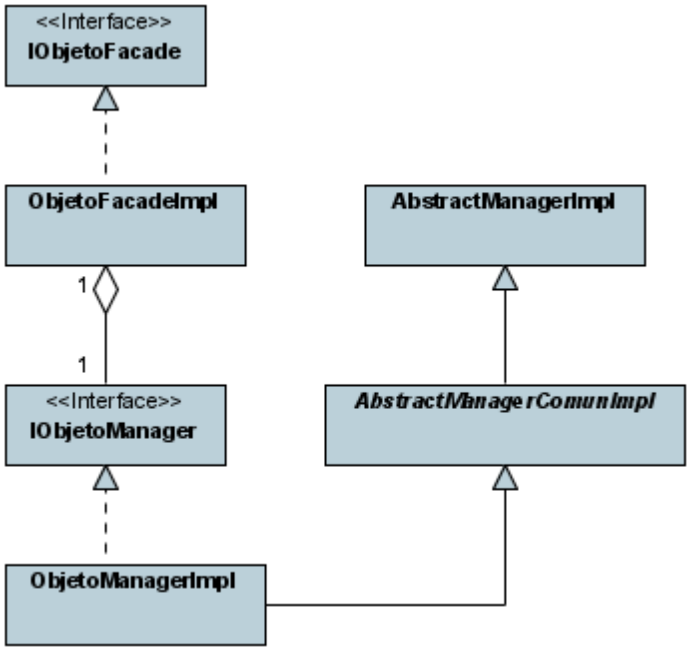


Fig. 2.22. Diagrama de Clases de la Capa Negocio del Paquete Gestionar Objeto.

Clase ObjetoManagerImpl

Propósito

Clase de servicio con el fin de darle una implementación concreta a las funciones que define la interfaz *IObjetoManager* sobre los objetos a nivel de lógica de negocio.

Atributos

- List<Objeto> listaObjetos: Maneja una lista de objetos de las clases de dominio que conforman la jerarquía de objetos.

Métodos

- String insertarObjeto(Objeto objeto, String nombre): Gestiona el conjunto de acciones necesarias para hacer la llamada al método de la capa de acceso a datos que permite insertar los datos del objeto dado.
- void modificarObjeto(Objeto objeto, String nombre): Gestiona el conjunto de acciones necesarias para hacer la llamada al método de la capa de acceso a datos que permite modificar los datos del objeto en cuestión.

- void eliminarObjeto(String idObjeto, String nombre): Gestiona el conjunto de acciones necesarias para hacer la llamada al método de la capa de acceso a datos que permite eliminar el objeto en cuestión.
- Objeto obtenerObjeto(String idObjeto, String nombre): Llama al método de la capa de acceso a datos que retorna los datos del objeto con identificador especificado.
- void insertarObjetoMemoria(Objeto objeto): Inserta el objeto especificado en la lista en memoria para almacenarlos en el momento en que estén completos los datos.
- void modificarObjetoMemoria(Objeto objeto): Modifica el objeto especificado en la lista en memoria para almacenarlos en el momento en que estén completos los datos.
- void eliminarObjetoMemoria(String idObjeto): Elimina el objeto especificado de la lista en memoria.
- Map<String, Object> obtenerListaObjetosMemoriaPaginado(int posicionInicial, int cantidad): Devuelve el listado de objetos paginados almacenados en memoria.

2.9.3. Diagrama de Clases de la Capa Acceso a Datos

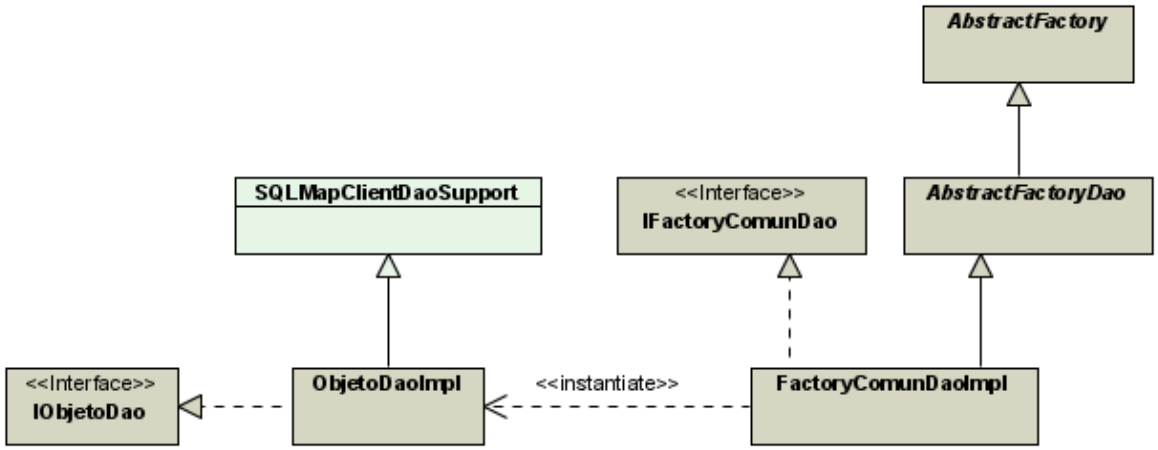


Fig. 2.23. Diagrama de Clases de la Capa Acceso a Datos del Paquete Gestionar Objeto.

2.9.4. Diagrama de Clases del Dominio

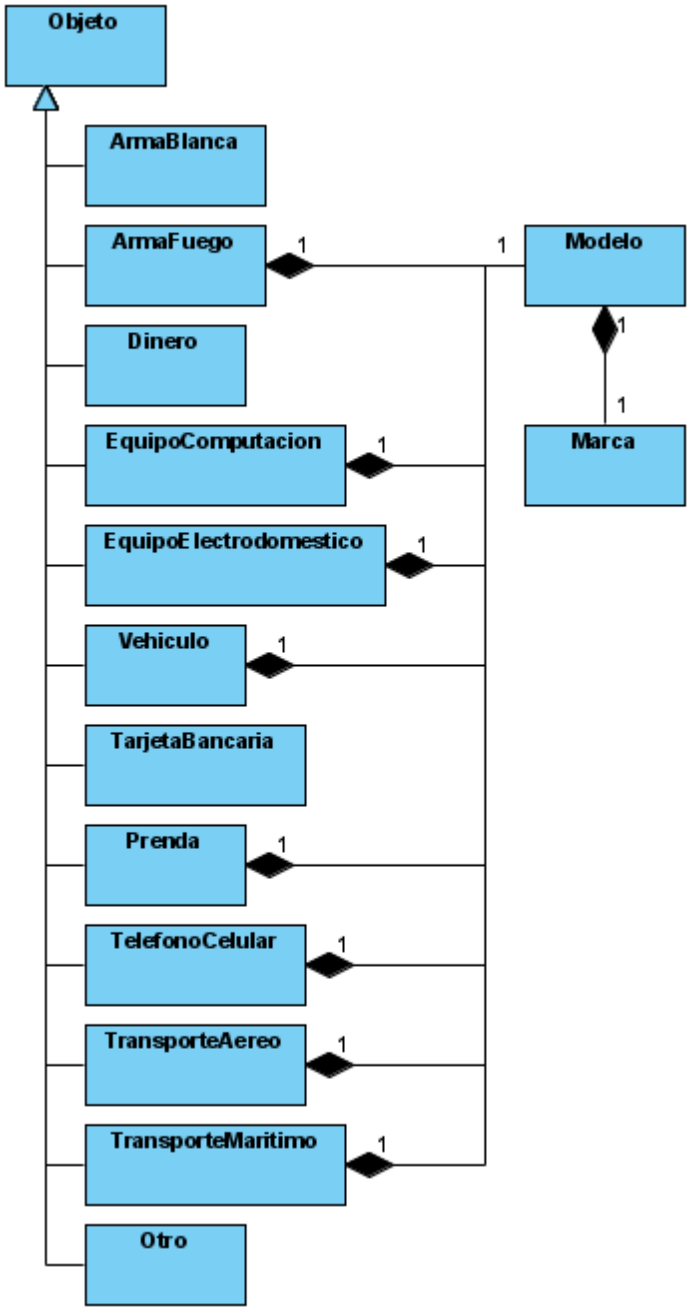


Fig. 2.24. Diagrama de Clases del Dominio del Paquete Gestionar Objeto.

2.10. Paquete Reporte

Este paquete proporcionará reportes sobre las denuncias registradas en la dependencia policial agrupadas bajo diversos criterios. Mostrará el reporte de los delitos y de las faltas denunciadas en un intervalo de tiempo y en una zona geográfica determinada. Permitirá mostrar el reporte de la cantidad de denuncias que se registran en un día en la dependencia.

2.10.1. Diagrama de Clases de la Capa Presentación

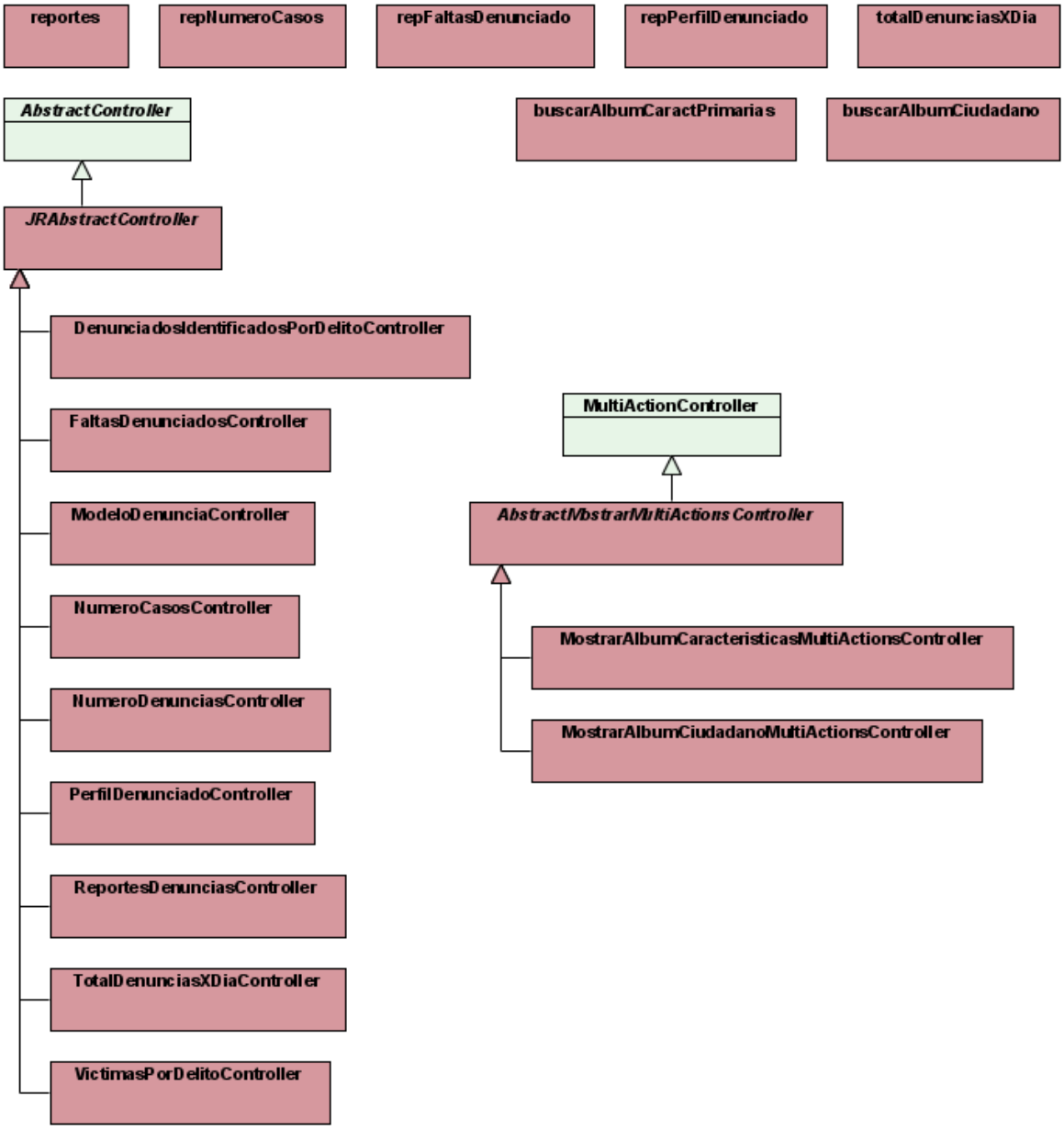


Fig. 2.25. Diagrama de Clases de la Capa de Presentación del Paquete Reporte.

Clase PersonaNaturalController

Propósito

Controlador que atiende la petición de mostrar reporte de denuncias en una dependencia determinada usando distintos criterios de agrupamiento.

Atributos

- IReportesFacade reportesFacade: Referencia a la fachada (interface) de Reporte que define las posibles acciones sobre la misma.
- INomencladorFacade nomencladorFacade: Referencia a la fachada (interface) de Nomencladores que define las posibles acciones sobre la misma.

Métodos

- Map<String, Object> referenceData(HttpServletRequest request): Permite suministrar datos que se necesitan en la vista del formulario.
- Map<String, Object> mostrarReporte(HttpServletRequest request, HttpServletResponse response): Se encarga de recopilar la información que se va a mostrar en la interfaz del reporte.

2.10.2. Diagrama de Clases de la Capa Negocio

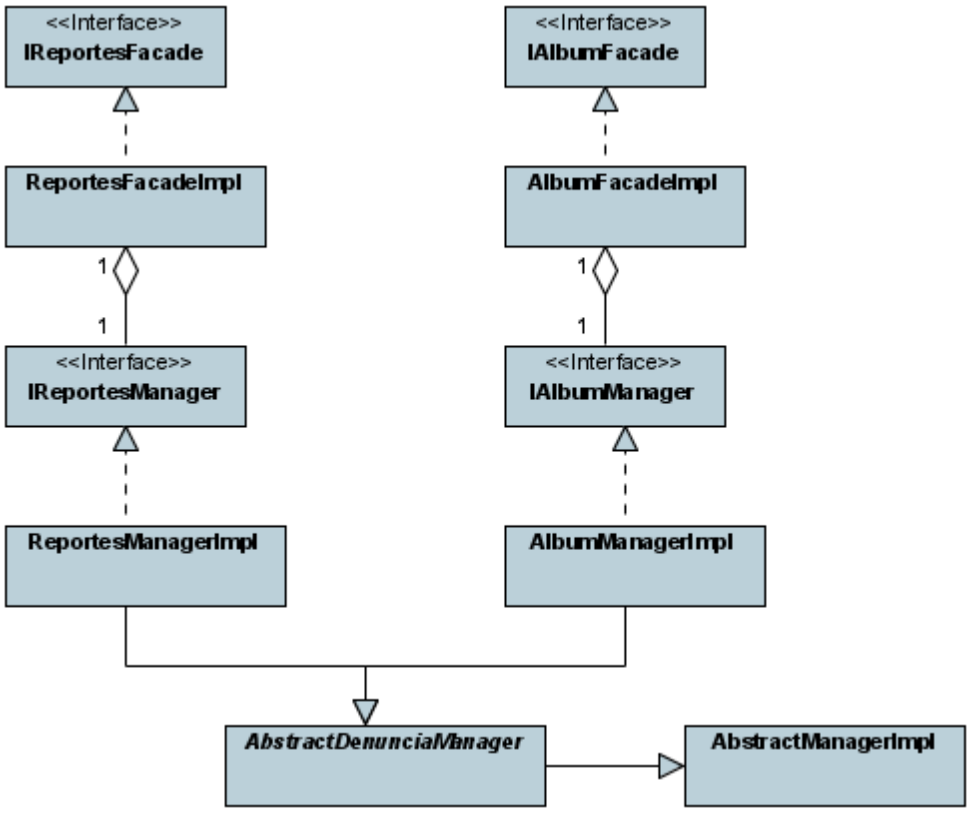


Fig. 2.26. Diagrama de Clases de la Capa Negocio del Paquete Reporte.

2.10.3. Diagrama de Clases de la Capa Acceso a Datos

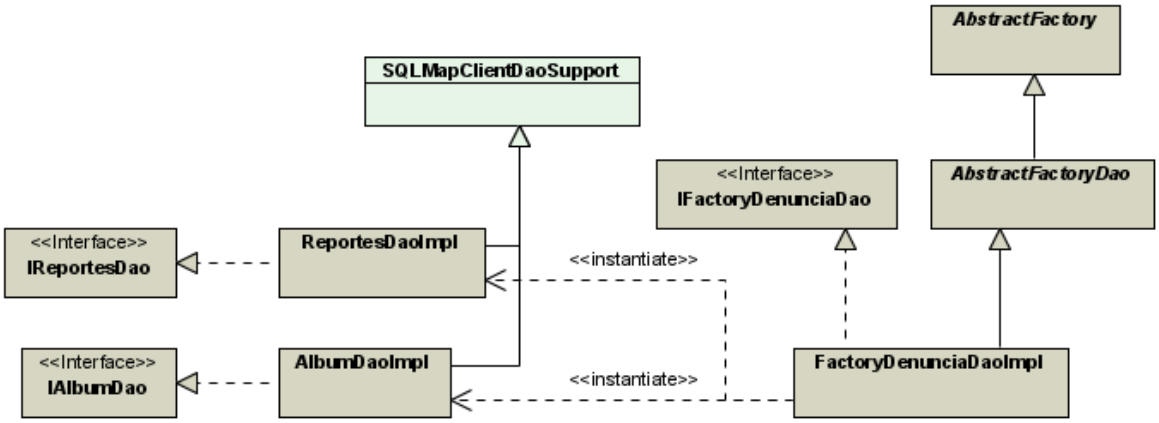


Fig. 2.27. Diagrama de Clases de la Capa Acceso a Datos del Paquete Reporte.

Clase ReportesDaoImpl

Propósito

Implementar los métodos definidos por la interfaz IReporteDao para el manejo de los distintos reportes al nivel de lógica de acceso a datos.

Atributos

No aplica.

Métodos

- List<ReporteResult> reporteNumeroCasosPorDelito(String usuario, String idDependencia, String idEstado): Devuelve la lista de Casos Policiales agrupados por delitos en la Dependencia especificada como parámetro.
- List<ReporteResult> reporteNumeroDenunciasPorDelito(String usuario, String idDependencia, String idEstado): Devuelve la lista de Denuncias agrupadas por delitos en la Dependencia especificada como parámetro.
- List<ReporteResult> reporteTotalDenunciasXDia(String usuario, String idDependencia, Date fechaInicial, Date fechaFinal): Devuelve la lista de Denuncias agrupadas por todos los días de la semana en el rango de fecha y la Dependencia especificados como parámetro.
- List<ReporteResult> reporteVictimasXDelito(String usuario, String idDependencia, String idEstado): Devuelve la lista de víctimas asociadas a la denuncia agrupadas por delito en la Dependencia especificada como parámetro.

- List<ResultDenuncia> obtenerDenunciasOrdenadasPor(BuscarDenuncia criterios, String groupBy): Devuelve la lista de Denuncias agrupadas por distintos criterios especificados como parámetros.
- List<FaltasDenunciadosDataSource> reporteFaltasDenunciados(Persona denunciado, String idDependencia, String usuario): Devuelve la lista de Faltas agrupadas por denunciados en la dependencia especificada como parámetro.
- int obtenerCantidadDenunciadosXDenuncia(String idDenuncia): Devuelve la cantidad de denunciados de la denuncia especificada como parámetro.

2.11. Paquete Común

Este paquete recoge el conjunto de clases que son utilizadas por todos los módulos que conforman SIGEPOL. Entre ellas tenemos las clases asociadas a la gestión de la dirección, del teléfono, del funcionario y de los delitos o faltas. Dichos elementos son usados a lo largo de todo el sistema y a su vez de gran utilidad en el Módulo de Denuncias.

2.11.1 Diagrama de Clases de la Capa Presentación

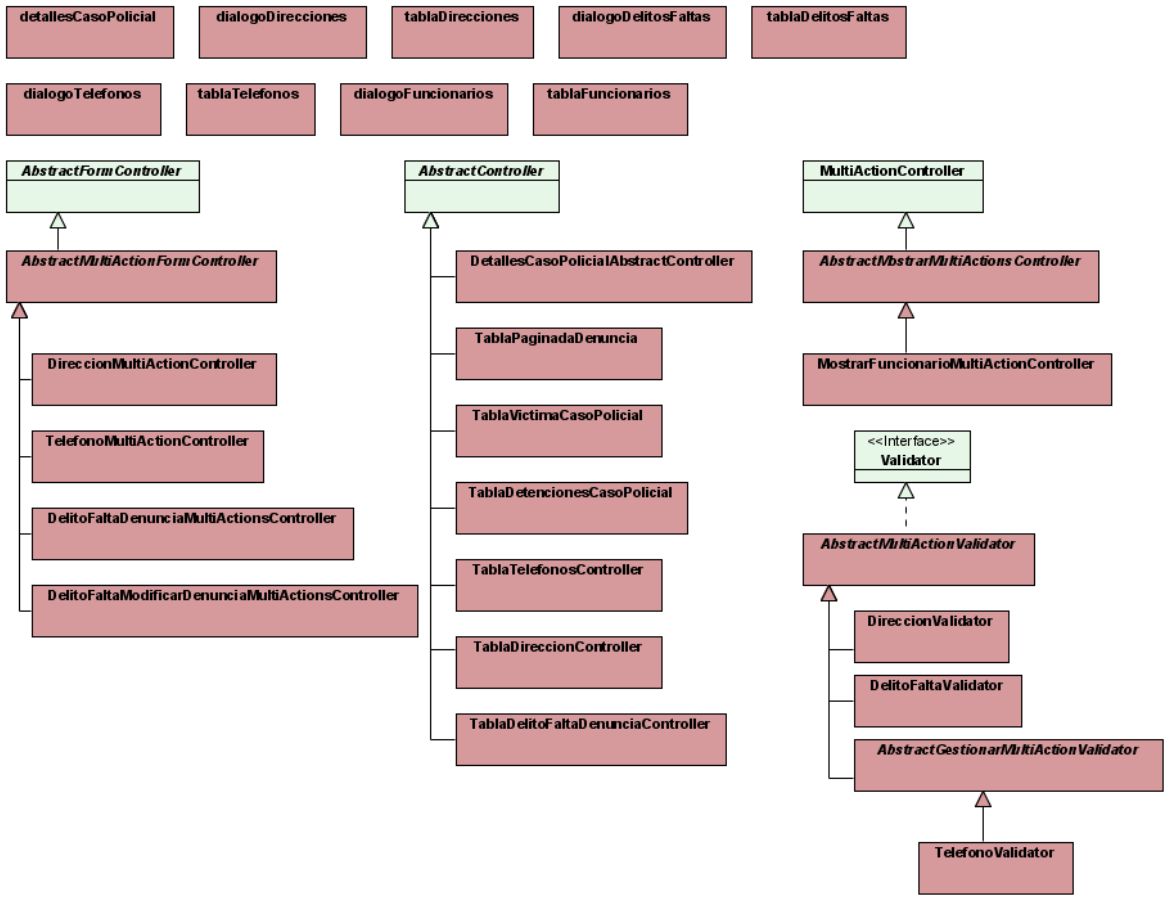


Fig. 2.28. Diagrama de Clases de la Capa Presentación del Paquete Común.

2.11.2. Diagrama de Clases de la Capa Negocio

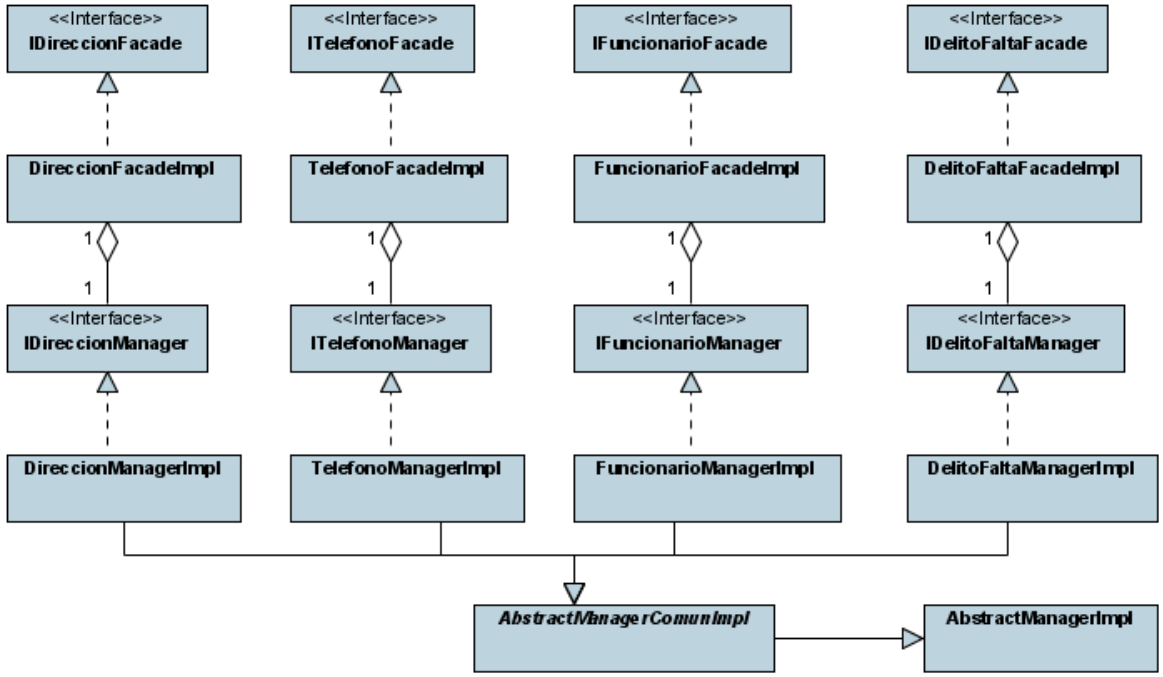


Fig. 2.29. Diagrama de Clases de la Capa Negocio del Paquete Común.

Clase FuncionarioManagerImpl

Propósito

Clase de servicio con el fin de darle una implementación concreta a las funciones que define la interfaz *IFuncionarioManager* sobre los funcionarios a nivel de lógica de negocio.

Atributos

- INomencladorManager nomencladorManager: Referencia a la interfaz de Nomenclador, con el fin de cargar el nomenclador jerarquía del funcionario.

Métodos

- List<NomencladorDependencia> dependenciasPorNombreUsuario(String nombreUsuario): Retorna el listado de dependencias a las que el usuario tiene acceso.
- boolean existeFuncionario(Funcionario funcionario): Verifica, auxiliándose de la capa acceso a datos, si el funcionario especificado ya existe en BD.
- Funcionario funcionarioPorNombreUsuario(String nombreUsuario): Devuelve el funcionario que tenga asociado el nombre de usuario especificado.
- void eliminarListFuncionario(String idFuncionario): Elimina de la lista de funcionarios el que tenga el identificador especificado.

- void insertarListFuncionario(Funcionario funcionario): Inserta en la lista de funcionarios en memoria el funcionario especificado.
- Funcionario obtener(String idFuncionario): Retorna el funcionario con el identificador especificado.
- Map<String, Object> listarFuncionarios(Map<String, Object> criterios): Devuelve la lista de funcionarios que concuerden con los criterios de búsqueda.

2.11.3 Diagrama de Clases de la Capa Acceso a Datos

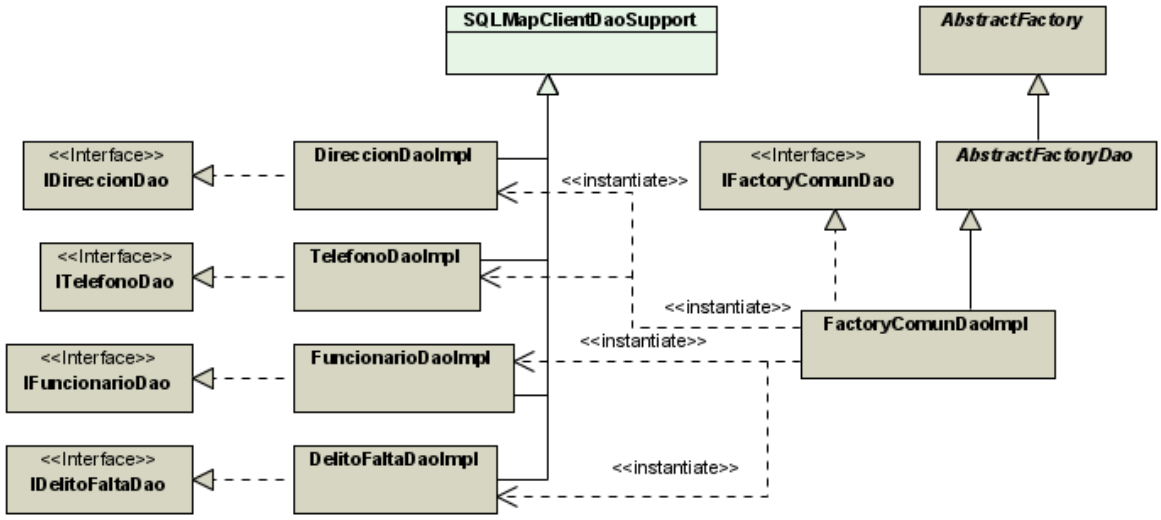


Fig. 2.30. Diagrama de Clases de la Capa Acceso a Datos del Paquete Común.

2.11.4. Diagrama de Clases del Dominio

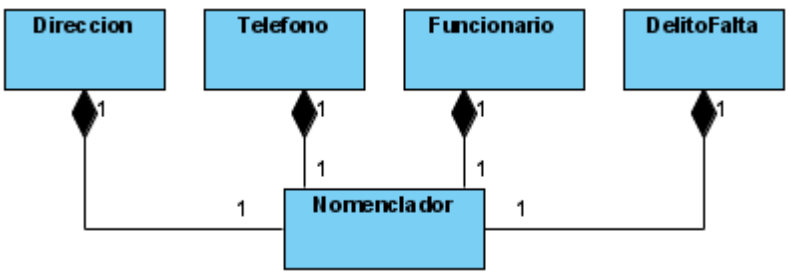


Fig. 2.31. Diagrama de Clases del Dominio del Paquete Común.

Clase DelitoFalta

Propósito

Representa el motivo de una acción delictiva.

Atributos

- Nomenclador motivo: Representa la información nomenclada referente a los delitos o faltas.
- Nomenclador gradoDelito: Representa la información nomenclada referente al grado del delito.
- boolean esDelito: Valor lógico que representa si el motivo en causa es un delito o una falta.

2.12. Conclusiones

En este capítulo se realizó una breve descripción de la arquitectura del sistema SIGEPOL, específicamente en lo relacionado al Módulo de Denuncias, se presentó el diagrama de paquetes para el diseño propuesto, así como los diagramas de clases de cada una de las capas correspondientes a estos. Además se realizó una descripción de las principales clases del diseño.

Capítulo 3 Implementación de la solución

3.1. Introducción

En el presente capítulo se representan en términos de componentes los elementos del modelo del diseño definidos en el capítulo anterior. A continuación se modela un diagrama de componentes, con los componentes principales que constituyen la aplicación del Módulo de Denuncias. Se realiza una breve descripción de cada componente, especificando su propósito y contenido.

3.2. Modelo de Implementación

Los componentes son la parte física y reemplazable de un sistema que está compuesto por un conjunto de interfaces y proporciona la realización de dicho conjunto. Se usan para modelar los elementos físicos que pueden hallarse en un nodo por lo que empaquetan elementos como clases, colaboraciones e interfaces. Son independientes entre ellos y tienen su propia estructura e implementación. Tienen relaciones de traza con los elementos del modelo que implementan.

3.2.1. Diagrama de Componentes.

Los diagramas de componentes se utilizan para modelar la vista estática de un sistema. Muestran la organización y las dependencias lógicas entre un conjunto de componentes software, sean éstos componentes de código fuente, librerías, binarios o ejecutables. El uso más importante de estos diagramas es mostrar la estructura de alto nivel del modelo de implementación.

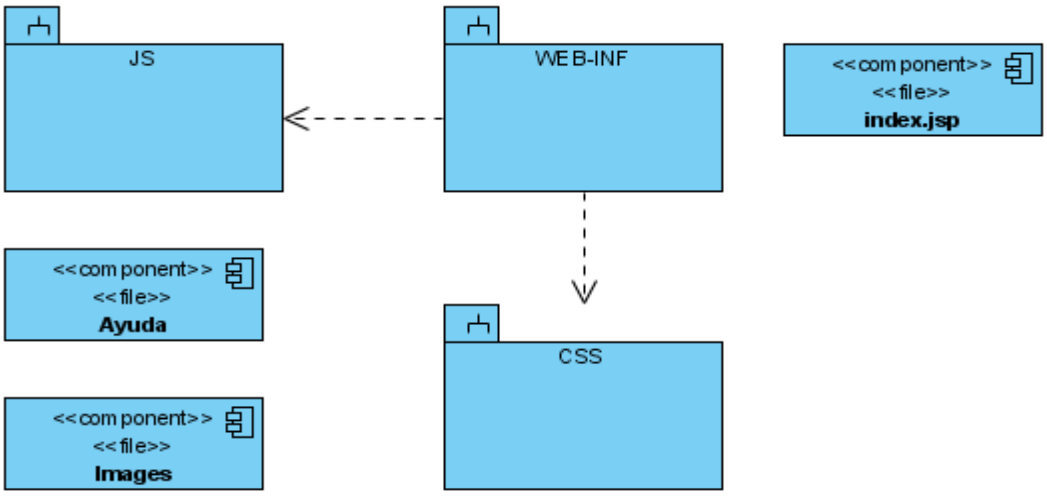


Fig. 3.1 Diagrama de Componentes.

3.2.2. Descripción de Componentes.

a) <u>Subsistema JS</u>

Propósito

Contenedor físico de los ficheros que contienen código JavaScript.

Contenido

- BuscarAlbumCaractPrimarias.js
- BuscarAlbumCiudadano.js
- BuscarCasoPolicial.js
- BuscarDenuncias.js
- DetallesDenuncia.js
- GestionarCaracteristicasPrimarias.js
- GestionarCaracteristicasSecundarias.js
- GestionarDenuncias.js
- GestionarEntrevistados.js
- GestionarObjetos.js
- GestionarPersonaJuridica.js
- GestionarPersonaNatural.js
- IdentificarEntrevistadoSAIME.js
- ModificarDenuncia.js
- RegistrarCasoPolicial.js
- RegistrarDatosBasicos.js
- TabsContainerGDD.js
- DetallesCasoPolicial.js
- DialogoDelitosFaltas.js
- DialogoDirecciones.js
- DialogoFuncionarios.js
- DialogoTelefonos.js
- Directions.js
- Undo.js
- util.js
- Tablas.js
- Comunes.js

- Objetos.js
- core.js
- index.js
- page.js
- validator.js
- check.js

b) Componente Ayuda

Propósito

Contenedor físico de los ficheros que conforman la ayuda de la aplicación.

c) Componente index.jsp

Propósito

Constituye el punto de entrada a la aplicación.

d) Componente Images

Propósito

Contenedor físico de los ficheros que conforman las imágenes de la aplicación.

e) Subsistema WEB-INF

Propósito

Contenedor físico de los ficheros que conforman el núcleo de la aplicación.

Contenido

- Subsistema JSP
- Subsistema Classes
- seguridad.jar
- web.xml

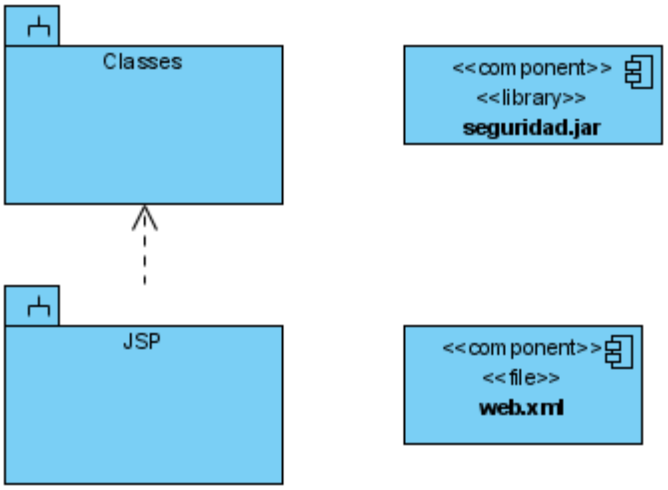


Fig. 3.2. Diagrama de Componentes del Subsistema WEB-INF.

e.1) <u>Subsistema JSP</u>

Propósito

Contenedor físico de las interfaces de usuarios, archivos con extensión JSP.

Contenido

- buscarAlbumCaractPrimarias.jsp
- buscarAlbumCiudadano.jsp
- buscarCasoPolicial.jsp
- buscarDenuncias.jsp
- detallesDenuncia.jsp
- detallesDenunciado.jsp
- detallesEntrevistado.jsp
- gestionarCaracteristicasPrimarias.jsp
- gestionarCaracteristicasSecundarias.jsp
- gestionarDenuncias.jsp
- gestionarEntrevistados.jsp
- gestionarObjetos.jsp
- gestionarPersonaJuridica.jsp
- gestionarPersonaNatural.jsp
- identificarEntrevistadoSAIME.jsp
- modificarDenuncia.jsp

- registrarCasoPolicial.jsp
- registrarDatosBasicos.jsp
- tabsContainerGDD.jsp
- repFaltasDenunciado.jsp
- repNumeroCasos.jsp
- reportes.jsp
- repPerfilDenunciado.jsp
- tabGestionarDenunciados.jsp
- tablaDenuncias.jsp
- tablaPersonaJuridica.jsp
- tablaPersonaNatural.jsp
- totalDenunciasXDia.jsp
- vistaPreviaDatosBasicos.jsp
- vistaPreviaEntrevistados.jsp
- vistaPreviaObjetos.jsp
- vistaPreviaPersonaJuridica.jsp
- vistaPreviaPersonasNaturales.jsp
- armaBlanca.jsp
- armaFuego.jsp
- dinero.jsp
- equipoComputacion.jsp
- equipoElectrodomestico.jsp
- otros.jsp
- prenda.jsp
- tarjetaBancaria.jsp
- telefonoCelular.jsp
- transporteAereo.jsp
- transporteMaritimo.jsp
- transporteTerrestre.jsp
- detallesCasoPolicial.jsp
- dialogoDelitosFaltas.jsp
- dialogoDirecciones.jsp

- dialogoFuncionarios.jsp
- dialogoTelefonos.jsp
- tabla.jsp
- tablaCaracteristicasPrimarias.jsp
- tablaDelitosFaltas.jsp
- tablaDenunciasDialogo.jsp
- tablaDirecciones.jsp
- tablaFuncionarios.jsp
- tablaTelefonos.jsp
- formArmaBlanca.jsp
- formArmaFuego.jsp
- formDinero.jsp
- formEquipoComputacion.jsp
- formEquipoElectrodomestico.jsp
- formOtros.jsp
- formPrenda.jsp
- formTarjetaBancaria.jsp
- formTelefonoCelular.jsp
- formTransporteAereo.jsp
- formTransporteMaritimo.jsp
- formTransporteTerrestre.jsp
- wipperObjetos.jsp

e.2) Subsistema Classes

Propósito

Contenedor físico del compilado de cada una de las clases de la aplicación, archivos con extensión CLASS, JASPER y XML.

Contenido

- AbstractDenunciaManager.class
- Acta.class
- Acta.xml
- Album.xml

- AlbumDaoImpl.class
- AlbumFacadeImpl.class
- Album ManagerImpl.class
- AsignarDirPersona.xml
- BuscarDenunciaController.class
- BuscarDenunciaModificarController.class
- BuscarDenunciaPController.class
- BuscarDenunciasModController.class
- CantidadDenunciadosPorDeitoDataSource.class
- CaracteristicaPrimaria.class
- CaracteristicaPrimaria.xml
- CaracteristicaPrimariaCiudadano.class
- CaracteristicasPrimariaDaoImpl.class
- CaracteristicasPrimariasController.class
- CaracteristicasPrimariasFacade.class
- CaracteristicasPrimariasManager.class
- CaracteristicasPrimariasValidator.class
- CaracteristicasSecundarias.class
- CaracteristicasSecundarias.xml
- CaracteristicasSecundariasController.class
- CaracteristicasSecundariasDaoImpl.class
- CaracteristicasSecundariasFacade.class
- CaracteristicasSecundariasManager.class
- ContarElementos.xml
- DatosBasicosCiudadano.class
- DelitoFaltaModificarDenunciaMultiActionsController.class
- Denuncia.class
- Denuncia.xml
- DenunciaDaoImpl.class
- DenunciaDataSource.class
- Denunciado.class
- DenunciadoController.class

- DenunciadoDaoImpl.class
- DenunciadosIdentificadosPorDelitoController.class
- DenunciaFacadeImpl.class
- DenunciaManagerImpl.class
- Denunciante.xml
- DenuncianteVictima.xml
- DenunciaService.xml
- DenunciaServiceManagerImpl.class
- DenunciasServiceFacadeImpl.class
- DenunciasXDiaDataSource.class
- DetallesCasoPolicialAbstractController.class
- DetallesDenunciaController.class
- DetallesDenunciadoController.class
- DetallesEntrevistadoController.class
- Entrevistado.class
- Entrevistado.xml
- EntrevistadoController.class
- EntrevistadoDaoImpl.class
- EntrevistadoFacadeImpl.class
- EntrevistadoManagerImpl.class
- EntrevistadoMDenunciaController.class
- EntrevistadoValidator.class
- FactoryDenunciaDaoImpl.class
- FaltasDenunciadosCommand.class
- FaltasDenunciadosController.class
- FaltasDenunciadosDataSource.class
- FotoPersonaNatural.class
- IAlbumDao.class
- IAlbumFacade.class
- IAlbum Manager.class
- ICaracteristicasPrimariaDao.class
- ICaracteristicasPrimariaManager.class

- ICaracteristicasPrimariasFacade.class
- ICaracteristicasSecundariasDao.class
- ICaracteristicasSecundariasFacade.class
- ICaracteristicasSecundariasManager.class
- IdentificarCiudadanoSaimeMultiActionsFormController.class
- IDenunciaDao.class
- IDenunciadoDao.class
- IDenunciaFacade.class
- IDenunciaManager.class
- IDenunciaServiceManager.class
- IDenunciasServiceFacade.class
- IEntrevistadoDao.class
- IEntrevistadoFacade.class
- IEntrevistadoManager.class
- IFactoryDenunciaDao.class
- ImprimirDenunciaController.class
- ImprimirDenunciadosController.class
- ImprimirEntrevistadosController.class
- ImprimirObjetosController.class
- ImprimirPersonaJuridicaController.class
- InsertarCasoPolicialFormController.class
- InsertarDenunciaController.class
- InsertarDenunciaValidator.class
- IPersonaJuridicaFacade.class
- IPersonaJuridicaManager.class
- IPersonaNaturalFacade.class
- IPersonaNaturalManager.class
- IReportesDao.class
- IReportesFacade.class
- IReportesManager.class
- ModeloDenuncia.jasper
- ModeloDenunciaController.class

- ModeloDenunciaDataSource.class
- ModeloDenuncia_listaObjetos.jasper
- ModeloDenuncia_subEntrevistados.jasper
- ModeloDenuncia_subEntrevistados_subTelefonos.jasper
- ModeloDenuncia_subPersNatural.jasper
- ModeloDenuncia_subPersNatural_subCaractPrimarias.jasper
- ModeloDenuncia_subPersNatural_subCaractSec.jasper
- ModeloDenuncia_subPersNatural_subDelFal.jasper
- ModeloDenuncia_subPersNatural_subDireccion.jasper
- ModeloDenuncia_subPersonaJur.jasper
- ModificarDenunciaMultiActionsFormController.class
- ModificarDenunciaValidator.class
- MostrarAlbumCaracteristicasMultiActionsController.class
- MostrarAlbumCiudadanoMultiActionsController.class
- MostrarCasoPolicialAbstractController.class
- MostrarCasoPolicialFormController.class
- NumeroCasosController.class
- NumeroDenunciasController.class
- ObjetoFormController.class
- PerfilDenunciadoController.class
- PerfilDenunciadoCriteria.class
- PerfilDenunciadoDataSource.class
- PersonaJuridica.class
- PersonaJuridica.xml
- PersonaJuridicaController.class
- PersonaJuridicaFacadeImpl.class
- PersonaJuridicaManagerImpl.class
- PersonaJuridicaMController.class
- PersonaJuridicaValidator.class
- PersonaNatural.class
- PersonaNatural.xml
- PersonaNaturalController.class

- PersonaNaturalFacadeImpl.class
- PersonaNaturalManagerImpl.class
- PersonaNaturalMDenunciaController.class
- PersonaNaturalValidator.class
- repDenunciadosIdentificadosPorDelito.jasper
- repDenunciasAreaGeografica.jasper
- repDenunciasFecha.jasper
- repDenunciasMotivoHecho.jasper
- repDenunciasRangoHora.jasper
- repDenunciasXDia.jasper
- repFaltasPersonaDenunciada.jasper
- repNumeroCasos.jasper
- ReporteCommand.class
- ReporteResult.class
- Reportes.xml
- ReportesDaoImpl.class
- ReportesDenunciasController.class
- ReportesFacadeImpl.class
- ReportesManagerImplPerfilDenunciado.class
- ReportesManagerImpl.class
- ReporteTotalDenunciasXDiaCommand.class
- repPerfilDenunciado.jasper
- repVictimasXDelito.jasper
- sigepol-denuncias-context.xml
- sigepol-denuncias-dataaccess-context.xml
- sigepol-denuncias-menu-context.xml
- sigepol-denuncias-report-context.xml
- sigepol-denuncias-security-context.xml
- sigepol-denuncias-servlet.xml
- sigepol-denuncias-ws-context.xml
- sqlMapConfig.xml
- TablaEntrevistadoAbstractController.class

- TablaPaginadaEntrevistado.class
- TablaPaginadaObjetos.class
- TablaPaginadaPesonaNatural.class
- TablaPaginadaTestigos.class
- TablaPaginadaVictimas.class
- TablaPersonaJuridica.class
- TablaPersonaJuridicaAbstractController.class
- TablaPersonaNaturalAbstractController.class
- TerminarDenunciaController.class
- TerminarDenunciaValidator.class
- Testigo.xml
- TotalDenunciasXDiaController.class
- Victima.xml
- VictimasPorDelitoController.class
- AbstractFactoryDao.class
- AbstractGestionarMultiActionValidator.class
- AbstractManagerComunImpl.class
- AbstractManagerImpl.class
- AbstractMultiActionValidator.class
- ArmaBlanca.class
- ArmaBlanca.xml
- ArmaBlancaFormController.class
- ArmaBlancaValidator.class
- ArmaFuego.class
- ArmaFuego.xml
- ArmaFuegoFormController.class
- ArmaFuegoValidator.class
- BuscarDenuncia.class
- CasoPolicial.class
- CasoPolicial.xml
- CasoPolicialDaoImpl.class
- CasoPolicialFacadeImpl.class

- CasoPolicialManagerImpl.class
- DelitoFalta.class
- DelitoFalta.xml
- DelitoFaltaDaoImpl.class
- DelitoFaltaDenunciaMultiActionsController.class
- DelitoFaltaFacadeImpl.class
- DelitoFaltaManagerImpl.class
- DelitoFaltaPersonaJuridicaController.class
- DelitoFaltaPersonaNaturalMultiActionsController.class
- DelitoFaltaValidator.class
- Denuncia.class
- DetallesCasoPolicialAbstractController.class
- DetallesObjetoAbstractController.class
- Dinero.class
- Dinero.xml
- DineroFormController.class
- DineroValidator.class
- Direccion.class
- Direccion.xml
- Direccion2MultiActionsController.class
- DireccionDaoImpl.class
- DireccionFacadeImpl.class
- DireccionManagerImpl.class
- DireccionMultiActionController.class
- DireccionValidator.class
- EquipoComputacion.class
- EquipoComputacion.xml
- EquipoComputacionFormController.class
- EquipoComputacionValidator.class
- EquipoDomesticoFormController.class
- EquipoElectrodomestico.class
- EquipoElectrodomestico.xml

- EquipoElectrodomesticoValidator.class
- Expediente.class
- Expediente.xml
- ExpedienteDaoImpl.class
- ExpedienteFacadeImpl.class
- ExpedienteManagerImpl.class
- FactoryComunDaoImpl.class
- Funcionario.class
- Funcionario.xml
- FuncionarioDaoImpl.class
- FuncionarioFacadeImpl.class
- FuncionarioManagerImpl.class
- ICasoPolicialDao.class
- ICasoPolicialFacade.class
- ICasoPolicialManager.class
- IDelitoFaltaDao.class
- IDelitoFaltaFacade.class
- IDelitoFaltaManager.class
- IDireccionDao.class
- IDireccionFacade.class
- IDireccionManager.class
- IExpedienteDao.class
- IExpedienteFacade.class
- IExpedienteManager.class
- IFactoryComunDao.class
- IFuncionarioDao.class
- IFuncionarioFacade.class
- IFuncionarioManager.class
- IndexController.class
- IObjetoDao.class
- IObjetoFacade.class
- IObjetoManager.class

- ITelefonoDao.class
- ITelefonoFacade.class
- ITelefonoManager.class
- JasperReportUtils.class
- JRAbstractController.class
- MostrarFuncionarioMultiActionController.class
- Objeto.class
- Objeto.xml
- ObjetoDaoImpl.class
- ObjetoFacadeImpl.class
- ObjetoManagerImpl.class
- ObjetoMultiActionsController.class
- Otro.class
- Otro.xml
- OtroFormFormController.class
- OtroValidator.class
- Prenda.class
- Prenda.xml
- PrendaFormController.class
- PrendaValidator.class
- ResultDenuncia.class
- Seguridad.xml
- TablaDelitoFaltaDenunciaController.class
- TablaDelitoFaltaPersonaJuridicaController.class
- TablaDenunciaMultiActionsController.class
- TablaDireccionController.class
- TablaDireccionPersonaNaturalController.class
- TablaPaginadaDenuncia.class
- TablaTelefonosController.class
- TablaVictimaCasoPolicial.class
- TarjetaBancaria.class
- TarjetaBancaria.xml

- TarjetaBancariaFormController.class
- TarjetaBancariaValidator.class
- Telefono.class
- Telefono.xml
- TelefonoCelular.class
- TelefonoCelular.xml
- TelefonoCelularFormController.class
- TelefonoCelularValidator.class
- TelefonoDaoImpl.class
- TelefonoFacadeImpl.class
- TelefonoManagerImpl.class
- TelefonoMultiActionController.class
- TelefonoValidator.class
- TransporteAereo.class
- TransporteAereo.xml
- TransporteAereoFormController.class
- TransporteAereoValidator.class
- TransporteMaritimo.class
- TransporteMaritimo.xml
- TransporteMaritimoFormController.class
- TransporteMaritimoValidator.class
- Vehiculo.class
- Vehiculo.xml
- VehiculoFormController.class
- VehiculoValidator.class

e.3) seguridad.jar

Propósito

Contenedor físico de las nuevas clases que modifican la seguridad original del framework Acegi, además de poseer nuevas funcionalidades que garantizan la comunicación entre aplicaciones.

e.4) web.xml

Propósito

Descriptor de despliegue para la aplicación Web.

3.3. Conclusiones

Los diferentes diagramas de componentes confeccionados en este capítulo, representan en términos de componentes y subsistemas de implementación, los elementos del modelo de diseño obtenidos en el capítulo anterior y establecen las dependencias entre uno y otro.

Conclusiones

El presente trabajo finaliza dando cumplimiento al objetivo general trazado de desarrollar una aplicación informática para gestionar el proceso de levantamiento de denuncias como parte del Sistema de Gestión Policial para la República Bolivariana de Venezuela. Además se concluye que:

- Se efectuó un estudio de las herramientas y tecnologías de desarrollo que se utilizaron en la confección de la solución, así como del proceso de levantamiento de denuncias.
- Se definió un diseño que modela el sistema y brinda soporte a todos los requisitos funcionales y no funcionales. Un aspecto de gran utilidad en la confección y concepción del diseño fue la comprensión y utilización de patrones, que permitieron aplicar buenas prácticas y brindar soluciones probadas a problemas comunes.
- Se estructuró un diagrama de componentes que representa la implementación del sistema en términos de componentes, quedando construida una aplicación capaz de garantizar rapidez y efectividad en el proceso de levantamiento de denuncias realizado por las dependencias policiales.

Recomendaciones

Realizar las pruebas a la versión operacional obtenida del Módulo de Denuncias, que permitan validar y evaluar la calidad del producto desarrollado.

Incorporar funcionalidades que permitan realizar el seguimiento a las denuncias efectuadas por los ciudadanos con el fin de tener un control de las investigaciones ejecutadas por los funcionarios así como obtener estadísticas de la efectividad del trabajo de los cuerpos policiales.

Añadir nuevos reportes relacionados con el proceso de levantamiento de denuncias de manera que el análisis de los mismos contribuya a perfeccionar la toma de decisiones para ayudar a combatir y prevenir el delito en la hermana República Bolivariana de Venezuela.

Referencias Bibliográficas

1. **Seguridad Ciudadana, Venezuela.** [Online] [Cited: Diciembre 12, 2007.] Disponible en: http://www.seguridadidl.org.pe/sistema/leysinasec.pdf.

2. **Monografías.com.** [Online] Monografias.com S.A. . [Cited: Febrero 4, 2007.] Disponible en: http://www.monografias.com/trabajos28/seguridad-ciudadana/seguridad-ciudadana.shtml.

3. **MIJ.** *Reportajes.htm.* 2006. nº. 17.

4. **GABALDÓN, LUIS GERARDO**. *Cfr.* p. 116. 1.

5. **ESCAMILLO, J. C.** Ministerio del Poder Popular para Relaciones de Interiores y Justicia. [Online] [Cited: Enero 7, 2008.] Disponible en: http://www.mpprij.gob.ve/spip.php?article=2071.

6. **Wikipedia.** [Online] [Cited: Diciembre 5, 2007.]. Disponible en: http://es.wikipedia.org/wiki/Software.

7. **Bustelo C, Amarilla R.** *Gestión del Conocimiento y Gestión de la Información.* 2001. VIII(34): 226-230 .

8. **Wikipedia.** ***Aplicaciones Web.*** [Online] [Cited: Diciembre 8, 2007.] Disponible en: http://es.wikipedia.org/wiki/Aplicaci%C3%B3n_web.

9. **Barzanallana, Rafael.** Universidad de Murcia. *Apuntes. Ingenieria del software. Sistemas Informaticos.* [Online] [Cited: Enero 7, 2008.] Disponible en: http://www.um.es/docencia/barzana/IAGP/Iagp02.pdf.

10. **JACOBSON, I. B., GRADY Y RUMBAUGH, JAMES**. *El Proceso Unificado de Desarrollo de Software.* La Habana : FÉLIX VARELA, 2004.

11. **Sanchez, María A. Mendoza**. Informatizate. [Online] Junio 7, 2004. [Cited: Enero 9, 2008.] Disponible en: http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html.

12. **Wikipedia.** ***Plataforma Java.*** [Online] [Cited: Febrero 5, 2008.] Disponible en: http://es.wikipedia.org/wiki/Plataforma_Java.

13. **FSF.** ***Fundación Software Libre.*** [Online] [Cited: Enero 8, 2008.] Disponible en: http://www.fsfla.org/svnwiki/legis/venezuela/3390.

14. **Allamaraju, Subrahmanyam, et al.** *Programación Java Server con J2EE.* 1.3. 1206.

15. **Wikipedia.** ***Herramienta CASE.*** [Online] [Cited: Febrero 4, 2008.] Disponible en: http://es.wikipedia.org/wiki/Herramienta_CASE.

16. **Vizcaíno, Aurora, García, Felix Oscar and Caballero, Ismael.** *Una Herramienta CASE para ADOO: Visual Paradigm.* [Documento] s.l. : UNIVERSIDAD DE CASTILLA-LA MANCHA.

17. **Visual Paradigm.** [Online] [Cited: Enero 23, 2008.] Disponible en: http://www.visual-paradigm.com/product/vpuml.

18. **Burbeck, Steve.** *Application Programming in Smalltalk-80: How to use Model-View-Controller.*

19. **Larman, Craig.** Introducción al análisis y diseño orientado a objetos. *UML y Patrones. Tomo I.*

20. **WM. *elwebmaster.com.*** [Online] [Cited: Enero 25, 2008.] Disponible en: http://www.elwebmaster.com/articulos/top-5-javascript-framework.

21. **C. Walls, R. Breidenbach**. *Spring in Action.* s.l. : Manning Publications, 2005.

22. **Clinton Begin, Brandon Goodin, Larry Meadors.** *iBatis in Action.* 2007. Manning Publications.

23. **Gutierrez, Juan.** Universidad de Valencia. *Eclipse (2.1) y Java.* [Online] 2004. [Cited: Febrero 10, 2008.] Disponible en: http://www.uv.es/~jgutierr/MySQL_Java/TutorialEclipse.pdf.

24. **García Puebla, Ivón.** Gestor de Contenidos. *Eclipse: una herramienta profesional al alcance de todos.* [Online] Agosto 2, 2005. [Cited: Febrero 11, 2008.] Disponible en: http://www.gui.uva.es/~laertes/nuke/index.php?option=com_content&task=view&id=56&Itemid=41.

Bibliografía

1. **Almaer, Dion**. *"The Dojo Toolkit in Practice".* 2006

2. **Becerril C., Francisco.** *Java a su alcance.* s.l. : McGRAW-HILL INTERAMERICANA EDITORES.

3. **C. Walls, R. Breidenbach**. *"Spring in Action", Manning Publications, 2005.*

4. **Catalogo de Antipatrones.** [En línea] [Citado el: 18 de 02 de 2008.] Disponible en: http://c2.com/cgi/wiki?AntiPatternsCatalog.

5. **Clinton Begin, Brandon Goodin, Larry Meadors.** *"iBATIS in Action", Manning Publications*.

6. **Constitución de la República Bolivariana de Venezuela.** [En línea] [Citado el: 16 de Diciembre de 2007.] Disponible en: http://www.venezuela-oas.org/Constitucion%20de%20Venezuela.htm.

7. **Craig Larman**. *UML y Patrones*, Introducción al análisis y diseño Orientado a Objetos.

8. **Darren Davison, Steven Devijver, Colin Yates**. *Expert Spring MVC and Web Flow.*

9. **DeMichiel**, L. *"Enterprise JavaBeans Specification, Version 2.1", Sun Microsystems, November 12, 2003*. Disponible en: http://java.sun.com/products/ejb/docs.html

10. **DEVX.** ***An Introduction to Antipatterns in Java Applications.*** [En línea] [Citado el: 02 de 03 de 2008.] Disponible en: http://www.devx.com/Java/Article/29162/1954.

11. **Gamma, Erich, y otros.** *Design Patterns*. Elements of Reusable Object Oriented Software, Adison-Wesley Longman,1995.

12. **Heilmann, Christian**. *"Beginning JavaScript with DOM Scripting and Ajax".* 2006

13. **Jacobson, I. and Booch, G. y Rumbaugh, J.** *"El Proceso Unificado de Desarrollo de software". 2000.* 2000.

14. **Java Anti-Patterns.** [En línea] [Citado el: 03 de 03 de 2008.] Disponible en: http://www.odi.ch/prog/design/newbies.php.

15. **Larman, Craig.** *Uml y Patrones.Introducción al análisis y diseño orientado a objetos*. [En línea] México, 1999. [Citado el: 24 de Enero de 2008.] Disponible en: http://bibliodoc.uci.cu/pdf/reg00061.pdf ISBN 970-17-0261-1

16. **VENEZUELA, G. D.** LEY DE COORDINACION DE SEGURIDAD CIUDADANA. 2001, nº Disponible en: http://www.mij.gov.ve/ley_coord_sc.htm.

17. **Visual Paradigm**. *Visual Paradigm for UML Model –Code-Deploy Platform*. [En línea][Citado el: 9 de Enero de 2008.] Disponible en: http://www.visual-paradigm.com/product/vpuml/ .

18. **Wikipedia.** ***La enciclopedia libre.*** [En línea] [Citado el: 03 de 03 de 2008.] Disponible en: http://es.wikipedia.org/wiki/Antipatr%C3%B3n_de_dise%C3%B1o .