Universidad de las Ciencias Informáticas

FACULTAD 2 “TELECOMUNICACIONES Y SEGURIDAD INFORMÁTICA”

Título: Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. Módulos: Administración y Mapas.

Trabajo de Diploma para optar por el título de
Ingeniero Informático

Autor: Carlos Manuel Gutíerrez Rodríguez.

Tutor: Ing. Liván Rodríguez Miranda.
Cotutor: Ing. Angela Gloria Gómez Peña.

Ciudad de La Habana, Junio 2011.
“Año 52 del Triunfo de la Revolución”.

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 _____.

Carlos Manuel Gutíerrez Rodríguez

_____

Firma del Autor

Ing. Liván Rodríguez Miranda

_____

Firma del Tutor

Ing. Angela Gloria Gómez Peña

_____

Firma del Cotutor

Datos de Contacto

Tutor:

Ing. Liván Rodríguez Miranda.

Graduado en la Universidad de las Ciencias Informáticas (UCI) como Ingeniero en Ciencias Informáticas en el año 2008. Profesor.

Email: lmiranda@uci.cu

Cotutor:

Ing. Angela Gloria Gómez Peña.

Graduada en la Universidad de las Ciencias Informáticas (UCI) como Ingeniera en Ciencias Informáticas en el año 2010. Recién Graduado en Adiestramiento.

Email: aggomez@uci.cu

DEDICATORIA

A mis padres por su guianza, cuidado y por anhelar este momento tanto como yo.
A mi esposa por haber contribuido a este gran sueño.

RESUMEN

En la actualidad se manifiesta un incremento considerable en el desarrollo de aplicaciones web orientadas a la gestión de información, esto proporciona un mejor funcionamiento y ayuda a la toma de decisiones y control de la organización.

A raíz de la necesidad de desarrollar un sistema informático, basado en software libre y con el objetivo de informatizar la recopilación, el procesamiento y almacenamiento de datos en las Coordinaciones Regionales (CR) de Prevención del Delito de la República Bolivariana de Venezuela, surge el presente trabajo, el cual muestra el desarrollo del Sistema de Gestión de Información de las Coordinaciones Regionales, específicamente en la gestión de la información referente a los módulos de Administración y Mapas. Para ello el autor desempeña los principales roles definidos por RUP en cada fase, además se utilizan herramientas y tecnologías de la plataforma J2EE que exponen tendencias de la programación web actual y otras herramientas como apoyo a la aplicación, además se emplea el estilo arquitectónico en tres capas y el mismo es implementado haciendo uso de los frameworks Dojo, Spring e Hibernate.

PALABRAS CLAVE: CR, Dojo, frameworks, Hibernate, módulos, RUP, software libre, Spring.

Índice

Introducción……1
Capítulo 1: Fundamentos Teóricos de la Investigación ……..4
1.1 Introducción ……4
1.2 Sistemas de Gestión de Información ……..4
1.3 Herramientas, Metodología y Tecnologías propuestas ……...5
1.4 Lenguajes de Programación ……...12
1.5 Plataforma de Desarrollo ……13
1.6 Marco de Trabajo o Frameworks ……...13
1.7 Tecnología para crear Reportes ……19
1.8 Conclusiones ……19
Capítulo 2: Diseño del sistema ……20
2.1 Introducción ……...20
2.2 Estilo Arquitectónico ……20
2.3 Patrones de Diseño ……24
2.4 Diagramas de Clases de Diseño ……28
2.5 Diagramas de Secuencia ……30
2.6 Conclusiones ……36
Capítulo 3: Implementación del sistema ……..37
3.1 Introducción ……37
3.2 Modelo de Implementación ……37
3.4 Diagrama de Componentes por Capas ……39
3.5 Código Fuente ……43
3.6 Validación ……46
3.7 Principales Interfaces de la Aplicación ……47
3.8 Conclusiones ……50
Capitulo 4: Pruebas al Sistema ……51
4.1 Introducción ……51
4.2 Pruebas de software ……51
4.3 Tipos de Pruebas ……52
4.4 Herramientas utilizadas ……61
4.5 Conclusiones ……67
Conclusiones ……..68
Recomendaciones ……..69
Referencia Bibliográfica ……70
Bibliografía ……..73
Anexos ……..75
Anexo 1: Arquitectura cliente servidor ……..75

Anexo 2: Esquema del patrón Controlador frontal ……..……...75
Anexo 3: Tabla de atributos y métodos pertenecientes al diagrama de clases de diseño ……76
Anexo 4: Lógica de validación aplicada ……..……82
Glosario de Términos ..……85

INTRODUCCIÓN

Con el triunfo del presidente Hugo Rafael Chávez Frías, la Revolución Bolivariana prioriza la agenda social, generando condiciones para la inclusión. La salud, la educación, el empleo, las viviendas, la seguridad ciudadana entre otros indicadores, comienzan a favorecer los sectores más pobres y excluidos, siendo estas variables macro-sociales elementos que impactan positivamente en la seguridad ciudadana y la prevención del delito.

En tal sentido el Ministerio del Poder Popular para Relaciones Interiores y Justicia (MPPRIJ), atendiendo a su misión de garantizar la seguridad ciudadana, mediante la proclamación de políticas dirigidas al resguardo de la paz pública, desarrollo territorial equilibrado y la estabilidad de la nación; promueve la formulación y funcionamiento del proyecto “Solución Integral para el Perfeccionamiento del Sistema de Prevención del Delito de la República Bolivariana de Venezuela”, partiendo de la premisa de que la seguridad ciudadana es una condición necesaria para el desarrollo humano, la cual es fundamental para desenvolverse en la vida cotidiana, con el menor nivel de amenaza a los derechos, la integridad personal y el disfrute de los bienes; así como también con el fin de brindar a las Coordinaciones Regionales (CRs) una herramienta que mejore su desempeño, y de esta forma garantizar una asistencia de mayor calidad al ciudadano.

Actualmente en dichas CRs, los procesos de gestión de información se realizan de forma manual, generando grandes volúmenes de información lo cual dificulta la organización y el control de la misma. Por otro lado la Dirección General de Prevención del Delito (DGPD) exige la creación de informes y el procesamiento de los datos recogidos en las CRs, los que son enviados en diferentes formatos, entorpeciendo el análisis de resultados finales. Esto afecta la toma de decisiones para la ejecución de los procesos internos así como la creación e implementación de otras alternativas que apoyen la labor preventiva.

Debido a esto, en el marco del convenio Cuba-Venezuela los directivos de la DGPD y la empresa cubana de software Albet contratan la realización de una herramienta informática. Posteriormente se realizó un levantamiento de requisitos, los cuales fueron aceptados, documentados y firmados por ambas partes.

El sistema está compuesto por fragmentos independientes, los cuales se especializan en funciones específicas, razón por la cual se hace imprescindible la gestión de la administración, con el objetivo de estandarizar la información y controlar el acceso a la misma.

Además, a petición del cliente se gestionan las imágenes correspondientes a la distribución geográfica de la República Bolivariana de Venezuela, permitiendo la posibilidad de observar de una forma detallada la cantidad de Consejos Comunales (CC) representados en las diferentes vistas.

Por lo antes expuesto, se plantea como **problema científico**: ¿Cómo garantizar el cumplimiento de los requisitos funcionales y no funcionales asociados a los módulos de Administración y Mapas del Sistema de Gestión de Información de las Coordinaciones Regionales?

El **objeto de estudio** determinado para el presente trabajo de diploma, se basa en la Gestión de Información en las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela y el **campo de acción** se enmarca en la Gestión de Información asociada al acceso y estandarización de la información y la representación cuantitativa de los Consejos Comunales.

Para dar solución al problema planteado, la investigación tiene como **objetivo general:** Desarrollar los módulos que gestionen la información referente a la administración y la gestión de imágenes de mapas del Sistema de Gestión de Información de las Coordinaciones Regionales.

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

- Realizar el diseño de las clases de los módulos: Administración y Mapas.
- Implementar los componentes correspondientes a los módulos: Administración y Mapas.
- Validar funcionalmente los módulos: Administración y Mapas.

Como fundamento de los objetivos específicos se planifican para su ejecución las siguientes **tareas de investigación**:

- Estudio y descripción de las herramientas y tecnologías seleccionadas para el desarrollo del Sistema Informático de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela.
- Realización de los Diagramas de clases de diseño de los módulos: Administración y Mapas.
- Realización de los Diagramas de secuencia del diseño de los módulos: Administración y Mapas.
- Realización de los Diagramas de componentes de los módulos: Administración y Mapas.
- Implementación de los componentes de los módulos: Administración y Mapas.
- Realización de pruebas funcionales a los módulos: Administración y Mapas.

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

Capítulo 1 "*Fundamentación Teórica*", este capítulo comprende el estudio del estado del arte acerca de los elementos fundamentales a tener en cuenta para la solución del problema planteado, se define la metodología de desarrollo, lenguaje de programación y plataforma de desarrollo entre otros aspectos.

Capítulo 2 "*Diseño del Sistema*", este capítulo describe la propuesta de solución para el problema planteado y el diseño de la aplicación. Se muestran los diagramas de clases del diseño y los diagramas de secuencias.

Capítulo 3 "*Implementación del Sistema*", en este capítulo se presentan los diagramas de componentes que se definieron durante la implementación de la aplicación, y además se muestran algunas imágenes de la misma.

Capítulo 4 "*Pruebas al Sistema*", en este capítulo se muestran las pruebas realizadas sobre la aplicación para comprobar sus funcionalidades.

CAPÍTULO 1

FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se abordan algunos conceptos necesarios que permiten dar a conocer qué es un sistema de gestión de información, para el desarrollo del Sistema de Gestión de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. También se hace un estudio de las tecnologías, herramientas y metodologías de desarrollo a utilizar en el diseño e implementación del sistema.

1.2 Sistemas de gestión de información.

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

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

Sistemas de gestión de información: "*Puede definirse como un conjunto de componentes interrelacionados que permiten capturar, procesar, almacenar y distribuir información para apoyar la toma de decisiones y el control de una institución, además de ayudar a dichos directivos y al personal a*

analizar problemas, visualizar cuestiones complejas y crear nuevos productos en un ambiente intensivo de información". [2]

En países como México, Argentina y Buenos Aires se han realizado sistemas de gestión de información con la finalidad de prevenir el delito, ejemplo de ello es el Sistema de Información para la Prevención Comunitaria del Delito y la Violencia (SIPREC), este es un programa de diagnóstico, prevención del delito y situaciones de violencia y conflicto en Buenos Aires.

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

1.3 Metodología, herramientas y tecnologías

Como antecedente se tienen los trabajos de diplomas del curso anterior, en los cuales se definieron las herramientas, tecnologías, metodología y la arquitectura que sigue el presente trabajo de diploma, entre estos trabajos de diplomas se encuentran: "Propuesta de arquitectura para el Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela" de los autores Eneysi Osorio y Asdrúbal Torres y "Sistema Informático de Gestión de Información de las Comunidades, los Consejos Comunales, Citas y Denuncias en las Coordinaciones Regionales" de los autores Osmany Cordero y Francisco Montada.

1.3.1 Metodología de desarrollo de software

Una metodología es un conjunto de procedimientos que permiten producir y mantener un producto de software, definiendo una serie de pasos a seguir para obtener un software de calidad. Las metodologías se utilizan con el objetivo de dar solución a problemas existentes en la producción de software. Estas abarcan procedimientos, técnicas, documentación y herramientas que se utilizan en la creación de un producto de software [3]. Sus procesos se descomponen a nivel de tareas o actividades elementales, donde cada tarea está identificada por un procedimiento que define la forma de llevarla a cabo.

Rational Unified Process (RUP)

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

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

Fig.1.1 Proceso de desarrollo de software.

Roles y artefactos

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

Análisis y diseño

- **Diseñador:** es el responsable del diseño del sistema que incluye: las restricciones de los requisitos, la arquitectura y el proceso de desarrollo del proyecto.

Artefactos

- **Diagrama de clases del diseño:** es una descripción de un grupo de objetos, que comparten las mismas responsabilidades, relaciones, operaciones, atributos y semánticas. A diferencia del modelo del dominio, un diagrama de clases de diseño muestra definiciones de entidades de software más que conceptos del mundo real.
- **Diagrama de secuencia:** es una representación que muestra un determinado escenario de un caso de uso, los eventos generados por actores externos, su orden y los eventos internos del sistema, contiene detalles de la implementación del escenario, incluyendo los objetos y clases que se utilizan para implementar el mismo, y los diversos mensajes intercambiados entre los objetos.

Implementación

- **Desarrollador:** es responsable de desarrollar y probar los componentes de acuerdo con los estándares adoptados por el proyecto, para la integración en subsistemas más grandes. Cuando los componentes de prueba, tales como drivers o las partes se crean para apoyar la prueba, el desarrollador es también responsable de probar los componentes y los subsistemas correspondientes.

Artefactos

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

Prueba

- **Analista de pruebas:** es el responsable de identificar y definir las pruebas requeridas, monitorear el progreso de las mismas y el resultado en cada ciclo de pruebas, evaluando la calidad total experimentada como un resultado de las actividades de prueba. Este rol lleva la responsabilidad de representar apropiadamente las necesidades de los trabajadores que no tienen representación regular y directa en el proyecto.
- **Probador:** conduce las pruebas necesarias y el registro del resultado de las mismas.

Artefactos

- **Registro de pruebas:** Una colección de resultados capturados durante una ejecución única de una o más pruebas.
- **Resultados de pruebas:** Este artefacto resume el análisis de uno o más registros de prueba y solicitudes de cambio, proporcionando una valoración relativamente detallada de la calidad de los elementos de destino de la prueba y el estado del esfuerzo de prueba.

1.3.2 Herramientas CASE

Las **Herramientas CASE** (**C**omputer **A**ided **S**oftware **E**ngineering, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software, reduciendo el coste de las mismas en términos de tiempo y dinero. Estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software, en tareas como: el proceso de realizar un diseño del proyecto, cálculo de costes, implementación automática de parte del código con el diseño dado, compilación automática, documentación o detección de errores entre otras. [3]

Visual Paradigm

Visual Paradigm es una herramienta que provee soporte para la generación de código, tiene integración con diversos Entornos de Desarrollo Integrados (IDE's) como NetBeans (de Sun

Microsystems), JDeveloper (de Oracle), Eclipse (de IBM), JBuilder (de Borland), así como la posibilidad de realizarse la ingeniería inversa para aplicaciones realizadas en JAVA, .NET, XML e Hibernate.
Dentro de sus características podemos citar que es portable y posee gran factibilidad de uso. Utiliza UML como lenguaje de modelado y soporta el ciclo de vida completo de desarrollo de software, ayuda a una rápida construcción de aplicaciones de calidad y a un menor coste [5].

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

Teniendo en cuenta las características antes mencionadas de Visual Paradigm, especialmente por su buena integración con los IDE's para el desarrollo con Java, ser multiplataforma y la ventaja de presentar una interfaz de usuario de fácil uso, además de posibilitar la realización de diagramas, la generación de los diferentes artefactos durante el desarrollo del software y realizar un control de las versiones durante todo el ciclo de trabajo, se decide utilizar como herramienta de modelado.

1.3.3 Unified Modeling Language (UML)

Es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como RUP), pero no especifica en sí mismo qué metodología o proceso usar. No obstante, es necesario además de modelar el software, definir las herramientas y tecnologías que deben utilizarse para la confección del software. [8]

1.3.4 Herramientas para el almacenamiento de datos

Sistema gestor de bases de datos

Los Sistemas de Gestión de Bases de Datos (Database Management System, DBMS) son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan, los cuales proveen a los usuarios de los medios necesarios para el proceso de definición, construcción, recuperación y manipulación de la base de datos, proporcionando un

acceso controlado y eficiente a la misma y asegurando su confidencialidad, seguridad, atomicidad, independencia física y lógica de los datos.

PostgreSQL

PostgreSQL es un avanzado sistema de base de datos relacional orientado a objetos y libre. Incluye características propias de la orientación a objetos, como puede ser: la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. A pesar de esto, PostgreSQL no es un sistema de gestión de bases de datos puramente orientado a objetos [6].

Este gestor se caracteriza principalmente por poseer alta concurrencia, mediante un sistema denominado Acceso Concurrente Multiversión (MVCC), PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos.

Por todo lo anteriormente expuesto se decide utilizar como Sistema Gestor de Bases de Datos (SGBD) PostgreSQL por presentar la característica de ser el gestor de bases de datos de código abierto más avanzado actualmente y liberado bajo licencia BSD (Berkeley Software Distribution), es decir; sus potencialidades están en constante perfeccionamiento, permitiendo su uso y distribución sin costo, además, continuamente se elimina y mejora cualquier brecha de seguridad que pueda aparecer, debido a que sus usuarios pueden acceder a su código y modificarlo a sus necesidades. En el caso que se esté trabajando con sistemas serios, la integridad referencial de los datos es muy buena, se pueden crear funciones complejas para validar los datos.

PgAdmin

PgAdmin es una herramienta de propósito general para diseñar, mantener, y administrar las bases de datos de Postgres. Funciona bajo Windows 95/98 y NT. Es multiplataforma y puede funcionar en sistemas operativos como: GNU\Linux, Mac, Windows y Solaris. Además es un software libre y permite desde ejecución de consultas SQL simples hasta la elaboración de bases de datos complejas, con el apoyo de las últimas características de PostgreSQL. El software es liberado con un instalador y no requiere ningún controlador adicional para comunicarse con el servidor de base de datos. [7]

1.3.5 Herramientas de desarrollo (IDE)

Un Entorno de Desarrollo Integrado o *Integrated Development Environment* (IDE) es un programa compuesto por un conjunto de herramientas de programación. Generalmente incluyen en una misma suite un buen editor de código, enlace transparente a compiladores, un depurador y un constructor de interfaz gráfica. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Eclipse

Eclipse es un Entorno de Desarrollo Integrado (IDE) debido a que provee herramientas para administrar áreas de trabajo (workspaces), para construir, lanzar y depurar aplicaciones, para compartir artefactos con un equipo y versionar el código fuente. Es multiplataforma, debido a que se ejecuta en gran cantidad de sistemas operativos incluyendo Windows y Linux, además es accesible ya que su diseño le permite ser extendido fácilmente por terceras partes. Emplea módulos (plug-in) para proporcionar toda su funcionalidad, a diferencia de otros entornos donde las funcionalidades están todas incluidas, las necesite el usuario o no. [8]

Para la realización del software se utilizará el Eclipse 3.5.0, por ser una herramienta de código abierto y por su factibilidad de uso, además de considerarse lo suficientemente estable y rápido; su diseño global permite tener las herramientas que necesitan los programadores inmediatamente al alcance de sus manos. Es fácilmente integrable con la herramienta CASE Visual Paradigm y soporta perfectamente la plataforma de desarrollo Java Enterprise Edition (JEE).

1.3.6 Servidor Web

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

Apache Tomcat 6.0.20

Para hospedar el sistema se utilizará el Apache Tomcat como servidor web y contenedor de servlet y jsp. El Apache Tomcat es un software de código abierto implementado para las tecnologías Java Servlet y Java ServerPages. Apache Tomcat es desarrollado en un entorno abierto y participativo y publicado bajo la licencia del software de Apache.

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

1.4 Lenguaje de programación

Un lenguaje de programación es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que éste pueda comunicarse con los dispositivos hardware y software existentes. [9]

Java

Java es un lenguaje de programación de alto nivel y entre sus principales características se encuentran:

- **Simple**: Elimina la complejidad de los lenguajes como "C" y da paso al contexto de los lenguajes modernos orientados a objetos.
- **Orientado a objetos:** Soporta las características esenciales del paradigma de la programación orientada a objetos: encapsulamiento, herencia y polimorfismo.
- **Robusto:** Elimina el uso de apuntadores para referenciar áreas de memoria, además, libera al desarrollador de la necesidad de desalojar la memoria que la aplicación ya no usa. También requiere la declaración explícita tanto de los tipos de datos como de los métodos.
- **Multiplataforma:** El mismo código Java que funciona en un sistema operativo, funciona en cualquier otro que tenga instalada la máquina virtual de Java.
- **Multitareas:** Permite la ejecución concurrente de varios procesos ligeros o hilos de ejecución.

Teniendo en cuenta las características antes mencionadas de Java, especialmente por ser multiplataforma y sumamente flexible, además de permitir la creación de programas modulares y de códigos reutilizables, se decide utilizar como lenguaje de programación.

1.5 Plataforma de desarrollo

Se refiere al ambiente de hardware y software donde un programa se ejecuta, por ejemplo, plataformas como GNU\Linux, Solaris, Windows 2003 y MacOS. En este caso se estudia la plataforma Java, ya que en el epígrafe anterior se escogió Java como lenguaje de programación. Esta se diferencia de las demás, ya que es sólo de software y se ejecuta sobre otras de hardware. [11]

Java Plataform Enterprise Edition (Java EE)

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

Java EE configura algunas especificaciones únicas para componentes, éstas incluyen: Enterprise JavaBeans, servlets, Java Server Pages y varias tecnologías de servicios web. Esto permite al desarrollador crear una aplicación de empresa portable entre plataformas, y a la vez que sea integrable con tecnologías anteriores. Otros beneficios añadidos son, el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes en lugar de en tareas de mantenimiento de bajo nivel. [11

1.6 Marco de trabajo o frameworks

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

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

Spring 3.0.2.

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

Es un framework basado en la Inversión de Control (IoC) y en la Programación Orientada a Aspectos (AOP). Se distribuye de forma libre y su código es abierto. Ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar. La simplificación del desarrollo de aplicaciones y de sus respectivas pruebas es una de las claves de su éxito. [14]

Arquitectura de Spring

Spring es un framework modular que cuenta con una arquitectura dividida en acerca de veinte módulos integrados en capas (Fig. 1.2), lo cual permite tomar y ocupar únicamente las partes que interesen para el proyecto y juntarlas con gran libertad. [16]

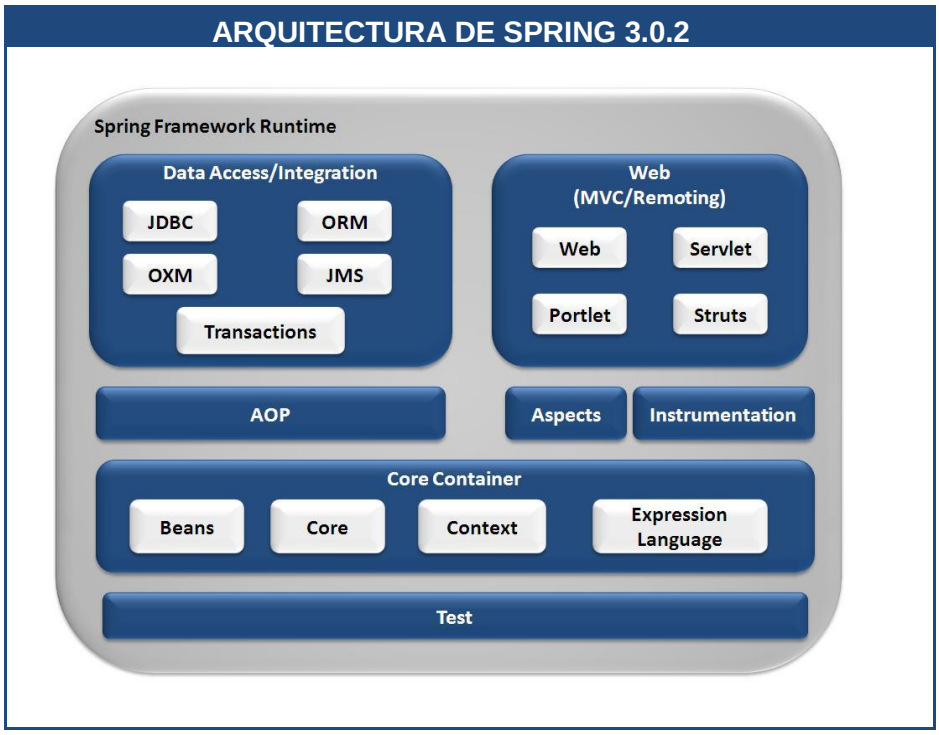

Fig.1.2 Arquitectura de Spring 3.0.2.

Capa Core Container

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

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

Capa Acceso a Datos/Integración

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

- El módulo **JDBC** proporciona una abstracción del modelo JDBC y elimina la necesidad de usar el JDBC básico que proporciona JAVA para acceder a la base de datos y controlar los errores.

- El módulo de **ORM** proporciona una capa de integración para las API's de mapeo entre objetos y el modelo relacional. Incluye integración con Hibernate, Java Persistence API (JPA), Java Data Objects (JDO) e iBatis. Este módulo permite integrar los anteriores frameworks con la funcionalidad y las características que ofrecen Spring.

Capa Web

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

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

Capa AOP (Aspect-Oriented Programming)

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

Capa Test.

La capa de prueba apoya el examen de los componentes de Spring con JUnit. Este es un marco simple para escribir pruebas repetibles. Proporciona carga constante de ApplicationContexts y el almacenamiento en caché de los contextos. También proporciona objetos mock que se pueden utilizar para probar el código en forma aislada.

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

A continuación, una representación gráfica del funcionamiento del MVC en Spring (Fig.1.3)

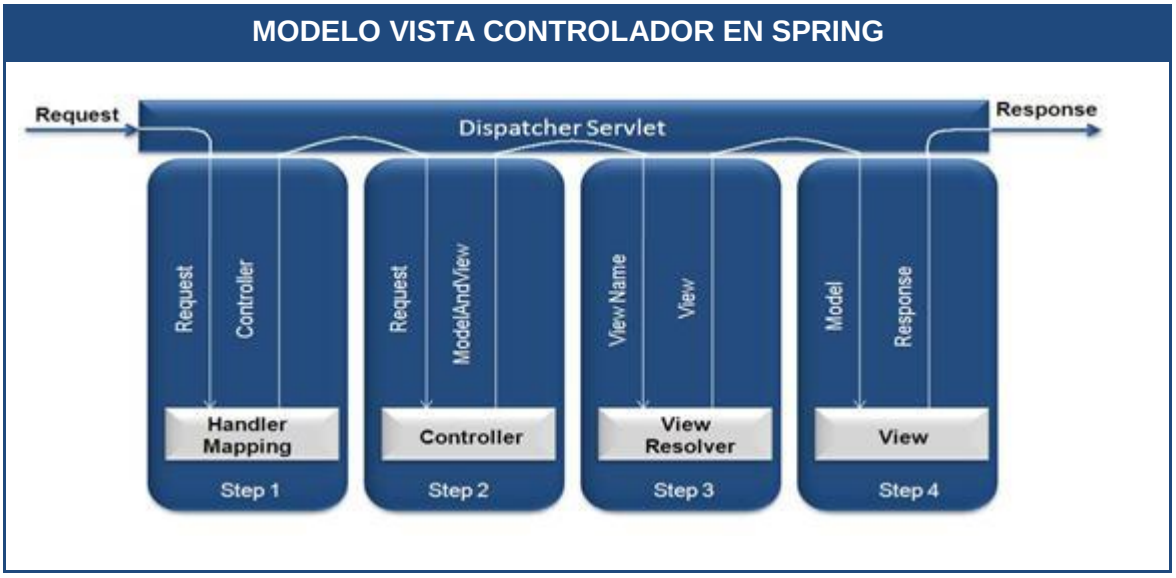

Fig.1.3 MVC en Spring

1.6.2 Framework Javascript

Dojo Toolkit 1.4.1

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

Dojo cuenta con librerías dojo, dijit y dojox. Se puede integrar fácilmente con Spring, framework de aplicación que se utilizará para desarrollar el sistema. [15]

A continuación, una breve descripción de las librerías mencionadas:

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

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

1.6.3 Framework Hibernate

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

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

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

1.7 Tecnología para crear reportes

¿Qué es un reporte?

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

JasperReport

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

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

1.8 Conclusiones

- Se describió la herramienta CASE, Visual Paradigm, la cual se utilizará para el modelado, así como materializar otras funcionalidades que este ofrece.
- El lenguaje de programación que se describió fue Java, pues es un lenguaje de programación Orientado a Objetos, que permite implementar multi-hilos, función necesaria para el presente software.
- El IDE de desarrollo que se describió fue el Eclipse en su versión 3.5.0. Es multiplataforma, lo cual es convenientemente cómodo. Además, posee otras características que lo convierten en una herramienta eficiente.

CAPÍTULO 2

DISEÑO DEL SISTEMA

2.1 Introducción.

En este capítulo se describe la solución del sistema a través del diseño de la aplicación, donde los requisitos se traducen a una especificación que describe cómo implementar el sistema, se realizan los diagramas de clases de diseño y de secuencias. Además se detallan los patrones de diseño y estilos arquitectónicos utilizados.

2.2 Estilo arquitectónico

Perry y Wolf establecen el razonamiento sobre estilos de arquitectura como uno de los aspectos fundamentales de la disciplina Arquitectura de Software. Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales. [18]

El estilo arquitectónico define las reglas generales de organización en términos de un patrón y las restricciones en la forma y la estructura de un grupo de sistemas de software. Los mismos sirven para sintetizar estructuras de soluciones, así como definir los patrones posibles de las aplicaciones. Estos estilos precisan qué forma tienen los componentes, qué forma tiene la comunicación entre esos componentes y qué restricciones se ponen a esa comunicación. Definen además, tanto un vocabulario de tipos de componentes y conectores, como un conjunto de restricciones sobre cómo combinar esos componentes y conectores.

¿Qué es un patrón de arquitectura?

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

Partiendo de la existencia de múltiples estilos arquitectónicos para la realización del Sistema Informático de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela se decidió utilizar el estilo arquitectónico: Arquitectura en Capas (Arquitectura en tres capas) perteneciente a la familia de estilos arquitectónicos de Llamada y Retorno, basándose en las características y peculiaridades del sistema a desarrollar. [20]

Esta arquitectura constituye una especialización de la arquitectura cliente-servidor, ver Anexo 1, donde la carga se divide en tres partes (Fig. 2.1) con un reparto claro de funciones. El desarrollo de cada paquete del Sistema Informático de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela responde a un modelo multicapas donde cada capa tiene funcionalidades y objetivos precisos, así su implementación se encuentra desacoplada de la programación de cualquier otra y la comunicación con una capa inferior ocurre a través de interfaces.

Fig. 2.1 Arquitectura en tres capas.

A continuación se realiza una breve descripción de cada una de las capas por las que está compuesto el sistema de acuerdo a la figura anterior.

2.2.1 Capa de acceso a datos

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

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

2.2.2 Capa de lógica de negocio

La capa de negocio define e implementa las funcionalidades que responden directamente a los requisitos de manera que se conserve la integridad del sistema y de los datos. Está constituida por los servicios y la fachada. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar datos de él.

2.2.3 Capa de presentación

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

Es importante destacar que en esta capa se encuentra presente el patrón arquitectónico **Modelo-Vista-Controlador (MVC)** el cual separa la presentación de los datos, ya que Spring lo utiliza en su funcionamiento como se abordó en el *Capítulo 1, epígrafe 1.6*. Este modelo se implementa como se describe a continuación:

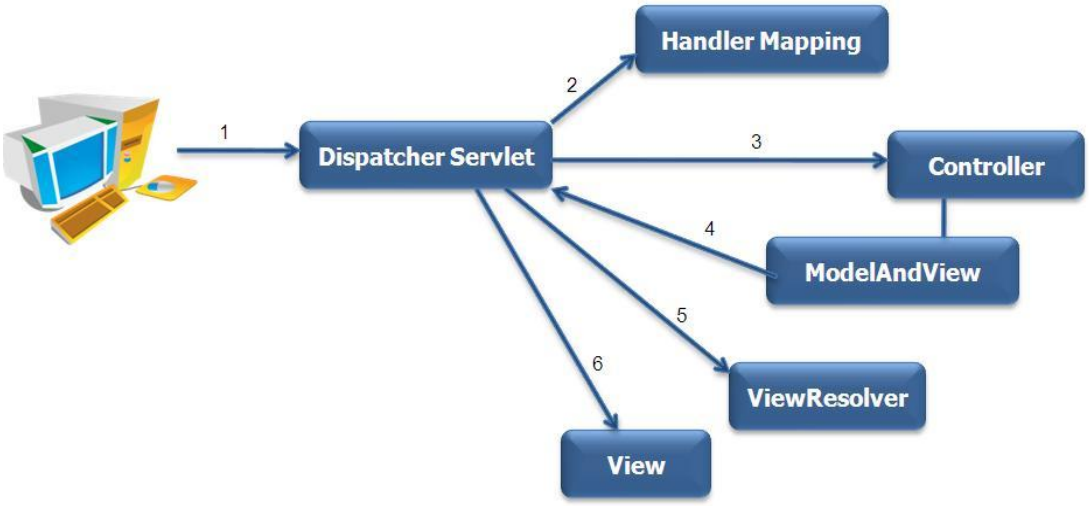

Fig. 2.2 Patrón MVC en Spring.

A grandes rasgos el patrón MVC en Spring funciona de la siguiente manera:

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

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

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

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

ModelAndView: Un objeto de esta clase engloba los datos a mostrar (el modelo, **Model**) en un objeto de la clase **Map** y el nombre de la vista que mostrará dichos datos, este nombre es en general un alias, o sea, no apunta directamente a un archivo físico (JSP).
ViewResolver: Esta clase es utilizada para resolver a partir del nombre de la vista, el alias contenido en el **ModelAndView**, la vista física que será dibujada (rendered) con los datos que le son pasados. Se tienen varias opciones a utilizar, independientes o combinadas y que son configuradas en archivos XML.
View: Este término se refiere las vistas físicas, básicamente a los archivos JSP, PDF, XLS.

2.3 Patrones de diseño

¿Qué es un patrón de diseño?

Los patrones de diseño son una solución probada para un problema general de diseño, en un contexto determinado. Encierran la experiencia que programadores e ingenieros han adquirido en la solución de problemas comunes. Los patrones de diseño no son fáciles de entender, pero una vez entendido su funcionamiento, los diseños serán mucho más flexibles, modulares y reutilizables.

Un patrón de diseño no es más que una solución estándar para un problema común de programación. Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios. Un proyecto o estructura de implementación que logra una finalidad determinada. Un lenguaje de programación de alto nivel. Una manera más práctica de describir ciertos aspectos de la organización de un programa. Conexiones entre componentes de programas. [21]

Benefician la documentación y el mantenimiento de los sistemas pues proveen una especificación de los objetos, las clases y sus relaciones. Es importante notar que un patrón de diseño representa experiencia y conocimiento. No es software ejecutable por lo que, cada vez que se use, debe ser implementado nuevamente. Los patrones a los que se hace referencia en este documento por su utilización en la solución técnica son:

2.3.1 Fachada

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

El uso de este patrón en el sistema se muestra en las clases de los paquetes **vnz.mpprij.prevencion.cr.admin.facade** las cuales sirven de fachada entre las clases de los paquetes **vnz.mpprij.prevencion.cr.admin.web** y **vnz.mpprij.prevencion.cr.admin.service**. El fragmento de diagrama de secuencia que se muestra en la (Fig. 2.2) correspondiente al caso de uso “Registrar Usuario” es un ejemplo de lo antes expuesto.

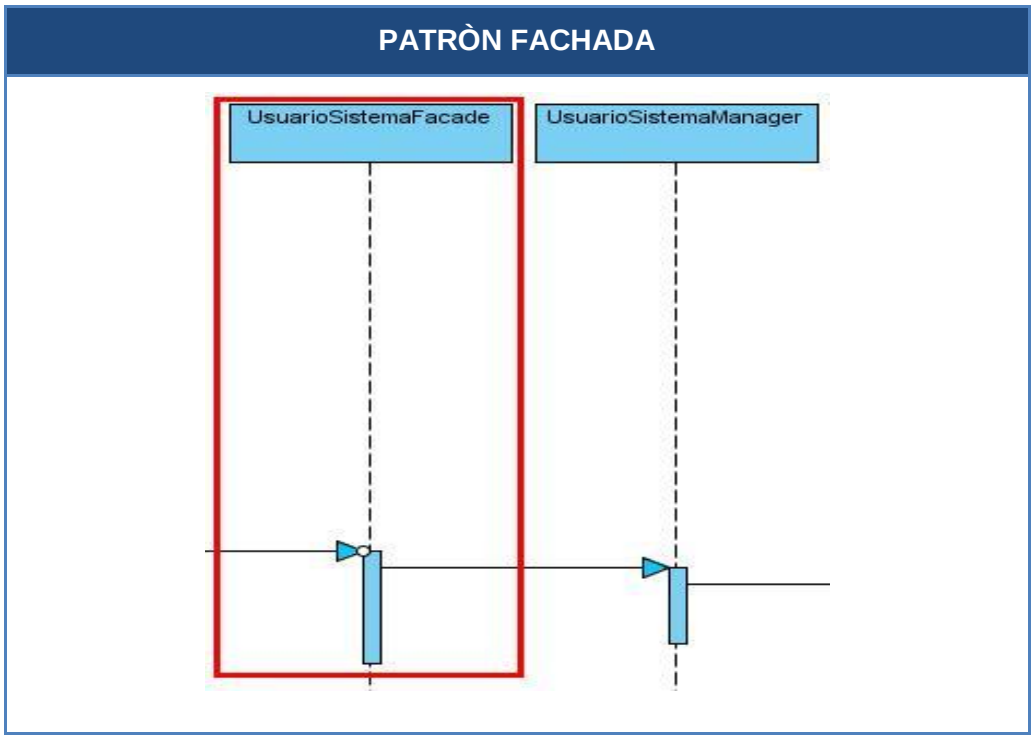

Fig. 2.3 Ejemplo del Patrón Fachada.

2.3.2 Data Access Object (DAO)

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

El uso de este patrón en el sistema se muestra en la capa de Acceso a Datos donde se encuentra un DAO Genérico encargado de realizar la gestión de los datos entre las clases que contienen la lógica de negocio (servicios) y el ORM Hibernate. El fragmento de diagrama de secuencia que se muestra en la (Fig. 2.3) correspondiente al caso de uso “Registrar Usuario” es un ejemplo de lo antes expuesto.

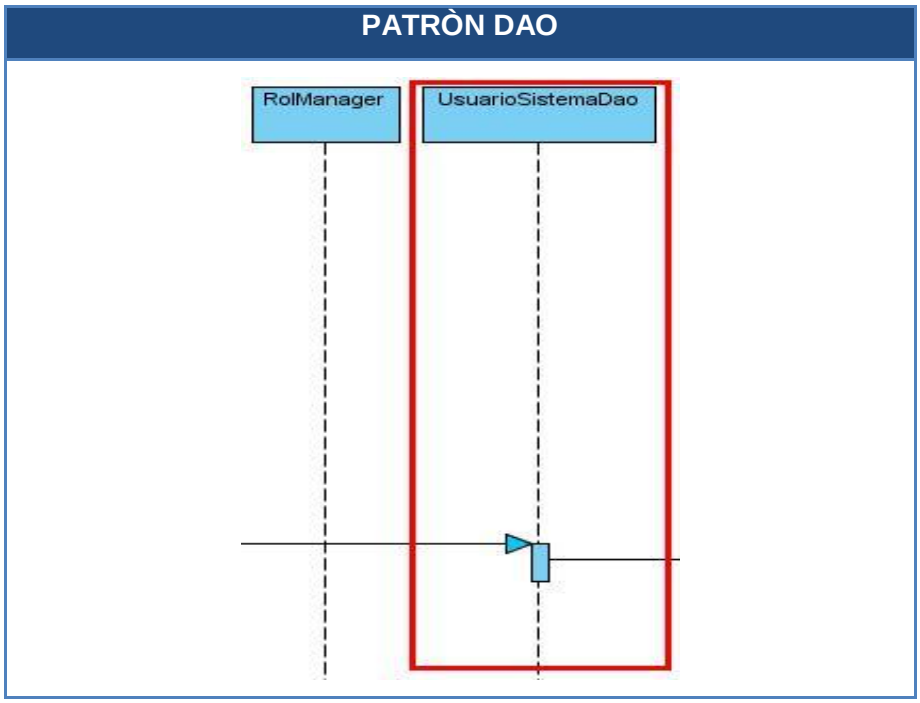

Fig. 2.4 Ejemplo del Patrón DAO.

2.3.3 Controller (Controlador)

Este patrón propone asignar la responsabilidad de controlar el flujo de eventos de un sistema, a clases específicas llamadas controladores. Los controladores no ejecutan las tareas sino que las delegan en otras clases, con las que mantiene un modelo de alta cohesión. En el sistema las clases a las cuales se les asignan estas responsabilidades se encuentran dentro de paquetes con la siguiente estructura

vnz.mpprij.prevencion.cr.admin.web y dentro de esta están los controladores. A modo de ejemplo se escoge la clase UsuarioMultiActionController.java, esta clase extiende del controlador AbstractController del framework Spring y se encarga de controlar la petición de visualizar una lista de usuarios, manipulando los datos (el identificador del programa) para en dependencia de estos, delegarlos a las clases responsables de ejecutar la acción requerida; devolviéndole luego a la vista física la JSP y los datos que serán mostrados al usuario.

2.3.4 Front-Controller (Controlador Frontal)

El patrón propone utilizar un controlador como el punto inicial de contacto para manejar las peticiones del usuario en una aplicación. El controlador maneja el control de peticiones, incluyendo la invocación de los servicios de seguridad como la autentificación y autorización, la elección de una vista apropiada, el manejo de errores, y el control de la selección de estrategias de creación de contenido. Este patrón es utilizado por Spring MVC a través del DispatcherServlet que se encuentra configurado en el **dispatcher-servlet, v**er Anexo 2. Un ejemplo de cómo funciona este patrón en el sistema sería el siguiente; si el usuario decide visualizar en el navegador la página gestionar usuarios, esta petición (la URL usuario.list) es recibida por el **dispatcher-servlet,** este posee un mecanismo para determinar cual controlador maneja esta petición (UsuarioMultiActionController en este caso), luego este controlador toma la petición y ejecuta la tarea (en realidad la delega a otras clases), devolviendo un ModelAndView al **dispatcher-servlet**, el que se encarga de despachar la petición a la Vista (usuario.list.jsp).

2.3.5 Inyección de dependencias

Es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase quien cree el objeto. Este radica en resolver las dependencias de cada clase (atributos) generando los objetos cuando se arranca la aplicación y luego inyectarlos en los demás objetos que los necesiten a través de métodos set o bien a través del constructor, pero estos objetos se instancian una vez, se guardan en una factoría y se comparten por todos los usuarios (al menos en el caso de Spring) y evitando tener que andar extendiendo clases. En el sistema está presente su aplicación ya que a todos los controladores se le inyectan las fachadas correspondientes para su funcionamiento mediante el uso de la anotación @Resource la cual provee de atributos a clases mediante los métodos set, de esta misma

manera se soluciona la dependencia entre las fachadas y sus servicios, dependientes estos últimos del DAO Genérico.

2.3.6 Alta Cohesión

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

2.3.7 Bajo Acoplamiento

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

2.4 Diagramas de clases de diseño.

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

A continuación se muestra en la (Fig. 2.4) el Diagrama de Clases de Diseño para el Caso de Uso del Sistema (CUS) Gestionar listado de usuarios perteneciente al Módulo Administración, dicho diagrama nos da una vista detallada de la ubicación de las clases por paquetes en las diferentes capas, para observar los atributos y métodos que pertenecen a cada clase, ver Anexo 3.

DIAGRAMA DE CLASES DE DISEÑO POR PAQUETES: CUS GESTIONAR LISTADO DE USUARIOS

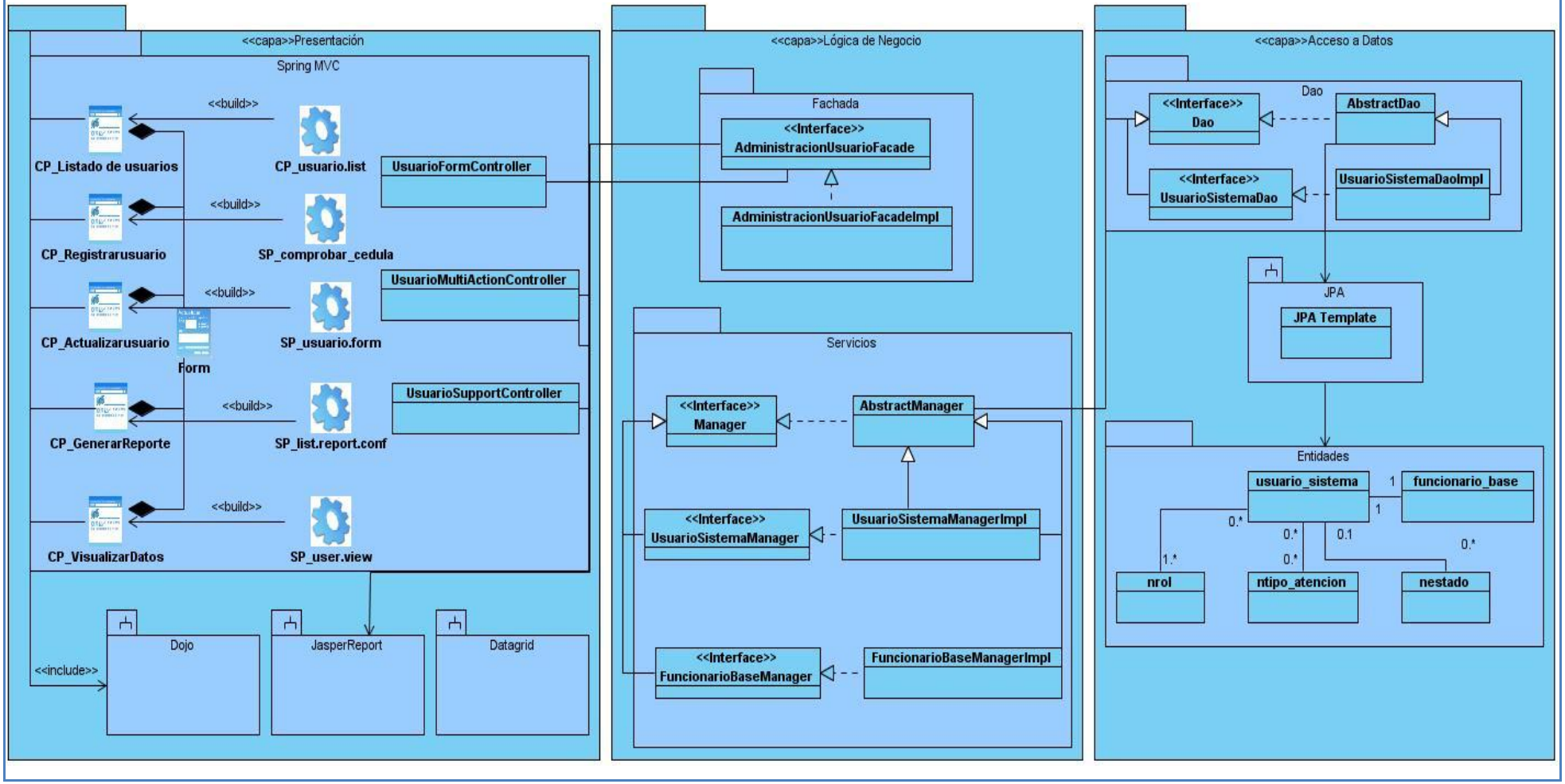

Fig. 2.5 Diagrama de clases de diseño por paquetes del CUS Gestionar listado de usuarios.

2.5 Diagramas de secuencia

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

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

A continuación se muestran los diagramas de secuencia correspondientes a los escenarios: Registrar Usuario (Fig. 2.5), Actualizar Usuario (Fig. 2.6), Visualizar Usuario (Fig. 2.7), Generar Reporte (Fig. 2.8) y Buscar Usuario (Fig. 2.8), todos del CUS Gestionar listado de usuarios.

DIAGRAMA DE SECUENCIA: REGISTRAR USUARIO

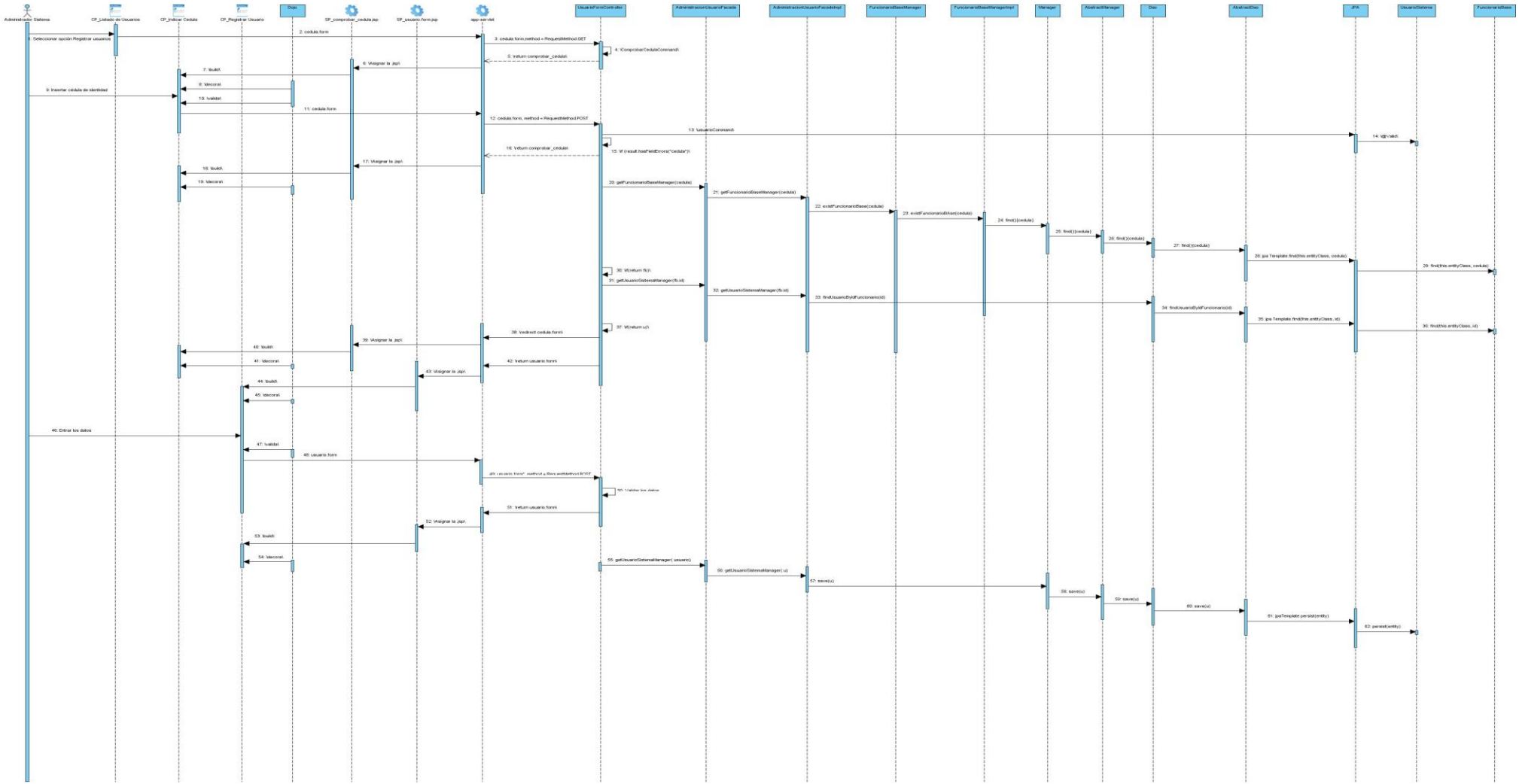

Fig. 2.5 Diagrama de secuencia del CUS Registrar usuario.

DIAGRAMA DE SECUENCIA: ACTUALIZAR USUARIO

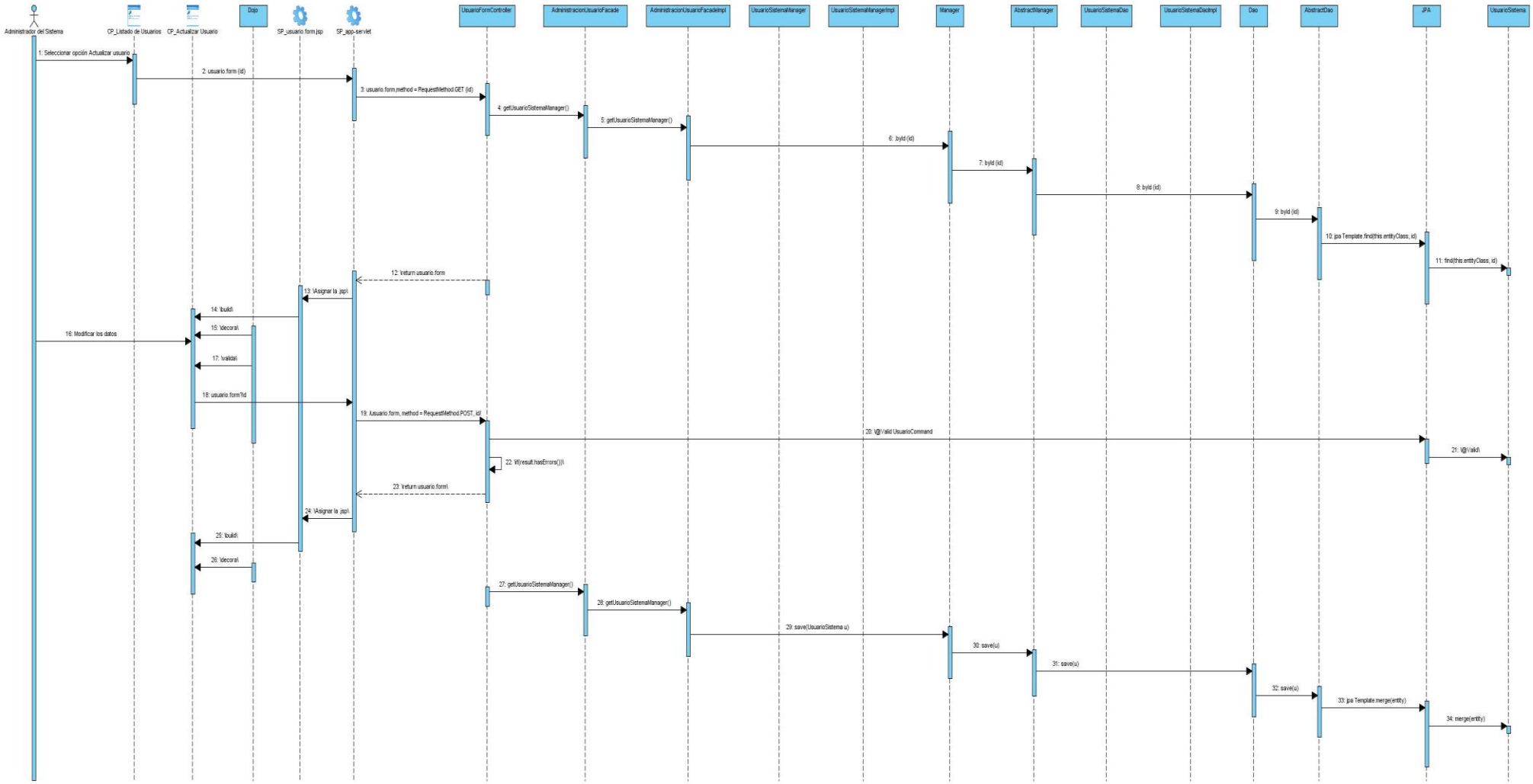

Fig. 2.6 Diagrama de secuencia del CUS Actualizar usuario.

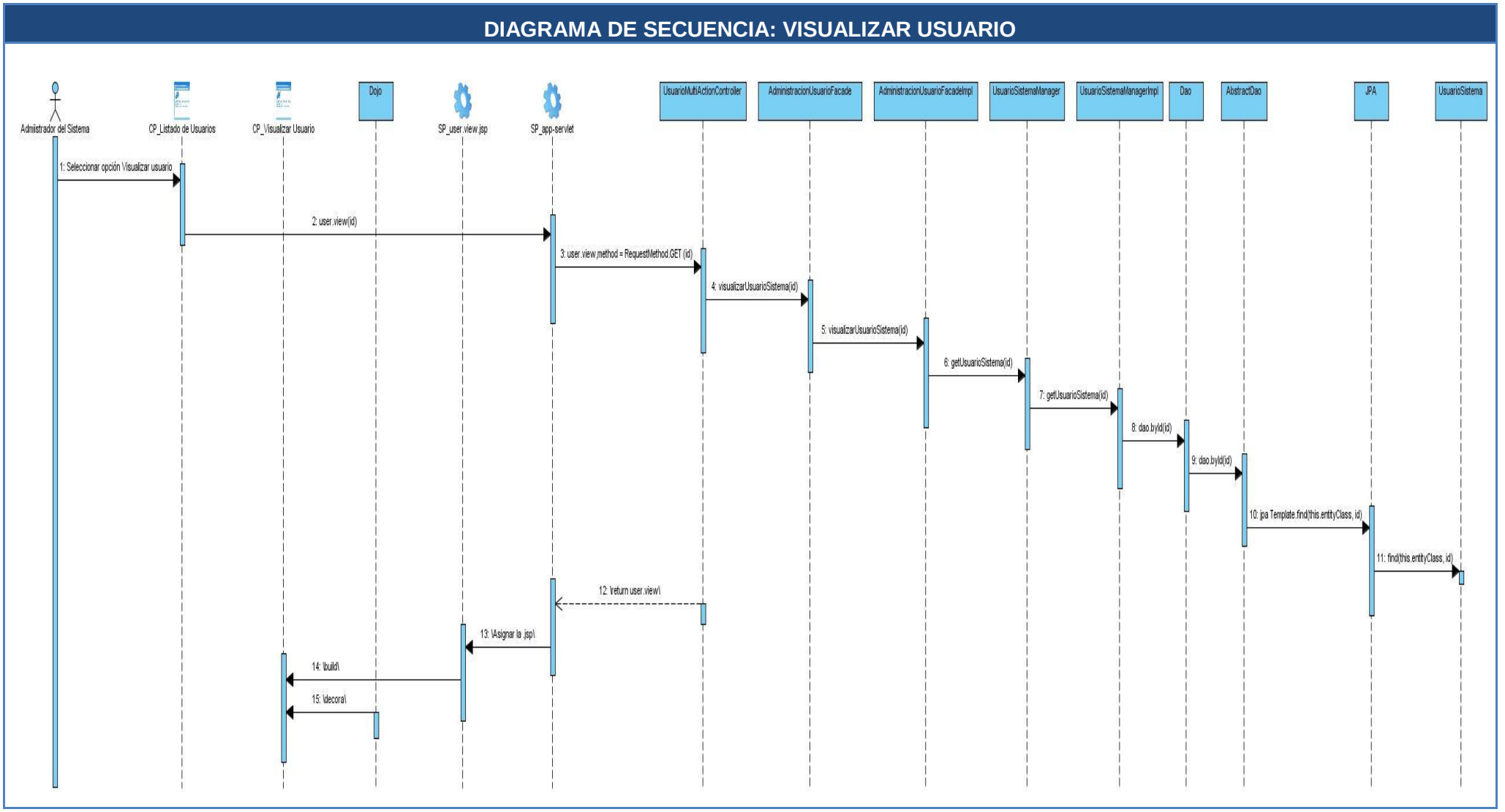

Fig. 2.7 Diagrama de secuencia del CUS Visualizar usuario.

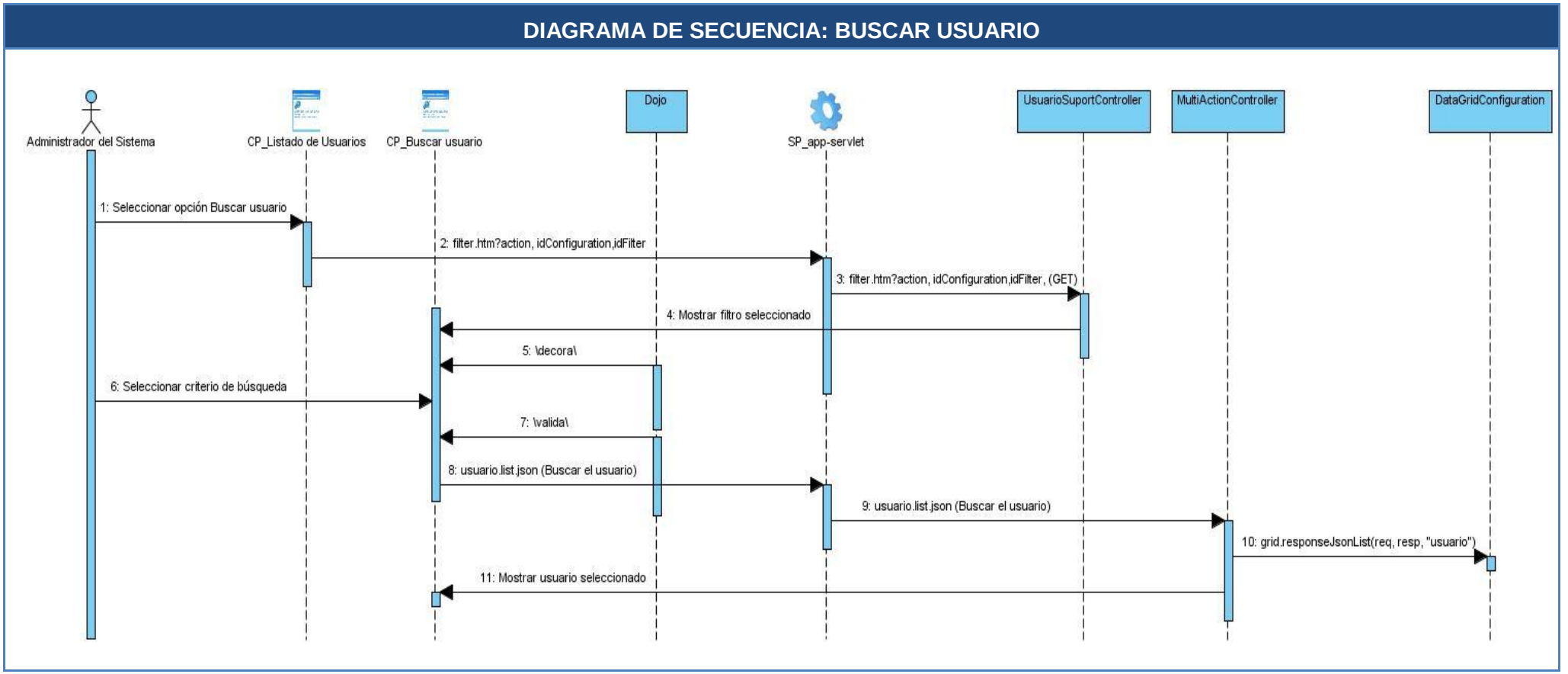

Fig. 2.8 Diagrama de secuencia del CUS Buscar usuario.

DIAGRAMA DE SECUENCIA: GENERAR REPORTE

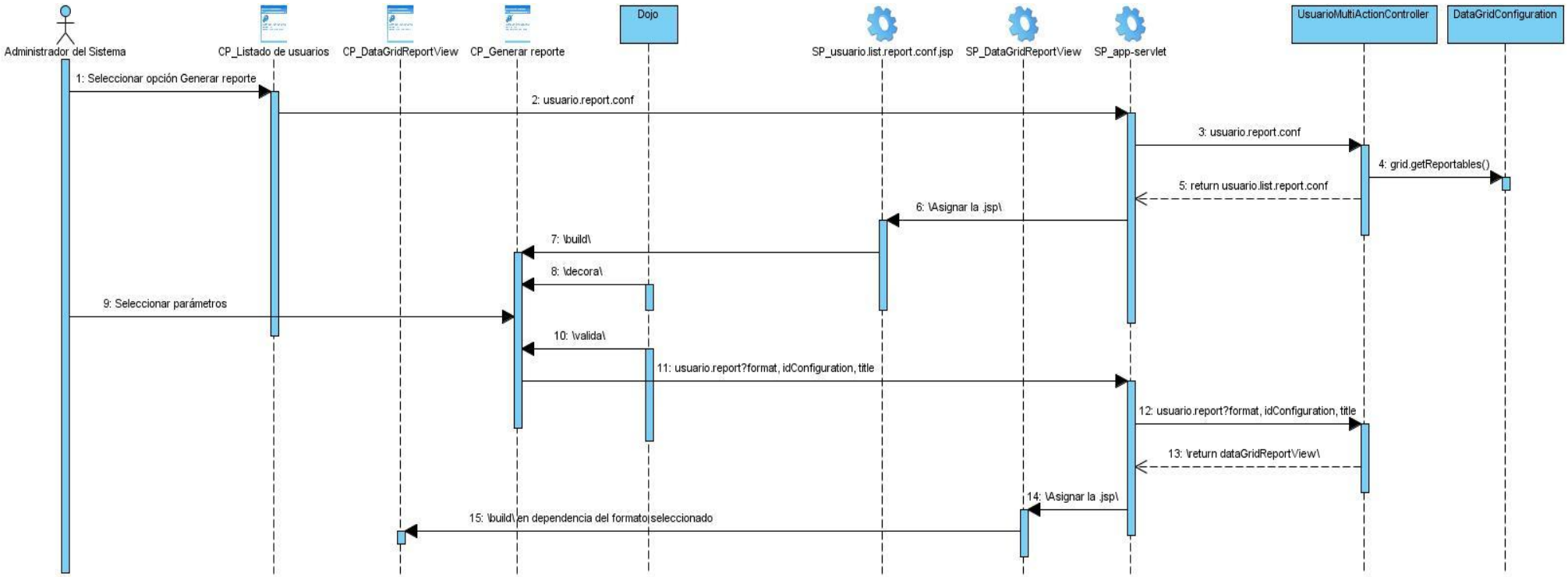

Fig. 2.9 Diagrama de secuencia del CUS Generar reporte.

2.6 Conclusiones

Como resultado de este capítulo:

- Se describió el estilo arquitectónico: Arquitectura en Capas (Tres Capas).
- Se describieron varios patrones de diseño aplicados.
- Se realizaron los diagramas de clases de diseño y los diagramas de secuencia más relevantes según los escenarios del CUS Gestionar listado de usuarios.

CAPÍTULO 3
IMPLEMENTACIÓN DEL SISTEMA

3.1 Introducción

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

3.2 Modelo de implementación

El Modelo de implementación representa la composición física de la implementación en términos de subsistemas de implementación y elementos de implementación. Describe como los elementos de diseño se implementan en componentes. Dicho modelo se considera el artefacto más significativo del flujo de trabajo de Implementación, debido a la importancia que tiene para los desarrolladores comprender el funcionamiento del sistema desde el punto de vista de componentes y sus relaciones. Este modelo está conformado por el diagrama de componentes. [3]

3.3 Diagrama de componentes

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

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

DIAGRAMA DE COMPONENTES: CUS GESTIONAR LISTADO DE USUARIOS

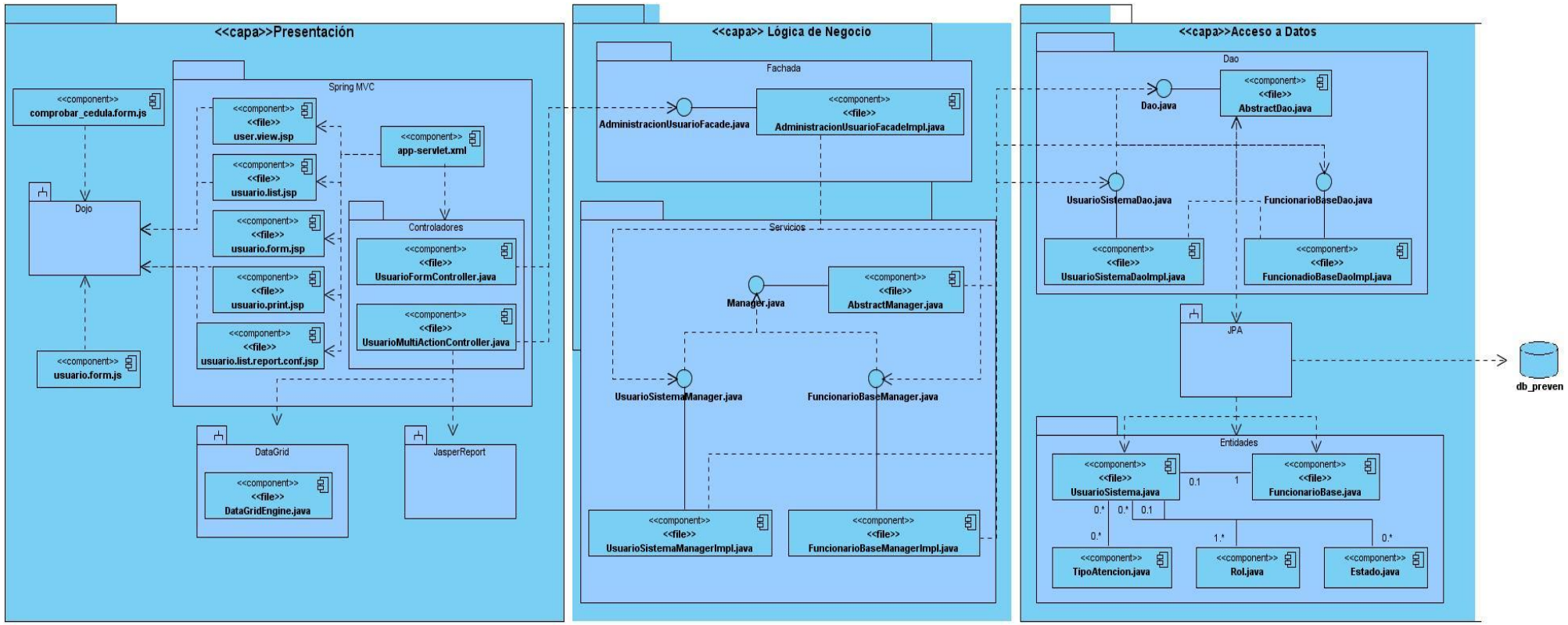

Fig. 3.1 Diagrama de componentes: CUS Gestionar listado de usuarios.

3.4 Diagrama de componentes por capas

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

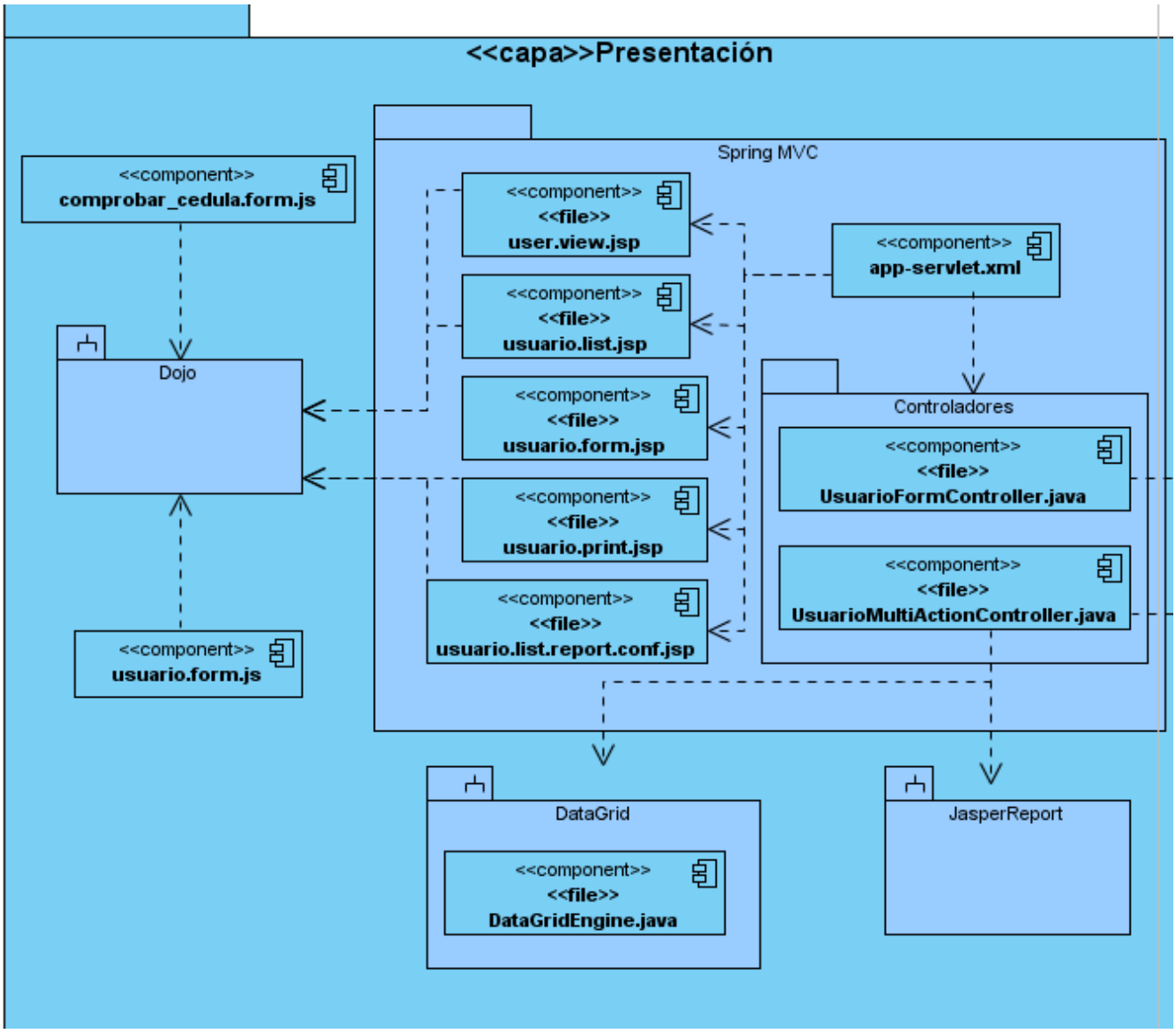

Fig. 3.2 Capa de presentación: CUS Gestionar listado de usuarios.

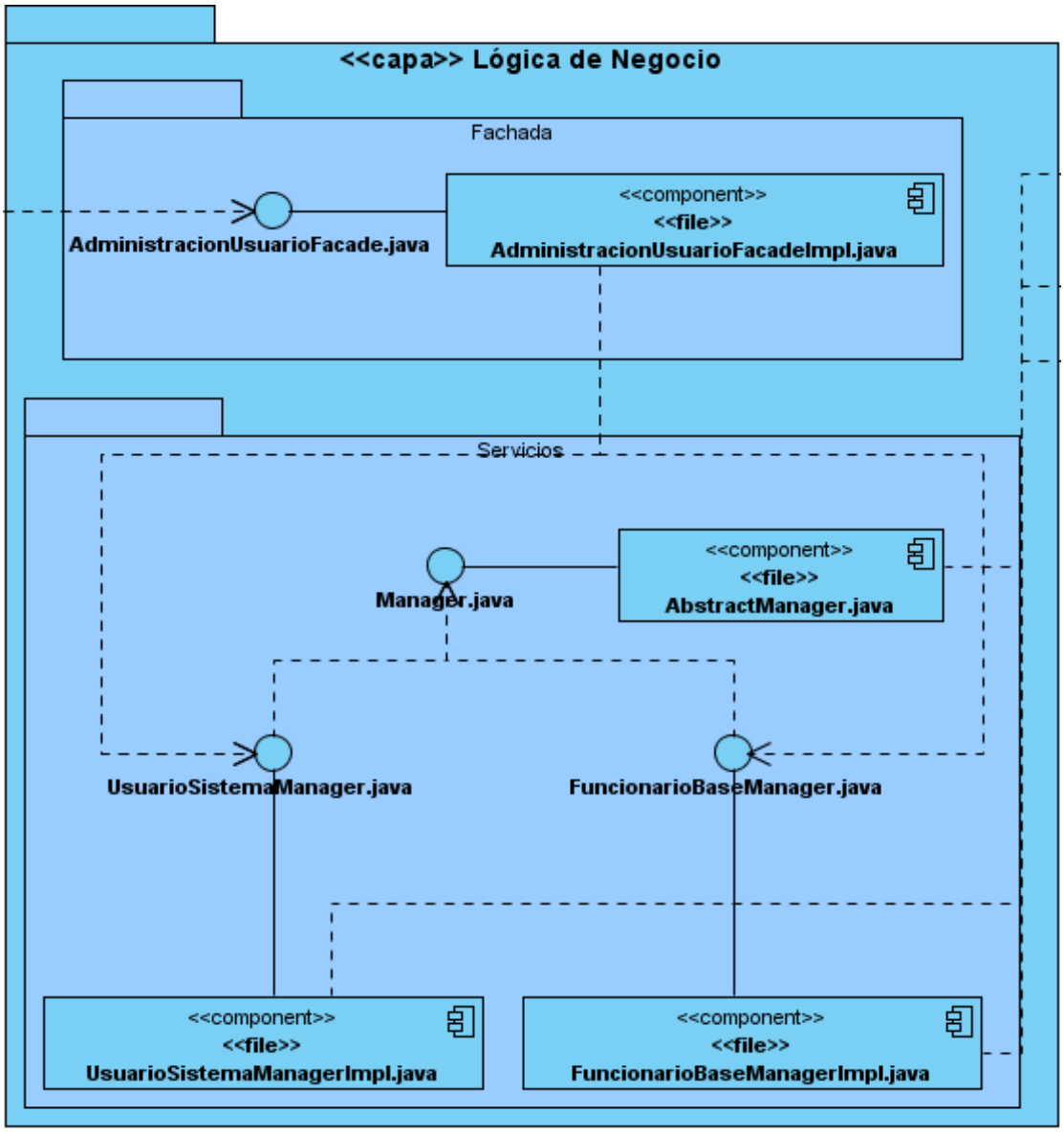

Fig. 3.3 Capa de lógica de negocio: CUS Gestionar listado de usuarios.

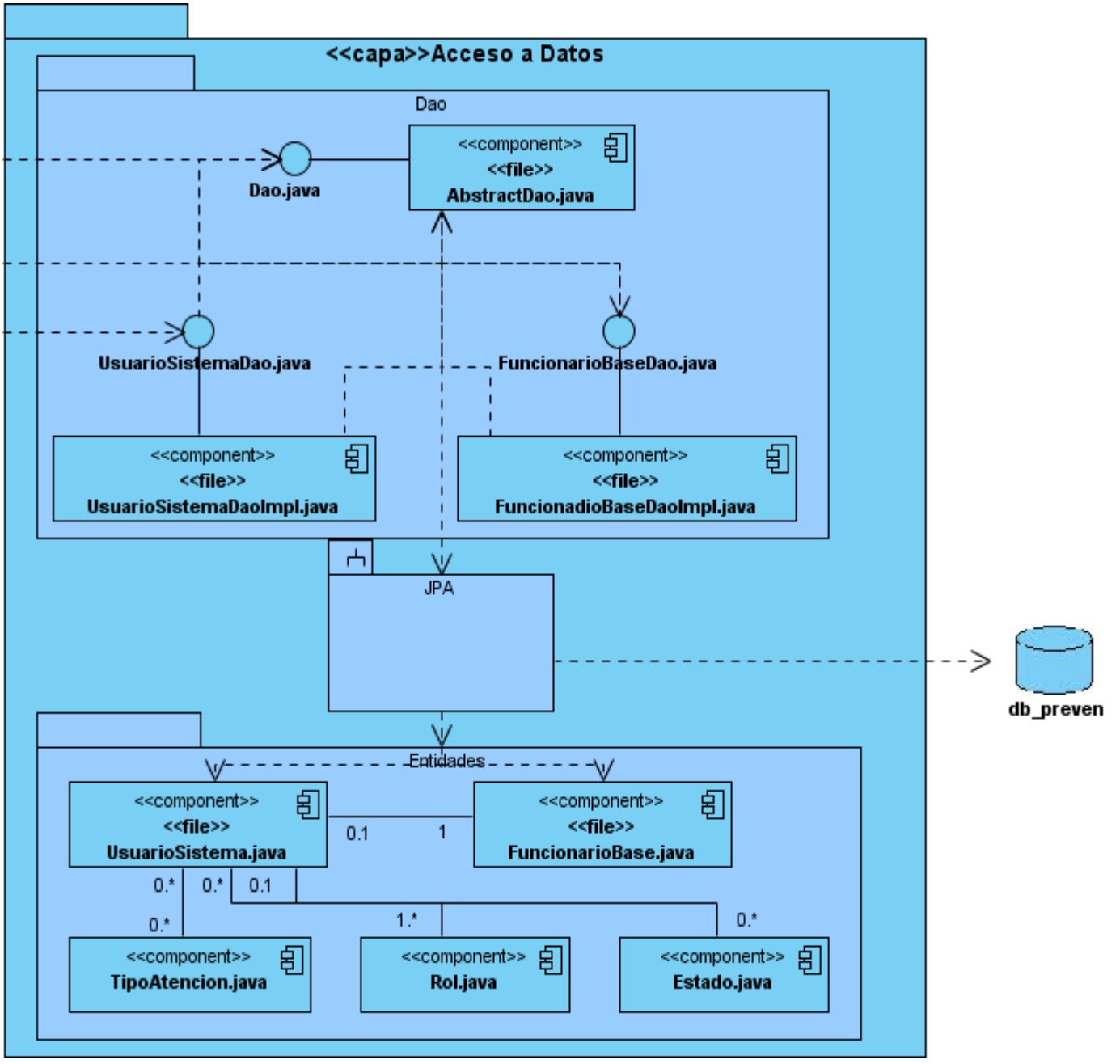

Fig. 3.4 Capa de acceso a datos: CUS Gestionar listado de usuarios.

Distribución de los componentes en el diagrama de despliegue.

A continuación se muestran los componentes distribuidos en los diferentes nodos definidos en el diagrama de despliegue (Ver Anexo 5)

PC Cliente:

Clases Javascript

- *comprobar_cedula.form.js*
- *usuario.form.js*

Componentes

- *Datagrid*

Frameworks

- *Dojo*
- *JasperReport*

Servidor Web:

Clases JSP

- *user.view.jsp*
- *usuario.list.jsp*
- *usuario.form.jsp*
- *usuario.print.jsp*
- *usuario.list.report.conf.jsp*

Clases Java

- *UsuarioFormController*
- *UsuarioMultiActionController*
- *AdministracionUsuarioFacade*

- *AdministracionUsuarioFacadeImpl*
- *Manager*
- *AbstractManager*
- *UsuarioSistemaManager*
- *UsuarioSistemaManagerImpl*
- *RolManager*
- *RolManagerImpl*
- *FuncionarioBaseManager*
- *FuncionarioBaseManagerImpl*
- *Dao*
- *AbstractDao*
- *UsuarioSistemaDao*
- *UsuarioSistemaDaoImpl*
- *RolDao*
- *RolDaoImpl*
- *FuncionarioBaseDao*
- *FuncionarioBaseDaoImpl*
- *UsuarioSistema*
- *Rol*
- *FuncionarioBase*

JPA

Servidor de bases de datos:

- *db_preven*

3.5 Código fuente

Para obtener una versión funcional de la aplicación se deben implementar los componentes que se han definido, como resultado se obtienen archivos que contienen el código fuente de la aplicación. Estos

archivos son un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa. Por tanto, en el código fuente de un programa está descrito por completo su funcionamiento.

3.5.1 Estándares de codificación

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

Organización de los ficheros

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

ESTRUCTURA DEL PROYECTO

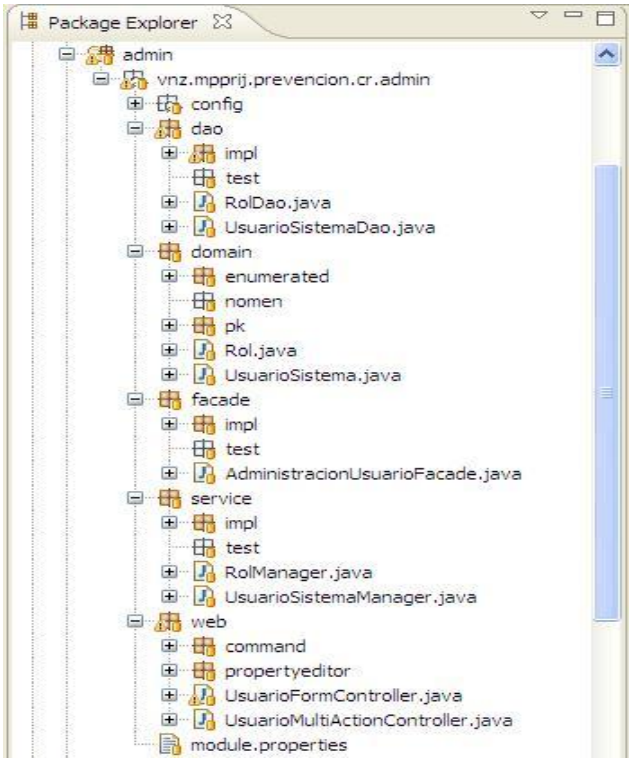

Fig. 3.5 Estructura del proyecto.

3.5.2 Ejemplo de código fuente

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

EJEMPLO DE CÓDIGO FUENTE


```
1  package vnz.mpprij.prevencion.cr.admin.web;
2
3  import java.util.ArrayList;
37 @Controller
38 @SessionAttributes("admin_administracion_usuario_command")
39 public class UsuarioFormController{
40     @Resource
41     private AdministracionUsuarioFacade facade;
42
43     @InitBinder
44     public void InitBinder(WebDataBinder binder) {
45         binder.registerCustomEditor(Rol.class, new RolEditor());
46         binder.registerCustomEditor(CoordinacionRegional.class, new CoordinacioRegionalEditor());
47         binder.registerCustomEditor(Estado.class, new EstadoEditor());
48       //  binder.registerCustomEditor(FuncionarioBase.class, new FuncionarioBaseEditor());
49         binder.registerCustomEditor(TipoAtencion.class, new TipoAtencionEditor());
50     }
51     @RequestMapping(value = "/cedula.form", method = RequestMethod.GET)
52     protected final String showFormC(Model model) throws Exception {
53
54         String commandName = "admin_administracion_usuario_command";
55         String title = "Indicar Cédula de Identidad";
56
57         ComprobarCedulaCommand command = new ComprobarCedulaCommand();
58         model.addAttribute("title", title);
59         model.addAttribute(commandName, command);
60        return "comprobar_cedula";
61     }
62
63     @RequestMapping(value = "/cedula.form", method = RequestMethod.POST)
64       protected final String onSubmit0(
65             @Valid @ModelAttribute("admin_administracion_usuario_command") ComprobarCedulaCommand command,
66             BindingResult result, SessionStatus status,
67             Model model, HttpSession sesion) throws Exception {
```

Fig. 3.6 Ejemplo de Código Fuente: Clase UsuarioFormController.

3.6 Validación

Validar los datos es una tarea común que se produce a través de cualquier aplicación, desde la capa de Presentación hasta la capa de Acceso a Datos. A menudo, la misma lógica de validación se aplica en cada capa, empleándose mucho tiempo en su implementación y propenso a errores. Para evitar la duplicación de estas validaciones en cada capa, se aplicó la lógica de validación directamente en el dominio, ver Anexo 4. Para este proceso se utiliza Hibernate Validator que implementa el API JSR-303 Bean Validation para la plataforma Java, posibilitando una validación declarativa a través de anotaciones

que se hacen cumplir en tiempo de ejecución. Spring contiene la anotación @Valid que se integra con Hibernate Validator garantizando la validación del lado del Servidor en cualquiera de las capas. Además, contiene clases para el manejo de los formularios las cuales se pueden comunicar con el dominio anotado.

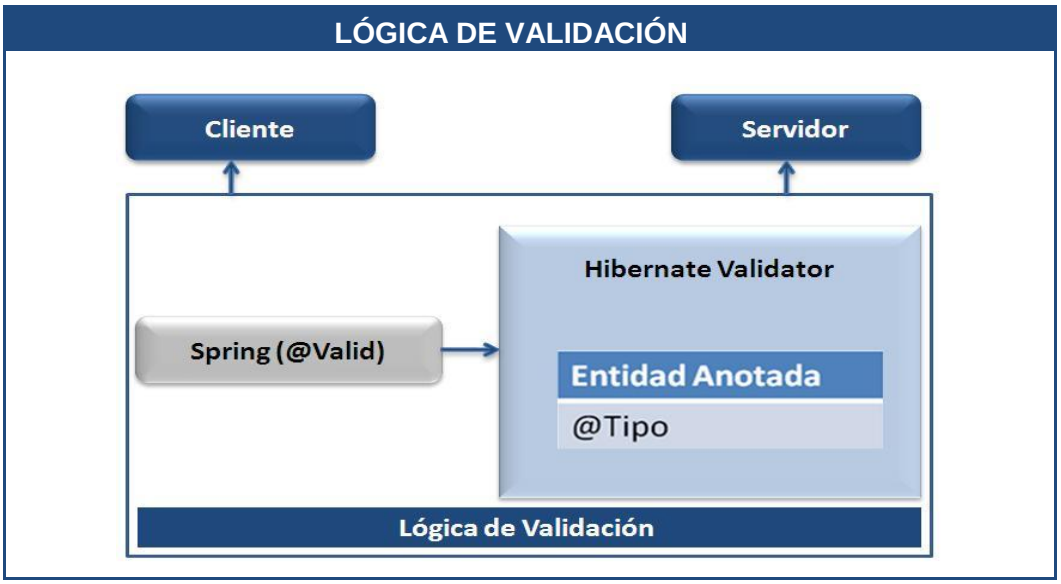

Fig. 3.7 Lógica de validación aplicada en el sistema.

3.7 Principales interfaces de la aplicación

A continuación se muestran algunas interfaces del sistema:

Fig. 3.8 Interfaz Listado de usuarios.

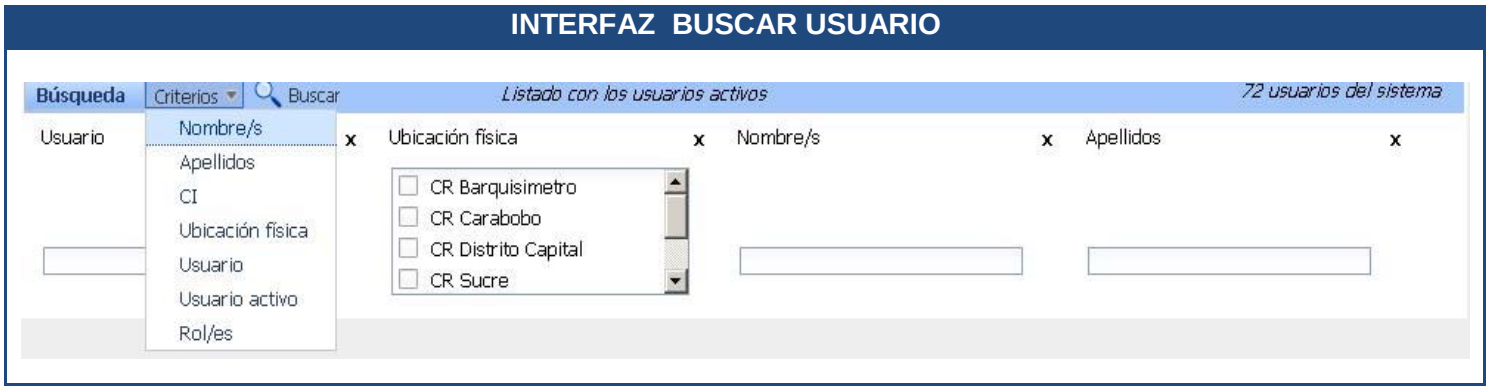

Fig. 3.9 Interfaz Buscar usuario.

Fig. 3.10 Interfaz Indicar cédula de identidad.

Fig. 3.11 Interfaz Visualizar datos de usuario.

INTERFAZ REGISTRAR USUARIO

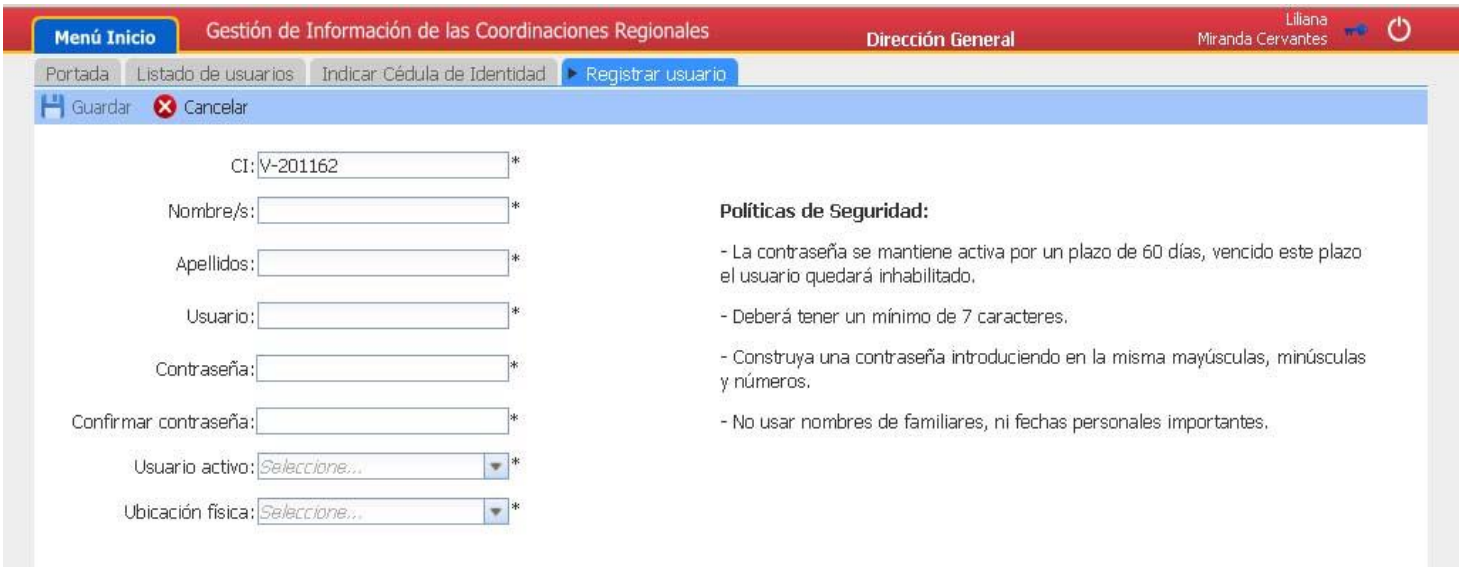

Fig. 3.12 Interfaz Registrar usuario.

INTERFAZ GENERAR REPORTE

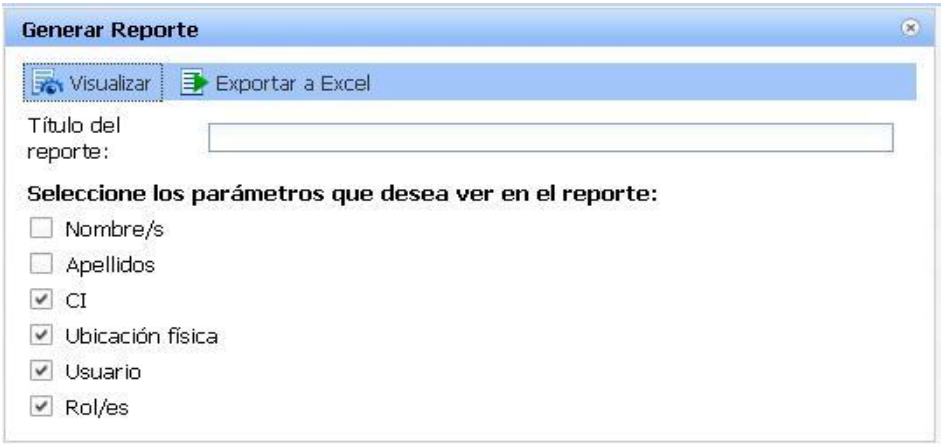

Fig. 3.13 Interfaz Generar reporte.

3.8 Conclusiones

- Esta etapa de desarrollo se caracteriza por resultados ya visibles para los clientes y gratificantes para los desarrolladores, ya que queda implementada la aplicación con las principales funcionalidades que se definieron para la iteración del producto.
- Se elaboraron los diagramas de componentes pertenecientes al CUS Gestionar listado de usuarios.
- Descripción de algunos métodos implementados para obtener un mejor entendimiento de los mismos.
- Se describió la arquitectura de validación aplicada en el sistema.

CAPÍTULO 4

PRUEBAS AL SISTEMA

4.1 Introducción

Durante este capítulo se realiza el flujo de trabajo de prueba, se define el método de prueba, se identifican las herramientas e instrucciones para implementar las pruebas necesarias, se monitorea el progreso de las mismas y el resultado en cada ciclo de pruebas.

4.2 Pruebas de software

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

- El proceso de ejecución de un programa con la intención de descubrir errores previos a la entrega al usuario final.
- Una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones específicas, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente.[22]

4.3 Niveles de pruebas

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

Pruebas de desarrollador

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

ejecutar pruebas de integración. Se recomienda que estas pruebas cubran más que las pruebas de unidad.

Pruebas al sistema

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

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

Pruebas de unidad

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

4.4 Tipos de pruebas

Cada nivel de prueba engloba una técnica de prueba específica según los atributos de calidad que se deseen verificar con las pruebas al software. De manera general las pruebas que se realizarán de acuerdo a su nivel son:

Pruebas de funcionalidad

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

- **Pruebas funcionales:**

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

  Método de Caja Negra: Se ejecuta cada caso de uso, flujo de caso de uso, o función, usando datos válidos e inválidos. Para verificar que se aplique apropiadamente cada regla de negocio, que los resultados esperados ocurran cuando se usen datos válidos, que sean desplegados los mensajes apropiados de error y precaución cuando se usan datos inválidos.

  Caso de prueba: Un caso de prueba se diseña según las funcionalidades descritas en los casos de usos. Este diseño se elabora previo a la realización de las pruebas funcionales de la aplicación. Cada planilla de caso de prueba recoge la especificación de un caso de uso, dividido en secciones y escenarios, para hacer más fructífera la ejecución de las pruebas.
  Los casos de prueba específicos del CUS Gestionar listado de usuarios se registran en el documento: Diseño de Caso de Prueba del CUS Gestionar listado de usuarios perteneciente al Módulo Administración, que se encuentran en el Expediente de proyecto del Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito. [23]

  No Conformidades: La plantilla de No Conformidades (NC) recoge los errores que son detectados durante la revisión de la documentación del sistema. Se elabora un documento por cada revisión que se haga y se controlan a través de versiones según se vayan eliminando los errores, hasta que finalmente se hayan erradicado todos los defectos que posea el elemento que se prueba. Además de estar inmerso en la planilla de diseño de casos de prueba, éstas NC se van registrando en un documento aparte para luego enviarlo al equipo de desarrolladores.

  En las siguientes tablas se expone un resumen de los resultados de las NC obtenidos para contabilizar a modo de resumen el total de NC de los módulos: Administración, Mapas y Planificación de Actividades, distinguiendo de ellas cuáles proceden, cuáles no proceden y las resueltas.

Módulo	Total de NC	# de NC No proceden	# de NC Resueltas
Iteración 1			
Administración	**86**	**11**	**75**
Administración de usuarios	55	5	50
Nomencladores	11	4	7
Estructura geográfica	20	2	18
Mapas	**20**	**0**	**20**
Administración de mapas	18	0	18
Mapas	2	0	2

Tabla 4.1 Resumen de NC del proceso de liberación interno.

Módulo	Total de NC	# de NC No proceden	# de NC Resueltas
Iteración 1			
Administración	**16**	**2**	**14**
Administración de usuarios	8	0	8
Nomencladores	3	1	2
Estructura geográfica	5	1	4
Mapas	**2**	**0**	**2**
Administración de mapas	1	0	1
Mapas	1	0	1

Tabla 4.2 Resumen de NC del proceso de liberación de Calisoft.

Tipos de errores
• Funcionalidad**(F)** • Validación **(V)** • Correspondencia con otro artefacto **(CA)** **-** Aplicación **(A)** • Error de Interfaz **(EI)** • Formato **(FO)** • Redacción **(R)** • Error Técnico **(ET)** • Excepción **(E)** • Diseño de Casos de prueba **(DCP)** • Ortografía **(O)**

ITERACIÓN 1

Módulo	F	V	CA(A)	EI	FO	R	ET	EX	O
Administración de mapas	-	-	2	2	-	-	-	-	-
Administración de usuarios	-	1	-	1	1	1	-	1	2
Estructura geográfica	-	2	-	1	-	2	-	-	1
Nomencladores	-	-	1	-	1	1	-	-	-

Tabla 4.3 Resultados de la Iteración #1

ITERACIÓN 2

Módulo	F	V	CA(A)	EI	FO	R	ET	EX	O
Administración de mapas	-	-	-	-	-	-	-	-	-
Administración de usuarios	-	-	1	-	-	-	-	-	-
Estructura geográfica	-	-	1	-	-	-	-	-	-
Nomencladores	4	-	-	-	-	-	-	-	-

Tabla 4.4 Resultados de la Iteración #2

ITERACIÓN 3

Módulo	F	V	CA(A)	EI	FO	R	ET	EX	O
Administración de mapas	-	-	-	-	-	-	-	-	-
Administración de usuarios	3	-	2	-	-	-	-	-	-
Estructura geográfica	-	-	1	1	-	-	-	-	-
Nomencladores	-	1	5	-	-	-	-	-	-

Tabla 4.5 Resultados de la Iteración #3

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

Clasificación de NC según el equipo de desarrollo
Baja (B) - Redacción, Ortografía. **Media (M)** - Error de interfaz, Formato. **Alta (A)** - Validación, Funcionalidad, Error técnico, Aplicación, Excepción.

Módulo	Alta	Media	Baja
Administración de mapas	2	2	2
Administración de usuarios	5	2	5
Estructura geográfica	2	2	5
Nomencladores	11	1	1

Módulo	Total de No Conformidades	# de No Conformidades que proceden	# de No Conformidades que no proceden	# de No Conformidades Resueltas
Iteración 1				
Administración de mapas	5	5	0	5
Administración de usuarios	8	8	0	8
Estructura geográfica	7	7	0	7
Nomencladores	3	3	0	3

Iteración 2				
Administración de mapas	1	1	0	3
Administración de usuarios	1	1	0	1
Estructura geográfica	1	1	0	1
Nomencladores	4	4	0	4
Iteración 3				
Administración de mapas	0	0	0	0
Administración de usuarios	5	5	0	4
Estructura geográfica	2	1	1	1
Nomencladores	6	4	2	4

Tabla 4.6 Descripción de NC.

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

  Objetivo: Se realizan para garantizar que los usuarios estén restringidos a funciones específicas o su acceso esté limitado únicamente a la información que se encuentre autorizado.

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

  Portada (Rol Administrador/a del sistema de la CR y la DGPD autenticado).

  Mapas

  - Consejos Comunales abordados.

Administrar usuarios

- Listado de usuarios.
- Registrar usuario.

Nomencladores

- Denuncia.
- Atención de casos.
- Diagnóstico.
- Recursos humanos.

Estructura geográfica.

- Estados.
- Municipios.
- Parroquias.
- Coordinaciones Regionales.

Pruebas de confiabilidad

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

- **Pruebas de estrés**

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

de que la carga real supere a la carga esperada.

Pruebas de rendimiento

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

- **Pruebas de carga**

Objetivo: Se realiza generalmente para observar el comportamiento de una aplicación bajo una cantidad de peticiones esperadas. Esta carga puede ser el número esperado de usuarios concurrentes utilizando la aplicación y que realizan un número específico de transacciones durante el tiempo que dura la carga.
Comprobar que se cumplen los requisitos funcionales establecidos, sea correcto el funcionamiento y rendimiento de las interfaces hardware, software y de usuario, sea apropiada la documentación de usuario y se verifique el rendimiento y respuesta en condiciones límite y de sobrecarga.

Pruebas unitarias

Objetivo: Aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el trozo de código debe satisfacer.

Ventajas básicas: facilitan el cambio de código para mejorar estructura así asegurarse de que los nuevos cambios no han introducido errores, simplifica la integración, separación de la interfaz y la implementación, se puede cambiar cualquiera de los dos sin afectar al otro y los errores están más acotados y son más fáciles de localizar: dado que tenemos pruebas unitarias que pueden desenmascararlos.

Pruebas exploratorias

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

Pruebas de regresión

Objetivo: Asegurar que cuando una NC encontrada en el sistema ha sido corregida ninguna de las funcionalidades liberadas previamente falla como resultado de las correcciones o que las características nuevamente agregadas no han creado conflicto con las versiones anteriores del software, es una manera de darle seguimiento y tratamiento a las NC. Estas se registran en el documento de NC.

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

4.5 Herramientas para automatizar las pruebas

JUnit

En el presente trabajo, se realizaron las pruebas unitarias a nivel de desarrollador, teniendo en cuenta, que las mismas deben ser automatizables, completas, reutilizables e independientes. Para realizar las pruebas se utilizó **JUnit** integrado al IDE Eclipse en forma de plug-in, lo cual permite que la generación de las plantillas necesarias para la creación de las pruebas de una clase Java se realice de manera automática, facilitando al programador, enfocarse en la prueba y el resultado esperado, dejando a la herramienta, la creación de las clases que permiten coordinar las pruebas.

A continuación se muestran los resultados de las pruebas realizadas a la funcionalidad Registrar usuario en las clase Fachada.

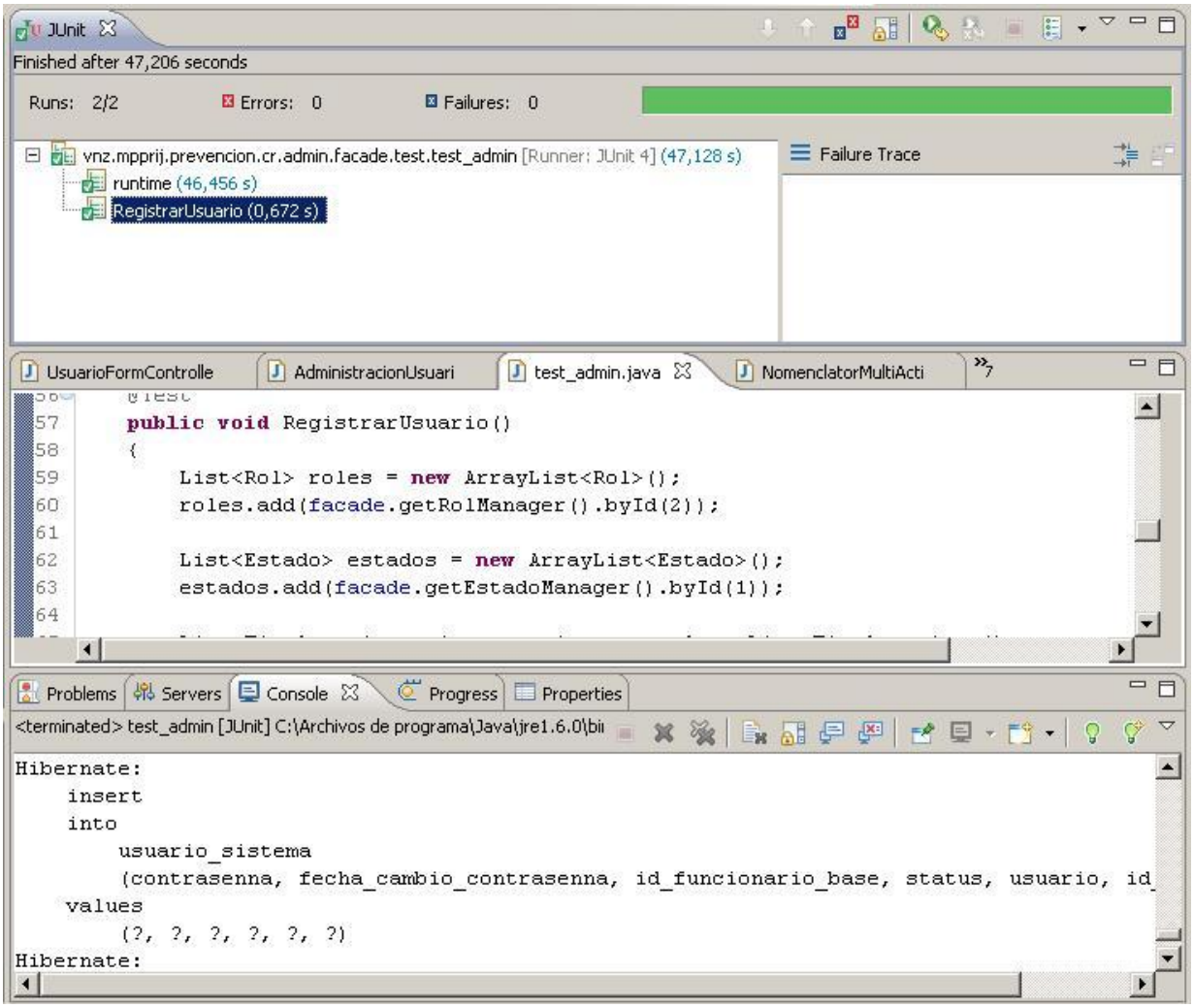

Fig. 4.1 Resultados de las pruebas JUnit en la clase Fachada.

JMeter

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

Una ventaja de JMeter es que permite realizar pruebas de carga sobre cada una de las capas que conforman la aplicación a probar. De esta manera, en un momento dado, es sencillo localizar el foco que está generando contención y está afectando los tiempos de respuesta de un caso de uso específico. [24]

- **Análisis de los resultados de las pruebas de carga y estrés con la herramienta JMeter.**

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

JMeter	RNF Sistema de Gestión de Información de las CR	Condiciones PC
	SOFTWARE:	
	Servidor Web	
Máquina Virtual Java 1.6	Sistema Operativo: Se recomienda Red Hat Enterprise Linux 4.0, aunque pudiera utilizarse otra distribución de Linux para servidores. Servidor web: Apache Jakarta Tomcat 6.0.20.	Microsoft Windows XP Profesional Versión 2002 Service Pack 3 Intel(R) Pentium(R) 4 CPU 3.00GHz 2.99 GHz, 0,99 GB de RAM
	PC Clientes	
	Navegador: para un funcionamiento óptimo del Sistema de Gestión se recomienda: Mozilla Firefox 3.5 o superior, Google Chrome 3.0 o superior.	
	HARDWARE:	

Servidor Web • Procesador Intel Pentium Xeon, 4 GHz, de 4 Gb de RAM. • 30 GB de HD. • NIC: 10/100/1000 bit Ethernet controller. **PC Cliente** • Procesador Intel Pentium IV, 2 GHz, de 256 Mb de RAM • 1 GB de HD disponible para caché del navegador (sin incluir el requerido para el sistema operativo). • Conectividad con el servidor correspondiente.

Tabla 4.3 Especificación de recursos y condiciones del entorno de pruebas.

Condiciones de las PCs:

Se emplearon para la ejecución de las pruebas 2 PCs con las siguientes características.

- Sistema Microsoft Windows XP Professional Versión 2002 Service Pack 3 y Ubuntu 10.10.
- Intel (R) Pentium 4, CPU 3.00 GHz, 0.99 GB de RAM.

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

- **Resumen de los resultados de la ejecución de las pruebas**

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

- **Servidor de aplicación sobre Sistema Operativo Windows**

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

Prueba realizada para un total de 30 usuarios

# Muestras	Media	Mediana	Línea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
2310	17	11	30	1	181	0.00%	1052.9/sec	10343696.4

Tabla 4.4 Resumen del Informe Agregado obtenido para 30 usuarios.

Prueba realizada para un total de 100 usuarios

# Muestras	Media	Mediana	Línea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
7700	17	11	30	1	181	0.00%	1052.9/sec	10343696.4

Tabla 4.5 Resumen del Informe Agregado obtenido para 100 usuarios.

Prueba realizada para un total de 150 usuarios

# Muestras	Media	Mediana	Línea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
11550	182	55	426	0	5223	0.00%	433.1/sec	4255122.1

Tabla 4.6 Resumen del Informe Agregado obtenido para 150 usuarios.

Servidor de aplicación sobre Sistema Operativo Linux

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

Prueba realizada para un total de 30 usuarios

# Muestras	Media	Mediana	Línea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
2310	17	13	29	1	168	0.00%	987.2/sec	9691730.8

Tabla 4.7 Resumen del Informe Agregado obtenido para 30 usuarios.

Prueba realizada para un total de 100 usuarios

# Muestras	Media	Mediana	Línea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
7700	77	54	136	1	729	0.00%	1036.3/sec	10174360.7.8

Tabla 4.8 Resumen del Informe Agregado obtenido para 100 usuarios.

Prueba realizada para un total de 100 usuarios

# Muestras	Media	Mediana	Línea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
11550	153	61	437	0	1509	0.00%	471.3/sec	4627162.7

Tabla 4.9 Resumen del Informe Agregado obtenido para 100 usuarios.

El resultado de la ejecución de las pruebas, es similar en ambos Sistemas Operativos. De igual manera, el rendimiento y la velocidad de carga son menores en los casos límites teniendo un número superior de peticiones. Por lo que se concluye que a medida que aumenta el número de usuarios disminuye el rendimiento de la aplicación.

4.5 Conclusiones

- Para lograr una mayor calidad y eficacia en el proceso de automatización se llevó a cabo la concepción de las pruebas con el régimen que dictaminan las pautas para su desarrollo y además un buen acondicionamiento del ambiente de las pruebas.
- Se describieron los principales tipos de prueba utilizadas.
- Se demostró que constituye una buena práctica utilizar herramientas como JUnit y JMeter que automaticen los procesos de pruebas, gracias a las ventajas que traen en conjunto con su utilización.

CONCLUSIONES

El desarrollo de los módulos Administración y Mapas del Sistema de Gestión de Información de Prevención del Delito de la República Bolivariana de Venezuela a partir del diseño de clases que cumplen con los estándares y patrones definidos, la implementación basada en la utilización de frameworks y la realización de pruebas que garantizan el cumplimiento de las funcionalidades definidas, posibilita a los especialistas a partir de la información gestionada en dichos módulos la estandarización de los informes generados, el control de acceso de los usuarios y la representación geográfica de los Consejos Comunales.

RECOMENDACIONES

1. Aplicar las facilidades que brinda el framework Dojo para el tratamiento de imágenes con el propósito de optimizar la implementación del módulo Mapas.
2. Aplicar posibles validaciones del lado del cliente del componente de selección múltiple, para mejorar el sistema en cuanto a su validación.

REFERENCIA BIBLIOGRÁFICA

1. Lourdes Aja. Gestión de la información, 2009, [En línea] [Citado el: 23 de Septiembre del 2010] Disponible en: http://www.google.com /gestión del conocimiento y gestion de la informacion.pdf
2. Cruz Carballo, Manuel Ivan. ” Sistema de información para el apoyo a la toma de decisiones gerenciales” [En línea] [Citado el: 23 de Septiembre del 2010] Disponible en: http://www.monografias.com/trabajos17/sistema-gerencial/sistema-gerencial.shtml
3. Jacobson, G. Booch y J. Rumbaugh, El Proceso Unificado de Desarrollo de Software. S.l.: PEARSON EDUCACIÓN S.A, 2000. [Citado el: 25 de Septiembre del 2010].
4. Guerrero Luis A. Universidad de Chile. Metodologías De Desarrollo De Software [En línea] [Citado el: 28 de Septiembre del 2010] Disponible en: http://www.eici.ucm.cl/ Academicos/R Villarroel/descargas/ing sw 1/RUP.pdf
5. Visual Paradigm 2005 [En línea] [Citado el: 30 de Septiembre del 2010] Disponible en: http://www.visual-paradigm.com/product/vpuml/..
6. Group, P.G.D. PostgreSQL. 2007 [En línea] [Citado el: 1 de Octubre del 2010] Disponible en: http://www.postgresql.org/.
7. PuroSoftware.com [En línea] [Citado el 1 de Octubre del 2010] Disponible en: http://www.purosoftware.com/programacion-bases-de-datos/11-pg-admin-3.html
8. BEATON W. Eclipse Platform Technical Overview [En Línea] [Citado el: 5 de Octubre del 2010] Disponible en: http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html.
9. Definición.org Definición Lenguaje de Programación [En Línea] [Citado el: 10 de Octubre del 2010] Disponible en: http://www.definicion.org/lenguaje-de-programacion
10. Schildt Herbert Java 2 Manual de referencia - Madrid: [s.n.], 2001. [Citado el: 20 de Octubre del 2010]
11. SQLManager.net. Disponible en: http://www.sqlmanager.net/products/postgresql/manager/ . Consultado el: (28 de Octubre del 2010).

12. Perrone Paul J and Krishna J2EE Developer's Handbook [Citado el: 1 de Noviembre del 2010] - Indianapolis, Indiana: Sam's Publishing, 2003.
13. Group, P.G.D. PostgreSQL. 2007 [En línea] [Citado el: 10 de Noviembre del 2010] Disponible en: http://www.postgresql.org/.Hibernate Validator JSR 303
14. hibernate.org [En línea] [Citado el: 15 de Noviembre del 2010] Disponible en: http://www.hibernate.org/
15. Framework Dojo [En línea] [Citado el: 21 de Noviembre del 2010] Disponible en: http://www.dojotoolkit.org/
16. hibernate.org [En línea] [Citado el: 30 de Noviembre del 2010] Disponible en: Disponible en: http://www.hibernate.org/
17. jasperforge.org Sitio Web Oficial de JasperReport [En línea] [Citado el: 30 de Noviembre del 2010] Disponible en: http://jasperforge.org/website/.
18. Reynoso Billy, Carlos. [En línea] Marzo 2004. [Citado el: 10 de Diciembre del 2010] Disponible en: http://www.willydev.net/descargas/prev/IntroArg.pdf
19. Evolución y Orientaciones de Patrones [En línea] [Citado el: 21 de Enero del 2011] Disponible en: http://www.monografias.com/trabajos27/evolucion-patrones/evolucion-patrones.shtml?monosearch
20. Osorio, Eneisy y Asdrúbal. Propuesta de arquitectura para el Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito. UCI. Ciudad de la Habana, 2010. Capítulos 2 y 3, Tesis.
21. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides**.** [Citado el: 21 de Enero del 2011]- *Design Patterns (Elements of Reusable Object-Oriented Software)*.
22. Conferencia # 5 Pruebas. Ingeniería del Software II. [Citado el: 1 de Marzo del 2011] Disponible en: http://teleformacion.uci.cu
23. Diseño de Caso de prueba .Caso de Prueba Gestionar denuncia perteneciente al Módulo Denuncias, Expediente de Proyecto Sistema Informático de Gestión de Información de las Coordinaciones Regionales para la Dirección General de prevención del Delito.
24. Mejoramiento del Proceso de Pruebas y Corrección de Defectos de Software en un ambiente globalizado.[En línea 2009] [Citado el: 3 de Marzo del 2011] Disponible en: http://chie.uniandes.edu.co/~gsd/index.php?option=com_content&task=view&id=129&Itemid=183

BIBLIOGRAFÍA

1. Guerrero Luis A. Universidad de Chile. Metodologías De Desarrollo De Software Disponible en: http://www.eici.ucm.cl/ Academicos/R Villarroel/ descargas/ ing_sw 1/RUP.pdf
2. Group, P.G.D. PostgreSQL. 2007 Disponible en: http://www.postgresql.org/.
3. Jacobson, G. Booch y J. Rumbaugh, El Proceso Unificado de Desarrollo de Software. S.l.: PEARSON EDUCACIÓN S.A, 2000. [Citado el: 25 de Septiembre del 2010]
4. Barban, Wilmar y Danoy. Sistema Informático para la Gestión de Información de los Recursos Materiales y Programas – Proyectos de las Coordinaciones Regionales de Prevención del Delito. UCI. Ciudad de la Habana, 2010. Capítulo 1, Tesis.
5. Booch, G. Rumbaugh, J. y Jacobson, I. (2000) "El Lenguaje Unificado de Modelado". Addison-Wesley.
6. Visual Paradigm 2005 Disponible en: http://www.visual-paradigm.com/product/vpuml/..
7. PuroSoftware.com Disponible en: http://www.purosoftware.com/programacion-bases-de-datos/11-pg-admin-3.html
8. BEATON W. Eclipse Platform Technical Overview Disponible en: http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html.
9. Definición.org Definición Lenguaje de Programación Disponible en: http://www.definicion.org/lenguaje-de-programacion
10. Schildt Herbert Java 2 Manual de referencia - Madrid: [s.n.], 2001.
11. Framework Dojo Disponible en: http://www.dojotoolkit.org/
12. Perrone Paul J and Krishna J2EE Developer's Handbook - Indianapolis, Indiana: Sam's Publishing, 2003.
13. hibernate.org Disponible en: http://www.hibernate.org/
14. Framework Dojo Disponible en: http://www.dojotoolkit.org/
15. Pressman, Roger; Ingeniería de software. Un enfoque práctico. McGraw.Hill/Interamericana de España.
16. Johson, Rod (2005)" Professional Java Development with the Spring Framework".

17. Johson, Rod (2008)" spring-reference version 2.5.6".
18. Lourdes Aja. Gestión de la información, 2009, Disponible en: http://www.eubca.edu.uy/materiales/planeamiento_de_servicios_Bibliotecarios/gestión_del_conocimiento_y_gestion_de_la_informacion.pdf
19. Cruz Carballo, Manuel Ivan. " Sistema de información para el apoyo a la toma de decisiones gerenciales" Disponible en: http://www.monografias.com/trabajos17/sistema-gerencial/sistema-gerencial.shtml
20. jasperforge.org Sitio Web Oficial de JasperReport Disponible en: http://jasperforge.org/website/.