UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 2 "TELECOMUNICACIONES Y SEGURIDAD INFORMÁTICA"

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

Título: Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. Módulos: Atención de Casos, Bienes Muebles y Materiales.

Autores: Dismey Pedroso Márquez.

Ernesto Wilfredo Ulloa Gómez.

Tutor: Ing. Liusmila Nieto Cervantes.

Tutor: Ing. Francisco Montada Aguilar.

Ciudad de La Habana, Mayo del 2011.

"Año 53 de la Revolución"

Declaración de Autoría

Declaramos que _____ y _____somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas (UCI) y a la Facultad (2) para que hagan el uso que estimen pertinente con este trabajo.

Para que así conste firmamos la presente a los ____ días del mes de ____del 2011.

_____	_____
Firma del Autor	Firma del Autor
Dismey Pedroso Márquez.	Ernesto Wilfredo Ulloa Gómez.
_____	_____
Firma del Tutor	Firma del Tutor
Liusmila Nieto Cervantes.	Francisco Montada Aguilar.

"Se puede adquirir conocimientos y conciencia a lo largo de toda la vida, pero jamás en ninguna otra época de su existencia una persona volverá a tener la pureza y el desinterés con que, siendo joven, se enfrenta a la vida."

Fidel Castro.

DATOS DE CONTACTO

Tutor:

Ing. Liusmila Nieto Cervantes.

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

Email: lnieto@uci.cu

Tutor:

Ing. Francisco Montada Aguilar

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

Email: fmontada@uci.cu

AGRADECIMIENTOS

A mi familia por hacer de mí quien soy ahora. A Reymundo por ser mi padre todo estos años. A mis tutores Francisco y Liusmila por soportarme y tener paciencia conmigo. A todos los profesores del proyecto. A todos mis amigos, los que conocí en los 5 años de la UCI y los que vienen cargando conmigo desde mucho antes.

Ernesto.

A mi familia por todo el apoyo y el cariño que me dan. A todos mis amigos que conocí durante la carrera y que me soportaron hasta el final. A mis tutores por ayudarme con todas las cosas de la tesis, en especial a Liusmila por soportarme y tener tanta paciencia conmigo. A mi amiguito Alejo que me soportó desde el primer día de primer año y hasta el día de hoy. A Yaime por sus cafés para mantenerme despierta en las noches de tesis. A todas aquellas personas que me han ayudado en la vida.Y en especial a mi novio por no perder la calma con todos mis ataque, por estar despierto conmigo cada noche que hizo falta, por facilitarme todo lo que necesitaba para que no me entretuviera en otras cosas, por estar siempre levantándome el ánimo y apoyándome hasta el final.

Dismey.

DEDICATORIA

A mi familia por estar siempre ahí para mí en todo lo que necesitaba y a mi madre por ser madre y padre a la vez.

Ernesto.

A mi papá por ser mi ejemplo a seguir y brindarme todo el apoyo que necesité durante toda la carrera. A mi mamita por todas sus preocupaciones durante estos 5 años y por todo lo que sacrificó en su vida por estar cuidándome desde que tenía 19 añitos.

Dismey.

RESUMEN

Con el crecimiento de las tecnologías y el almacenamiento de grandes cantidades de informaciones, las empresas y entidades están optando por tener aplicaciones web que les permitan gestionar la información con un mejor funcionamiento y una mayor rapidez en cuanto a organización y toma de decisiones.

Teniendo en cuenta dicha necesidad, el objetivo principal del presente trabajo de diploma es desarrollar un Sistema Informático para la Gestión de Información. Este sistema será el encargado de gestionar toda la información referente a la Atención de Casos, control de Materiales y el inventario de Bienes Muebles en las Coordinaciones Regionales (CRs) de Prevención del Delito, pertenecientes al Ministerio del Poder Popular para Relaciones Interiores y Justicia de la República Bolivariana de Venezuela, permitiendo además mantener actualizada a la Dirección General de Prevención del Delito (DGPD) mediante reportes que periódicamente se generan resumiendo todas las actividades realizadas.

Se desarrolló el sistema empleando el estilo arquitectónico Arquitectura en Capas y haciendo uso de los frameworks Dojo, Spring e Hibernate fundamentalmente.

PALABRAS CLAVE: Prevención, CRs, DGPD, delito, frameworks, Spring, Hibernate y Dojo.

ÍNDICE

INTRODUCCIÓN

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

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

Partiendo de la nueva estructura organizacional el gobierno atiende en primer orden la agenda social, siendo este un elemento que impacta positivamente en la dimensión amplia de 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 institucional de garantizar la seguridad ciudadana, promueve la formulación y puesta en marcha del proyecto Solución Tecnológica Integral para el Perfeccionamiento del Sistema de Prevención del Delito de la República Bolivariana de Venezuela, orientado a mejorar el funcionamiento de los procesos internos de todas las direcciones pertenecientes al mismo.

La Dirección General de Prevención del Delito (DGPD) está adscrita al Viceministerio de Seguridad Ciudadana perteneciente al Ministerio del Poder Popular para Relaciones Interiores y Justicia de la República Bolivariana de Venezuela. Este organismo posee 22 Coordinaciones Regionales (CRs); 18 adscritas al MPPRIJ, ubicadas en los Estados: Anzoátegui, Aragua, Barinas, Bolívar, Carabobo, Cojedes, Falcón, Guárico, Lara, Mérida, Miranda, Monagas, Portuguesa, Táchira, Trujillo, Yaracuy y Zulia y en el Distrito Capital; además de cuatro que se han creado de forma descentralizada en los estados Delta Amacuro, Nueva Esparta, Sucre y Vargas.

Las CRs están conformadas por un equipo de profesionales y técnicos (Coordinador/a General, Secretaria/o, Trabajador Social, Administradores, entre otros funcionarios) que se encargan de la

atención e implementación de los programas de prevención del delito, como el Programa de Educación y El Proyecto de la Comunidad. Las CRs deben ejecutar en sus áreas de acción diferentes programas y proyectos, los cuales son propuestos por la DGPD acorde a las necesidades de cada región, grupo o problema a enfrentar. Además, cada Coordinación Regional (CR) puede realizar los proyectos que considere necesarios según las necesidades que presente el entorno, siempre y cuando dichos proyectos sean aprobados previamente por la DGPD.

Es necesario destacar que los programas generan diferentes actividades, dentro de estas actividades se encuentra la de Atención de Casos como una actividad de carácter social, con el objetivo de proporcionar servicios que ayuden y guíen a la población. Para la realización de estas actividades las CRs utilizan Materiales (pelotas, sillas, mesas, guantes, juegos de ajedrez, entre otros), los cuales le son asignados por la DGPD, la gobernación u otras instituciones. También le son asignados a las CRs Bienes Muebles (mobiliaria) para el desempeño de su trabajo diario.

Por estas razones son las CRs las encargadas de gestionar la información asociada a la Atención de Casos, así como velar por el almacenamiento, organización y control de los Materiales y los Bienes Muebles que le fueron asignados. Para llevar a cabo estas tareas las CRs registran toda la información correspondiente a los servicios que brinda para tener un mayor control y análisis de los mismos, además se le realizan inventarios a los Bienes Muebles y se controla la entrada y salida de los Materiales, con el objetivo de poder informar a la DGPD de estas actividades.

Todo este proceso genera gran volumen de información y debido a que en la actualidad se realizan de forma manual, esto dificulta la organización y gestión de la misma, la cual llega a la DGPD con diversidad de formatos, dificultando la creación de informes y el procesamiento de los datos, lo cual provoca demora en la toma de decisiones afectando así la ejecución de sus procesos internos y la efectividad en el logro de resultados dirigidos al desarrollo de acciones que contribuyan a la prevención de la violencia y el fortalecimiento de la convivencia ciudadana, para alcanzar la calidad de vida adecuada en las comunidades venezolanas.

Debido a esto en el marco del convenio Cuba-Venezuela los directivos de la DGPD y Albet contratan la realización de una herramienta informática. Se decidió realizar un Sistema de Gestión de Información para las CRs de Prevención del Delito de la República Bolivariana de Venezuela. Posteriormente se realizó un levantamiento de requisitos, los cuales fueron aceptados, documentados y firmados por estos directivos, quedando pendiente su implementación.

Por todo lo antes expuesto se plantea como **Problema Científico** a resolver: ¿Cómo garantizar el cumplimiento de los requerimientos funcionales y no funcionales asociados a los módulos Atención de Casos, Bienes Muebles y Materiales del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela?

A partir del problema científico se define como **Objeto de estudio** para esta investigación: Gestión de información en las Coordinaciones Regionales de la Dirección General de Prevención del Delito de la República Bolivariana de Venezuela, y como **Campo de acción**: Gestión de la Información asociada a los Bienes Muebles, los Materiales y la Atención a Casos en las Coordinaciones Regionales de la Dirección General de Prevención del Delito de la República Bolivariana de Venezuela.

La presente investigación tiene como **Objetivo General**: Desarrollar los módulos que gestionen la información referente a la Atención de Casos, el inventario de Bienes Muebles y el control de Materiales del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela.

Una vez definido el objetivo general y en aras de darle cumplimiento, se plantean los siguientes **Objetivos Específicos:**

- Realizar diseño de las clases de los módulos: Atención de Casos, Bienes Muebles y Materiales**.**
- Implementar los componentes correspondientes a los módulos: Atención de Casos, Bienes Muebles y Materiales.
- Validar funcionalmente los módulos: Atención de Casos, Bienes Muebles y Materiales.

Para cumplir con los objetivos específicos antes descritos se han trazado las siguientes **Tareas Investigativas:**

1. Estudio y descripción de las herramientas y tecnologías seleccionadas para el desarrollo del Sistema Informático de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. Módulos: Atención de Casos, Bienes Muebles y Materiales.
2. Realización de los Diagramas de Clases del Diseño de los módulos: Atención de Casos, Bienes Muebles y Materiales.

3. Realización de los Diagramas de Secuencia del Diseño de los módulos: Atención de Casos, Bienes Muebles y Materiales.
4. Realización de los Diagramas de Componentes de los módulos: Atención de Casos, Bienes Muebles y Materiales.
5. Implementación de los componentes de los módulos Atención de Casos, Bienes Muebles y Materiales.
6. Realización de pruebas funcionales a los módulos implementados.

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

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

Capítulo 2 Diseño del Sistema: En este capítulo se explica el estilo arquitectónico aplicado y los patrones de diseños utilizados, además se muestran los diagramas de clases del diseño y los diagramas de secuencia de los módulos a desarrollar: Atención de Casos, Bienes Muebles y Materiales.

Capítulo 3 Implementación del Sistema: En este capítulo se muestran los diagramas de componentes que se definieron durante la implementación de los módulos: Atención de Casos, Bienes Muebles y Materiales, también se muestran algunas de las principales interfaces del sistema.

Capítulo 4 Pruebas del Sistema: En este capítulo se explican las pruebas realizadas al sistema con el objetivo de garantizar su correcto funcionamiento.

1) CAPÍTULO1. FUNDAMENTACIÓN TEÓRICA

1. Introducción

El presente capítulo abordará primeramente los principales conceptos que permitan entender qué es un Sistema de Gestión de Software, luego se argumentan las principales características de la metodología de desarrollo que se utilizó para guiar el proceso de software así como las herramientas en las que se apoyará el desarrollo del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. Módulos: Atención de Casos, Bienes Muebles y Materiales.

2. Sistemas de Gestión de Información

La gestión de información como actividad surgió en las últimas décadas del siglo XX como resultado de la necesidad que existía en las organizaciones de perfeccionar el uso y manejo de la información. Ese interés estuvo determinado no solo por el reconocimiento de la información y el conocimiento como activos intangibles de gran valor para la empresa, sino también por los beneficios derivados de una adecuada planificación, organización, conducción y control de recursos, estrategias y acciones relacionadas con la información, es decir, una adecuada gestión de este recurso.

Gestión de la información: La gestión de información constituye un "conjunto de actividades realizadas con el fin de controlar, almacenar y, posteriormente, recuperar adecuadamente la información producida, recibida o retenida por cualquier organización en el desarrollo de sus actividades". [1]

El proceso de gestión de información en las empresas al realizarse de forma manual puede generar problemas tales como: el deterioro, pérdida o duplicidad de la información, gasto innecesario de papel y otros recursos materiales, afectando así el logro de sus resultados finales. Por estas razones, muchas de estas empresas que manejan grandes volúmenes de información de las cuales depende su éxito están optando por Sistemas de gestión de información.

Sistemas de gestión de información: Puede definirse como un conjunto formal de procesos que, operando sobre una colección de datos e información estructurados según las necesidades de la organización, recopilan, elaboran y distribuyen la información (o parte de ella) necesaria para las operaciones, las actividades de dirección y la toma de decisiones. [2]

3. Sistemas de gestión de Información para la Prevención del Delito

3.1 Programa Institucional de Prevención del Delito

Consiste en una página web cuyo objetivo es promover y fortalecer hábitos de prevención del delito entre la comunidad del Distrito Federal (México), a través de orientación, capacitación y asesoría continua en la detección y conocimiento de factores protectores y de riesgo, que influyen en la inhibición o facilitación de conductas delictivas y/o antisociales. Lo anterior, con el fin de que los capitalinos puedan implementar y difundir medidas preventivas en distintos espacios, como parte de una labor conjunta con la Secretaría de Seguridad Pública del Distrito Federal. [3]

3.2 Comité de Consulta y Participación Ciudadana

En el estado de Chiapas[1] en México, se han organizado Comités Municipales, Regionales y el Comité Estatal de Consulta y Participación Ciudadana. Estos Comités están integrados por representantes de la sociedad y realizan acciones de información, vigilancia y participación en funciones de seguridad públicas. Estos Comités son órganos consultivos que dan a conocer la opinión de la comunidad y la transmiten a los Consejos Municipales de Seguridad Pública, de manera informada, responsable y documentada. Esta página pretende ser una herramienta de información y comunicación útil para la población en general, interesada en la prevención del delito y las adicciones. [4]

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

El Programa Integral de Gestión e Investigación para la Prevención del Delito se creó con el objetivo de contar con un Sistema de Información socio-delictiva que cuente con los datos necesarios para elaborar diagnósticos y estudios sobre los factores que inciden en la violencia y la delincuencia, a fin de diseñar planes y programas que respondan a la realidad de los estados y municipios. Este sistema está dirigido por el Gobierno Federal de México. [5]

Los sistemas que hasta aquí hemos visto son más bien de carácter informativo para los estados y municipios de la ciudad de México, brindan espacios donde la sociedad en general puede informarse respecto a temas delictivos, realizan encuestas en la población para identificar factores conflictivos y poder así mantener a los ciudadanos al tanto de cómo prevenirlos. Estos sistemas aparte de que se centran en las características y problemas propios de su país, no almacenan toda la información

[1] **Chiapas** es uno de los 31 estados que, junto con el Distrito Federal, conforman las 32 entidades federativas de México.

digitalmente, se realizan las consultas, conferencias y demás actividades en la población, luego estos sistemas de información brindan solamente los resultados de los sucesos ocurridos y de cómo poder integrarte a ellos o simplemente reconocerlos.

Por tanto, la novedad que se busca con el Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela, módulos: Atención de Caso, Bienes Muebles y Materiales, es que no sea solo un sistema que le permita a la DGPD estar informada en cuanto a los resultados de los servicios proporcionados mediante la Atención de Casos, la realización del inventario de los Bienes Muebles y el control de los Materiales. Sino que sea una herramienta que al permitirles un mejor control, optimización y organización de la información a los trabajadores de las CRs, se obtengan excelentes resultados enfocados a mejorar la delincuencia y la seguridad ciudadana en las comunidades.

Ya considerándose necesaria la realización de dicho sistema el cual abarcará toda la información referente a la Atención de Casos, Bienes Muebles y Materiales en las CRs, se necesitará estructurar, planificar y controlar el proceso de desarrollo del mismo, para lo cual se describirá la metodología de desarrollo de software que nos permitirá realizar dichas acciones.

4. Herramientas, metodología y tecnologías propuestas

Como antecedente se tienen trabajos de diplomas del curso anterior los cuales definieron las herramientas, tecnologías, metodología y la arquitectura que sigue el presente trabajo. [6]

4.1 Metodología de desarrollo

4.1.1 RUP

Por la complejidad del proceso a automatizar, la cantidad de requisitos que deben ser implementados en el sistema y la cantidad de información que se maneja se usará la metodología RUP (Rational Unified Procces). RUP se encuentra dentro de las Metodologías Tradicionales o Pesadas, las cuales hacen un mayor énfasis en la planificación y control del proyecto, así como en la especificación precisa de requisitos y modelado.[7]

RUP es un proceso de ingeniería de software capaz de ser aplicado a cualquier proyecto sin importar la magnitud del mismo, está estructurado y permite adaptarse a cada entorno de trabajo (Ver Anexo 1). Proporciona un acercamiento disciplinado a la asignación de tareas y responsabilidades en una

organización de desarrollo. Su propósito es asegurar la producción de software de alta calidad que se ajuste a las necesidades de sus usuarios finales con unos costos y calendarios predecibles.

El proceso de software propuesto por RUP tiene tres características esenciales que lo hacen único: está dirigido por los Casos de Uso, está centrado en la arquitectura, y es iterativo e incremental. [8]

Un proceso de desarrollo de software define quién hace qué, cómo y cuándo. RUP define cuatro elementos: los roles, que responden a la pregunta ¿Quién?, las actividades que responden a la pregunta ¿Cómo?, los productos, que responden a la pregunta ¿Qué? y las disciplinas de trabajo que responden a la pregunta ¿Cuándo?

4.1.1.1 Disciplinas de Trabajo

Las disciplinas de trabajo proponen la realización de un gran número de actividades y la elaboración de un amplio conjunto de artefactos, que usualmente los proyectos de desarrollo de software se comprometen a desarrollar pero que no realizan en su totalidad debido a la carencia de tiempo y personal. 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 las disciplinas de mayor interés para la realización del presente trabajo de diploma.

- **Análisis y Diseño:** Los objetivos de la disciplina Análisis y Diseño son transformar los requisitos de software en un diseño del sistema, desarrollar una arquitectura robusta y adaptar el diseño al ambiente de implementación. En esta disciplina el rol desempeñado fue el de diseñador, realizando los artefactos: Diagramas de clases del diseño y diagrama de secuencia.
- **Implementación:** Su objetivo principal es la implementación del diseño, además de la realización de pruebas unitarias a los componentes. En la implementación se comienza con el resultado del diseño y se implementa el sistema en término de componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables, y similares. La implementación es el centro durante las iteraciones de construcción, aunque también se lleva a cabo trabajo de implementación durante la fase de elaboración, para crear la línea base ejecutable de la arquitectura, y durante la fase de transición para tratar defectos tardíos. En esta disciplina el rol desempeñado fue el de desarrollador, realizando el artefacto: Modelo de implementación.

- **Prueba:** Su objetivo principalmente es validar que el software trabaje como fue diseñado y que los requisitos fueron implementados correctamente, además de encontrar fallas de calidad en el software y documentarlas. En esta disciplina los roles desempeñados fueron el de analista de pruebas y el probador, realizando los artefactos: Registro de pruebas y Resultados de prueba.

RUP junto con el Lenguaje Unificado de Modelado (Unified Modeling Language), constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

4.2 Lenguaje de Modelado

4.2.1 Lenguaje Unificado de Modelado (UML)

Es un lenguaje para la **Visualización**, **Especificación**, **Construcción**, y **Documentación** de los artefactos que se crean durante el proceso de desarrollo. [9]

UML ante todo es un lenguaje. Un lenguaje proporciona un vocabulario y unas reglas para permitir una comunicación, en este caso este lenguaje se centra en la representación gráfica de un sistema. Permite representar a mayor o menor medida todas las fases de un proyecto informático: desde el análisis de los casos de usos, el diseño con los diagramas de clases, la implementación con los diagramas de componentes, hasta la configuración con los diagramas de despliegue, entre otros.

UML brinda el lenguaje gráfico para modelar, por lo que se necesita de una herramienta que como mínimo facilite la creación de diagramas y la relación entre los componentes de estos, por lo que se describirá Visual Paradigm como herramienta CASE para el modelado.

4.3 Herramienta CASE

El nombre CASE proviene de Computer-Aided Software Engineering (Ingeniería de Software Asistida por Computadora) la cual está diseñada para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de un software.

4.3.1 Visual Paradigm for UML 6.4 Enterprise Edition

Esta herramienta está desarrollada por Visual Paradigm Internacional una de las principales compañías de herramientas CASE. Su mayor éxito consiste en la capacidad de ejecutarse sobre diferentes sistemas operativos lo que le confiere la característica de ser multiplataforma. Es muy fácil de usar y presenta un ambiente gráfico agradable para el usuario.

Visual Paradigm for UML se utiliza para el diseño de software usando el Lenguaje Unificado de Modelado (UML). Proporciona un entorno unificado de diseño de software para el analista del sistema y el desarrollador de software para analizar, diseñar y mantener aplicaciones de software de manera disciplinada y en labor común. [10]

Visual Paradigm es una herramienta que provee soporte para la generación de código, ingeniería inversa para Java, se integra con Eclipse, para soportar las fases de implementación en el desarrollo de software. Es portable y posee gran facilidad de uso. Dicha herramienta ofrece un entorno para la creación de diagramas que se obtienen durante todo el ciclo del desarrollo del software.

4.4 Lenguajes de programación

Para la realización del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. Módulos: Atención de Casos, Bienes Muebles y Materiales, se describió Java como lenguaje de programación.

4.4.1 Java

El lenguaje Java es un lenguaje de propósito general, de alto nivel, que utiliza el paradigma de orientación a objetos. Su sintaxis y tipos están basados principalmente en C++, sin embargo, las diferencias principales con este son la administración de memoria, siendo ejecutada por la máquina virtual automáticamente y no por el código de cada programa, y el soporte de procesos livianos o hilos a nivel del lenguaje, que ayuda a controlar la sincronización de procesos paralelos. Estas características dan al lenguaje Java las propiedades de robustez y seguridad, evitando por ejemplo problemas de desbordamiento de buffer utilizados en ataques a sistemas. [11]

Los programas escritos en Java son compilados como archivos ejecutables de una máquina virtual llamada Java Virtual Machine (JVM). Existen implementaciones de esta máquina para múltiples plataformas, permitiendo ejecutar en diferentes arquitecturas el mismo programa ya compilado. La característica de independencia de la plataforma hace posible el libre intercambio de software desarrollado en Java sin necesidad de modificaciones.

4.5 Plataforma de Desarrollo

En este caso se estudia la plataforma Java, ya que en el epígrafe anterior se describió Java como lenguaje de programación.

4.5.1 Java EE

Java Platform Enterprise Edition (antes J2EE, ahora Java EE) es un estándar para el desarrollo de aplicaciones empresariales (portables, robustas, escalables y seguras) usando tecnología Java. [12]

Java EE se basa en componentes bien definidos para proporcionar soporte del lado del servidor y del lado del cliente para el desarrollo de aplicaciones web de varios niveles. Las aplicaciones Java EE pueden formar parte de tres o cuatro niveles, que son generalmente considerados como partes de la arquitectura de tres niveles, debido a su distribución en tres capas: una capa de cliente, una de nivel medio y un último nivel correspondiente a los datos (Ver Anexo 2).

Java EE configura algunas especificaciones únicas para componentes, éstas incluyen: Enterprise JavaBeans, servlets, JavaServer Pages (JSP) 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. En el nivel de cliente pueden ofrecer apoyo a una multiplicidad de tipos de cliente, tales como páginas HTML generado por JSP o applets de Java. En el nivel intermedio se mantiene los servicios del cliente a través de contenedores Web (por ejemplo se usan los componentes Java Servlets y JSP para crear los datos que se enviarán al cliente) y además apoya con los servicios de componente de la lógica a través de Enterprise JavaBeans (EJB). Ya en el último nivel se aloja la base de datos y los datos del proyecto que son accesibles por el camino de la API estándar (por ejemplo, JDBC[2]).

A continuación, una breve descripción de los componentes de Java EE que hemos venido mencionando:

- **JSP:** Es una tecnología orientada a crear páginas web con programación en Java, es decir, permite fragmentos de código Java incrustado en la página web. Por tanto, las páginas JSP están compuestas de código HTML/XML mezclado con etiquetas especiales para programar scripts de servidor en sintaxis Java.
- **Java Servlets:** Proporcionan un método para escribir programas del lado del servidor. Su uso común es la generación de páginas web dinámicas.

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

- **EJB:** Un EJB es una clase java que tiene varias características por ejemplo: son distribuidos, usan transacciones, son Multi Hilos, persistentes, entre otras más. Muchas de las características de los EJB son proporcionadas por los distintos servidores de aplicación.

4.6 Entorno de Desarrollo Integrado (IDE)

IDE, **I**ntegrated **D**evelopment **E**nvironment en español Entorno de Desarrollo Integrado, es un entorno de programación empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de usuario (GUI). Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Python, Java, C#, Delphi, Visual Basic, entre otros lenguajes.

4.6.1 Eclipse

Eclipse es una herramienta que permite integrar diferentes tipos de aplicaciones, es un proyecto de la comunidad eclipse.org conformado por empresas proveedoras de software como IBM, Borland, Rational, RedHat y SuSE, que tiene como objetivo crear una plataforma de desarrollo de nivel comercial con código abierto y altamente integrable. Eclipse está estructurado en base a plug-ins que pueden ser creados utilizando una API especial para ello y que pueden ser agregados fácilmente (sólo ubicarlos dentro de un directorio). Así, existe una gama amplia y creciente de plug-ins desarrollados para Eclipse, ya sea por empresas o por usuarios comunes. [12]

El sistema de plug-ins agiliza el trabajo del equipo de desarrollo en la implementación. Algunos de los plug-ins más utilizados en desarrollo del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. Módulos: Atención de Casos, Bienes Muebles y Materiales en on:

- **Hibernate Tools:** Constituye un conjunto de herramientas para facilitar el uso del framework Hibernate. Las principales funcionalidades que brinda son: un editor de mapeos de los metadatos de la base de datos a las clases y una consola para la ejecución de consultas HQL[3]. Permite realizar ingeniería inversa a partir de la base de datos de las clases y de los mapeos de las entidades.

[3]**HQL (Hibernate Query Lenguaje):** Este lenguaje es similar a SQL y es utilizado para obtener objetos de la base de datos según las condiciones especificadas en el HQL.

- **JPA Tools:** Java Persistence API, un modelo de programación más simple para la persistencia de la Entidad, desarrollada para la plataforma Java EE. Esta API busca unificar la manera en que funcionan las utilidades que proveen un mapeo objeto-relacional con el objetivo de no perder las ventajas de la orientación a objetos al interactuar con una base de datos, y permitir usar objetos regulares (conocidos como POJO's).
- **JUnit**: es un marco simple para escribir pruebas repetibles. 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.

4.7 Sistema Gestor de Base de Datos

Los Sistemas de Gestión de Bases de Datos (SGBD) es un conjunto de programas que permiten crear y mantener una base de datos, asegurando su integridad, confidencialidad y seguridad.

4.7.1 PostgreSQL 8.3

PostgreSQL es un Sistema de Gestión de Bases de Datos Objeto-Relacional (ORDBMS) basado en la versión 4.2 de Postgres, desarrollado en la Universidad de California en Berkeley del Departamento de Ciencias de la Computación. Postgres fue pionera en muchos conceptos que sólo estuvo disponible en algunos sistemas de bases de datos comerciales mucho más tarde.

PostgreSQL es un descendiente de código abierto del código original de Berkeley. Soporta una gran parte del estándar SQL y ofrece muchas características modernas como son: consultas complejas, claves externas, puntos de vista, integridad transaccional y control de concurrencia multi- versión. [13]

El SGBD PostgreSQL ya que es de código abierto trabaja bajo la licencia BSD[4], es decir, sus potencialidades están en constante perfeccionamiento, permitiendo su uso y distribución sin costo. Es también utilizado porque es un SGBD potente y multiplataforma.

[4] **BSD:** Es la licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Es una licencia de software libre permisiva, además permite el uso del código fuente en software no libre.

4.8 Herramienta para las pruebas de Carga y Estrés al Sistema

4.8.1 Apache JMeter

Apache JMeter es un software de código abierto, una aplicación de escritorio Java 100% pura, diseñada para cargar el comportamiento de pruebas funcionales y la medición del rendimiento. Originalmente fue diseñada para probar las aplicaciones Web, pero desde entonces se ha expandido a otras funciones de prueba. Apache JMeter puede ser utilizado para probar el rendimiento tanto en los recursos estáticos como en los dinámicos (archivos, Servlets, bases de datos y consultas, servidores FTP y mucho más). Puede ser utilizado para simular una carga pesada en un servidor, de red o un objeto para poner a prueba su resistencia o para analizar el rendimiento general en diferentes tipos de carga.[14]

5. Marcos de Trabajo o Frameworks

En el desarrollo de Software, un framework es una estructura conceptual y tecnológica de soporte definida, normalmente, con artefactos o módulos de software concretos, en base a la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado, entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio. [15]

Existen varias clasificaciones para los framework de acuerdo a diferentes criterios:

- **Framework de aplicación:** Proveen un amplio rango de funcionalidades típicamente usadas en una aplicación. Interviene en varias capas de la aplicación como Interfaz de Usuario, Acceso a Datos, entre otros.
- **Framework de dominio:** Proporciona funcionalidades para un dominio específico de aplicación.
- **Framework de soporte:** Se dirigen a dominios muy específicos dentro de una aplicación como manejo de memoria, reportes, entre otros.

En el desarrollo de aplicaciones similares al Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela, el uso de frameworks se ha convertido en algo verdaderamente importante porque implica ahorro en tiempo

de diseño y código, puesto que implementa un conjunto de patrones de diseño, lo que permite su reutilización como componentes de software y a su vez suelen implementar las partes más engorrosas y difíciles del dominio del problema, permitiendo que el desarrollador se concentre en implementar las tareas propias de la aplicación. Algunos de los frameworks utilizados en el desarrollo del presente trabajo son:

5.1 Spring

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 JEE (Java 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. [16]

Además, se considera a Spring como un framework ligero o liviano ya que no es una aplicación que requiera de mucho para su ejecución. Una de las características que ayuda a su éxito es que es una aplicación de código abierto, lo que significa que no tiene ningún costo ni se necesita una licencia para utilizarlo, por lo tanto, da la libertad a muchas empresas y desarrolladores a incursionar en la utilización de esta aplicación. Spring no intenta “reinventar la rueda” sino integrar las diferentes tecnologías existentes, en un solo framework para el desarrollo más sencillo y eficaz de aplicaciones JEE portables entre servidores de aplicación.

5.1.1 Arquitectura de Spring

El framework Spring se compone de elementos organizados en 20 módulos. Estos módulos se agrupan en las capas Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, y Test, como se muestra en la siguiente imagen. [17]

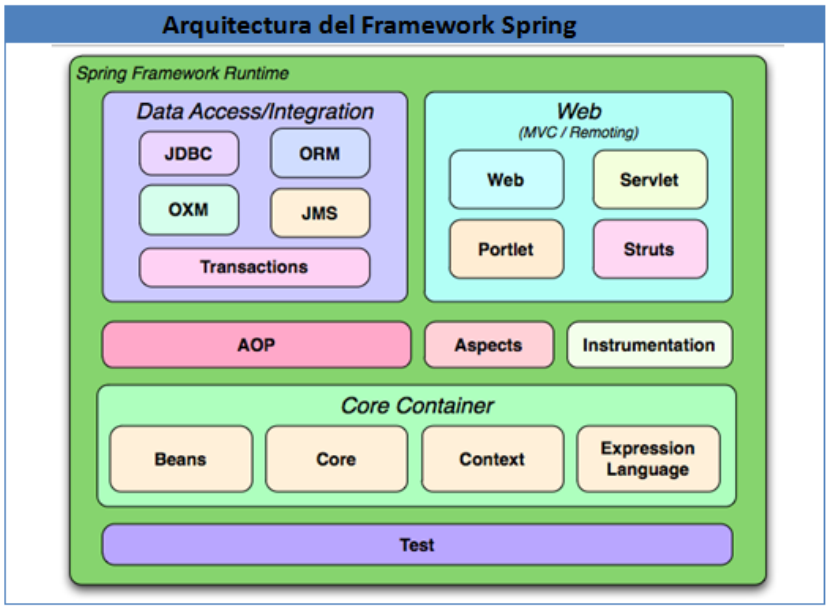


Fig. 1. Arquitectura del Framework Spring

Al ser Spring modular, se pueden utilizarse solo aquellas partes que se necesiten, sin necesidad de recurrir al resto, a continuación se dará una breve explicación de las capas que componen la arquitectura de Spring y de los módulos más utilizados en la realización del presente trabajo.

Capa Core Container o Núcleo de Contenedores

Esta capa es la que provee la funcionalidad esencial del framework, está compuesta por los módulos Core, Beans, Context, y Expression Language.

- **Core y Beans:** Estos módulos contenidos son los que provee la funcionalidad esencial del framework, está compuesta por el Bean Factory, el cual utiliza el patrón de Inversión de Control (IoC[5]) y configura los objetos a través de Inyección de Dependencia. Este módulo es el que convierte a Spring en un contenedor y el módulo Context es el que lo convierte en un framework.
- **Context**: Este módulo es un archivo de configuración que provee de información contextual a todo el framework en general, es decir, a todos los componentes. Además, provee servicios

[5]**IoC (Inversión de Control)**: Permite al framework que lo utilice llamar al código, en lugar de que el código de la aplicación llame alguna librería, invierte el proceso y se termina llamando al código.

enterpise JNDI[6], EJB, e-mail, validación y funcionalidad de agenda. Brinda diferentes funcionalidades algunas de ellas son: cargar múltiples contextos, contextos de herencia, acceso a recursos, propagación de eventos, entre otros más.

Capa Data Access/Integration o 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 framework 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 Framework Spring.
- El módulo **Web-Portlet** proporciona la implementación MVC para ser utilizado en un entorno de portlets[7] y refleja la funcionalidad del módulo Web-Servlet. . El Web de MVC de Spring tiene muchas similitudes con otro framework para web pero este posee características que lo vuelven único, algunas de ellas son: Spring hace una clara división entre controladores, modelos JavaBeans y vistas, además está basado en interfaces y es bastante flexible, Spring no obliga a utilizar JSP como única tecnología de vistas también se pueden utilizar otras. En la arquitectura base de Spring MVC el DispatcherServerlet es el corazón y los controladores son el cerebro ya

[6] **JNDI**, Java Naming and Directory Interface **(**Interfaz de Nombrado y Directorio Java**),** es un API para servicios de directorios.Esto permite a los clientes descubrir y buscar objetos y nombres a través de un nombre.

[7] **Portlets**: Son componentes modulares de las interfaces de usuario gestionadas y visualizadas en un portal web. Los portlets producen fragmentos de código de marcado que se agregan en una página de un portal.

que son especializados en tareas particulares, pero siempre con el fin de procesar la petición (Ver Anexo 3).

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. Es una técnica .que permite a los programadores descartar ya sean las preocupaciones cross-cutting, o el comportamiento de divisiones de responsabilidad, como logging y manejo de transacciones. El núcleo de construcción es el Aspect, que encapsula comportamiento que afectan a diferentes clases, en módulos que pueden ser reutilizados.

Capa Test

- La capa de prueba apoya el examen de los componentes de Spring con JUnit o TestNG[8]. Proporciona carga constante de Application Contexts y el almacenamiento en caché de los contextos. También proporciona objetos mock que se pueden utilizar para probar el código en forma aislada.

5.2 Dojo Toolkit

El conjunto de herramientas “Dojo” ofrece un desarrollo rápido y modular con excelentes herramientas para probar y optimizar la productividad. No necesita el paso de “Compilación” de las aplicaciones de internet (WEB) e internet móvil. Este incluye rápidos y eficientes APIs para cada aplicación de internet (WEB): Ajax, Events, DOM, Querying, effects y muchos más. Puedes lograr gran accesibilidad del usuario con sofisticado pero a la vez sencillos widgets (controles). [18]

Dojo es un framework escrito en lenguaje de programación javascript, que permite a los desarrolladores de aplicaciones para internet enriquecer de funcionalidades y efectos dinámicos las páginas que programan.

Dojo cuenta con dojo, dijit y dojox, proporcionan librerías totalmente de código abierto muy bien pensadas y diseñadas. También se puede integrar fácilmente con Spring, framework de aplicación que se utilizará para desarrollar el sistema. [19]

[8] **TestNG**: Es un framework de pruebas inspiradas en JUnit y NUnit pero introduciendo algunas nuevas funcionalidades que lo hacen más potente y más fácil de usar.

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[9] Document Object Model (DOM) API, scripting remoto, Firebug Lite, dragand 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.

5.3 Hibernate

Hibernate es de alto rendimiento para la persistencia objeto-relacional y servicio de consulta. La más flexible y potente solución objeto-relacional en el mercado. Hibernate se encarga de la asignación de las clases Java a tablas de bases de datos y de tipos de datos Java a tipos de datos SQL. Ofrece consulta y recuperación de datos que reducen significativamente el tiempo de desarrollo. El objetivo de diseño de Hibernate es relevar al desarrollador del 95% de las tareas comunes de programación de datos relacionadas con la persistencia, al eliminar la necesidad manual, el procesamiento de datos a mano utilizando SQL y JDBC. Sin embargo, a diferencia de muchas otras soluciones de persistencia, Hibernate no oculta el poder de SQL de ti y garantiza que su inversión en la tecnología relacional y el conocimiento sean tan válidos como siempre. [20]

Algunas de las características que hacen que muchos desarrolladores se decidan por este framework son:

- Hibernate permite hacer un mapa con la representación de los datos del modelo de objeto hacia un modelo de datos relacional y su correspondiente esquema de base de datos, evitando así la

[9] **W3C: E**s el abreviado de World Wide Web Consortium, es un consorcio internacional que produce recomendaciones para la World Wide Web.

diferencia entre como los datos son presentados en “Objetos” frente a las Base de Datos Relacionales.

- Hibernate es licenciado bajo la fuente abierta LGPL[10] y se puede descargar gratis para ambos, despliegue de desarrollo y producción. La licencia también permite insertar los ISV[11] y distribuir Hibernate libre de cargos.
- Hibernate elimina el trabajo tedioso y repetitivo de codificar, permitiendo así que los desarrolladores se enfoquen en el problema del negocio. Hibernate puede reducir significativamente el tiempo de desarrollo y como un proveedor de solución Objeto-Relacional persistente reducirá significativamente la cantidad de líneas de códigos.

5.4 JasperReports

Es importante señalar que es el 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.

JasperReports es un framework bastante completo para desarrollar reportes tanto web como desktop en Java. En el mundo de código abierto es el motor de informes más popular, está escrito completamente en Java y es capaz de utilizar los datos procedentes de cualquier tipo de fuente de datos y presentar los documentos con precisión de píxel que se puede ver, imprimir o exportar en una variedad de formatos de documentos incluyendo HTML, PDF, Excel, OpenOffice y Word. [21]

Algunas de las características que provee JasperReports son:

- Amplios archivos de ejemplos que te permiten prepararte y ponerte en marcha rápidamente.
- Fuentes completas y transparentes con códigos APIs documentados y listos para ejecutar los ficheros de construcción.

---

[10] **LGPL (Lesser General Public License o Licencia Pública General Menor):** Permite realizar versiones comerciales de un producto final que contenga como herramienta adicional un programa LGPL. Por lo tanto, LGPL puede ser utilizada o enlazada con software propietario. Exige registrar todos los cambios realizados por terceros, a manera de no afectar la reputación del autor original del software.

[11] **ISV (Independent Software Vendor o Proveedores de Software Independientes)**: Es un término de negocios para empresas especializadas en la fabricación o venta de software

- JasperReports puede utilizar cualquier proveedor de origen de datos, lo que le permite extender las capacidades de presentación de informes para casi cualquier aplicación de terceros.

6. Conclusiones

Se utilizó como guía para el proceso de desarrollo del software RUP que utiliza como lenguaje de modelado UML y Visual Paradigm como herramienta CASE para el modelado de los artefactos. Se realizó una descripción de las herramientas utilizadas para la implementación, especificando Java como lenguaje de programación, JEE como plataforma de desarrollo, Eclipse como IDE de desarrollo, Dojo como framework de interfaz, Spring como framework de aplicación, Hibernate como framework ORM, Jasper Report para la generación de los reportes, PostgreSQL como gestor de base de datos. Además se describieron JUnit y Apache JMeter que se usaron en la realización de las pruebas.

2) CAPÍTULO 2. DISEÑO DEL SISTEMA

1. Introducción

En el presente capítulo se describe la solución para la problemática planteada a través del diseño, se enfoca a cómo el sistema cumple sus objetivos teniendo en cuenta los requisitos funcionales y no funcionales, Se explica la arquitectura y los principales patrones de diseño utilizados, además se realizan los diagramas de clases del diseño y los diagramas de secuencias de los módulos: Atención de Casos, Bienes Muebles y Materiales.

2. Estilo Arquitectónico utilizado

Arquitectura de Software

La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución. [22]

Es una vista estructural de alto nivel, ocurre muy tempranamente en el ciclo de vida y define los estilos o grupos de estilos adecuados para cumplir con los requerimientos no funcionales.

El estilo arquitectónico y los patrones de diseño que se mencionan en el presente capítulo y que fueron los utilizados en el desarrollo del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. Módulos: Atención de Caso, Bienes Muebles y Materiales fueron definidos en investigaciones anteriores. [6]

Estilo Arquitectónico

Un estilo arquitectónico define las reglas generales de organización y las restricciones en la forma y la estructura de un grupo numeroso y variado de sistemas de software. [23]

Se puede concluir que: [24]

- Un estilo contempla las cuatro “C” de la Arquitectura de Software (componentes, conectores, configuraciones y restricciones (constraints), por sus siglas en inglés).
- Sirve para sintetizar estructuras de soluciones que luego serán refinadas a través del diseño.
- Pocos estilos abstractos encapsulan una enorme variedad de configuraciones concretas.

- Definen los patrones posibles de las aplicaciones, evitando errores arquitectónicos.
- Permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales.

Existen diferentes estilos arquitectónicos por ejemplo: Estilos de Flujos de Datos, Estilos Centrado en Datos, Estilos Peer to Peer, Estilo de Llamada y Retorno entre otros más. Dentro de los Estilos de Llamada y retorno se encuentra el estilo Arquitectura en Capas el cual fue seleccionado para la realización del sistema.

2.1 Arquitectura en Capas

El estilo arquitectónico Arquitectura en Capas organiza el modelo de diseño a través de capas, que pueden estar físicamente distribuidas, donde los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores. Este estilo es importante porque simplifica la comprensión y la organización del desarrollo de sistemas complejos, reduciendo las dependencias de forma que las capas más bajas no son conscientes de ningún detalle o interfaz de las superiores (Ver Anexo 4).

A continuación una breve explicación de las capas que componen al estilo arquitectónico Arquitectura en Capas:

2.1.1 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 JPA con el soporte del 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.

2.1.2 Capa de Lógica de Negocio

Es donde se localiza la lógica del negocio. Esta capa recibe la petición del usuario a través de la capa de presentación y se encarga de darle curso. En esta capa se definen e implementan 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.

2.1.3 Capa de Presentación

Define e implementa todo lo relacionado con la interfaz gráfica de usuario. Aquí reside la definición de las peticiones que el usuario puede realizar sobre la aplicación, los controladores que manejan el flujo web y la comunicación con las interfaces de la capa de negocio, un formulario es un ejemplo de una capa de presentación. Las responsabilidades principales de la capa de presentación son: la navegabilidad del sistema, el formateo de los datos de salida, la validación de los datos de entrada y la construcción de la interfaz gráfica de usuario.

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

MVC es un concepto para separar la presentación de los datos. En el caso de Spring, este modelo se implementa como se describe a continuación:

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. [25]

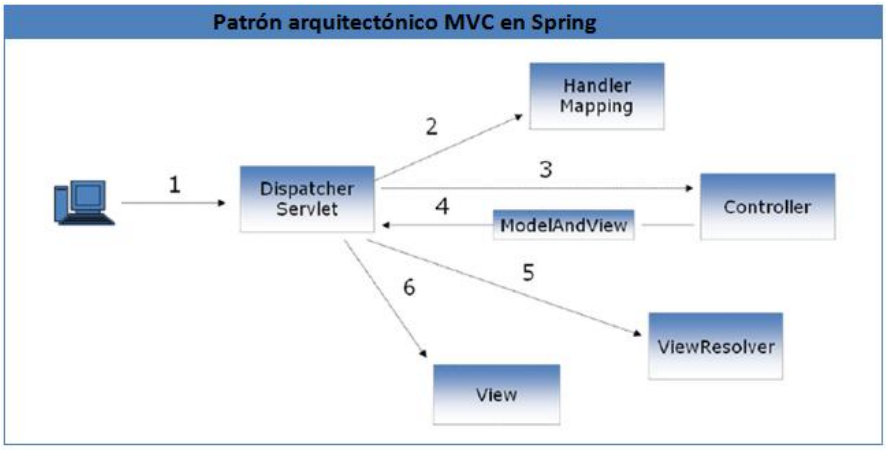


Fig. 2. Implementación del patrón arquitectónico MVC en Spring

A continuación se describen los componentes que utiliza el MVC en su implementación, ya anteriormente mencionados:

- **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 sistema en el archivo **app_servlet.xml**.
- **HandlerMapping:** Permite mapear las peticiones HTTP, utilizando las URLs[12] o partes de estas, 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 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.

3. Patrones de Diseños Utilizados

Patrones de Diseño

Los patrones de diseño no son fáciles de entender, pero una vez entendido su funcionamiento, los diseños serán mucho más flexibles, modulares y reutilizables. Son soluciones simples y elegantes a

[12] **URL (**Uniform Resource Locator**)** son las siglas de **Localizador de Recurso Uniforme**, es la dirección global de documentos y de otros recursos en la World Wide Web.

problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan.

[Gamma] Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. [26]

Se puede decir que, un patrón de diseño es:

- Una solución estándar para un problema común de programación.
- Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios.
- Una manera más práctica de describir ciertos aspectos de la organización de un programa.
- Conexiones entre componentes de programas.
- La forma de un diagrama de objeto o de un modelo de objeto.

A continuación se describen los patrones de diseños que permitieron que el Sistema Informático de Gestión de Información para las CRs de Prevención del Delito de la República Bolivariana de Venezuela tuviera una mejor organización y un código flexible a la hora de implementar los módulos: Atención de Caso, Bienes Muebles y Materiales.

3.1 Patrón de diseño Facade (Fachada)

Este patrón sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Utilizado para reducir la dependencia entre clases, ya que ofrece un punto de acceso al resto de las clases. La idea principal es la de ocultar todo lo posible la complejidad de un sistema, el conjunto de clases o componentes que lo integran, de forma que solo se ofrezca un (o unos pocos) punto de entrada al sistema tapado por la fachada.

Una ventaja más de usar una clase fachada para comunicar las dos partes o componentes, es la de aislar los posibles cambios que se puedan producir en alguna de las partes, es decir, fortalece su modificabilidad. Si cambias, por poner un ejemplo, el medio de comunicación o de almacenamiento de una de las partes, la otra, que por ejemplo hace la presentación, no tiene por qué enterarse, y viceversa.

Fachada no oculta las clases sino que ofrece una forma más sencilla de acceder a ellas, en los casos en que se requiere se puede acceder directamente a ellas. El uso de este patrón en el sistema se

evidencia en las clases de los paquetes **vnz.mpprij.prevencion.cr.aten.facade** las cuales sirven de fachada entre las clases de los paquetes **vnz.mpprij.prevencion.cr.aten.web** y **vnz.mpprij.prevencion.cr.aten.service.** El fragmento del diagrama de secuencia del escenario "Actualizar Cita" correspondiente al caso de uso "Gestionar Cita" del Módulo Atención de Casos que se muestra en la (Fig. 3) es un ejemplo de lo antes expuesto.

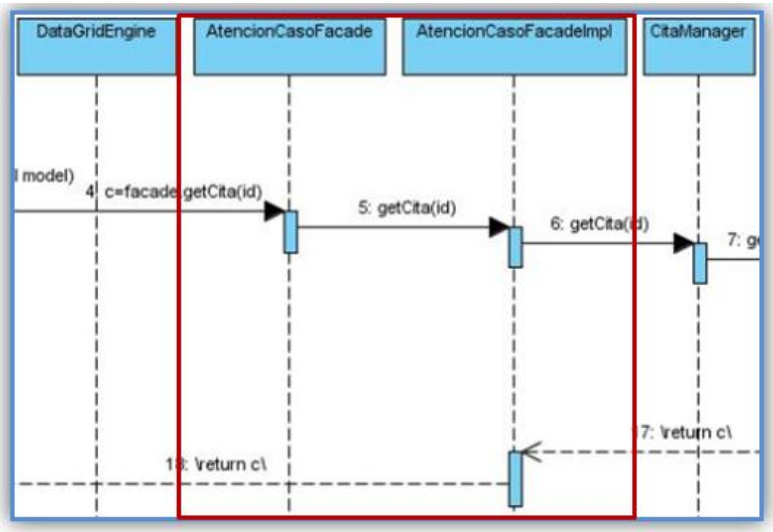


Fig. 3. Ejemplo del patrón Fachada

3.2 Patrón de diseño Dependency Injection (Inyección de dependencias)

Es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase quien cree el objeto. Es una forma de Inversión de Control, que está basada en constructores de Java. Este radica en resolver las dependencias de cada clase (atributos) generando los objetos cuando se arranca la aplicación y luego inyectarlos en los demás objetos que los necesiten a través de métodos set o bien a través del constructor, pero estos objetos se instancian una vez, se guardan en una factoría y se comparten por todos los usuarios (al menos en el caso de Spring) y evitando tener que estar 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 @Autowired como se muestra en la (Fig. 4) 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.

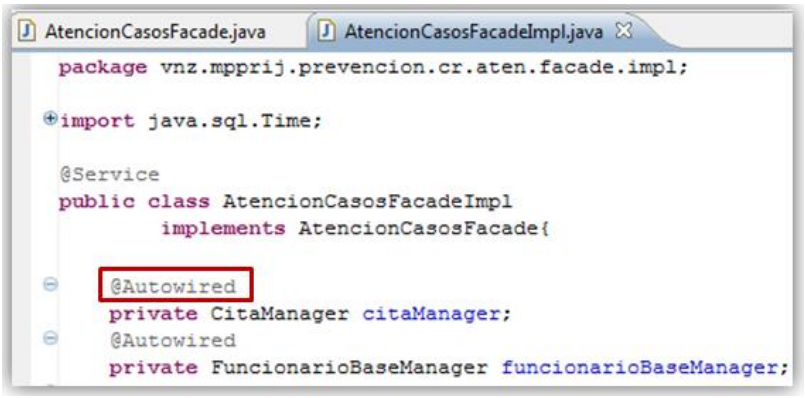


Fig. 4. Ejemplo del patrón Inyección de Dependencia

3.3 Patrón de diseño Front-Controller (Controlador Frontal)

El patrón propone utilizar un controlador como el punto inicial de contacto para manejar las peticiones del usuario en una aplicación. El controlador maneja el control de peticiones, incluyendo la invocación de los servicios de seguridad como la autentificación y autorización, la elección de una vista apropiada, el manejo de errores, y el control de la selección de estrategias de creación de contenido.

Este patrón es utilizado por Spring MVC a través del DispatcherServlet que en el sistema se encuentra configurado en el **app-servlet.** Un ejemplo de cómo funciona este patrón en el sistema sería el siguiente; si el usuario decide visualizar en el navegador la página Actualizar citas, esta petición (la URL cita.form.html) es recibida por el **app-servlet,** este posee un mecanismo para determinar cuál controlador maneja esta petición (**CitaFormController** en este caso), luego este controlador toma la petición y ejecuta la tarea, devolviendo un ModelAndView al **app-servlet**, el que se encarga de despachar la petición a la Vista (cita.form.jsp). El fragmento del diagrama de secuencia del escenario "Actualizar Cita" correspondiente al caso de uso "Gestionar Cita" del Módulo Atención de Casos que se muestra en la (Fig. 5) es un ejemplo de lo antes expuesto.

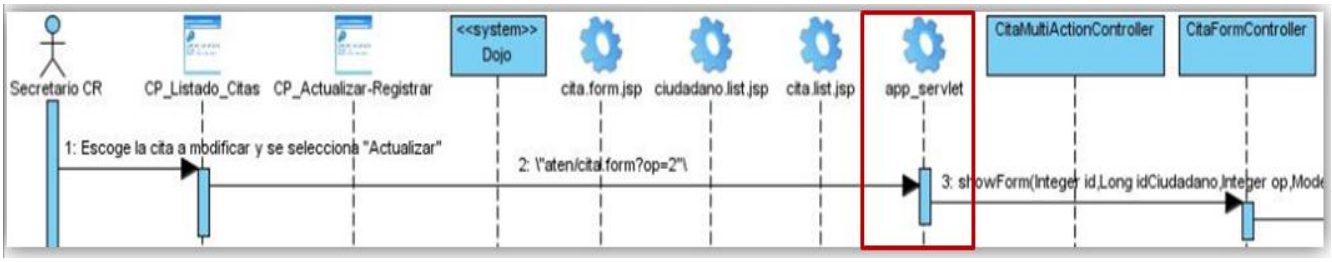


Fig. 5. Ejemplo del patrón Controlador Frontal.

3.4 Patrón de diseño Data Access Object (DAO)

Patrón de Diseño del Core de JEE que centraliza todo el acceso a datos en una capa independiente, aislándolo del resto de la aplicación. Sus principales beneficios son que reduce la complejidad de los objetos de negocio al abstraerlos de la implementación real de la comunicación con la fuente de datos y que permite una migración más fácil de fuente de datos. Se trata de que el software cliente se centre en los datos que necesita y se olvide de cómo se realiza el acceso a los datos o de cuál es la fuente de almacenamiento.

El uso de este patrón en el sistema se evidencia en la capa de Acceso a Datos donde se encuentra un DAO Genérico encargado de realizar la gestión de los datos entre las clases que contienen la lógica de negocio (servicios) y el API JPA. El fragmento del diagrama de secuencia del escenario “Actualizar Cita” correspondiente al caso de uso “Gestionar Cita” del Módulo Atención de Casos que se muestra en la (Fig. 6) es un ejemplo de lo antes expuesto.

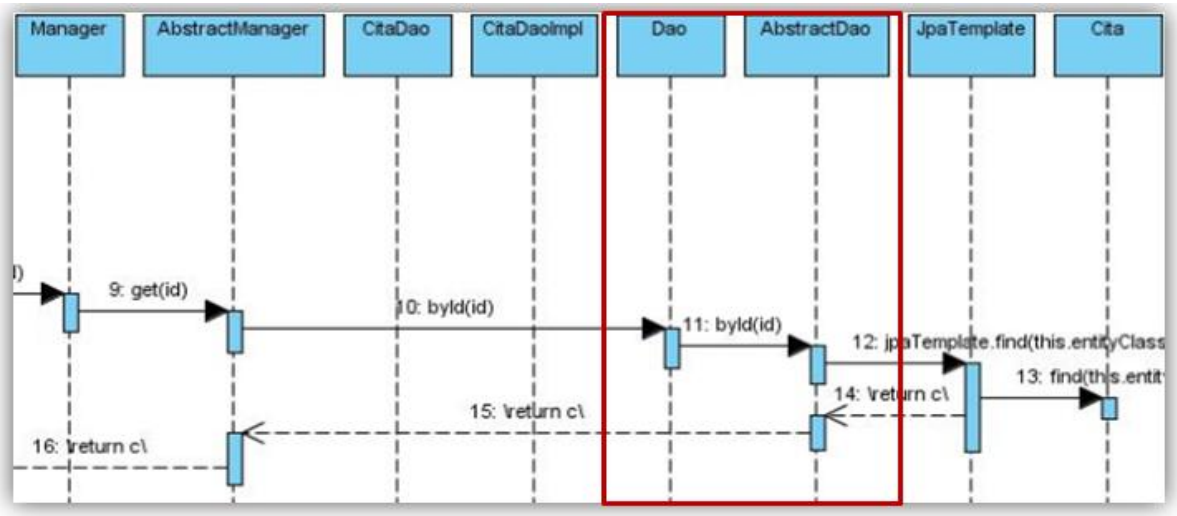


Fig. 6. Ejemplo del patrón DAO

Otros de los patrones que se describieron forma parte de los **P**atrones **G**enerales de **S**oftware para la **A**signación de **R**esponsabilidades (**GRASP General Responsibility Assignment Software Patterns**). Aunque se consideran que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software.

3.6 Patrón de diseño Controller (Controlador)

El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado.

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.aten.web** y dentro de esta están los controladores.

A modo de ejemplo se escoge la clase **CitaFormController.java**, se encarga de controlar la petición de Actualizar una cita , manipulando los datos para en dependencia de estos delegarlos a las clases responsables de ejecutar la acción requerida; devolviéndole luego a la vista física la JSP los datos para presentárselos al usuario. El fragmento del diagrama de secuencia del escenario “Actualizar Cita” correspondiente al caso de uso “Gestionar Cita” del Módulo Atención de Casos que se muestra en la (Fig. 7) es un ejemplo de lo antes expuesto.

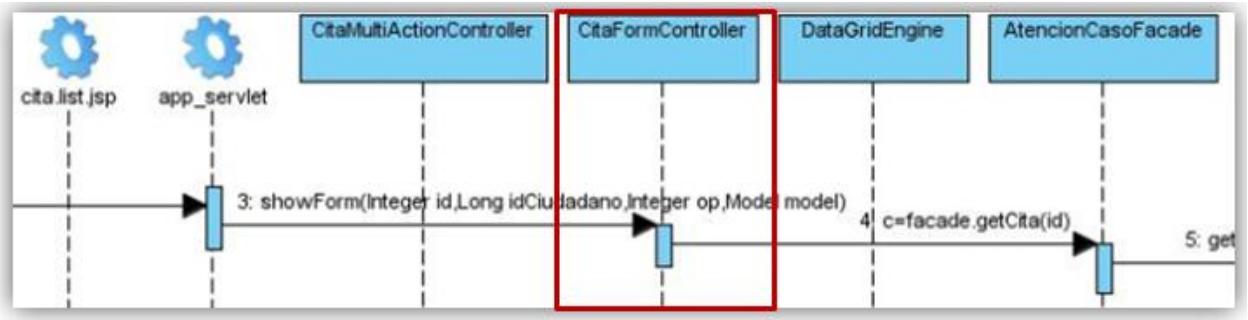


Fig. 7. Ejemplo del patrón Controlador

3.7 Alta Cohesión

La cohesión es la medida de la fuerza que une a las responsabilidades de una clase. Nos dice que la información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la clase. Una clase con baja cohesión es aquella que hace muchas cosas no afines o muchas tareas, lo que trae como resultado dificultades para entender, reutilizar y conservarla. Son delicadas y las afectan constantemente los cambios. Una clase con alta cohesión mejora la claridad y

la facilidad de su uso, su mantenimiento se simplifica y es fácil de reutilizar. A menudo, se genera un bajo acoplamiento.

El sistema fue diseñado en módulos definidos a partir de que en los mismos se manejaran la menor cantidad de entidades posibles de manera tal que las clases pertenecientes a las diferentes capas se le asignaran las responsabilidades necesarias y bien delimitadas para garantizar una alta cohesión generando por ende **bajo acoplamiento**.

3.8 Bajo Acoplamiento

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.

En el sistema esto se evidencia ya que se implementa para cada clase de acceso a datos y de servicio una interfaz que defina sus funcionalidades y luego sus implementaciones concretas. El uso de este patrón en el sistema se evidencia por ejemplo, en las clases de los paquetes **vnz.mpprij.prevencion.cr.aten.facade** donde nos encontramos con la interfaz **AtencionCasosFacade** con todas las funcionalidades definidas y luego nos encontramos con su implementación en la clase **AtencionCasosFacadeImpl**. Todo esto es de gran utilidad, sobre todo por el bajo acoplamiento que permite lo que facilita enormemente el trabajo en equipo.

4. Diagramas de Clases del Diseño

Los diagramas de clases del diseño son un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Definen bien las relaciones de dependencia, generalización y asociación de las clases que involucran el sistema.

El diseño de las clases partió de los requerimientos funcionales definidos. A continuación se muestra en la (Fig. 8) el Diagrama de clases del diseño para el caso de uso Gestionar Cita perteneciente al Módulo de Atención de Casos, el cual abarca los siguientes requisitos funcionales: Registrar cita, Actualizar cita, Eliminar cita y Buscar horarios.

Las funcionalidades correspondientes a los restantes casos de uso del módulo Atención de Casos y los módulos Bienes Muebles y Materiales pueden ser consultadas (Ver Anexo 5). Los diagramas de clases del diseño se encuentran detallados en el documento Modelo de diseño contenido en el Expediente de proyecto.

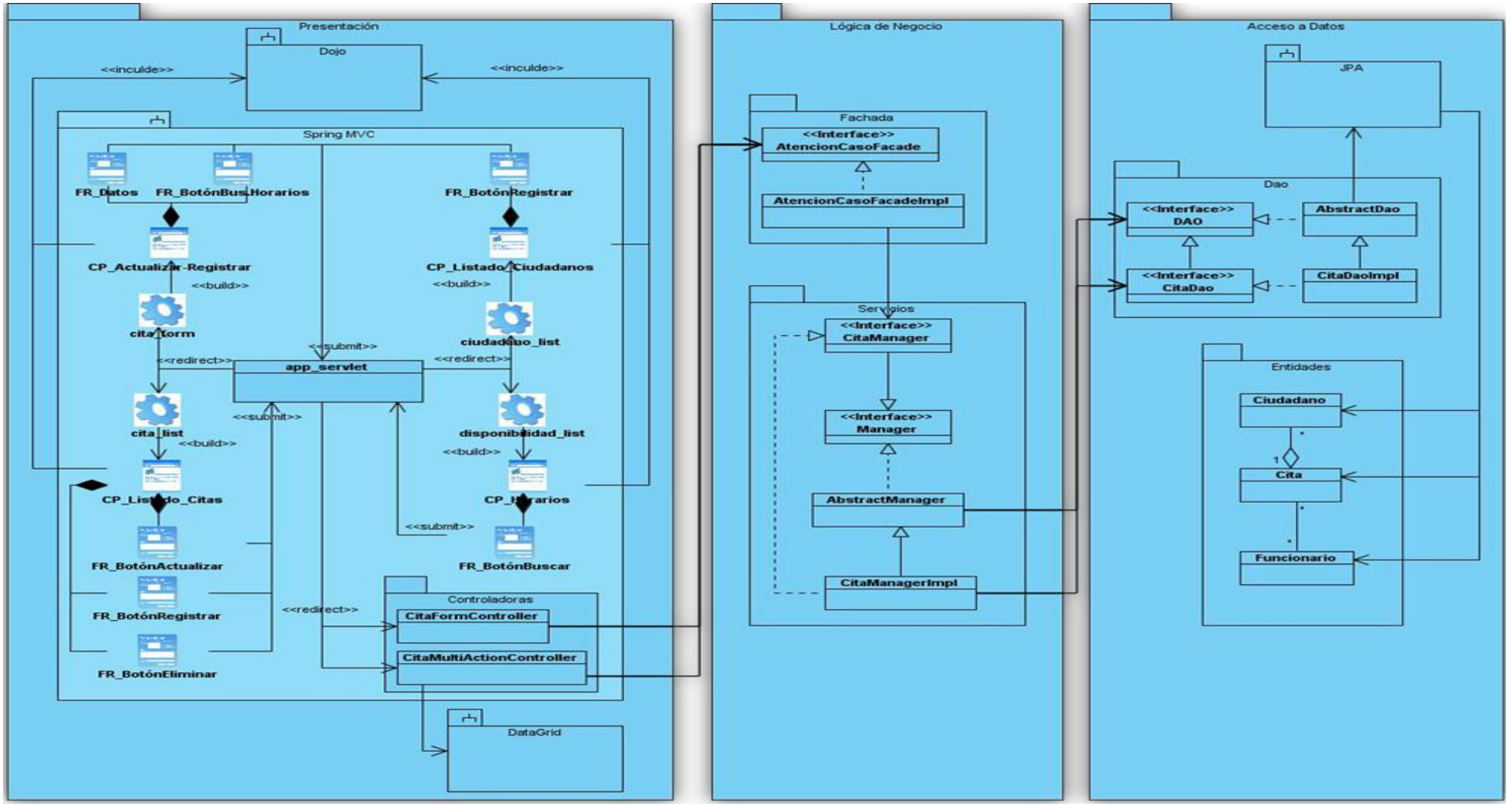


Fig. 8. Diagrama de clases del diseño para el CUS Gestionar Cita.

4.1 Distribución de Clases por Capas

Para una mejor vista del diagrama de diseño del CUS Gestionar cita no se muestran los atributos y métodos de las clases, la distribución de los mismos según la capa a la que pertenecen se muestran en la siguiente tabla (Ver Anexo 6).

5. Diagramas de Secuencia

Los diagramas de Iteración consisten en un conjunto de objetos y sus relaciones, incluyendo los mensajes que se pueden enviar entre ellos. Dentro de estos diagramas se encuentran los diagramas de secuencia y los diagramas de colaboración.

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 del CUS Gestionar Cita: Registra Cita (Fig. 9), Actualizar Cita (Fig. 10), Eliminar Cita (Fig. 11) y Buscar Horarios (Fig. 12) todos pertenecientes al Módulo de Atención de Caso. Para una mayor comprensión se dividieron en imágenes más detalladas, las cuales se encuentran en los anexos pertenecientes a cada diagrama: Registra Cita (Ver Anexo 7), Actualizar Cita (Ver Anexo 8), Eliminar Cita (Ver Anexo 9) y Buscar Horarios (Ver Anexo 10).

Para ver los demás diagramas de secuencia de los módulos: Atención de Casos, Bienes Muebles y Materiales (Ver expediente de proyecto).

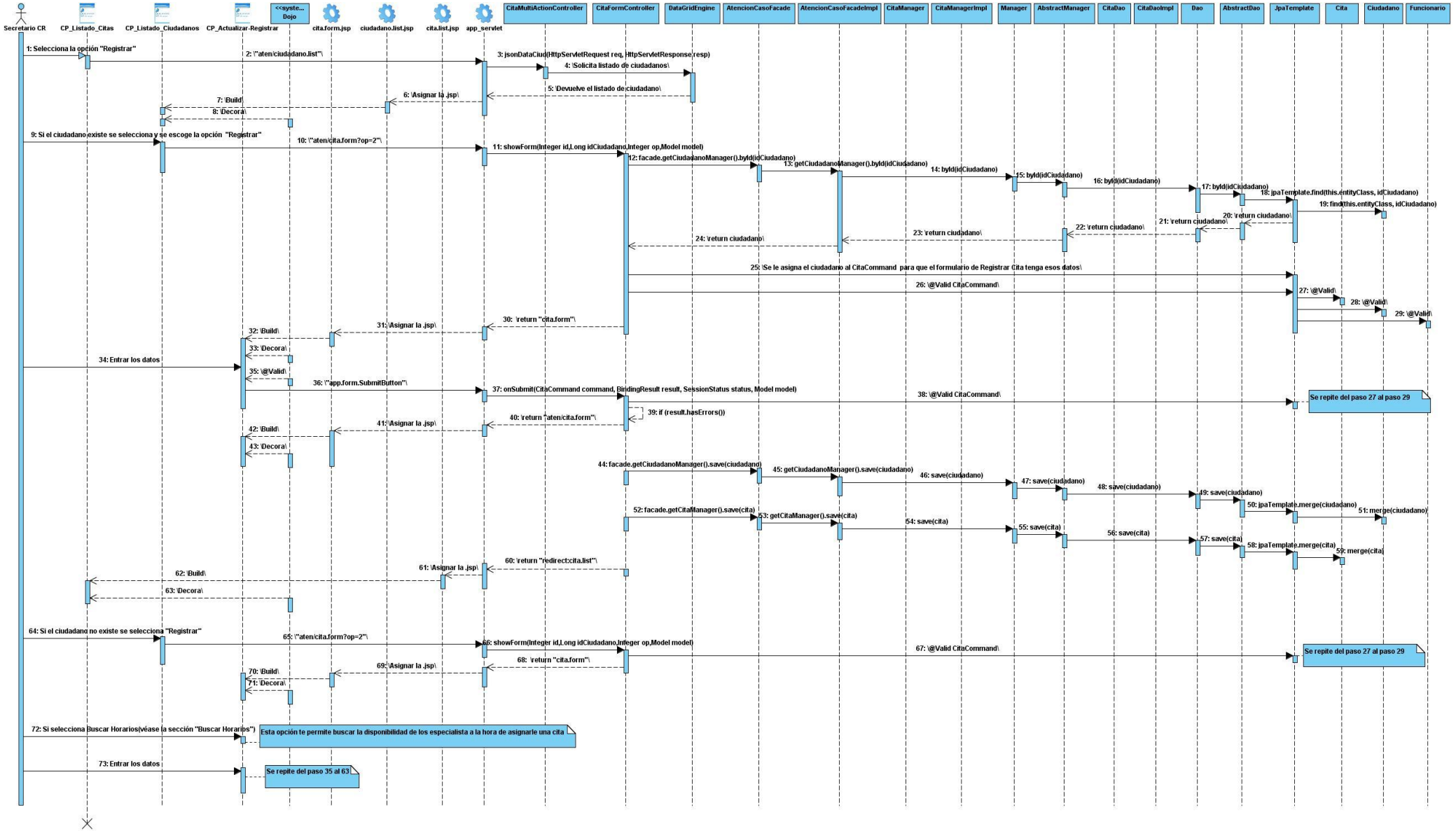


Fig. 9. Diagrama de Secuencia del CUS Gestionar Cita, sección Registra Cita.

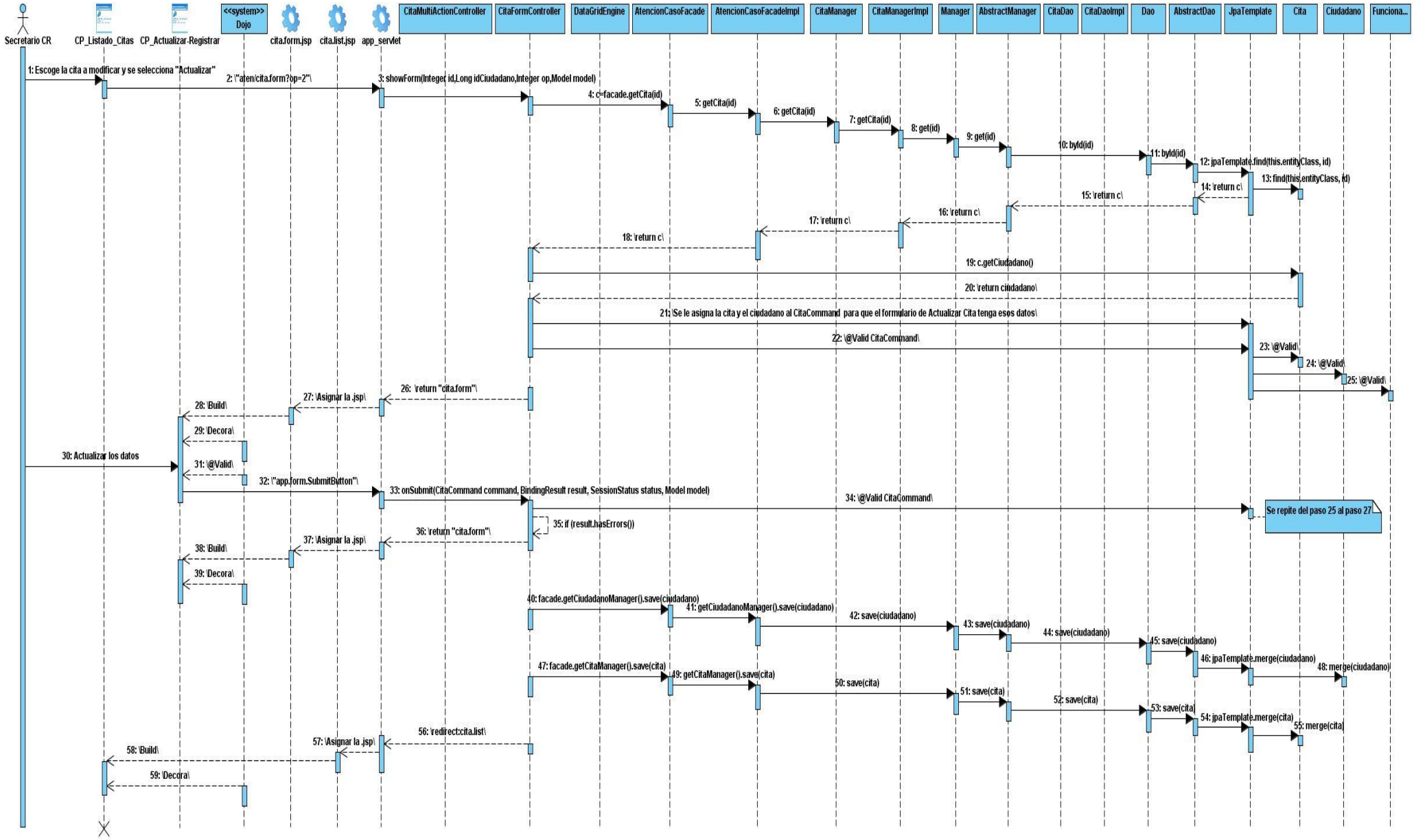


Fig. 10. Diagrama de Secuencia del CUS Gestionar Cita, sección Actualizar Cita.

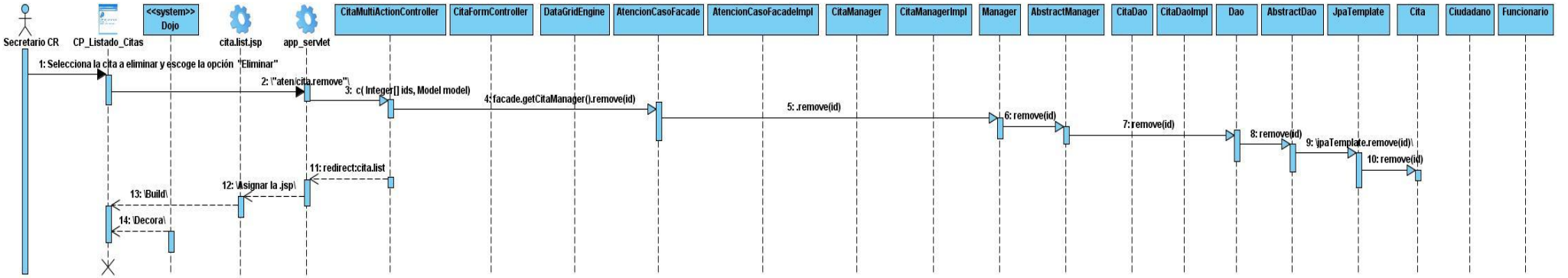


Fig. 11. Diagrama de Secuencia del CUS Gestionar Cita, sección Eliminar Cita.

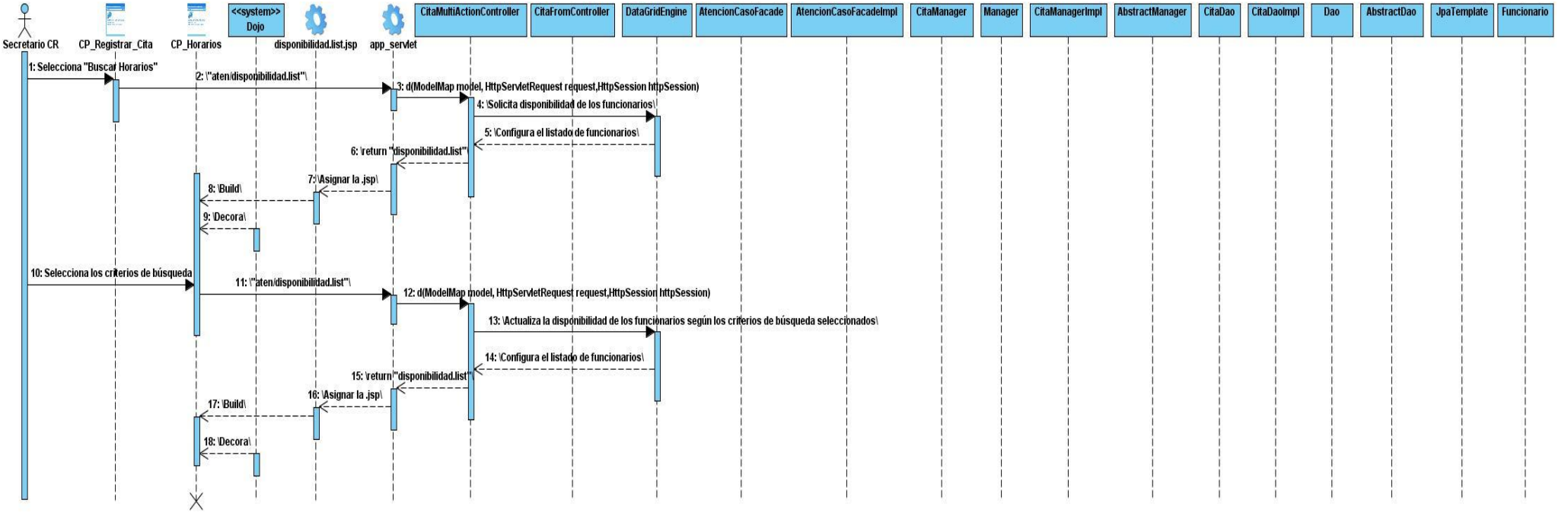


Fig. 12. Diagrama de Secuencia del CUS Gestionar Cita, sección Buscar Horarios.

6. Conclusiones

Los diagramas de clases del diseño, asociados a los casos de usos pertenecientes a los módulos: Atención de Caso, Bienes Muebles y Materiales se realizaron cumpliendo con el estilo arquitectónico “Arquitectura en capas” y aplicándoles los patrones de diseño Controlador Frontal, Controlador, Fachada, Inyección de Dependencias, Alta Cohesión, Bajo Acoplamiento y DAO.

Además se realizaron los diagramas de secuencia de los diferentes escenarios de cada caso de uso perteneciente a los módulos: Atención de Caso, Bienes Muebles y Materiales los cuales muestran la interacción de las clases diseñadas permitiendo una mejor comprensión para la posterior implementación.

3) CAPÍTULO 3. IMPLEMENTACIÓN DEL SISTEMA

1. Introducción

Durante este capítulo se realiza el flujo de trabajo de implementación, el cual comenzó con el resultado del diseño y se implementó el sistema en término de componentes. Se realizaron los diagramas de componentes de los módulos: Atención de Casos, Bienes Muebles y Materiales, además se muestra cómo se van a ejecutar estos componentes en el entorno de hardware donde se situará la aplicación.

2. Modelo de Implementación

El modelo de implementación describe como los elementos del modelo de diseño, por ejemplo, como las clases se implementan en términos de ficheros de código fuentes, ejecutables, etc. Además describe también como se organizan los componentes y como dependen unos de otros [27].

El modelo de implementación se compone de un sistema de implementación, que a su vez consta de varios subsistemas. Estos subsistemas son una forma de organizar el modelo de implementación en partes más manejables y pueden estar formados por componentes, interfaces y subsistemas.

Los componentes son la parte física del sistema (base de datos, ejecutables, librerías, entre otros). Se puede decir que un componente es la materialización de una o más clases, porque una abstracción con atributos y métodos pueden ser implementados en los componentes.

Las interfaces constituyen una forma de separar la especificación de la funcionalidad en términos de operaciones de sus implementaciones en términos de métodos. Se consideran relevantes ya que definen las iteraciones entre los subsistemas.

2.1 Diagrama de componentes

Dentro del Modelo de Implementación se encuentran los diagramas de componentes, estos contienen interfaces, relaciones y componentes. Los mismos describen la descomposición física del sistema (y eventualmente de su entorno organizativo), se realiza en términos de componentes y de relaciones entre los mismos.

A continuación se muestra el Diagrama de Componentes del CUS Gestionar Cita perteneciente al Módulo de Atención de Casos (Fig. 13). Para ver los demás diagramas de componentes de los módulos: Atención de Casos, Bienes Muebles y Materiales (Ver expediente de proyecto).

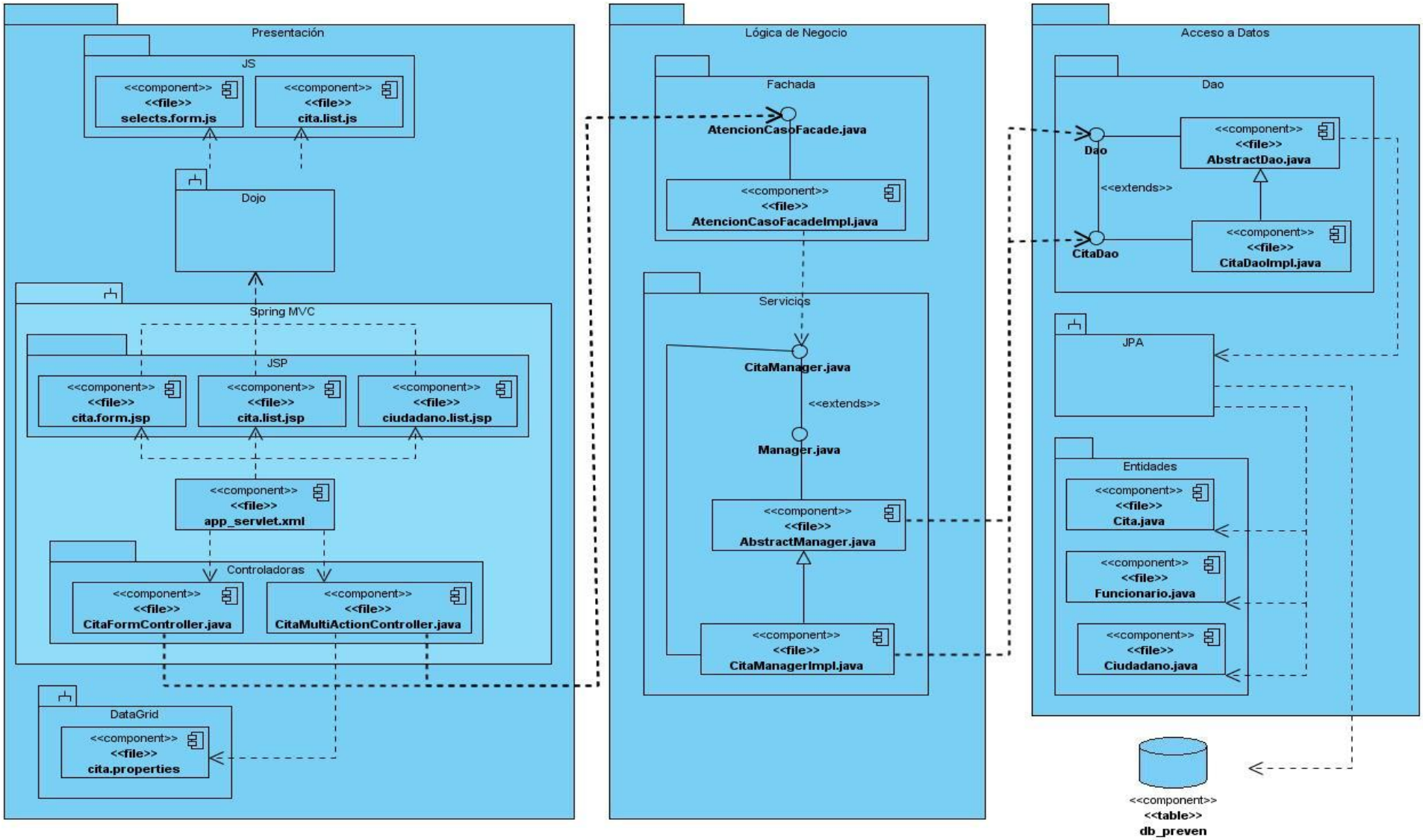


Fig. 13. Diagrama de Componentes del CUS Gestionar Cita.

2.2 Diagrama de Despliegue

El diagrama de despliegue nos muestra una vista de cómo quedará distribuido físicamente el sistema implementado, pudiendo así situar los componentes lógicos desarrollados en el hardware que lo contiene

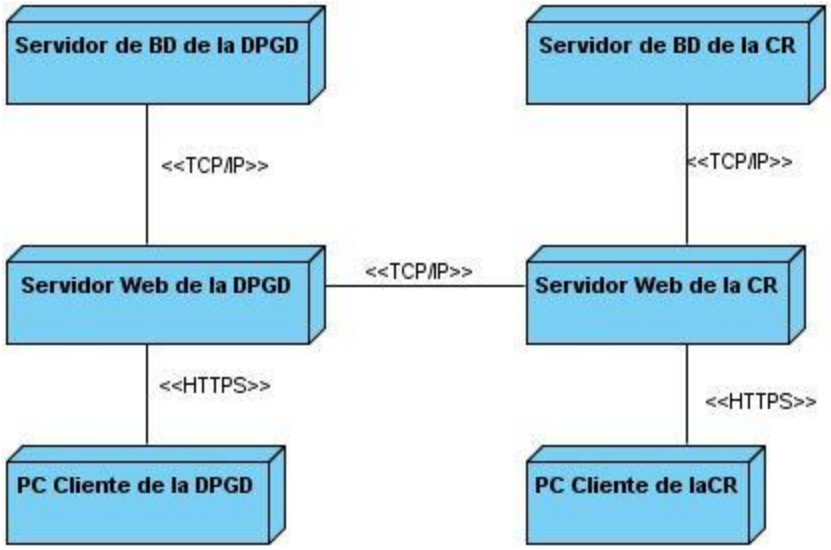


Fig. 14. Diagrama de Despliegue del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela.

Partiendo del diagrama de despliegue del sistema, el diagrama de componentes correspondiente al CUS Gestionar Cita del módulo Atención de Casos anteriormente presentado quedaría distribuido de la siguiente manera:

- **Servidor de BD de la DGPD y la CR:** Estará situada la base de datos (db_preven).
- **Servidor Web de la DGPD y la CR:** Se ejecutarán los subsistemas Spring, DataGrid y JPA, así como los componentes pertenecientes a los paquetes:
  - **JSP:** cita.form.jsp, cita.list.jsp y ciudadano.list.jsp.
  - **Controladoras:** CitaFormController.java y CitaMultiActionController.java
  - **Fachada:** AtencionCasoFacade.java y AtencionCasoFacadeImpl.java
  - **Servicios:** CitaManager.java, CitaManagerImpl.java, Manager.java y AbstractManager.java

- **Dao:** CitaDao.java, CitaDaoImpl.java, Dao.java y AbstractDao.java
- **Entidades:** Cita.java, Funcionario.java y Ciudadano.java

- **PC Cliente de la DPCN y la CR:** Se ejecutará el subsistema Dojo y los componentes del paquete JS (select.form.js y cita.list.js).

3. Código Fuente

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

Estas instrucciones son escritas en un lenguaje de programación , en este caso Java, el cual fue descrito en el Capítulo 1 como lenguaje de programación para la realización del sistema.

3.1 Estándares de Codificación

Un estilo de codificación es una guía por la que todos los desarrolladores deben seguir su trabajo de forma tal que sea comprensible por todos, los estilos de codificación no son patrones preestablecidos que deben seguirse de forma íntegra, sino que, aunque ya existen algunos que son seguidos según el lenguaje de programación en el que se implementa la aplicación, es posible establecer su propio estilo de codificación.

Lo más importante es que el desarrollo de la aplicación se guíe por algún estilo de codificación, pues este permite la comprensión de los programas, haciendo disminuir el número de errores en el desarrollo, contribuye a la documentación del código y al ahorro de tiempo y recursos en la comprensión de lo escrito, ya que cualquier otra persona debería ser capaz de leer el código y entender su funcionamiento.

A continuación alguno de los Estilos de Codificación utilizados en la implementación del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela, módulos: Atención de Casos, Bienes Muebles y Materiales.

Declaración de variables:

- Codificar cada variable con un nombre que intente describir lo más cercano posible la utilidad que va a tener. Ejemplo: dirección, asistencia, observaciones, funcionarios, entre otros.

- Deben empezar con minúscula y en caso de nombres compuestos la siguiente se escribe con mayúscula. Ejemplo: fechaCita, tiempoDuracion, horaAtencion.
- En el caso de las variables que constituyen instancias de clases persistentes deben llamarse del mismo modo que la clase que representa empezando con minúscula. Ejemplo: ciudadano.

```
public class Cita implements Serializable {

    private static final long serialVersionUID = 1L;
    private Integer id;
    private Asistencia asistencia;
    private String direccion;
    private Date fechaCita;
    private Time horaAtencion;
    private String observaciones;
    private Integer tiempoDuracion;

    private Ciudadano ciudadano;
```

Nombre de Métodos:

- El nombre debe describir lo más posible la utilidad que va tener. Ejemplo: RegistrarCita (Cita cita).
- Deben empezar con mayúscula y en caso de nombres compuestos la siguiente también se escribe con mayúscula. Ejemplo: ActualizarCita (Cita cita), EliminarCita (Integer cita), entre otros.
- Los métodos deben ser agrupados por funcionalidad en lugar de por el alcance o la accesibilidad.

```
public interface CitaManager extends Manager<Cita, Integer> {
    Cita getCita(Integer id);
    void RegistrarCita(Cita cita);
    void ActualizarCita(Cita cita);
    void EliminarCita(Integer cita);
}
```

Clases de la capa de Acceso a Datos:

- Las interfaces que representan las operaciones de acceso a datos, correspondientes al patrón de diseño Data Access Object terminan con la palabra “Dao”. Ejemplo: CitaDao.

- Las implementaciones reales de las interfaces DAO comienza con el nombre de la interfaz correspondiente y terminan con la palabra “Impl”. Ejemplo: CitaDaoImpl.

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

```
public interface CitaDao extends Dao<Cita, Integer> {

}

public class CitaDaoImpl extends AbstractDao<Cita, Integer> implements CitaDao{

}
```

Clases de la capa de Negocio:

- Las interfaces que representan las operaciones del negocio terminarán con la palabra “Facade”. Ejemplo: AtencionCasosFacade.
- Las implementaciones de las interfaces de negocio comenzarán con el nombre de la interfaz correspondiente y terminaran con la palabra “Impl”. Ejemplo: AtencionCasosFacadeImpl.

```
public interface AtencionCasosFacade {

    CitaManager getCitaManager();
    CasoManager getCasoManager();

@Service
public class AtencionCasosFacadeImpl
        implements AtencionCasosFacade{
```

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

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

```
public interface CitaManager extends Manager<Cita, Integer> {
    Cita getCita(Integer id);
    void RegistrarCita(Cita cita);
    void ActualizarCita(Cita cita);
    void EliminarCita(Integer cita);
}

@Service
public class CitaManagerImpl extends AbstractManager<Cita, Integer> implements CitaManager {
```

Clases de la capa de Presentación:

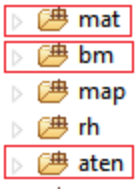
- Las clases que manejan el flujo de la capa de presentación, es decir, las controladoras, terminarán con la palabra “Controller”.

```
public class CitaFormController{

public class CitaMultiActionController{
```

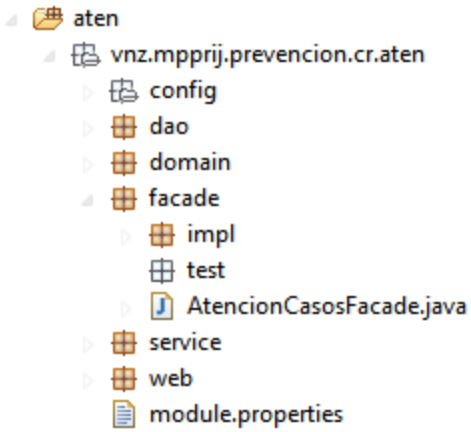
Estructura del sistema

Para cada módulo que conforme el sistema la nomenclatura quedaría de la siguiente forma: un paquete central con el nombre que representa al módulo, en este caso se utilizó la abreviatura **aten** para hacer referencia al módulo Atención de Casos, **mat** para hacer referencia al módulo Materiales y **bm** para hacer referencia al módulo Bienes Muebles.



Luego dentro de estas carpetas se encuentran los paquetes **vnz.mpprij.prevencion.cr. <Abreviatura que identifica al Módulo>**, que serían los paquetes padres, los demás paquetes serán nombrado en

correspondía a las funciones de las clases que contendrán. Los ficheros se agruparan por paquetes, cada paquete se corresponderá con cada una de las capas a implementar. La capa de presentación estará agrupada en la capeta **web**, en el caso de la capa de lógica de negocio, las interfaces estarán agrupadas en las carpetas **service** y **facade**, las clases que implementan estas interfaces estarán en las subcarpetas **impl**. Las clases que representan el dominio estarán en la carpeta **domain**, en caso de alguna configuración o clases auxiliares se tendrán en la carpeta **config**, y si es necesario realizar una operación que no esté definida en el Dao Genérico se creará un Dao que herede de este y el mismo se tendrá en la carpeta **dao**. Ejemplo:



4. Imágenes de la Aplicación

A continuación se muestran algunas de las interfaces correspondientes al CUS Gestionar Citas del módulo Atención de Casos.

Registrar Cita

Para registrar una cita, en primer lugar se accede al Listado de citas (Fig.15) con el objetivo de buscar la función que nos va a permitir realizar dicha operación (botón Registrar). Una vez seleccionada la opción para registrar la cita el sistema muestra una interfaz con un Listado de ciudadanos (Fig.16), permitiendo este, buscar si el ciudadano al que se le realizará la cita ya se encuentra en el sistema. Si la búsqueda resulta exitosa se selecciona el ciudadano y se pasa a registrarle la cita (botón Registrar), en caso contrario se pasa a registrar una cita (botón Registrar) que al mismo tiempo nos permitirá insertar un nuevo ciudadano al sistema.

Fig. 15. Imagen de la aplicación correspondiente al CUS Gestionar Citas (Registrar Cita).

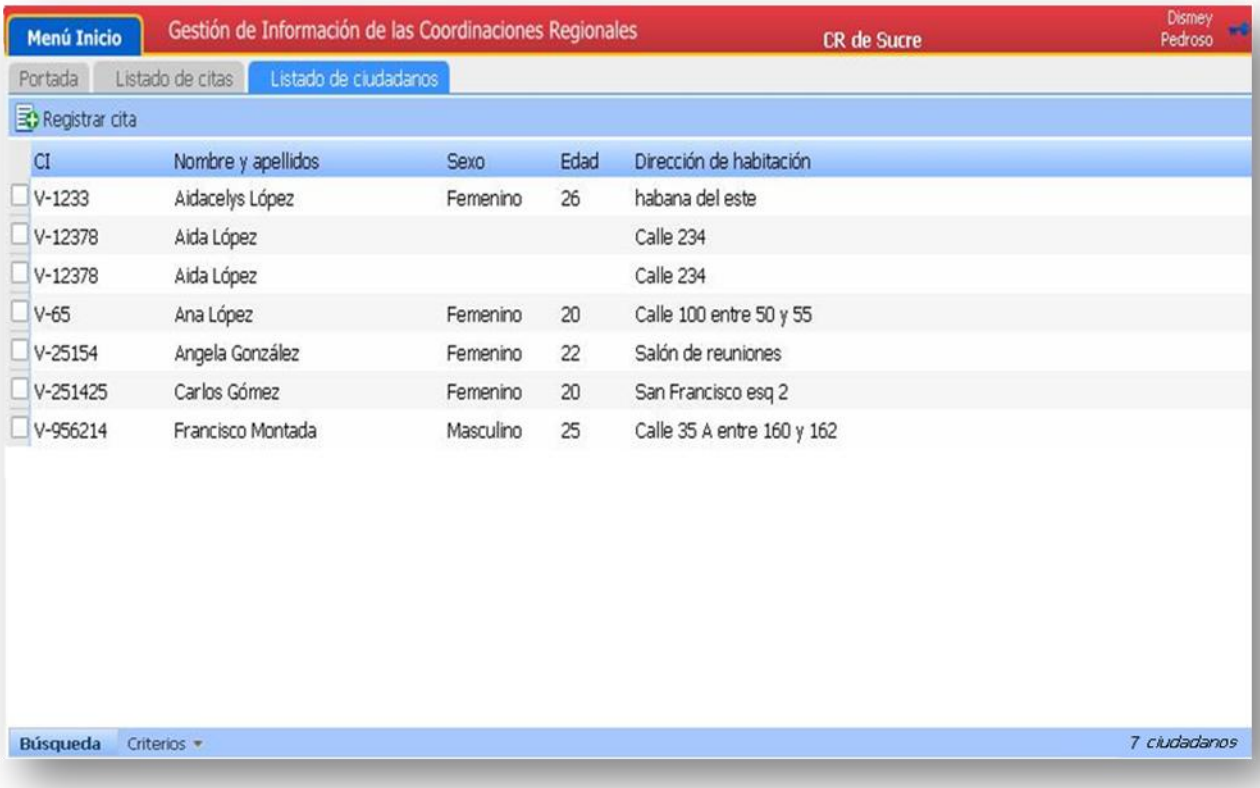


Fig. 16. Imagen de la aplicación correspondiente al CUS Gestionar Citas (Registrar Cita).

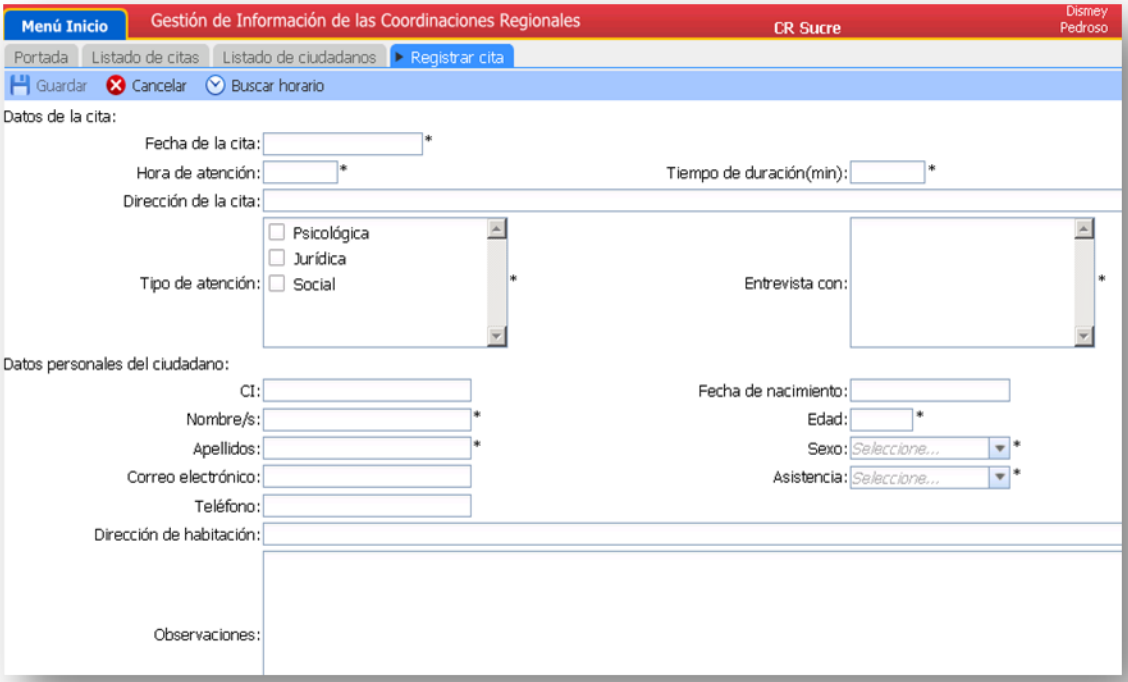


Fig. 17. Imagen de la aplicación correspondiente al Formulario de Registrar Cita.

Eliminar Cita

Para eliminar una cita, en primer lugar se accede al Listado de citas con el objetivo de buscar la función que nos va a permitir realizar dicha operación (botón Eliminar), luego se selecciona la cita que se desea eliminar y se ejecuta la operación, en este caso el sistema muestra un mensaje de confirmación (Fig.18) con el objetivo de simplificar posibles errores futuros.

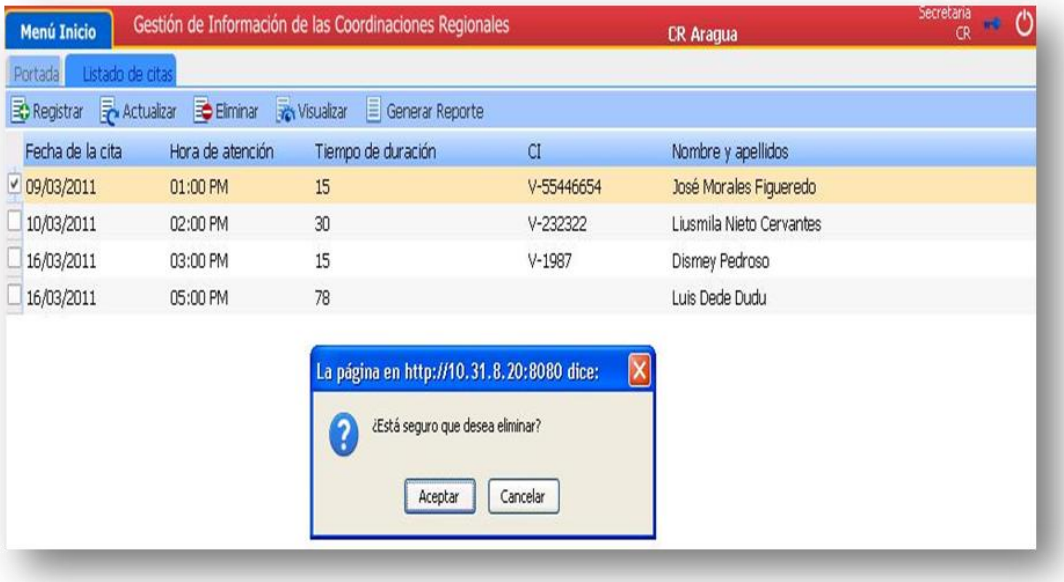


Fig. 18. Imagen de la aplicación correspondiente al CUS Gestionar Citas.

5. Conclusiones

Se realizaron los diagramas de componentes, los cuales describen cómo los elementos del diseño se implementan en términos de ficheros de código fuentes. Posteriormente se implementaron los componentes siguiendo estándares de codificación, utilizando Dojo para las validaciones del lado del cliente, llamadas asíncronas que mejoran los tiempos de respuesta y usabilidad y la generación de listados, filtros y reportes personalizados.

4) CAPÍTULO 4. PRUEBAS DEL SISTEMA

1. Introducción

La realización de pruebas a un software son de vital importancia ya que proporciona un alto grado de confianza y seguridad no solo en el producto sino en los resultados que se obtienen en implantarlo. Por todas estas razones se le realizaron al Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. Módulos: Atención de Casos, Bienes Muebles y Materiales diferentes tipos de pruebas asociadas a medir una serie de atributos de calidad en este caso: Funcionalidad, Fiabilidad y Rendimiento con el objetivo de obtener resultados que sirvieran para la detección de errores futuros.

2. Pruebas de Software

Para contribuir con la calidad del software es recomendable que el producto vaya siendo evaluado a medida que se va construyendo. Posteriormente, se hace necesario llevar a cabo, paralelo al proceso de desarrollo, un proceso de evaluación y comprobación de los distintos productos o modelos que se van generando, en el que participarán desarrolladores y clientes. 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. [28]

Las pruebas de software se hacen con el objetivo de encontrar y documentar los errores que puedan afectar la calidad del software, verificar que el mismo trabaje como fue diseñado, así como validar y probar que los requisitos que debe cumplir se hayan implementado correctamente.

Al evaluar un software se debe comenzar por los componentes más simples y más pequeños e ir avanzando progresivamente hasta probar todo el software en su conjunto. Para cumplir con este objetivo se organizan las pruebas por niveles, permitiendo así especificar el alcance que tendrán las mismas.

3. Niveles de Pruebas

A los módulos Atención de Casos, Bienes Muebles y Materiales del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del delito de la República Bolivariana de Venezuela se le realizaron pruebas en los niveles de Unidad y Sistema.

3.1 Prueba de Unidad

Este nivel se enfoca 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 mismo. 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. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera. [28]

JUnit para la realización de las pruebas unitarias

Para realizar las pruebas en el nivel de Unidad se utilizó la herramienta JUnit, integrada al entorno de desarrollo Eclipse. Las pruebas unitarias realizadas contribuyeron a la detección de errores en la implementación de las clases, permitiendo al desarrollador corregir los mismos, garantizando el cumplimiento de los objetivos de estas.

A partir de la utilización de la herramienta JUnit, se programaron un conjunto de pruebas para verificar el funcionamiento de las principales clases de la capa de acceso a datos y lógica de negocio. Un ejemplo de las pruebas realizadas fue **TestAtencionCasosFacade.java**, la cual se enfocó en evaluar algunas de las principales funcionalidades (registrarCita (), registrarCaso(), ExisteCedula() y TieneCita()) de la clase AtencionCasosFacade del módulo Atención de Casos.

Todas las clases creadas para las pruebas comenzaron con la palabra **Test,** estas se ubicaron en la carpeta **test** que posee cada una de las carpetas que estructuran el sistema, en este caso ubicadas en la carpeta **facade**:

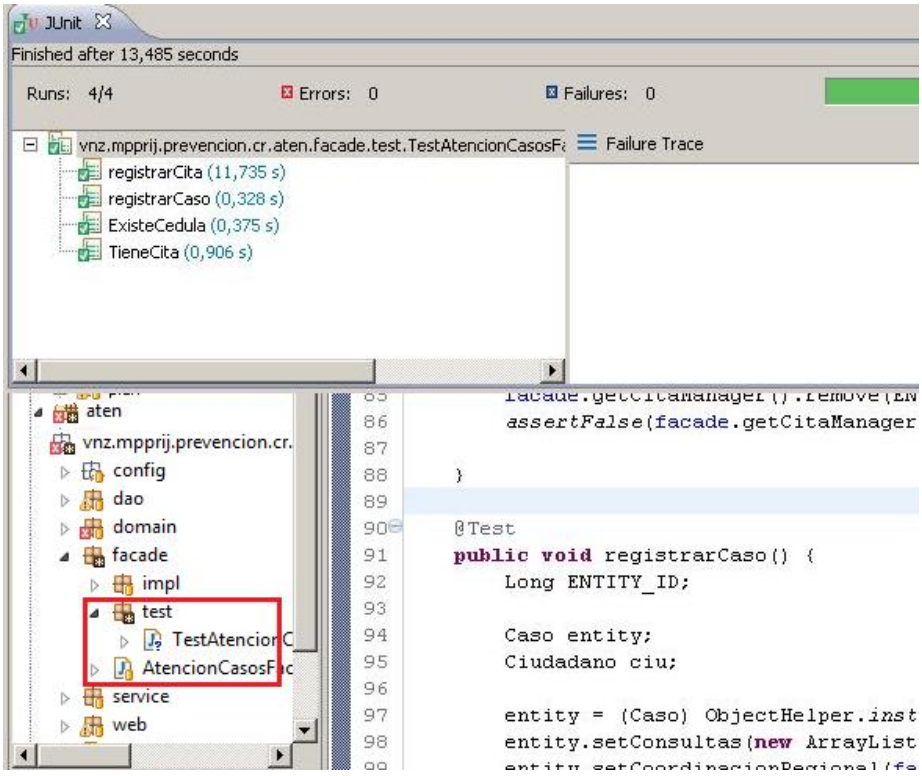


Fig. 19. Ubicación de la clase prueba TestAtencionCasosFacade.

En versiones anteriores de JUnit no existían las anotaciones pero estas se han incluido en su versión 4 para intentar simplificar más la labor del programador. Se utilizó la anotación @Test para indicar los métodos que iban a ser probados, también fueron utilizaron los métodos assert (assertTrue () y assertFalse ()) de JUnit para verificar que las expresiones se evalúen de forma correcta o falsa.

3.2 Prueba de Sistema

Son las pruebas que se hacen cuando el software está funcionando como un todo. Es la actividad de prueba dirigida a verificar el programa final, después que todos los componentes de software y hardware han sido integrados. [28]

Con el objetivo de demostrar que el software cumple con la calidad requerida, se realizaron diferentes tipos de pruebas asociadas a una serie de atributos de calidad, los cuales no son más que características que sirven para medir un software. En la siguiente tabla se muestra la relación entre los atributos de calidad y los tipos de pruebas que se realizaron para medirlos.

Tipos de Pruebas		
Atributos de Calidad	**Característica de los Atributos**	**Tipos de Pruebas**
Funcionalidad	Grado en que las necesidades asumidas o descritas se satisfacen.	- Pruebas de Funcionalidad - Pruebas de Control de Acceso al Sistema
Fiabilidad	Probabilidad de operación libre de fallos de un programa de computadora en un entorno determinado y durante un tiempo específico.	- Pruebas de Estrés
Rendimiento	Pruebas que se realizan, desde una perspectiva, para determinar lo rápido que realiza una tarea un sistema en condiciones particulares de trabajo.	- Prueba de Carga

3.2.1 Pruebas de Funcionalidad

Las pruebas funcionales se realizan con el objetivo de asegurar el trabajo apropiado de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados. Para la realización de las pruebas funcionales al Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. Módulos: Atención de Casos, Bienes Muebles y Materiales, se utilizaron los casos de prueba realizados por el equipo de calidad del proyecto de desarrollo en el cual se enmarca este trabajo de diploma.

Las pruebas de funcionalidad realizadas estuvieron divididas en pruebas internas y pruebas de liberación. Las pruebas internas fueron realizadas al sistema por el equipo de desarrollo, con el fin de entregar un producto lo más libre de errores posible a Calisoft, mientras que las pruebas de liberación fueron las realizadas por Calisoft, institución encargada de validar que el software cuente con la calidad requerida para ser entregado a los clientes finales

Durante las pruebas internas se realizaron varias iteraciones, siempre verificando que las No Conformidades (NC[13]) encontradas en la iteración anterior estuvieran resueltas, esto posibilitó de

[13] **NC:** Defecto o error detectado durante las pruebas.

forma significativa un aumento de la calidad del software dando así un margen de error menor durante las pruebas de liberación.

Las NC detectadas durante el proceso de prueba se clasificaron atendiendo a los tipos de errores definidos por Calisoft (Ver Anexo 11) y a partir del impacto que tienen estas para su solución por el equipo de desarrollo se agruparon en altas (funcionalidad, validación, error técnico, excepción.), medias (error de interfaz, seguridad, correspondencia con otro artefacto.) y bajas (redacción, ortografía, recomendaciones, formato). A partir de la agrupación de las NC detectadas se muestran los resultados del proceso de prueba en los siguientes gráficos:

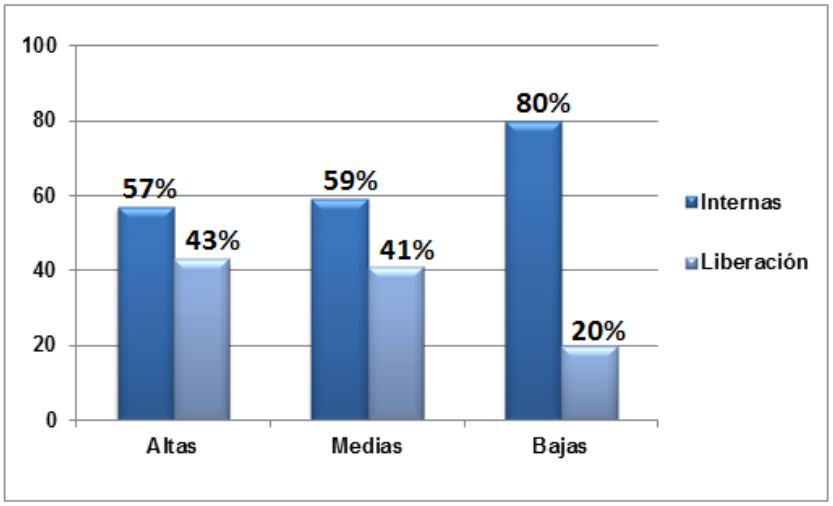


Fig. 20. Resultados de las pruebas internas y liberación del módulo Atención de Caso.

Como resultado del análisis de las NC detectadas durante el proceso de prueba, en el módulo Atención de Casos, se evidenció que las pruebas internas garantizaron una reducción de un 57% en los errores de clasificación alta, un 59% en los medios y un 80% en los bajos.

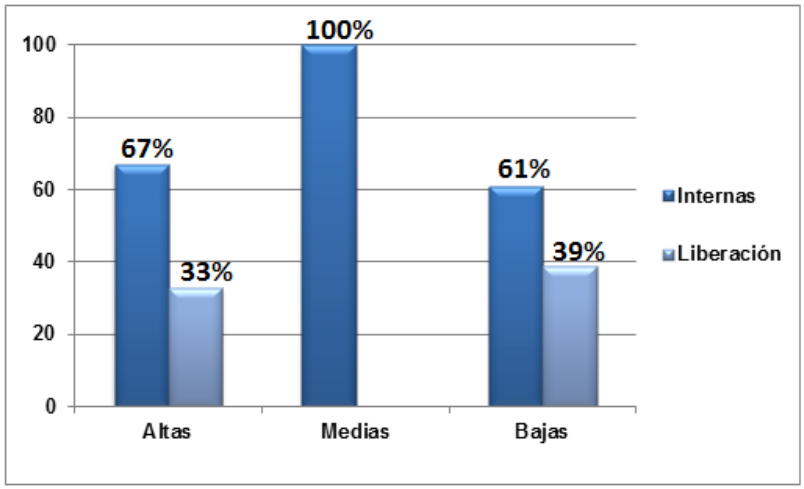


Fig. 21. Resultados de las pruebas internas y liberación del módulo Bienes muebles.

Como resultado del análisis de las NC detectadas durante el proceso de prueba, en el módulo Bienes Muebles, se evidenció que las pruebas internas garantizaron una reducción de un 67% en los errores de clasificación alta, un 100% en los medios y un 61% en los bajos.

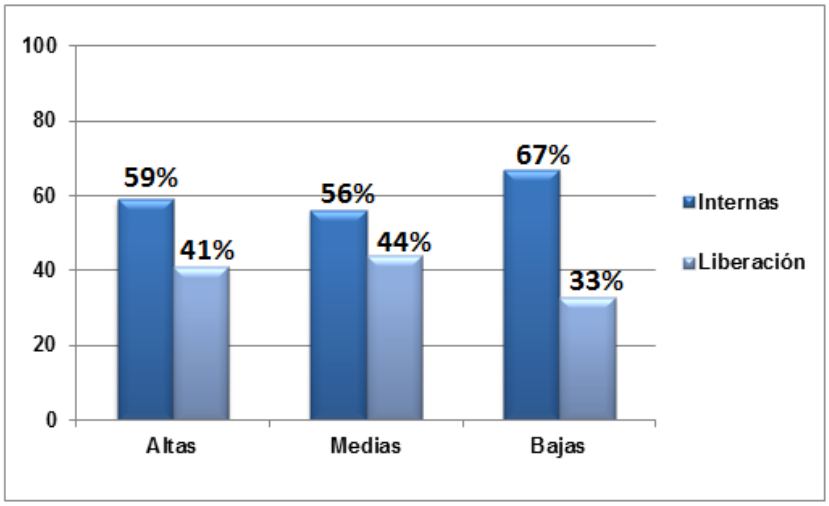


Fig. 22. Resultados de las pruebas internas y liberación del módulo Materiales.

Como resultado del análisis de las NC detectadas durante el proceso de prueba, en el módulo Materiales, se evidenció que las pruebas internas garantizaron una reducción de un 59% en los errores de clasificación alta, un 56% en los medios y un 67% en los bajos.

3.2.2 Prueba de Control de Acceso al Sistema

El acceso a cualquier manipulación del Sistema de Gestión de Información solo estará disponible para los usuarios autorizados. El sistema contará con un mecanismo de protección que incluye seguridad a nivel de usuarios y roles, requiriéndose como primera acción en la aplicación, la autenticación. Cada usuario deberá proporcionar sus credenciales (usuario y contraseña) y en dependencia del rol o roles que le fueron asignados, tendrá acceso a la información.

Durante el negocio se establecieron los roles (Coordinador/a Regional, Secretaria/o de la Coordinación Regional, Administrador/a de la Coordinación Regional, ETP, Director/a de línea de la DPCN y Supervisor/a) y las funcionalidades que pueden realizar los mismos. Con las Pruebas de Control de Acceso al Sistema se verifica que solo los usuarios autorizados puedan acceder solo a las funcionalidades e informaciones que su rol le tiene permitido

Para demostrar como fue el proceso de prueba de Control de Acceso al Sistema para los módulos: Atención de Caso, Bienes Muebles y Materiales, se tomó como ejemplo los CUS Gestionar cita y Gestionar listado de citas pertenecientes al módulo Atención de Casos, donde se seleccionaron los roles de Coordinador/a Regional y Secretaria/o de la Coordinación Regional.

Rol/es	CUS	Funcionalidades
Secretaria/o de la CR	Gestionar citas	Registrar cita
Secretaria/o de la CR		Actualizar cita
Secretaria/o de la CR		Eliminar cita
Secretaria/o de la CR		Buscar horarios
Secretaria/o y Coordinador/a de la CR	Gestionar listado de citas	Visualizar listado de citas
Secretaria/o y Coordinador/a de la CR		Buscar cita
Secretaria/o y Coordinador/a de la CR		Generar reporte de citas
Secretaria/o y Coordinador/a de la CR		Exportar a formato Excel el reporte de citas
Secretaria/o y Coordinador/a de la CR		Visualizar cita
Secretaria/o y Coordinador/a de la CR		Imprimir reporte de citas

Una de las responsabilidades que ejecuta la Secretaria/o de la CR es atender al personal que acude a las CRs en busca de apoyo, ya sea Psicológico, Jurídico, Criminológico o Social. Además, se encarga también de mantener informado al Coordinador/a de la CR y de cumplir con funciones que el mismo le delegue. Por estas razones tiene acceso a todas las funcionalidades que abarca la gestión de citas que atenderán luego los Especialistas y a todas las funcionalidades que permitan mantener un control de las mismas.

En el caso del Coordinador/a de la CR al ser este el máximo responsable de las CRs es el encargado de organizar y controlar que el trabajo se esté realizando correctamente, por tanto, tiene acceso a las funcionalidades que le permitan visualizar las tareas que realizan otros roles, además de poder tener acceso a la información que necesita para enviar los informes a la DGPD.

A continuación se muestran dos pantallas del sistema donde se puede comprobar que entrando con los roles antes mencionado y accediendo al listado de citas solo se visualizan las operaciones con las que pueden trabajar los mismos.

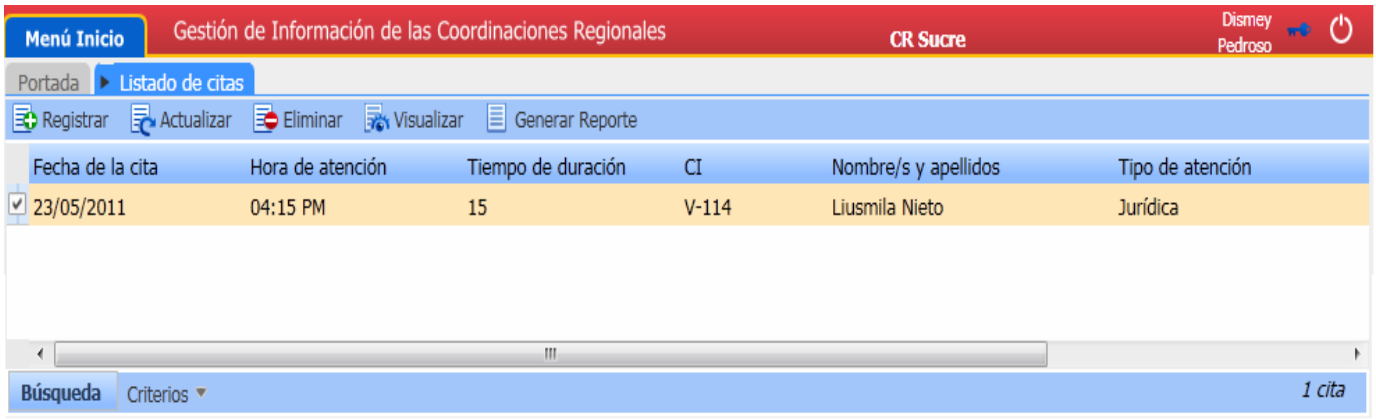


Fig. 23. Acceso al Sistema con el rol de "Secretaria/o de la CR".

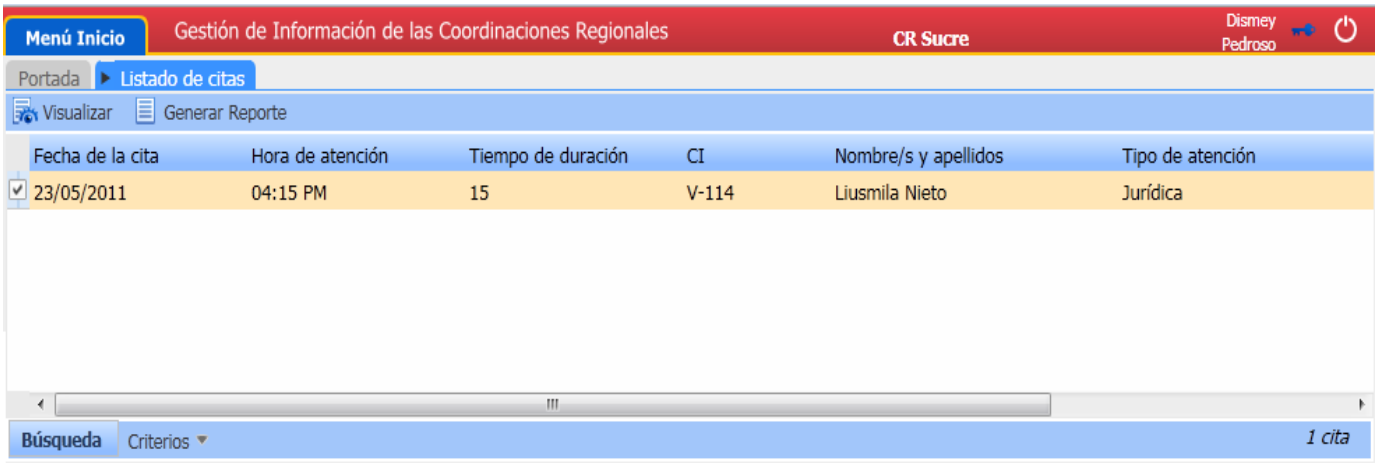


Fig. 24. Acceso al Sistema con el rol de "Coordinador/a de la CR".

3.2.3 Pruebas de carga y estrés

El objetivo central de las pruebas de carga y estrés es esencialmente obtener un índice de resultados de comportamiento del sistema en determinadas condiciones. Al realizar estas pruebas a la aplicación, se obtienen resultados que pueden ser muy cercanos a un comportamiento real. A partir de estos resultados se puede prever respuestas del sistema ante cantidades específicas de usuarios y altos niveles de concurrencia.

Al realizar las pruebas de carga y estrés es necesario tener declarados explícitamente por parte del equipo de desarrollo las especificaciones de rendimiento (Ver Anexo 12), dígase aquellos requerimientos de eficiencia propios del sistema que describen las características que se supone deba tener la aplicación en un ambiente real de ejecución.

JMeter para la realización de las pruebas de cargas y estrés

Para la realización de estas pruebas se utilizó la herramienta JMeter, con la cual se grabó una prueba que abarcó más de 70 peticiones, estas incluían desde la autenticación en el sistema, el acceso al menú, así como a las principales funcionalidades del sistema, como Registrar, Actualizar, Generar Reporte, entre otras.

En un primer momento se realizaron 3 iteraciones con el servidor de aplicación corriendo en el sistema operativo Linux, probando todos los escenarios grabados. Para una primera iteración se definieron 30 usuarios ya que este es el máximo promedio de personas que pueden trabajar en una

CR, donde se especifica 1 segundo como tiempo de conexión entre cada uno de estos. En una segunda y tercera iteración se simularon 100 y 150 usuarios respectivamente, en similares condiciones a la anterior. Por cada iteración se registraron los resultados según la cantidad de usuarios a que se sometió el sistema. Lo anteriormente explicado se realizó de igual manera en el sistema operativo Windows, con el fin de comparar el comportamiento del software en ambos sistemas operativos.

A continuación se muestran los datos más significativos resultantes de las pruebas realizadas, para el análisis del rendimiento del sistema bajo una cantidad de peticiones esperadas y diferentes cantidades de usuarios realizándolas.

Plataforma	Cantidad Usuarios	% Error	Rendimiento	Kb/sec
Windows	30	0.00 %	1052.9 /sec	10343696.4
	100	0.00 %	784.8 /sec	7710416.9
	150	0.03%	433.1 /sec	4255122.1
Linux	30	0.00 %	987.2 /sec	9691730.8
	100	0.00 %	1036.3 /sec	10174360.7
	150	0.00%	471.3 /sec	4627162.7

Descripción de los aspectos mostrados en la tabla de los Resultados:

- **%Error:** Por ciento de error de las páginas que no se llegaron a cargar de manera satisfactoria.
- **Kb/Seg:** Velocidad de carga de las páginas.
- **Rendimiento:** Representa el número de muestras por unidad de tiempo.

A mayor cantidad de usuarios conectados por Windows se produjo un error de 0.03% y el rendimiento disminuyó considerablemente, así como también la velocidad de respuesta de las peticiones

realizadas. Por otra parte, las pruebas realizadas por Linux no arrojaron ningún error, el rendimiento y la velocidad de respuesta se comportaron mejor ante situaciones de uso intenso.

El 0.03% de las páginas que no se llegaron a cargar de manera satisfactoria en el sistema operativo Windows no se considera significativo, debido a que las características de hardware de la computadora donde se realizaron las pruebas son inferiores a las del entorno mínimo especificado (Ver Anexo 13), además de ser Linux el sistema operativo en el cual se ejecutara la aplicación. Por lo anteriormente planteado se considera que los resultados obtenidos demuestran un comportamiento satisfactorio del sistema en determinadas condiciones de trabajo.

Características de la computadora donde se realizaron las pruebas

- Microprocesador: Intel Pentium 4.
- RAM: 1GB.

4. Conclusiones

Para la validación de las funcionalidades implementadas se realizaron pruebas a nivel de desarrollador, unidad y sistema. En el nivel de pruebas de unidad se utilizó el framework JUnit para probar las funcionalidades principales de clases como los Dao, los managers y la fachada. Para las pruebas a nivel de sistema se diseñaron Casos de Prueba basados en los casos de uso del sistema para evaluar el software atendiendo a los atributos de calidad funcionalidad, fiabilidad. Una vez integrado el sistema se utilizó la herramienta JMeter para evaluar el rendimiento del sistema y su comportamiento ante situaciones de estrés. Todos los errores detectados durante estas pruebas fueron corregidos y se verificó que el sistema responde ante las condiciones extremas evaluadas.

5) CONCLUSIONES

Aplicando el estilo arquitectónico Arquitectura en capas y los patrones de diseño Controlador Frontal, Controlador, Fachada, Inyección de Dependencias, Alta Cohesión, Bajo Acoplamiento y DAO se realizó el diseño de las clases de los módulos Atención de casos, Bienes Muebles y Materiales lo que proporcionó un mejor entendimiento para la implementación de sus funcionalidades.

Se implementaron las funcionalidades correspondientes a los módulos Atención de casos, Bienes Muebles y Materiales a partir de las especificaciones modeladas en los diagramas de componentes, haciendo uso de frameworks como Dojo, Spring e Hibernate, los cuales facilitaron las validaciones del lado del cliente, el control de acceso, el trabajo con formularios, la abstracción del gestor de base de datos, entre otras.

Se realizaron pruebas para validar las funcionalidades implementadas en los módulos Atención de casos, Bienes Muebles y Materiales, haciendo uso de las herramientas caso de prueba, JUnit y JMeter las cuales contribuyeron a la verificación del cumplimiento de los atributos de fiabilidad, funcionalidad y rendimiento. Estas pruebas se dividieron en pruebas internas (realizadas por el equipo de desarrollo) y pruebas de liberación (realizadas por Calisoft) logrando que los módulos pasaran a las pruebas de liberación con la menor cantidad de errores posibles y fueran liberados por esta institución.

6) RECOMENDACIONES

- Optimizar el trabajo con los componentes de selección múltiple de Dojo que sus datos dependen de la selección de otro componente de selección múltiple.
- Llevar al lado del cliente el tratamiento de errores con el fin de garantizar mejores tiempos de respuestas de la aplicación.

7) REFERENCIAS BIBLIOGRÁFICAS

1. Yunier Rodríguez Cruz y Ailín Martínez Rodríguez, Máster en Ciencias de la Información, "Comportamiento de la producción científica sobre gestión de información en revistas del *Web of Science* (1995-2008)". Disponible en: http://scielo.sld.cu/scielo.php?pid=S1024-94352009001200002&script=sci_arttext Consultado el: (1 de noviembre del 2010).
2. Maydelín Díaz Pérez, Yimian de Liz Contreras y Soleydis Rivero Amador, Máster en Bibliotecología y Ciencias de la Información, "Características de los sistemas de información que permiten la gestión oportuna de la información y el conocimiento constitucional". Disponible en: http://scielo.sld.cu/scielo.php?pid=S1024-94352009001100006&script=sci_arttext Consultado el: (1 de noviembre del 2010).
3. Programa Institucional de Prevención del Delito del Distrito Federal, México. Disponible en: http://portal.ssp.df.gob.mx/Portal/ProgramasyCampanas/Institucional.htm Consultado el: (11 de noviembre del 2010).
4. Comité de consulta y participación ciudadana, Chiapas, México. Disponible en: http://www.prevencion.chiapas.gob.mx/sitio/index.php?pag=2 Consultado el: (11 de noviembre del 2010).
5. Programa Integral de Gestión e Investigación para la Prevención del Delito, Gobierno Federal, México. Disponible en: http://www.presidencia.gob.mx/programas/seguridad/?contenido=35019 Consultado el: (12 de noviembre del 2010).
6. Eneysi Osorio, Asdrúbal Torres, "Propuesta de arquitectura para el Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela" Consultado el: (15 de noviembre del 2010).
7. Letelier Patricio, "Proceso de desarrollo de software". Disponible en: https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Forms/DispForm.aspx?ID=6&Source=https%3A%2F%2Fpid.dsic.upv.es%2FC1%2FMaterial%2Fdefault.aspx&RootFolder=%2FC1%2FMaterial%2FDocumentos%20Disponibles Consultado el: (18 de noviembre del 2010).
8. Letelier Patricio, Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia, "Introducción a RUP". Disponible en: https://pid.dsic.upv.es/C1/Material/default.aspx Consultado el: (20 de noviembre del 2010).

9. Glosario de términos técnicos de internet, (En línea) Disponible en: http://tecnologia.glosario.net/terminos-tecnicos-internet/uml-1655.html Consultado el: (20 de noviembre del 2010).
10. Sitio Web Oficial de Visual Paradigm [En línea]. Disponible en: http://www.visual-paradigm.com/product/vpuml/provides/ Consultado el: (18 de noviembre de 2010).
11. Nuevas Tecnologías de la Programación, Universidad de Huelva, Disponible en: http://www.uhu.es/josel_alvarez/NvasTecnProg/recursos/tTema1.pdf Consultado el: (25 de noviembre de 2010).
12. Juan Manuel Barrios Núñez, Universidad de Chile Facultad de Ciencias Físicas y Matemáticas, Departamento de Ciencias de La Computación, "Investigación de la plataforma J2EE y su Aplicación Práctica", Disponible en: http://www.dcc.uchile.cl/~jbarrios/J2EE/MemoriaJ2EE.pdf Consultado el: (25 de noviembre de 2010).
13. Sitio Web Oficial de PostgreSQL [En línea]. Disponible en: http://www.postgresql.org/docs/8.3/static/intro-whatis.html Consultado el: (30 de noviembre de 2010).
14. Sitio Web Oficial de The Apache Jakarta Project [En línea]. Disponible en: http://jakarta.apache.org/jmeter/ Consultado el: (16 de marzo del 2011).
15. EcuRed: Enciclopedia cubana. Disponible en: http://www.ecured.cu/index.php/Framework Consultado el: (26 de noviembre del 2010).
16. Sánchez Rico, Mario Alfredo, Spring, un framework de aplicación [En línea]. Disponible en:http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/sanchez_r_ma/capitulo3.pdf Consultado el: (26 de noviembre del 2010).
17. Spring-framework-reference. Disponible en: La documentación del proyecto \\10.31.8.12\preVen\Documentación_Implementación\spring framework 3.0.2\docs Consultado el: (26 de noviembre del 2010).
18. Sitio Web Oficial de Dojo [En línea].Disponible en: http://dojofoundation.org/project/dojo/ Consultado el: (29 de noviembre de 2010).

19. Sitio Web Oficial de Dojo [En línea].Disponible en: http://www.dojotoolkit.org/reference-guide/dojox/index.html Consultado el: (29 de noviembre de 2010).
20. Sitio Web Oficial de Hibernate [En línea].Disponible en: http://www.hibernate.org/about Consultado el: (29 de noviembre de 2010).
21. Sitio Web Oficial de JasperReport [En línea]. Disponible en: http://jasperforge.org/projects/jasperreports Consultado el: (29 de noviembre de 2010).
22. Carlos Billy Reynoso, "Introducción a la Arquitectura de Software", Universidad de Buenos Aires. Disponible en: http://www.willydev.net/descargas/prev/IntroArq.pdf Consultado el: (29 de enero de 2011).
23. Carlos Canal Velasco, "Un Lenguaje para la Especificación y Validación de Arquitecturas de Software", Áreas de Lenguajes y Sistemas Informáticos. Disponible en: http://www.lcc.uma.es/~canal/papers/tesis/canal_tesis.pdf Consultado el: (2 de febrero de 2011).
24. Yoan Arlet Carrascoso Puebla, Enrique Chaviano Gómez, "Propuesta de Arquitectura orientada a servicios para el módulo del ERP cubano". [En línea]. Disponible en: http://www.gestiopolis.com/administracion-estrategia/erp-arquitectura-orientada-a-servicios.htm Consultado el: (2 de febrero de 2011).
25. Arquitectura de Software Patrones de diseño y framework, Case I. Disponible en: El expediente de proyecto \\10.31.8.12\preVen\Documentación_Implementación_Viejo\Arquitectura, patrones y framework (SPRING)\Clase I Consultado el: (2 de febrero de 2011).
26. Yupanqui García, Glen Jasper, "Diseño con Patrones". [En línea]. Disponible en: http://www.scribd.com/doc/17767790/Patrones-de-Diseno-Recopilacion Consultado el: (2 de febrero de 2011).
27. Ivar Jacobson, Grady Booch, James Rumbaugh, "El Proceso Unificado de Desarrollo de Software". S.l.: PEARSON EDUCACIÓN S.A, 2000. Consultado el: (20 de Febrero del 2011).
28. Materiales Básicos de la Conferencia # 7 Disciplina de Prueba "Sobre la disciplina de Prueba", Ingeniería del Software II. Disponible en: http://eva.uci.cu/mod/resource/view.php?id=14103 Consultado el: (11 de marzo del 2011).

8) BIBLIOGRAFÍA

1. Alejandro Martínez y Raúl Martínez, "Guía de Rational Unified Process". Disponible en: http://www.dsi.uclm.es/asignaturas/42551/trabajosAnteriores/Trabajo_Guia%20RUP.pdf [Citado el: 1 de Noviembre de 2010]
2. Letelier Patricio, "Proceso de desarrollo de software". Disponible en: https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Forms/DispForm.aspx?ID=6&Source=https%3A%2F%2Fpid.dsic.upv.es%2FC1%2FMaterial%2Fdefault.aspx&RootFolder=%2FC1%2FMaterial%2FDocumentos%20Disponibles Consultado: (18 de Noviembre de 2010).
3. Juan Manuel Barrios Núñez, Universidad de Chile Facultad de Ciencias Físicas y Matemáticas, Departamento de Ciencias de La Computación, "Investigación de la plataforma J2EE y su Aplicación Práctica", Disponible en: http://www.dcc.uchile.cl/~jbarrios/J2EE/MemoriaJ2EE.pdf Consultado el: (25 de Noviembre de 2010).
4. Framework Dojo [En línea] Disponible en: http://www.dojotoolkit.org/ [Citado el: 29 de Noviembre de 2010].
5. Group, P.G.D. PostgreSQL. 2007 [En línea] Disponible en: http://www.postgresql.org/. Consultado el: (30 de Noviembre de 2010)
6. hibernate.org [En línea]. Disponible en: http://www.hibernate.org/ Consultado el: (4 de Diciembre de 2010).
7. jasperforge.org Sitio Web Oficial de JasperReport [En línea] Disponible en: http://jasperforge.org/website/. Consultado el: (6 de Diciembre del 2010).
8. Ivar Jacobson, Grady Booch, James Rumbaugh, "El Proceso Unificado de Desarrollo de Software". S.l.: PEARSON EDUCACIÓN S.A, 2000 Consultado el: (20 de Febrero del 2011).
9. Joseph Schmuller, "Aprendiendo UML en 24 horas" Consultado el: (21 de Febrero del 2011).
10. Programación en Castellanos, Diseño web y desarrollo web [En línea]. Disponible en: http://www.programacion.com/articulo/primeros_pasos_con_junit_265 Consultado el: (5 de mayo del 2011).
11. Linet Lores Sánchez, Diana Monné Roque, "Aplicación de las Pruebas de Liberación al Sistema Informático de Genética Médica". Consultado el: (12 de marzo del 2011).

12. Materiales Complementarios de la Conferencia # 7 Disciplina de Prueba "Material de caja blanca y caja negra", Ingeniería del Software II. Disponible en: http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_9/Conferencia_7/Materiales_Complementarios/Material_de_caja_b_y_caja_n.pdf Consultado el: (12 de marzo del 2011).
13. EcuRed: Enciclopedia cubana. Disponible en: http://www.ecured.cu/index.php/Pruebas_de_Calidad_de_Software Consultado el: (14 de marzo del 2011).
14. Sitio Web Oficial de JUnit [En línea]. Disponible en: http://junit.sourceforge.net/doc/faq/faq.htm#overview_1 Consultado el: (14 de marzo del 2011).