UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 2

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS.

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

Autores: Humberto Yera Lorenzo.
Orlando Herrera González.

Tutor: Ing. René Vega Gorgoso.
Cotutor(es): Ing. Sucel Ochoa Ochoa.
Ing. Linet Lores Sánchez.

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 _____.

Humberto Yera Lorenzo

_____

Firma del Autor

Orlando Herrera González

_____

Firma del Autor

Ing. René Vega Gorgoso

_____

Firma del Tutor

Ing. Linet Lores Sánchez

_____

Firma del Cotutor

Ing. Sucel Ochoa Ochoa

_____

Firma del Cotutor

Datos de Contacto

Tutor:

Ing. René Vega Gorgoso

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

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

e-mail: rvega@uci.cu

Cotutores:

Ing. Sucel Ochoa Ochoa

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

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

e-mail: sochoa@uci.cu

Ing. Linet Lores Sánchez.

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

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

e-mail: lloress@uci.cu

AGRADECIMIENTOS

Es imposible plasmar todos los nombres de las personas que están ligadas indisolublemente a nuestras vidas, agradecemos a todos los que nos han apoyado, respaldado, animado y han reído con nosotros en estos cinco años. Especialmente agradecemos a nuestra familia, por contribuir en nuestra formación y por toda la confianza que depositaron en nosotros, muy en especial agradecemos a nuestros padres, guía e inspiración en todo momento.

Hacemos llegar este agradecimiento también a la Revolución y al Comandante en Jefe Fidel Castro Ruz, creador de este magnífico proyecto que es la Universidad de las Ciencias Informáticas, la cual se convirtió durante este tiempo en nuestro hogar, y nos acogió como sus hijos.

De forma especial queremos agradecer a nuestros tutores René, Sucel y Linet, por dedicar parte de su tiempo a asesorarnos, guiarnos y aconsejarnos.

A todos los profesores del proyecto Prevención del Delito, como son Liusmila, Aidacelys, Ángela, Ballester, Liván, Dairon, Francisco, Manuel, Dany, a todos gracias por su abnegada y constante preocupación por este trabajo.

A aquellos profesores que en estos años de estudio pusieron tanto empeño en educarnos, formarnos como profesionales, ya que somos el resultado del trabajo esmerado de cada uno de ellos.

A todos nuestros compañeros y amigos, que de una forma u otra contribuyeron con el desarrollo del presente trabajo.

A todos, muchas gracias…

DEDICATORIA

Hay una persona que ocupa el primer lugar en mi vida y esa es mi mamá. Eres tu mami a quien quisiera comenzar dedicando todo mi esfuerzo y sacrificio, porque eres tú mi inspiración, porque sé que cuando te levantas en las mañanas lo primero que piensas es en mí, te pasas el día pensando en mí, en las noches esperas mi llamada, si no te llamo no duermes y si lo hago igual te duermes pero pensando en mí, es por eso que hoy y siempre serás la primera persona en mi vida.

Papá, te dedico este trabajo que es parte de mi sueño que siempre ha sido ser como tú, ser un militar sencillo como Camilo Cienfuegos, llegar a ser muy grande de gloria como de cuerpo, saber ser primero, ser un buen ejemplo, pelear por los pobres contra todos los imperios, ser internacionalista y como el Ché ¡un gran guerrero!. Saber de todos los hombres célebres del mundo entero, llegar a estudiar todo aquello que tú no has podido hacerlo, saber de las conquistas de todos los grandes pueblos, amar mucho a Camilo, Fidel, Martí y Maceo y a todos los combatientes que la libertad nos dieron.

Abuela, dime si aún me ves como un niño, porque recuerdo que siempre me decías que nunca ibas a llegar a verme hecho un hombre. Tienes 93 años, yo solo tengo 23 y quisiera llegar a la edad que tienes con tus fuerzas, tus energías, tu voluntad de acero y brincando balcones como tú lo haces para recoger mangos o limones.

Mis hermanitas, soy sacrificado porque ustedes me han enseñado a serlo, verlas salir adelante enfrentando malos y buenos momentos, batallando día a día, minuto a minuto, segundo a segundo, a solas. Ustedes son mis heroínas, son mi fuente de esperanzas, porque nunca se rinden, ustedes lo malo lo convierten en bueno, de lo bueno sacan lo mejor, la tristeza la convierten en alegría, ríen cuando se debe llorar y lloran cuando nadie las ve.

Mis sobrinitas, son ustedes las niñas más lindas del mundo, y cuando yo estoy triste pues pienso en ustedes, porque me dan alegría. Hay veces que cuando camino me canso mucho, me dan deseos de descansar y de no caminar más, pero de repente ustedes me vienen a la mente y me entran muchas fuerzas para seguir caminando e incluso correr. A ustedes les queda un camino largo y difícil por seguir, muchas cosas por conocer, pero piensen que yo también fui un niño y atravesé por ese camino largo, pero llegué al final, porque ese era mi sueño y los sueños se hacen realidad, solo hay que creer en ellos.

Le doy gracias a mi compañero de tesis, inigualable amigo, por el grandioso trabajo en equipo que realizamos, donde no hubo cuestionamientos y siempre una plena confianza en el trabajo de cada cual.

Agradecer a mis tutores por su guía constante, por dedicar parte de su tiempo a asesorarnos, orientarnos y aconsejarnos.

A los profesores del proyecto Prevención del Delito por su abnegada, desinteresada y constante preocupación por este trabajo.

A mis amigos pues gracias por estar siempre presentes en las buenas y en las malas, gracias por el apoyo que me brindaron, por el granito de arena que aportó cada uno, de una manera u otra, para ayudarme a sembrar la semilla que se convirtió en un árbol resistente que hoy da sus frutos. A muchos no los volveré a ver jamás, pero siempre los llevaré en el corazón a dondequiera que vaya y me acompañarán los recuerdos de los momentos vividos junto a cada uno de ustedes. Gracias a todos especialmente a Buti, Sandy e Israel, que a kilómetros de distancia siempre me hacían llegar su apoyo incondicional.

Orlando

Dedico este trabajo a

RESUMEN

En la actualidad se manifiesta un incremento considerable en el desarrollo de aplicaciones web orientadas a la gestión de información, éstas proporcionan un mejor funcionamiento, ayudan a la toma de decisiones y al control de las organizaciones.

El Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela**.** Módulos: Actividades y Administración de Programas, contribuye a la gestión de la información referente a los programas, proyectos y actividades, garantizando la estandarización de los datos y el intercambio adecuado del flujo de información entre la Coordinación Regional y la Dirección General de Prevención del Delito. Posibilita a los especialistas, a partir de la gestión realizada por dichos módulos, identificar fortalezas y debilidades de las actividades realizadas y proyectar su labor preventiva en las comunidades.

Para el desarrollo de los módulos se utilizó el lenguaje de programación Java, empleando el estilo arquitectónico en capas y haciendo uso de diferentes frameworks como Hibernate, Dojo y Spring fundamentalmente.

Palabras clave: Actividades, Administración de Programas, Coordinación Regional, Dojo, Hibernate, Spring.

ÍNDICE

INTRODUCCIÓN

Desde el momento en que el actual Presidente de la República Bolivariana de Venezuela Hugo Rafael Chávez Frías asumió el gobierno, la Revolución comenzó a andar. Cuando el Presidente juró sobre una moribunda Constitución que existía en aquel entonces y convocó a la Asamblea Constituyente no dejó margen de dudas de que se avecinaba un proceso de profundas transformaciones. Una etapa bien clara del proceso bolivariano fue marcada desde la puesta en vigencia de la nueva Constitución de la República Bolivariana de Venezuela hasta la implementación de leyes que priorizaban una serie de reformas en la esfera social en la que se encuentran sectores como la salud, la educación, el empleo y la seguridad ciudadana.

Partiendo de estas premisas el Ministerio del Poder Popular para las Relaciones Interiores y Justicia (MPPRIJ), atendiendo a su misión de garantizar la seguridad y protección integral de los ciudadanos contra hechos delictivos, mediante la formulación de políticas dirigidas al resguardo de la paz pública, desarrollo territorial equilibrado y los derechos políticos y civiles de los venezolanos; promueve la formulación del proyecto “Solución Integral para el Perfeccionamiento del Sistema de Prevención del Delito de la República Bolivariana de Venezuela”, partiendo de que la seguridad ciudadana es una condición necesaria del desarrollo humano que propicia el mejoramiento de la calidad de vida de todos los ciudadanos que habitan en el territorio venezolano.

La Dirección General de Prevención del Delito (DGPD) está adscripta al Viceministerio de Seguridad Ciudadana perteneciente al MPPRIJ (Anexo 1, Figura 1). Tiene como misión el establecimiento, la promoción y la coordinación de políticas, programas y proyectos que atiendan a la prevención de la violencia criminal y no criminal del país con la integración y participación de la institucionalidad y la sociedad civil. Esta institución posee 22 Coordinaciones Regionales en los estados: Anzoátegui, Aragua, Barinas, Bolívar, Carabobo, Cojedes, Falcón, Guárico, Lara, Mérida, Miranda, Monagas, Portuguesa, Táchira, Trujillo, Yaracuy, Zulia, Delta Amacuro, Nueva Esparta, Sucre, Vargas y en el Distrito Capital (Anexo 2, Figura 2), conformadas por un jefe y un equipo de profesionales y técnicos que se encargan de la atención e implementación de los programas de prevención del delito.

La Dirección General de Prevención del Delito y sus Coordinaciones Regionales (CR) proponen una serie de programas y proyectos acorde a las necesidades de cada región, grupo o problema a enfrentar; que las CR deben ejecutar en sus áreas de acción; a la vez que las CR pueden realizar sus propios proyectos, los cuales deben ser aprobados previamente por la DGPD. Es necesario destacar que los programas generan actividades diversas las cuales son orientadas a cada CR por la supervisora de la DGPD correspondiente a ese estado, cada región adapta las actividades según su realidad, por lo que las CR deben rendir cuenta de su ejecución generando un informe mensual que

contiene todo lo realizado en el período, así como otra información demostrando sus logros y desempeños.

En este contexto las CR no cuentan con una manera efectiva de gestionar la información referente a los programas, proyectos y a las actividades realizadas. Esta información es hoy manejada y archivada de forma manual, cada CR crea sus propios formatos para la realización de sus informes lo que no garantiza la estandarización de la información y dificulta el análisis de la misma que cada supervisora debe hacer de los estados que atiende. Es importante destacar además que cada supervisora atiende varios estados(un estado tiene una CR); es decir, posee la misma información en diferentes formatos además de los datos propios de cada coordinación, dificultando el intercambio y adecuado flujo de información entre la CR y la DGPD, lo cual afecta 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, el fortalecimiento de la convivencia ciudadana y la paz, 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 CR y Albet[1] contratan la realización de una herramienta informática. Posteriormente se realizó un levantamiento de requisitos, los cuales fueron aceptados, documentados y firmados por estos directivos.

Por las dificultades anteriormente expuestas se plantea como **problema a resolver:** ¿Cómo garantizar el cumplimiento de los requisitos funcionales y no funcionales asociados a los módulos Actividades y Administración de Programas del Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela?

El problema planteado se enmarca en el siguiente **objeto de estudio:** Gestión de Información en las CR de Prevención del Delito de la República Bolivariana de Venezuela; teniendo como **campo de acción:** Gestión de Información asociada a las Actividades y la Administración de los Programas en las CR.

Para dar solución al problema planteado, esta investigación tiene como **objetivo general:** Desarrollar los módulos que gestionen la información referente a las Actividades y la Administración de Programas del Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela y la DGPD.

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

[1] **Albet:** Albet Ingeniería y Sistemas, S.A. es una empresa cubana, cuyo origen y desarrollo se vincula estrechamente a la Universidad de Ciencias Informáticas (UCI).

1. Realizar el diseño de clases de los módulos: Actividades y Administración de Programas.
2. Implementar los componentes correspondientes a los módulos: Actividades y Administración de Programas.
3. Validar funcionalmente los módulos: Actividades y Administración de Programas.

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

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.
2. Realización de los Diagramas de Clases del Diseño de los módulos: Actividades y Administración de Programas.
3. Realización de los Diagramas de Secuencia del Diseño de los módulos: Actividades y Administración de Programas.
4. Realización del los Diagramas de Componentes de los módulos: Actividades y Administración de Programas.
5. Implementación de los componentes de los módulos: Actividades y Administración de Programas.
6. Realización de pruebas funcionales a los módulos implementados.

A continuación se describe de manera resumida el contenido que se expone en cada uno de los 4 capítulos del presente trabajo de diploma:

Capítulo 1 Fundamentación Teórica: En este capítulo se ven reflejadas las características principales de la metodología de desarrollo utilizada, lenguaje de programación y herramientas fundamentales a tener en cuenta para la solución del problema planteado.

Capítulo 2 Diseño del Sistema: En este capítulo se describe el diseño de los módulos Actividades y Administración de Programas. Se reflejan los patrones de diseño empleados, los diagramas de clases y de secuencia más relevantes. Además se detallan los patrones y estilos arquitectónicos utilizados en el desarrollo de los módulos.

Capítulo 3 Implementación del Sistema: En este capítulo se muestran los diagramas de componentes que se definieron durante la implementación de los módulos Actividades y Administración de Programas, así como las principales pantallas correspondientes a las interfaces de estos módulos.

Capítulo 4 Pruebas del Sistema: En este capítulo se realiza el diseño de los casos de pruebas de los módulos Actividades y Administración de Programas para verificar si el producto satisface los requerimientos del usuario. Además se detallan las pruebas realizadas a la aplicación para comprobar sus funcionalidades.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

1.1 Introducción

En este capítulo se abordan primeramente algunos conceptos importantes para lograr un mejor entendimiento de los sistemas de gestión de información. Se especifica la metodología de desarrollo utilizada en el diseño y la implementación de los módulos Actividades y Administración de Programas, así como las principales herramientas, lenguajes de programación y frameworks utilizados para el desarrollo de los mismos.

1.2 Sistema de Gestión de la Información

A medida que se desarrolla la sociedad, las empresas, organismos, e instituciones, va en aumento creciente el volumen de información que se maneja sobre los mismos. El desarrollo por igual de las nuevas tecnologías de la información y las comunicaciones ha permitido transmitir, gestionar y compartir muchos de estos datos, pero el exceso de información hace que tengamos que invertir mucho tiempo en ella, de esta manera surge la necesidad de gestionarla, que no es más que el proceso de analizar, utilizar, recuperar y almacenar la información que se ha obtenido y registrado, para permitir el aprovechamiento de la misma de una manera más eficiente, rápida y organizada.

Existen en latinoamérica varios sistemas de prevención del delito como son:

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 sociodelictiva 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. [1]

Sistema de Información para la Prevención Comunitaria del Delito y la Violencia (SIPREC)

SIPREC desarrolla técnicas y metodologías que, mediante la participación ciudadana, posibilitan elaborar muestreos poblacionales que proporcionan información cuantitativa aproximada de hechos delictivos y situaciones de violencia y conflictos ocurridos en el territorio de la Ciudad Autónoma de Buenos Aires y las características de los mismos. [2]

Los sistemas mencionados anteriormente requieren de una coordinación entre los diferentes encargados de enviar información sobre los programas de prevención del delito, enviando esta por correo electrónico hacia el usuario final que sería el encargado de publicarlos. Estos sistemas requieren además de una coordinación entre responsables y enlaces para poder mantener

actualizado el sistema. Son mayormente informativos y la realización de los informes que se remiten en algunos de los casos son anualmente.

El Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. Módulos Actividades y Administración de Programas, permite de manera efectiva gestionar la información referente a los programas, proyectos y las actividades, garantizar la estandarización de la información y el intercambio adecuado del flujo de información entre la CR y la DGPD.

1.3 Herramientas y metodología

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

1.3.1 Metodología de desarrollo de software

RUP

Un proceso de desarrollo de software es el conjunto de actividades necesarias para transformar los requisitos del usuario en un sistema de software (**Figura 1**). Sin embargo el Proceso Unificado es más que un simple proceso; es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto [3].

Figura 1. Un Proceso de desarrollo de software.

Este proceso de desarrollo de software hace uso del **Lenguaje Unificado de Modelado** (UML) para preparar todos los esquemas del proyecto, de hecho, UML es una parte esencial del proceso, ya que

proporciona las bases para emprender de manera eficiente el ciclo de desarrollo. Este lenguaje es un estándar diseñado para visualizar, especificar, construir y documentar software orientado a objetos [4]. El modelado visual ayuda a mantener la consistencia entre los artefactos del sistema: requisitos, diseños e implementaciones. Este ayuda a mejorar la capacidad del equipo para gestionar la complejidad del software.

Roles y Artefactos

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

Análisis y Diseño

Diseñador: Es el responsable del diseño de una parte del sistema que incluye: las restricciones de los requisitos, la arquitectura y el proceso de desarrollo del proyecto.

Artefactos

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

Implementación

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

Artefactos

- **Elementos de implementación:** Son las partes físicas que componen la implementación, incluyendo archivos y directorios. Además, los archivos de código del software (binario o ejecutable), archivos de datos y documentación.

- **Artefactos de instalación:** Se refieren al software y a las instrucciones documentadas requeridas para instalar el producto.

Prueba

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

Probador: Conduce las pruebas necesarias y el registro del resultado de la mismas.

Artefactos

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

1.3.2 Herramienta CASE

En los últimos años se ha trabajado para encontrar técnicas que permitan incrementar la productividad y el control de la calidad en cualquier proceso de elaboración de software, hoy en día las herramientas **CASE** (Computer Aided Software Engineering) sustituyen métodos como el papel y el lápiz, por el ordenador, para convertir la actividad de desarrollar software en un proceso automatizado. La ingeniería de sistemas asistida por ordenador (CASE) es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo, su objetivo es acelerar el proceso para el que han sido diseñadas, en el caso de CASE para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de software.

Visual Paradigm

Esta herramienta CASE de modelado profesional soporta el ciclo de vida completo del desarrollo de software. Provee soporte para la generación de código, ingeniería inversa para Java, se integra con Eclipse, Borland JBuilder y Oracle JDeveloper, para soportar las fases de implementación en el desarrollo de software. Tiene dentro de sus características su buena integración con los IDE para el desarrollo de java y la ventaja de que presenta una interfaz de usuario de fácil uso. Dicha herramienta ofrece un entorno de creación de diagramas para UML 2.0, diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad y uso de un lenguaje estándar común, a todo el equipo de desarrollo, que facilita la comunicación.

1.3.3 Servidor Web

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

Apache Tomcat 6.0.20

El Apache Tomcat es un software de código abierto implementado para las tecnologías Java Servlet y Java Server Pages. Apache Tomcat es desarrollado en un entorno abierto y participativo y publicado bajo la licencia del software de Apache. El mismo se ejecuta en varios Sistemas Operativos. Es un servidor configurable de diseño modular, con diversidad que permiten garantizar una elevada seguridad y buenas prestaciones. Además, brinda soporte para la plataforma de desarrollo JEE. Existe bastante documentación asociada al mismo, cuenta con una gran comunidad de desarrollo y es uno de los más usados internacionalmente.

1.3.4 Lenguaje de Programación

Los lenguajes de programación son herramientas que permiten crear programas y software facilitando la tarea de programación ya que disponen de formas adecuadas que permiten ser leídas y escritas por personas que resultan independientes del modelo de computadora a utilizar.

Java

Java es un Lenguaje de Programación Orientado a Objetos (POO) ideado por un equipo de desarrollo de Sun Microsystems (adquirida por Oracle Corporation). Es un lenguaje de propósito general con gran aceptación en la web, especialmente en las aplicaciones Cliente-Servidor y el uso de JSP (Java Server Pages). Entre sus principales características tenemos:

Orientado a Objeto

En este aspecto Java fue diseñado partiendo de cero, no siendo derivado de otro lenguaje anterior y no tiene compatibilidad con ninguno de ellos. En Java el concepto de objeto resulta sencillo y fácil de ampliar. Además se conservan elementos “no objetos”, como números, caracteres y otros tipos de datos simples.

Robusto

Java verifica su código al mismo tiempo que lo escribe, y una vez más antes de ejecutarse, de manera que se consigue un alto margen de codificación sin errores. Se realiza un descubrimiento de

la mayor parte de los errores durante el tiempo de compilación, ya que Java es estricto en cuanto a tipos y declaraciones, y así lo que es rigidez y falta de flexibilidad se convierte en eficacia.

Multiplaforma

Java está diseñado para que un programa escrito en este lenguaje sea ejecutado correctamente independientemente de la plataforma en la que se esté actuando (Macintosh, PC, UNIX…). Este tipo de aplicaciones son conocidas como “escribir una vez, y ejecutar en cualquier parte” (write once, run anywhere).Para conseguir esto, se utiliza una compilación en una representación intermedia que recibe el nombre de Bytecode que puede interpretarse en cualquier sistema operativo con un intérprete de Java.

1.3.5 Plataforma de Desarrollo

Una plataforma de desarrollo es el entorno de software común en el cual se desenvuelve la programación de un grupo definido de aplicaciones. Comúnmente se encuentra relacionada directamente a un sistema operativo, sin embargo, también es posible encontrarla ligada a un lenguaje de programación como por ejemplo JEE (Java Enterprise Edition) la cual goza de gran aceptación a nivel internacional.

JEE

La plataforma JEE ha sido diseñada para aplicaciones distribuidas con base en componentes o unidades funcionales de software que interactúan entre sí para formar parte de una aplicación empresarial JEE. Esta plataforma provee una arquitectura robusta encaminada a la construcción de aplicaciones basadas en n-capas que permite entre otras cosas una mejor escalabilidad, seguridad, concurrencia y gestión de los componentes desplegados. La plataforma JEE añade a Java la funcionalidad necesaria para convertirse en un lenguaje orientado al desarrollo de servicios en Internet. Mediante JSP y Servlets se pueden desarrollar sitios Web bajo la tecnología Java. [5]

1.3.6 Entorno de Desarrollo Integrado

Un entorno de desarrollo integrado o en inglés Integrated Development Environment (IDE) es un programa compuesto por un conjunto de herramientas para un ordenador. Generalmente incluyen en una misma suite un buen editor de código, enlace transparente a compiladores y debuggers e integración con sistemas controladores de versiones o repositorios. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Eclipse

Eclipse es un armazón (workbench) sobre el que se pueden montar herramientas de desarrollo para cualquier lenguaje, mediante la implementación de los plugins adecuados. La arquitectura de plugins de Eclipse permite integrar diversos lenguajes sobre un mismo IDE, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces, ayuda en línea para librerías, etc. [8]

El IDE Eclipse es multiplataforma. Permite compilación incremental de código. Modifica e inspecciona valores de variables. Además, avisa de los errores cometidos mediante una ventana secundaria y depura código que resida en una máquina remota. Eclipse constituye una de las mejores herramientas para el desarrollo de aplicaciones libres en java.

1.3.7 Sistema Gestor de Base de Datos

Un Sistema Gestor de Bases de Datos (SGBD) es una colección de programas cuyo objetivo es servir de interfaz entre la base de datos, el usuario y las aplicaciones. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. Un SGBD permite definir los datos a distintos niveles de abstracción y manipular dichos datos, garantizando la seguridad e integridad de los mismos. Algunos ejemplos de SGBD son Oracle, DB2, PostgreSQL, MySQL, MS SQL Server, etc. Un SGBD debe permitir:

- Definir una base de datos: especificar tipos, estructuras y restricciones de datos.
- Construir la base de datos: guardar los datos en algún medio controlado por el mismo SGBD.
- Manipular la base de datos: realizar consultas, actualizarla y generar informes.

PostgreSQL 8.4

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD (Berkeley Software Distribution) y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado. PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando. [9]

1.4 Frameworks

Los frameworks son soluciones completas que contemplan herramientas de apoyo a la construcción (ambiente de trabajo o desarrollo) y motores de ejecución (ambiente de ejecución). Tienen como objetivo brindarles a los desarrolladores una mejor organización y estructura de sus proyectos proporcionando una arquitectura definida la cual ayuda hacer sus aplicaciones con mayor rigidez. [6]

1.4.1 Spring 3.0.2

Spring Framework aparece en el año 2003 como parte de un proyecto Open Source (OS) como alternativa para el desarrollo de aplicaciones empresariales. Es un contenedor de peso ligero, basado en inyección de dependencias y orientado a aspectos. A continuación se muestran las principales características del mismo:

Framework: Spring hace posible configurar y componer complejas aplicaciones a partir de simples componentes. En Spring, los objetos de aplicación son compuestos declarativamente, generalmente en un archivo XML (Extensible Markup Language). Además Spring brinda abundante funcionalidad de infraestructura (gestión de transacciones, integración con frameworks de persistencia, etc), dejando el desarrollo de la lógica de la aplicación al programador.

Peso ligero: Spring es ligero en términos de tamaño y en general. La parte principal de Spring puede ser distribuida en un solo archivo JAR que pesa sobre los 2.5 MB y el procesamiento general requerido por Spring es insignificante. Además, Spring no es intrusivo: a menudo los objetos de una aplicación basada en Spring no tienen dependencias con clases específicas de Spring.

Contenedor: Spring es un contenedor en el sentido de que contiene y gestiona el ciclo de vida y configuración de objetos de aplicación. En Spring se puede declarar como cada uno de los objetos de la aplicación deben ser creados, configurados y asociados entre ellos.

Inyección de Dependencias: Spring promueve el bajo acoplamiento a través de una técnica conocida como Inyección de Dependencias. Es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase quien cree el objeto.
Este radica en resolver las dependencias de cada clase (atributos) generando los objetos cuando se arranca la aplicación y luego inyectarlos en los demás objetos que los necesiten a través de métodos set o bien a través del constructor, estos objetos se instancian una vez y se comparten por todos los usuarios (al menos en el caso de Spring) evitando tener que andar extendiendo clases.

Orientación a Aspectos: Spring viene con un gran soporte para programación orientada a aspectos (AOP) que habilita un desarrollo cohesivo a través de la separación de la lógica de negocio de los servicios del sistema. Los objetos de la aplicación hacen lo que les corresponde (ejecutar la lógica de negocio) y nada más. Ellos no son responsables ni incluso conscientes de otros asuntos del sistema.

Módulos de Spring

Spring está formado por una serie de módulos, bien definidos, que en su conjunto brindan el soporte y las facilidades necesarias para desarrollar una aplicación empresarial en JEE. No es necesario usar todos los módulos en la aplicación ya que por una parte, aunque los módulos se integran entre sí, son bastante independientes y por otro lado Spring brinda posibilidades de integración con una serie de frameworks y librerías, los cuales pueden ser usados en defecto de algunos módulos.

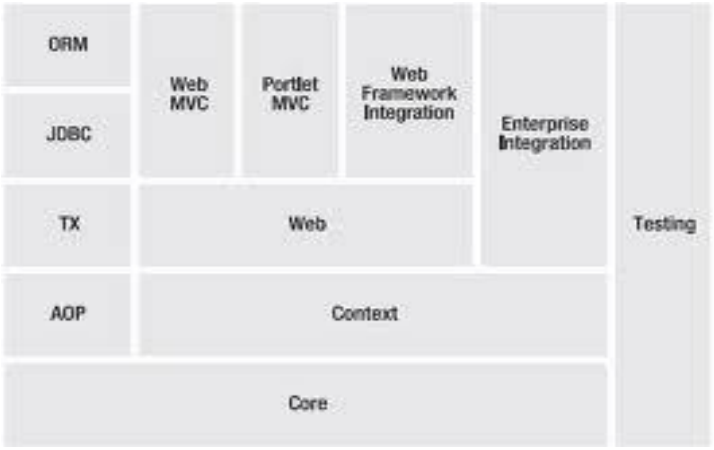

Figura 3. Arquitectura de Spring

El contenedor del núcleo:

Este módulo brinda las funcionalidades fundamentales del framework. Contiene el Bean Factory, que es el contenedor fundamental y el basamento de la Inyección de Dependencias en Spring.

Contexto de la Aplicación:

Este módulo está construido sobre el núcleo y así como el núcleo hace de Spring un contenedor, el módulo de contexto hace de Spring un framework. Extiende el concepto de BeanFactory añadiendo soporte para mensajes de internacionalización, manejar el ciclo de vida y validación de los beans en la aplicación. Además provee servicios como correo, acceso JNDI, integración con EJB, entre otros.

Programación Orientada a Aspectos (AOP)

Spring brinda un rico soporte para AOP en este módulo. Al igual que DI, AOP soporta el bajo acoplamiento entre los objetos de la aplicación. Con AOP sin embargo, los asuntos globales de la aplicación como transacciones y seguridad están desacoplados de los objetos a los que se les aplica.

Capa Test

La capa de prueba apoya el examen de los componentes de Spring con JUnit. Este es un marco simple para escribir pruebas repetibles. Proporciona carga constante de ApplicationContexts y el

almacenamiento en caché de los contextos. Para realizar las pruebas se utiliza JUnit integrado al IDE Eclipse en forma de plug-in.

JDBC

El módulo de abstracción JDBC (Java Database Connectivity) permite la separación de todo el código repetitivo asociado al trabajo con JDBC en cuanto a conexiones, sesiones en la base de datos, etc. Prevé problemas que resultan a partir del cierre de recursos de la base de datos y construye una capa de significativos errores sobre los errores brindados por varios servidores de bases de datos. Además usa AOP para brindar servicios de manejo de transacciones para los objetos en la aplicación.

Integración con Mapeo Relacional de Objetos (ORM)

El soporte para ORM está construido en el soporte para DAO (Data Access Object), brindando una manera conveniente de crear DAO's para varias soluciones ORM incluyendo Hibernate, Java Persistente API, Java Data Objects y iBATIS SQL Maps. El manejo de transacciones de Spring también soporta estos frameworks.

Modelo Vista Controlador (MVC)

El paradigma Modelo-Vista-Controlador, es usado habitualmente para el desarrollo de aplicaciones Web en las que se separa la interfaz del usuario de la lógica de la aplicación. Sobre Java existen varios frameworks entre los que se encuentran Apache Struts, JSF, WebWork y Trapestry. Aún cuando Spring brinda la posibilidad de integrarse con la mayoría de estos frameworks, él brinda su propia solución MVC que impulsa las técnicas de bajo acoplamiento de Spring.

Spring Web

Este módulo brinda clases de soporte especial para Spring MVC. Además contiene soporte para varias tareas orientadas a la web como peticiones, upload, conexión de parámetros de respuesta con objetos del negocio, etc. Además contiene soporte de integración con Java Server Faces (JSF) y Apache Struts.

Como puede apreciarse Spring cubre una gran cantidad de aspectos para el desarrollo de una aplicación empresarial y sobre todo facilita la integración con los mejores frameworks y API de cada una de las áreas involucradas en la aplicación dígase, persistencia, transacciones, comunicación remota, etc.

1.4.2 Spring Security 3.0.2

Spring Security es un framework que provee las declaraciones de seguridad de una aplicación basada en Spring. Ofrece una completa solución de seguridad, la manipulación de autenticación y autorización en la solicitud web y en la invocación de método.

1.4.3 Hibernate 3.5

Hibernate es una capa de persistencia objeto-relacional y un generador de sentencias SQL (Structured Query Language). Permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones, y un gran número de tipos de datos. De una manera muy rápida y optimizada se pueden generar Bases de Datos en cualquiera de los entornos soportados: Oracle, DB2, MySql, PostgreSql, etc. [7]

1.4.4 Dojo 1.4.3

DOJO es un framework escrito en JavaScript. Se compone de un conjunto de librerías al estilo Java. Su idea es la de abstraer al desarrollador de las complejidades del DHTML y de las discrepancias existentes entre navegadores, que hacen que el código JavaScript a utilizar sea diferente. Algunas de las características que tiene son:

- Maneja incompatibilidades entre navegadores.
- Soporte AJAX.
- Oculta el manejo del XMLHttpRequest.
- Soporte de backward y forward.
- Sistema de eventos orientado a aspectos.

1.4.5 JasperReport 3.5.2

JasperReport es una poderosa herramienta para generar reportes en java, con la habilidad de reproducir contenido completo para la pantalla, directo para impresora o en diferentes formatos de archivos (PDF, XLS, CSV, XML, RTF, entre otros). Es una librería 100% java y puede reutilizarse tanto en aplicaciones cliente-servidor como aplicaciones web, JEE, etc. JasperReport permite organizar la información obtenida desde una base de datos relacional a través de conectores JDBC en diseños de reportes predefinidos en formato XML.

1.4.6 Hibernate Validator 4.0.2

Es una implementación que permite definir validaciones usando XML y anotaciones, validación completa y recursiva de objetos, definición a medida de restricciones y validadores así como

personalización de mensajes de error. Trae un conjunto predefinido de validaciones típicas que son definidas con anotaciones.

1.4.7 JPA 2.0

Java Persistence API (JPA) proporciona un modelo de persistencia basado en POJO's (Plain Old Java Object) para mapear bases de datos relacionales en Java. El Java Persistence API fue desarrollado por el grupo de expertos de EJB 3.0 (Enterprise JavaBeans) como parte de JSR 220 (Java Specification Requests), aunque su uso no se limita a los componentes software EJB. También puede utilizarse directamente en aplicaciones web y aplicaciones clientes; incluso fuera de la plataforma JEE, por ejemplo, en aplicaciones Java SE (Standard Edition).

En su definición, se han combinado ideas y conceptos de los principales frameworks de persistencia como Hibernate, Toplink y JDO (Java Data Objects), y de las versiones anteriores de EJB. Todos estos cuentan actualmente con una implementación JPA.

El mapeo objeto/relacional, es decir, la relación entre entidades Java y tablas de la base de datos, se realiza mediante anotaciones en las propias clases de entidad, por lo que no se requieren ficheros descriptores XML. También pueden definirse transacciones como anotaciones JPA.

1.5 Conclusiones

En el capítulo se abarcaron una serie de conceptos necesarios para el desarrollo de los módulos. Se describieron las herramientas y tecnologías en las que se desarrollará el sistema así como los diferentes frameworks, garantizando de esta forma un software robusto, escalable y orientado a las tecnologías Open Source. Se utilizarán otras tecnologías como PostgreSQl, Hibernate, JPA, entre otras. Además como IDE principal para el desarrollo de la aplicación se utilizará Eclipse.

CAPÍTULO 2: DISEÑO DEL SISTEMA.

2.1 Introducción

En este capítulo se detalla el estilo arquitectónico utilizado, se describen los patrones de diseño empleados y se muestran los diagramas de clases del diseño así como los diagramas de secuencia del caso de uso Gestionar Actividad Realizada del módulo Actividades.

2.2 Estilo arquitectónico utilizado

Arquitectura de software

Dentro de las cientos de definiciones de Arquitectura de Software, mundialmente se ha reconocido como la oficial la que brinda el documento de IEEE Std 1471-2000 donde refleja: “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”. Es una definición bastante abstracta pero hay varias empresas que se han regido por dicho documento incluyendo Microsoft.

Estilo arquitectónico

Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales. [10]

Para el desarrollo del Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela se seleccionó el estilo arquitectónico en Capas, que se encuentra dentro de los estilos de Llamada y Retorno. [11]

Este estilo define cómo organizar el modelo de diseño a través de capas, que pueden estar físicamente distribuidas, lo cual quiere decir que los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores. Particularmente se utilizó la especialización arquitectura de tres capas, la cual consiste en dividir la carga en tres capas lógicas fundamentales: Capa de Presentación, Capa de Lógica de Negocio, y Capa de Acceso a Datos **(Figura 4).**

Figura 4. Arquitectura en tres capas.

Capa de Presentación

Es la encargada de interactuar con el usuario y se corresponde con lo que tradicionalmente se conoce como interfaz de usuario, esta capa se comunica únicamente con la capa de negocio y en ella es donde se implementa todo lo relacionado con la interfaz gráfica. Aquí reside la definición de las peticiones que el usuario puede realizar sobre la aplicación, los controladores que manejan el flujo web y la comunicación con las interfaces de la capa de negocio. Además se encuentran las vistas HTML, XML, PDF, XLS que se muestran al usuario y la implementación del comportamiento dinámico de los documentos HTML a través de Javascript. Las responsabilidades principales de la capa de presentación son: la navegabilidad del sistema, el formateo de los datos de salida, la validación de los datos de entrada y la construcción de la interfaz gráfica de usuario. Se debe destacar que en esta capa va a estar presente el patrón arquitectónico **Modelo-Vista-Controlador (MVC)** ya que Spring lo utiliza en su funcionamiento. 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:

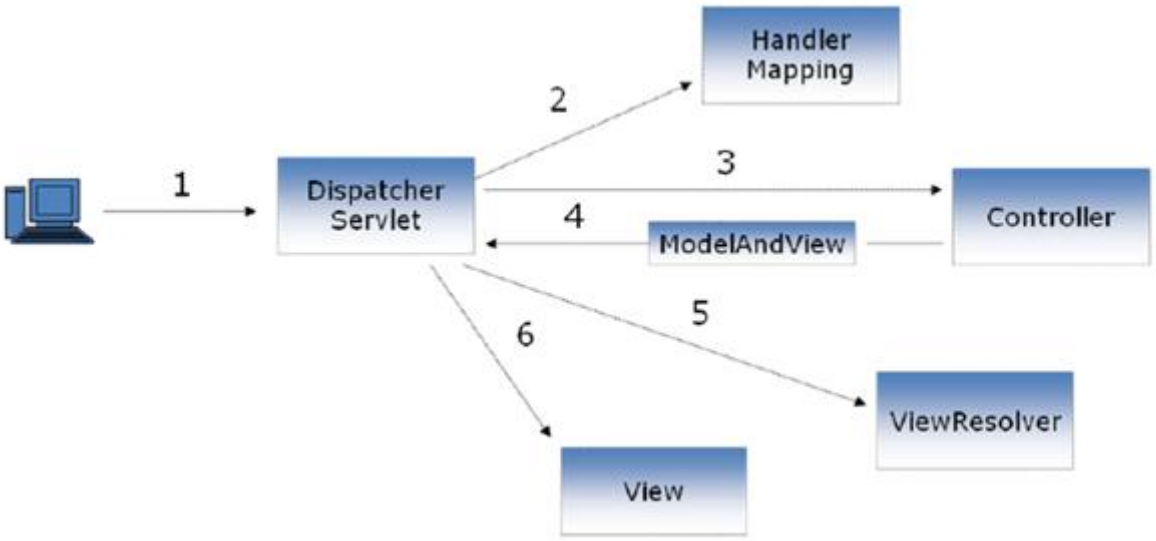

Figura 5. Patrón arquitectónico MVC en Spring.

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, esta clase es configurada en el archivo **app_servlet.xml**.

HandlerMapping: Permite mapear las peticiones HTTP, utilizando las URLs o partes de éstas, 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.

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.

Capa de Lógica de Negocio

Establece una comunicación con la capa de presentación para recibir las solicitudes, procesarlas y presentar los resultados, además de interactuar con la capa de acceso a datos para persistir o recuperar información, por lo que define e implementa las funcionalidades que responden

directamente a los requisitos de manera que se conserve la integridad del sistema y de los datos. Está constituida por los servicios y la fachada.

Capa de Acceso a Datos

Es la encargada de manejar la información desde y hacia la base de datos. Contiene las clases del dominio representadas como clases entidades, los objetos que encapsulan la lógica de acceso a datos (DAO) y sus interfaces que se mantienen independientes de Spring e Hibernate; así como las implementaciones de los DAOs que extienden de clases de soporte del framework Spring para el uso de este patrón usando el framework ORM Hibernate.

2.3 Patrones de diseño utilizados

Patrón de Diseño

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. Brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares. Se debe tener presente los siguientes elementos de un patrón: su nombre, el problema (cuando aplicar un patrón), la solución (descripción abstracta del problema) y las consecuencias (costos y beneficios). [14]

A continuación se describen los patrones de diseño utilizados en la implementación del Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela.

Patrón Controlador Frontal

El patrón Controlador Frontal propone utilizar un controlador como punto inicial de contacto para manejar las peticiones del usuario en una aplicación, 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. El framework Spring MVC a través del DispatcherServlet, que en el caso de la aplicación sería el **app-servlet.xml,** hace uso de este patrón.

El funcionamiento del patrón Controlador Frontal se evidencia de la siguiente manera: cada vez que entra una petición por la URL es recibida por el DispatcherServlet activando el mecanismo para determinar cual controlador ejecuta la petición. El controlador ejecuta la acción y devuelve un ModelAndView al DispatcherServlet el que se encarga de despachar la petición a la Vista. Un ejemplo del uso de este patrón se ve expuesto en la **Figura 6** correspondiente al diagrama de secuencia del caso de uso Registrar Actividad Realizada, donde el app-servlet maneja las peticiones de entrada y

las envía a los correspondientes controladores, que ejecutan la acción y devuelven la vista con el modelo de datos, es el app-servlet el encargado nuevamente de manejar las peticiones y seleccionar la vista indicada para mostrársela al usuario con los datos enviados del controlador.

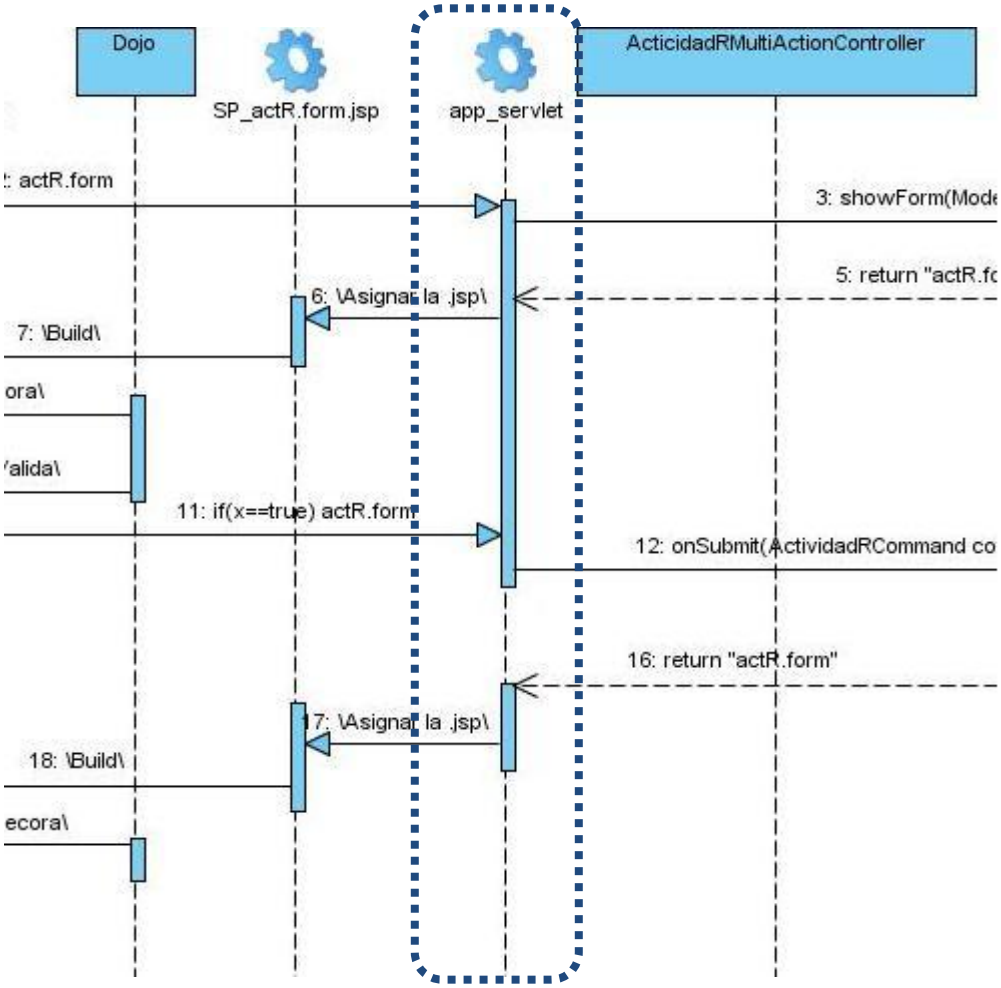

Figura 6. Uso del patrón Controlador Frontal.

Patrón Controlador

El Patrón Controlador 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 mantienen un modelo de alta cohesión. Dentro del paquete **vnz.mpprij.prevencion.cr.act.web,** es donde se encuentran los controladores del módulo Actividades. Un ejemplo del uso de este patrón se ve expuesto en la **Figura 7** correspondiente al diagrama de secuencia del caso de uso Registrar Actividad Realizada, donde una vez que el app-servlet envía la información a los controladores, pues éstos se las delegan a la fachada que se encarga de encapsular la lógica de negocio.

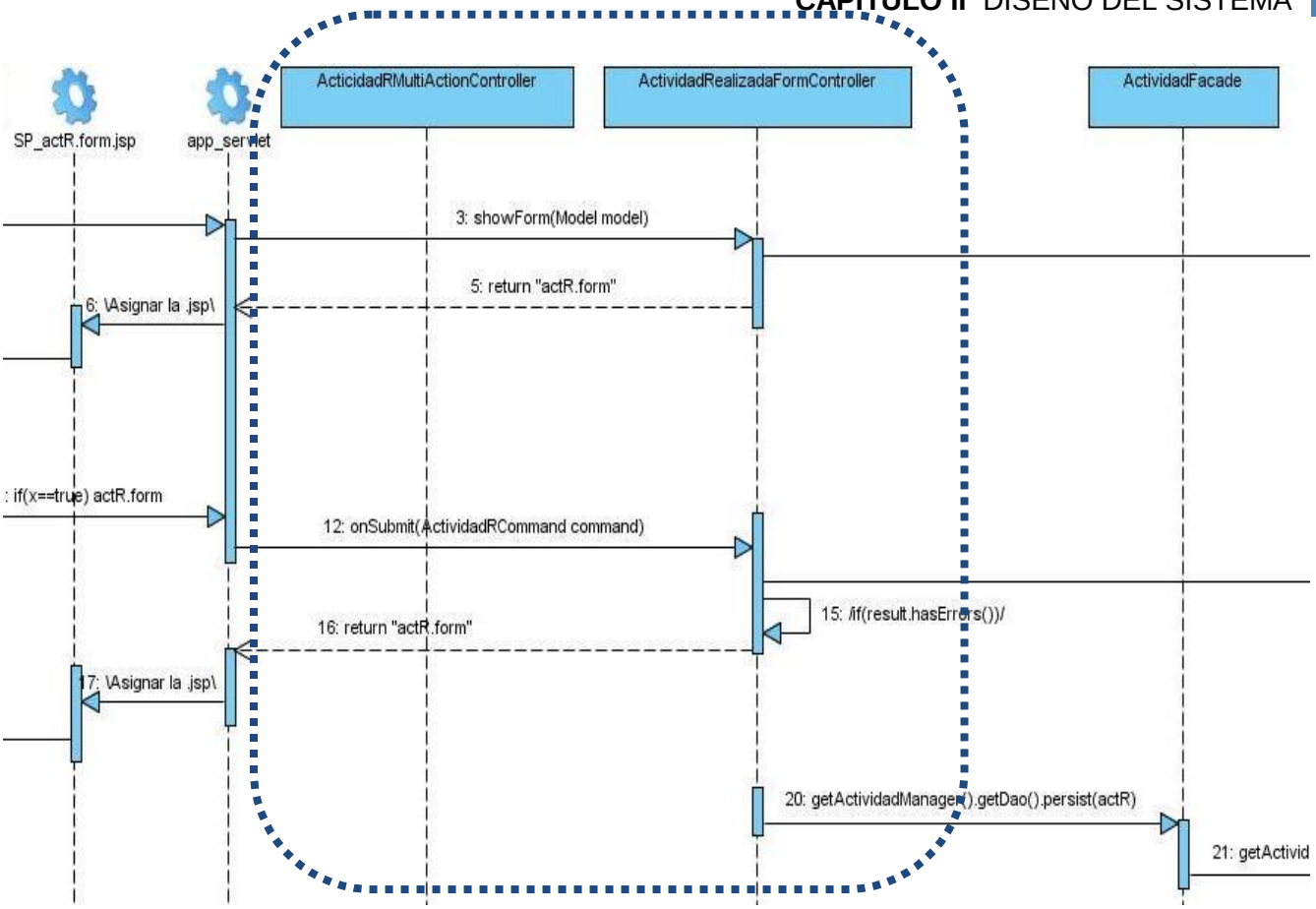

Figura 7. Uso del patrón Controlador.

Patrón Fachada

Este patrón se utiliza para proveer al sistema de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Utilizado para reducir la dependencia entre clases, ya que ofrece un punto de acceso al resto de las clases, si estas cambian o se sustituyen por otras solo hay que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones clientes. Fachada no oculta las clases sino que ofrece una forma más sencilla de acceder a ellas, en los casos en que se requiere se puede acceder directamente a ellas. El uso de este patrón en el sistema se evidencia en las clases de los paquetes **vnz.mpprij.prevencion.cr.act.facade** las cuales sirven de fachada entre las clases de los paquetes **vnz.mpprij.prevencion.cr.act.web** y **vnz.mpprij.prevencion.cr.act.manager** del módulo Actividades**.** Un ejemplo del uso de este patrón se ve expuesto en la **Figura 8** correspondiente al diagrama de secuencia del caso de uso Registrar Actividad Realizada.

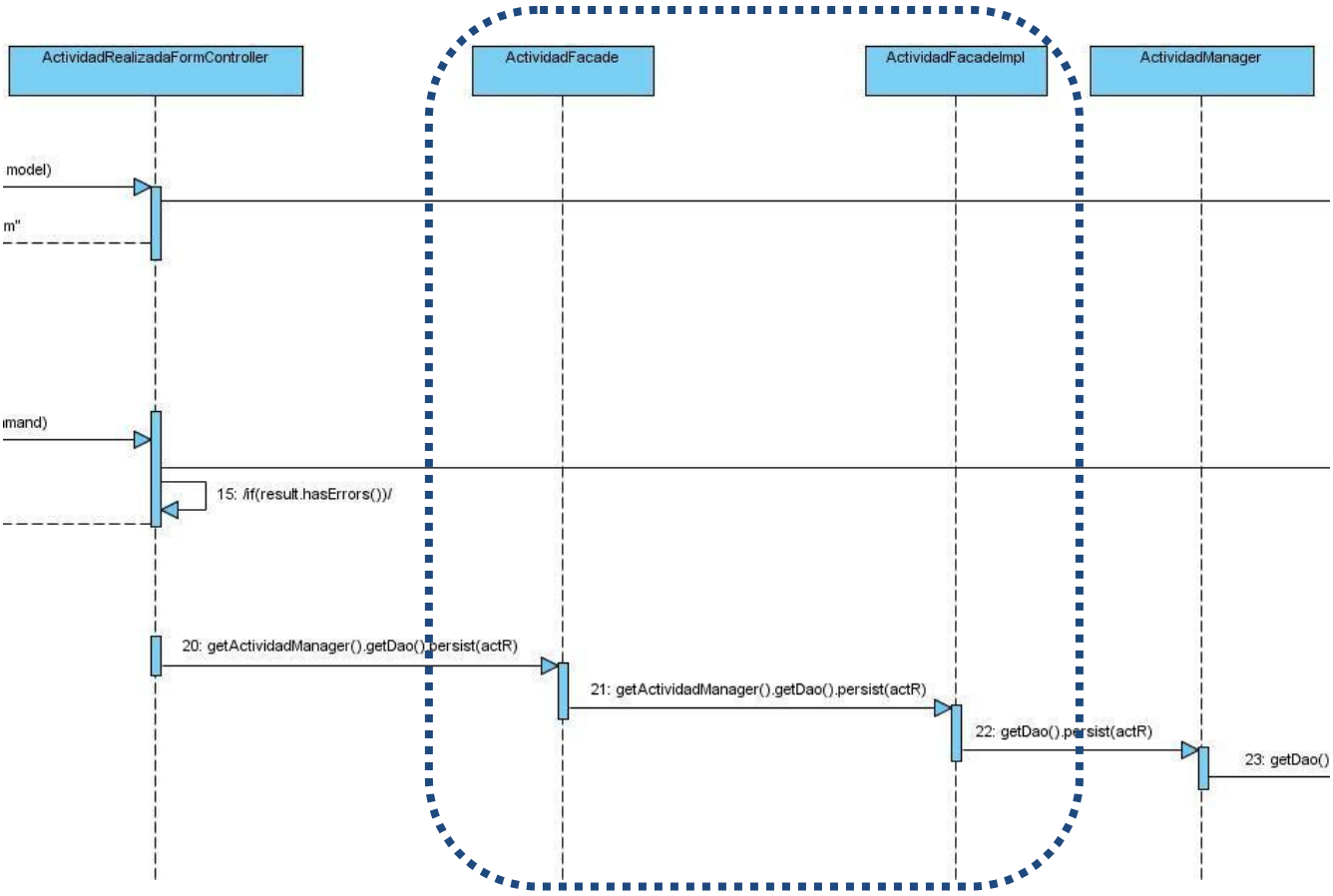

Figura 8. Uso del patrón Fachada.

Patrón Inyección de Dependencias

Patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase quien cree el objeto. Es una forma de Inversión de Control, que está basada en constructores de Java. Este radica en resolver las dependencias de cada clase (atributos) generando los objetos cuando se arranca la aplicación y luego inyectarlos en los demás objetos que los necesiten a través de métodos set o bien a través del constructor, pero estos objetos se instancian una vez, se guardan en una factoría y se comparten por todos los usuarios (al menos en el caso de Spring) y evitando tener que andar extendiendo clases. [15]

Haciendo referencia de este patrón, vemos como se encuentra presente en todos los controladores de los módulos a los cuales se le inyectan las fachadas correspondientes para su funcionamiento mediante el uso de la anotación @Resource la cual provee de atributos a clases mediante los métodos set.

Patrón Objeto de Acceso a Datos

Patrón de Diseño J2EE que centraliza todo el acceso a datos en una capa independiente, aislándolo del resto de la aplicación. Como beneficios 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. Este patrón se evidencia en la capa de Acceso a Datos mediante el DAO genérico que es el encargado de gestionar los datos entre las clases que contienen la lógica de negocio y el ORM Hibernate. Un ejemplo del uso de este patrón se ve expuesto en la **Figura 9**, correspondiente al diagrama de secuencia del caso de uso Registrar Actividad Realizada.

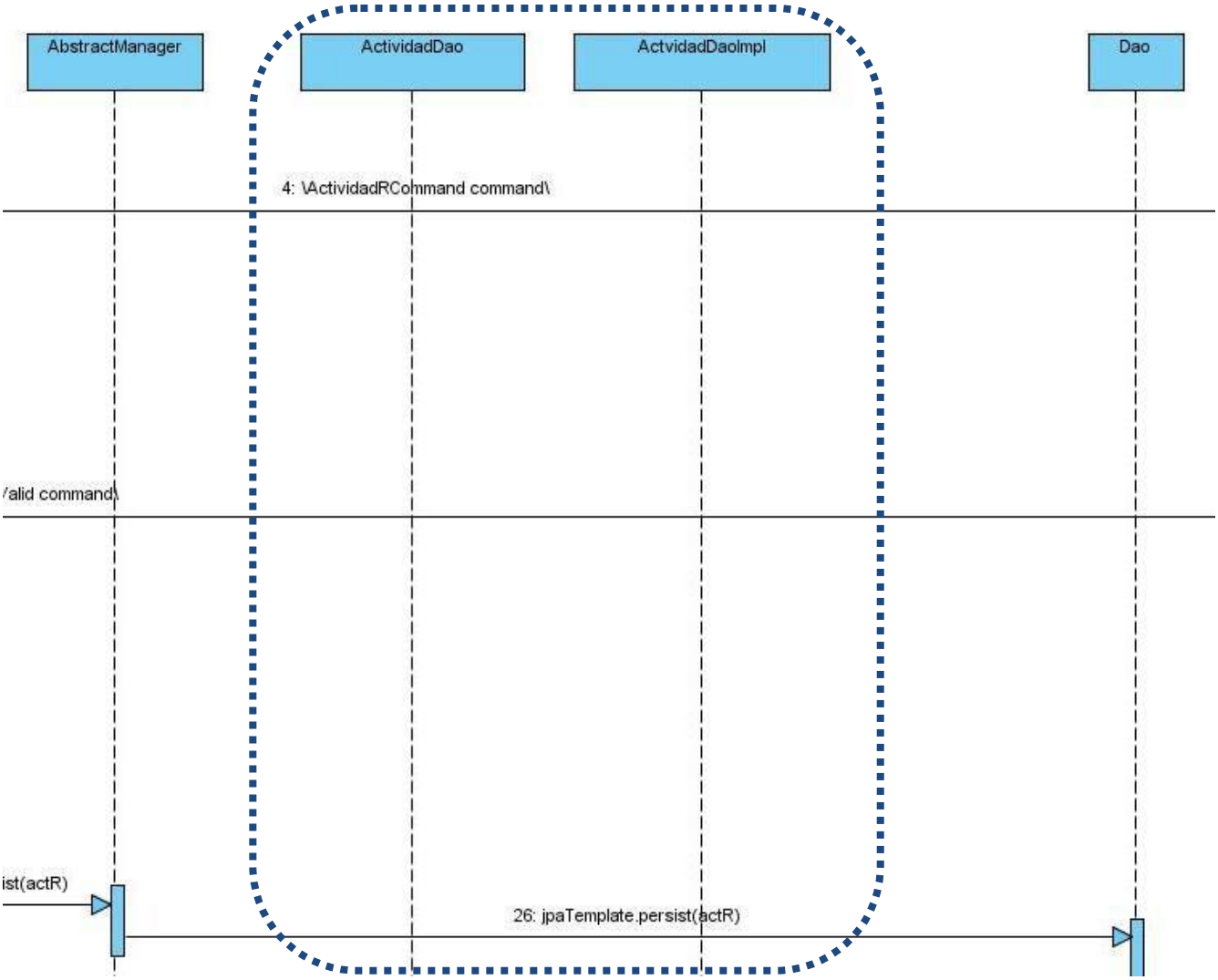

Figura 9. Ejemplo del patrón Objeto de Acceso a Datos.

Patrón Alta Cohesión

La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme, esto 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 que significa que una clase no depende de muchas clases. En el diseño de los módulos se logró que a partir de la definición de los mismos cada uno manejara la menor cantidad de entidades posibles. De esta manera a las clases pertenecientes a las diferentes capas se le asignaron las responsabilidades

necesarias para asegurar que la cohesión siga siendo alta y generando además un bajo acoplamiento.

2.5 Diagramas de Clases del Diseño

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

El diseño de las clases partió de los requerimientos funcionales definidos. A continuación se muestra en la **Figura 10** el Diagrama de clases del diseño para el caso de uso Gestionar Actividad Realizada perteneciente al módulo Actividades, donde se refleja la ubicación de las clases por capas. Este diagrama abarca los siguientes requisitos agrupados para este caso de uso:

RF - Registrar actividad realizada en la Coordinación Regional.
El sistema permitirá registrar los datos de una actividad realizada en la Coordinación Regional.

RF- Actualizar actividad realizada en la Coordinación Regional.
El sistema permitirá actualizar los datos de una actividad realizada en la Coordinación Regional.

Las funcionalidades correspondientes a los restantes casos de uso del módulo Actividades así como las funcionalidades del módulo Administración de Programas, pueden ser consultadas en el (Anexo 3).

En la **Figura 11** se puede observar específicamente la capa de Presentación, en la **Figura 12** la capa de Lógica de Negocio, y en la **Figura 13** la capa de Acceso a Datos. Es válido destacar que los restantes diagramas de clases de diseño pertenecientes a los Módulos Actividades y Administración de Programas se encuentran en el Expediente de proyecto del Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito.

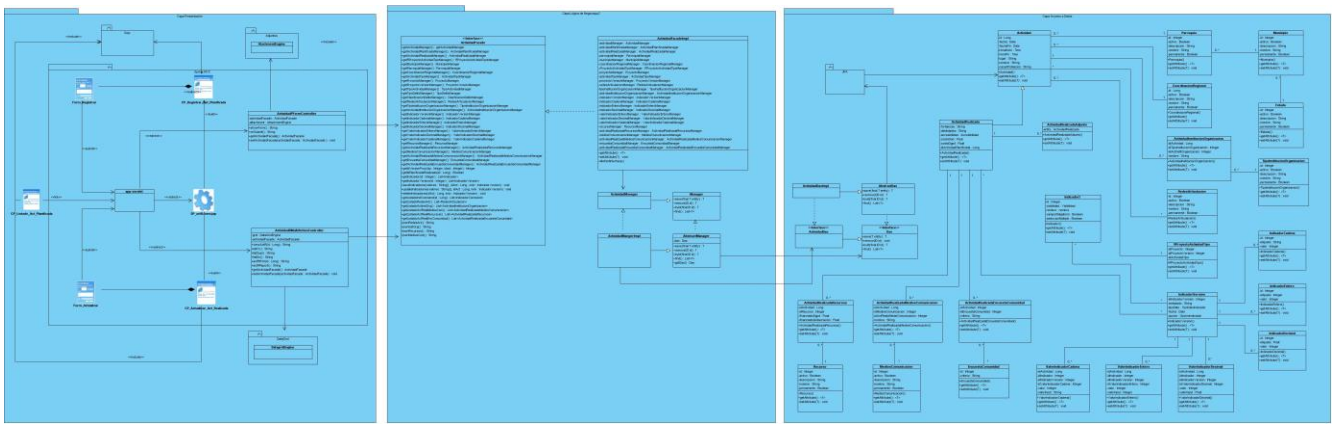

Figura 10. Diagrama de clases de diseño para el caso de uso Gestionar Actividad Realizada.

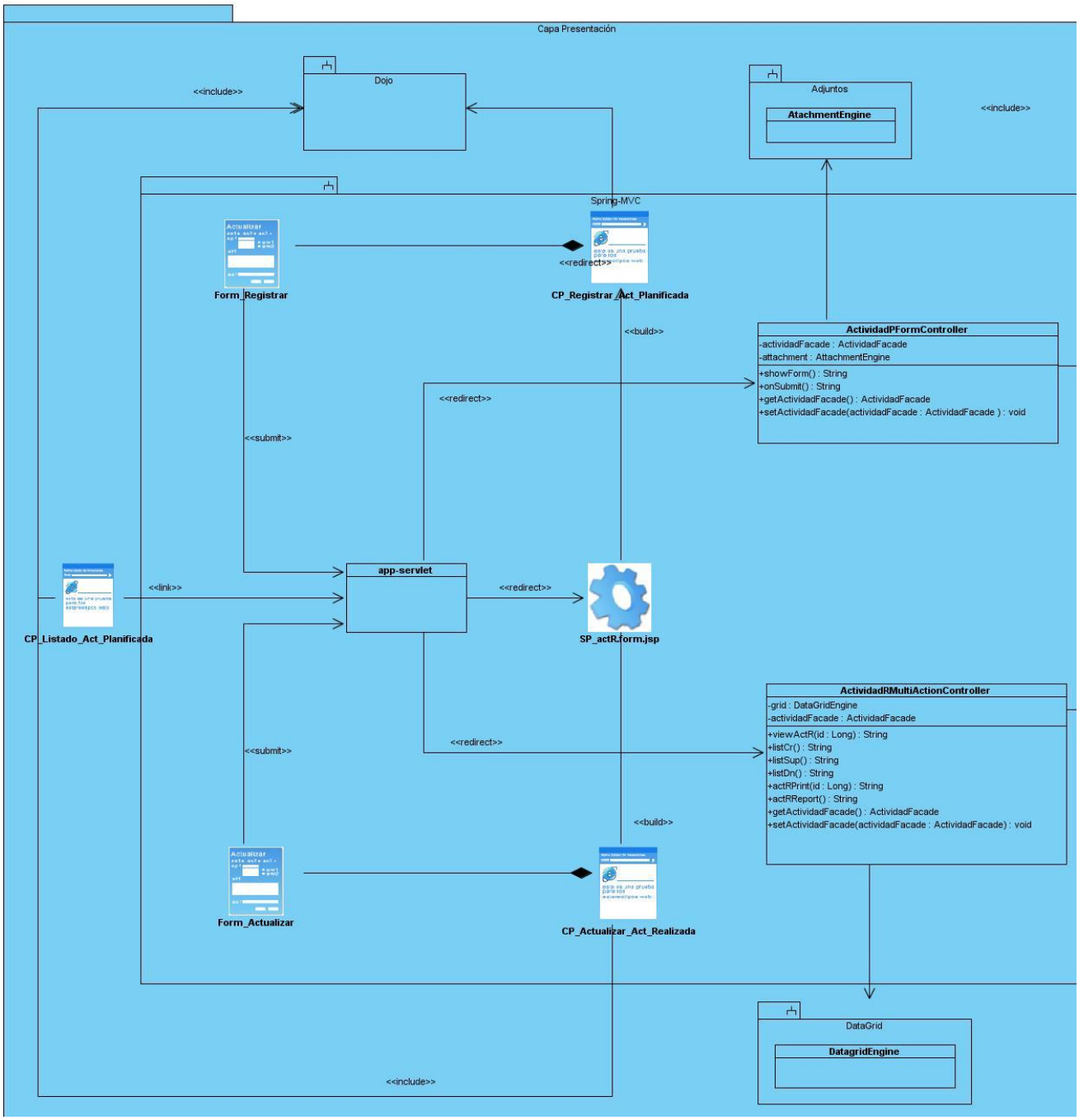

Figura 11. Diagrama de clases de diseño para el caso de uso Gestionar Actividad Realizada: Capa de Presentación.

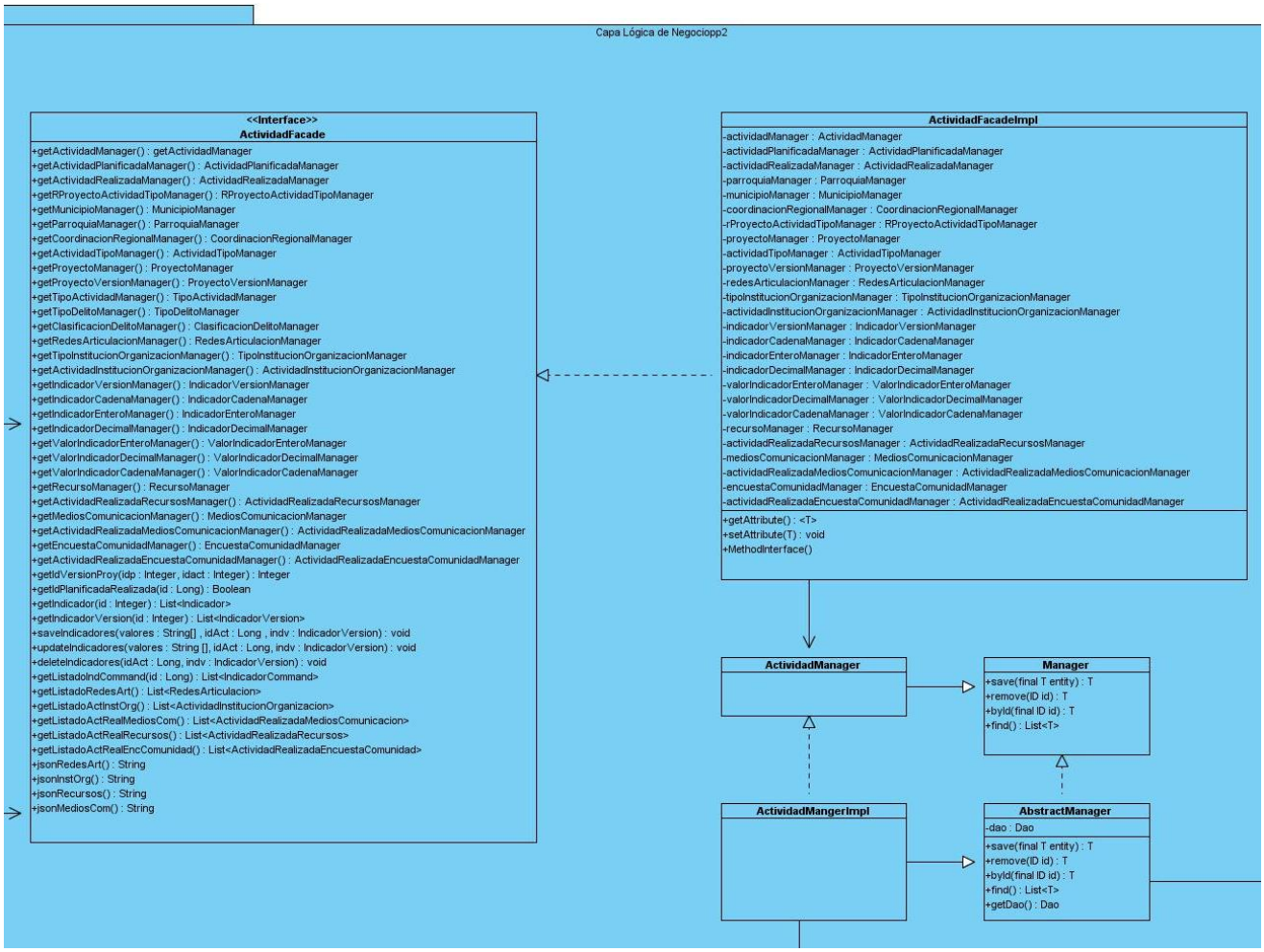

Figura 12. Diagrama de clases de diseño para el caso de uso Gestionar Actividad Realizada: Capa de Lógica de Negocio.

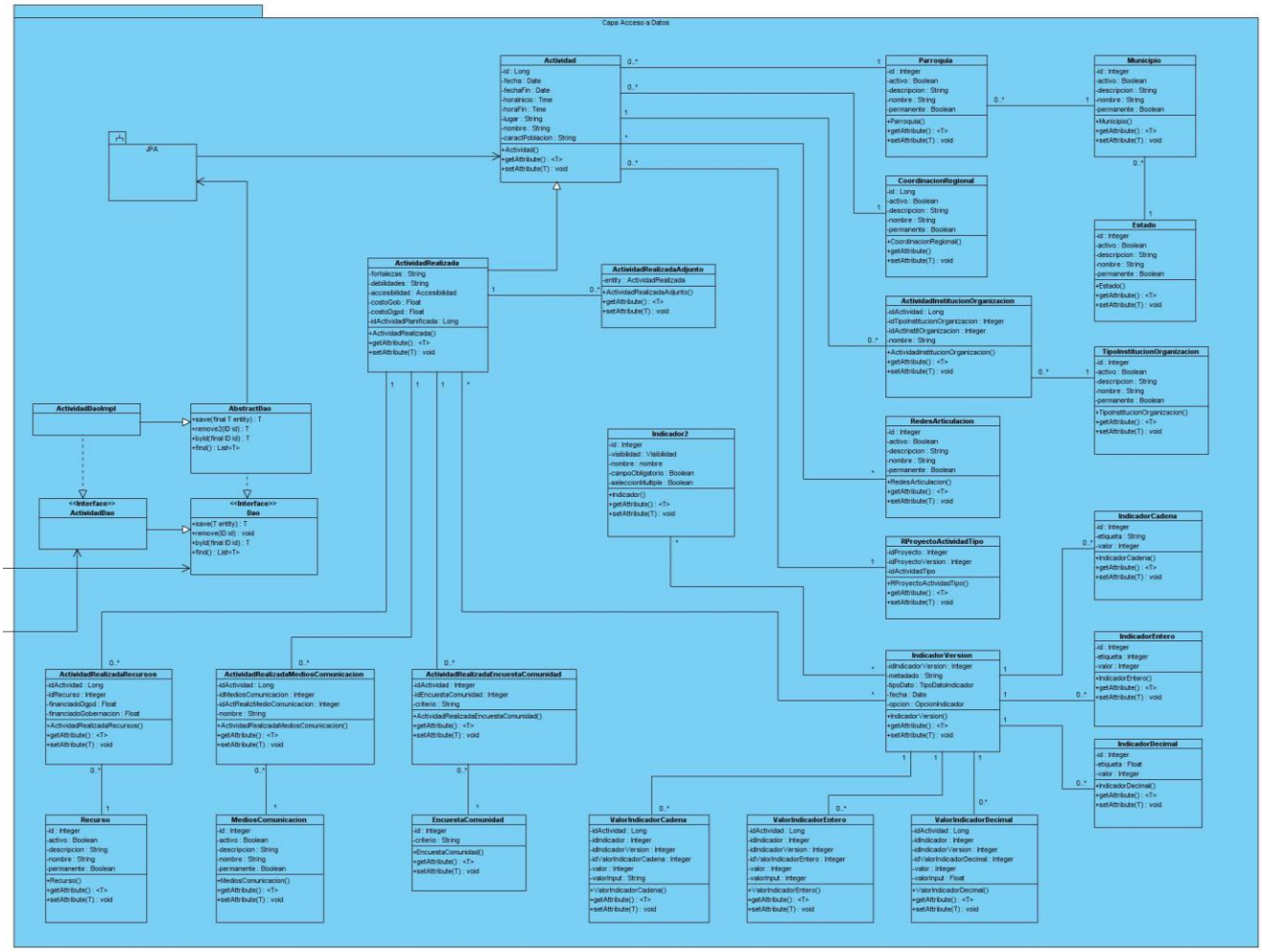

Figura 13. Diagrama de clases del diseño para el caso de uso Gestionar Actividad Realizada: Capa de Acceso a Datos.

2.6 Diagramas de Secuencia

Se muestran a continuación los diagramas de secuencia correspondientes al caso de uso Gestionar Actividad Realizada. En los escenarios de Registrar Actividad Realizada (**Figura 14**) y Actualizar Actividad Realizada (**Figura 15**). Es válido destacar que los restantes diagramas de secuencia pertenecientes a los Módulos Actividades y Administración de Programas se encuentran en el Expediente de proyecto del Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito.

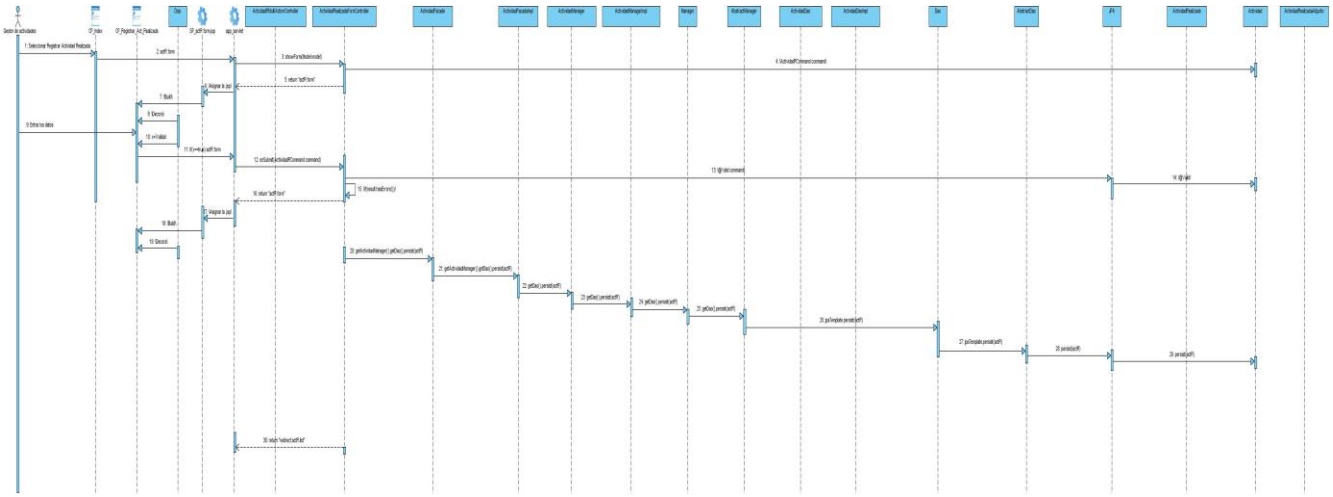

Figura 14. Diagrama de secuencia del caso de uso Gestionar Actividad Realizada escenario: Registrar Actividad Realizada.

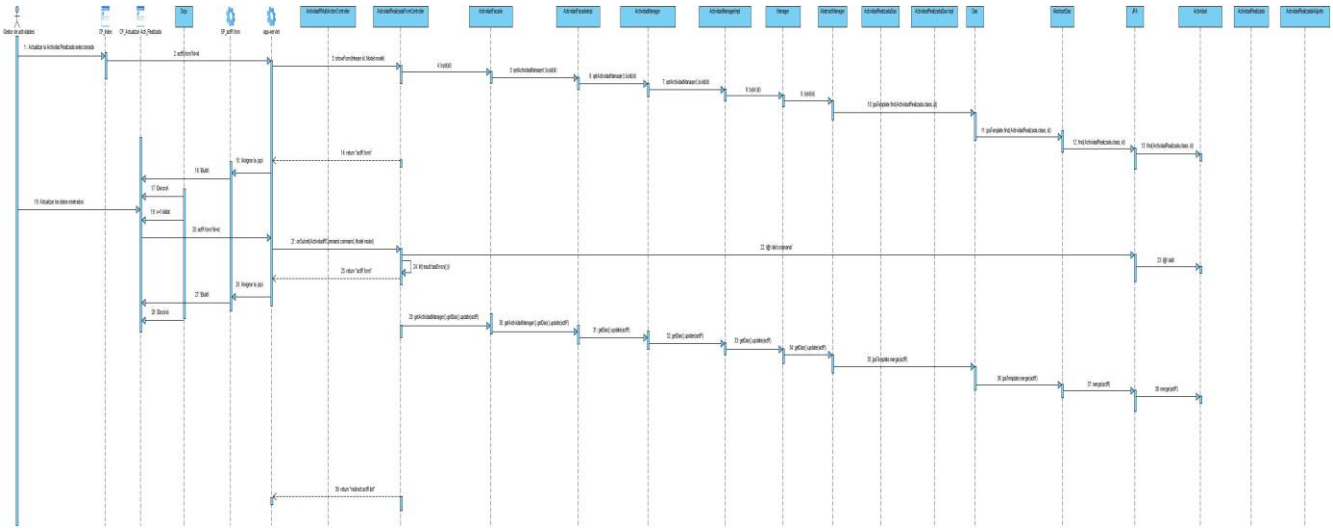

Figura 15. Diagrama de secuencia del caso de uso Gestionar Actividad Realizada escenario: Actualizar Actividad Realizada.

2.7 Conclusiones

En el capítulo se especificó el estilo arquitectónico del sistema, donde se describió el uso de estilo en Capas (3 capas). Se pudieron ver además los diferentes patrones de diseño empleados y la aplicación de los mismos. Se realizaron los diagramas de clases de diseño así como los diagramas de secuencia de diseño.

CAPÍTULO 3: IMPLEMENTACIÓN DEL SISTEMA.

3.1 Introducción

En este capítulo se describe cómo fue implementado el sistema. Se presenta el diagrama de componentes del caso de uso Gestionar Actividad Realizada, que muestra la distribución física de los componentes para la implementación del sistema. Se describen los estándares de codificación utilizados, así como la lógica de validación aplicada y se muestran algunas interfaces del sistema.

3.2 Diagrama de Componentes

En el flujo de trabajo de implementación se crea como artefacto el diagrama de componentes que describe los elementos físicos del sistema y sus relaciones. Los componentes representan todos los tipos de elementos de software que entran en la fabricación de aplicaciones informáticas. Estos pueden ser simples archivos, paquetes, bibliotecas cargadas dinámicamente, entre otros. A continuación **(Figura 16)** se muestra el diagrama de componentes del caso de uso Gestionar Actividad Realizada.

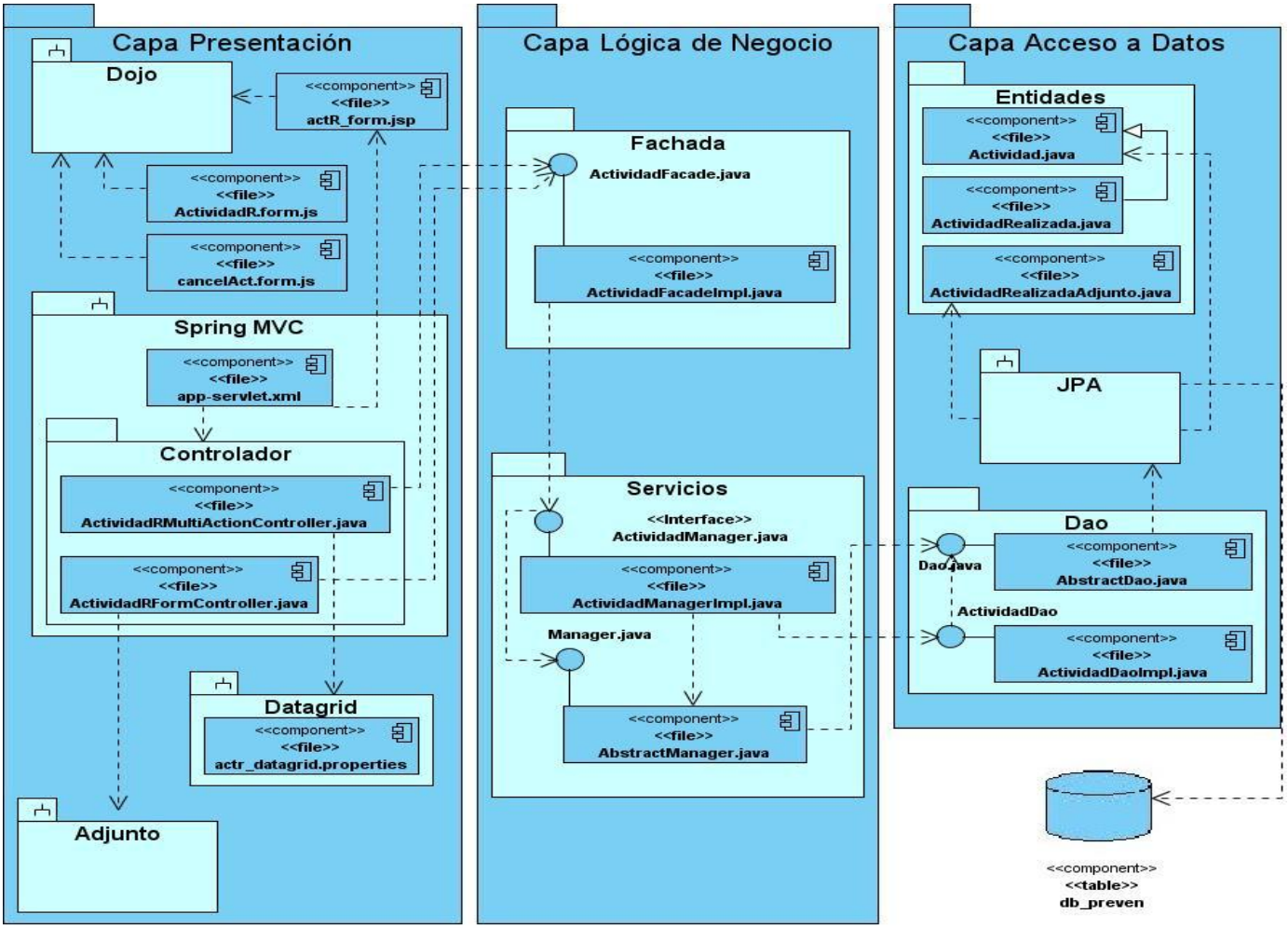

Figura 16. Diagrama de componentes del caso de uso Gestionar Actividad Realizada.

3.3 Diagrama de Despliegue

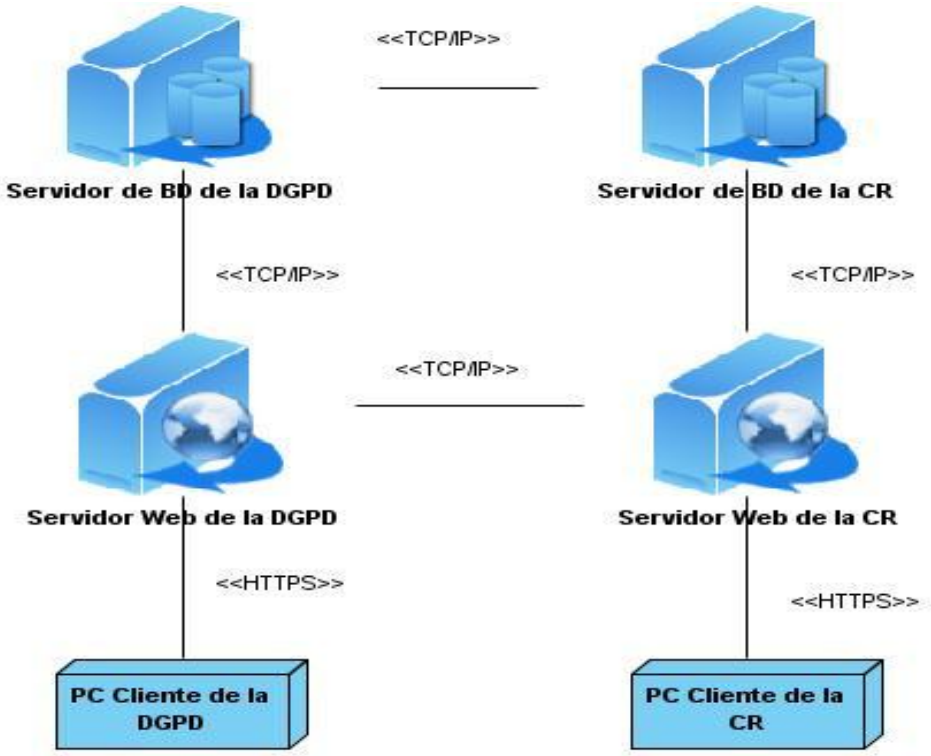

Figura 17. Diagrama de Despliegue del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela.

El diagrama de despliegue da una vista de cómo quedará distribuido físicamente el sistema implementado, logrando de esta manera situar los componentes lógicos desarrollados en el hardware que lo contiene, de esta forma el diagrama de componentes correspondiente al Caso de Uso Gestionar Actividad Realizada anteriormente presentado quedaría distribuido de la siguiente manera:

- **Servidor de BD de la DGPD:** Estará situada la base de datos (db_preven).
- **Servidor Web de la DGPD:** Estarán situados los subsistemas Spring, DataGrid y JPA y además los componentes pertenecientes a los paquetes:
  - JSP: actR.form.jsp.
  - Controladoras: ActividadRFormController.java y ActividadRMultiActionController.java
  - Fachada: ActividadFacade.java y ActividadFacadeImpl.java
  - Servicios: ActividadManager.java, ActividadManagerImpl.java, Manager.java y AbstractManager.java
  - Dao: ActividadDao.java, ActividadDaoImpl.java, Dao.java y AbstractDao.java

- Entidades: Actividad.java, ActividadRealizada.java y ActividadRealizadaAdjunto.java

- **PC Cliente de la DPCN y la PC Cliente de la CR:** Se ejecutará Dojo y los componentes del paquete JS (ActividadR.form.js y cancelAct.form.js).
- **Servidor de BD y Web de la CR:** Se ejecutarán los mismos componentes que en el Servidor de BD de la DGPD y el Servidor Web de la DPGD.

3.4 Código Fuente

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

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

Estándares de Codificación

Un estándar de codificación comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Debe ser una guía por la que todos los desarrolladores deben seguir su trabajo de forma tal que sea comprensible por todos. Es de gran importancia que el desarrollo de cualquier aplicación se guíe por algún estilo de codificación, ya que cualquier otra persona debería ser capaz de leer el código y entender su funcionamiento. En la implementación del Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela. Módulos: Actividades y Administración de programas, se siguieron una serie de estándares que se describen a continuación.

- **Paquetes:**

El prefijo de un nombre de paquete único se escribe siempre en letras minúsculas y debería ser uno de los nombres de dominio de nivel superior (actualmente com, edu, gov, mil, net, org ó uno de los códigos de país de dos letras, como se especifica en el estándar ISO 3166). Los siguientes componentes del nombre del paquete varían de acuerdo a las propias convenciones de nombrado internas de las organizaciones. Dichas convenciones pueden especificar que ciertos componentes de nombre de directorio sean división, departamento, proyecto, máquina o nombres de usuario. 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 **act** para hacer referencia al módulo Actividades y **admp** para hacer referencia al módulo Administración de Programas.

Figura 18. Estructura del proyecto.

Dentro de la aplicación cada módulo representa un paquete con un nombre característico que identifica el mismo. Los ficheros, dentro de cada módulo, se agruparan por paquetes igualmente, donde cada paquete se corresponderá con cada una de las capas a implementar. La capa de presentación estará agrupada en la capeta **web** del módulo, en el caso de la capa de lógica de negocio, las interfaces estarán agrupadas en las carpetas **service** y **facade** respectivamente, las clases que implementan estas interfaces estarán en las subcarpetas **impl**. Las clases que representan la capa de acceso a datos estarán ubicadas en la carpeta **domain**, las interfaces estarán en la carpeta **dao** y las clases que implementan estas interfaces estarán en la subcarpeta **impl**, los archivos de configuraciones se tendrán en la carpeta **config**.

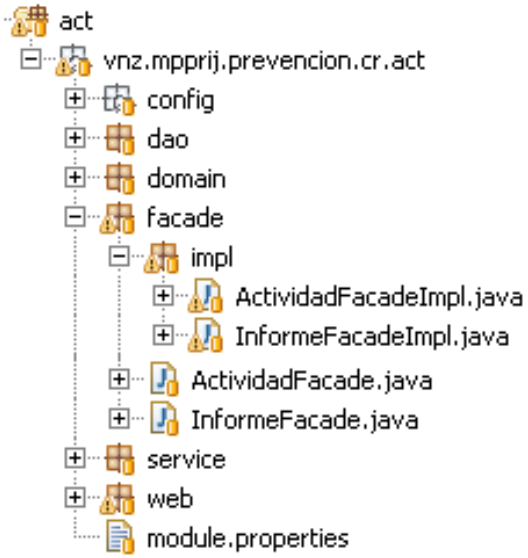

Figura 19. Estructura del los módulos.

- **Clases:**

Los nombres de clases deberían ser sustantivos, escritos con la primera letra en mayúscula. Tratar de mantener los nombres de clases simples y que describan su utilidad. Usar palabras completas,

evitar acrónimos y abreviaturas (a menos que la abreviatura sea mucho más usada que la forma larga, como URL o HTML).

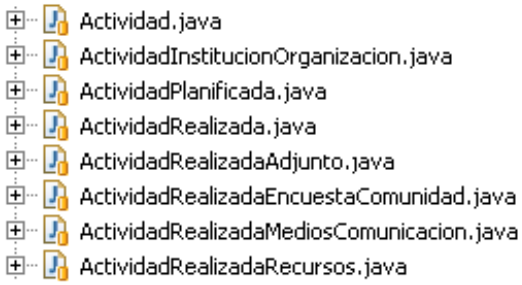

Figura 20. Nombre de las clases.

- **Métodos**

Los nombres de los métodos deberían ser verbos, escritos con la primera letra en minúscula y deben describir la utilidad que van tener.

```
Programa registrarPrograma(Programa programa);
void actualizarPrograma(Programa programa);
Programa visualizarPrograma(Integer idPrograma);
List<Programa>listadoProgramas();
```

Figura 21. Nombre de los métodos.

- **Variables:**

Todos los nombres de variable deben estar escritos con la primera letra en minúscula. No deberían comenzar con un símbolo de subrayado (_) o un signo de dólar ($), aunque ambos están permitidos. Los nombres de variables deberían ser cortos aunque significativos. La elección de un nombre de variable debería ser pensada para indicar la intención de su uso a un posible observador ocasional.

```
public class Actividad implements Serializable {

    private Long id;
    private Date fecha;
    private Date fechaFin;
    private Time horaFin;
    private Time horaInicio;
    private String lugar;
    private String nombre;
    private String caractPoblacion;
```

Figura 22. Declaración de variables.

Ejemplo de Código Fuente

A continuación se muestra un fragmento de código del controlador **ActividadRFormController** donde se ve el uso de la anotación **@Controller**, dando comportamiento de Controlador a la clase. Para solucionar la dependencia de ésta con la clase **ActividadFacade** se le inyecta mediante la anotación **@Resource** esta instancia, igualmente sucede con la clase **AttachmentEngine** lo que en este caso se hace con la anotación @**Autowired**.

```
@Controller
@SessionAttributes("act_actr_command")
public class ActividadRFormController{

    private ActividadFacade actividadFacade;

    @Autowired
    private AttachmentEngine attachment;

    @Resource
    public void setActividadFacade(ActividadFacade actividadFacade) {
        this.actividadFacade = actividadFacade;
    }

    public ActividadFacade getActividadFacade() {
        return actividadFacade;
    }
```

Figura 23. Ejemplo de código fuente.

3.5 Validación

La correcta validación de los datos de entrada es una tarea fundamental para el correcto funcionamiento de una aplicación. Muchas veces la validación se realiza desde la capa de presentación hasta la capa de acceso a datos, lo que trae como consecuencias que a menudo la misma lógica de validación se aplica en cada capa, empleándose mucho tiempo en su implementación y es propenso a errores. A manera de hacerse necesario evitar la duplicación de estas validaciones en cada capa, se aplicó la lógica de validación directamente en las entidades del dominio. Para este proceso se utiliza Hibernate Validator que implementa el API JSR-303 Bean Validation para la plataforma Java, posibilitando una validación declarativa a través de anotaciones que se hacen cumplir en tiempo de ejecución. Spring contiene la anotación @Valid que se integra con Hibernate Validator garantizando la validación del lado del Servidor en cualquiera de las capas. Además, contiene clases para el manejo de los formularios las cuales se pueden comunicar con el dominio anotado, ventaja que fue utilizada para generar todas las validaciones del lado del Cliente de forma dinámica.

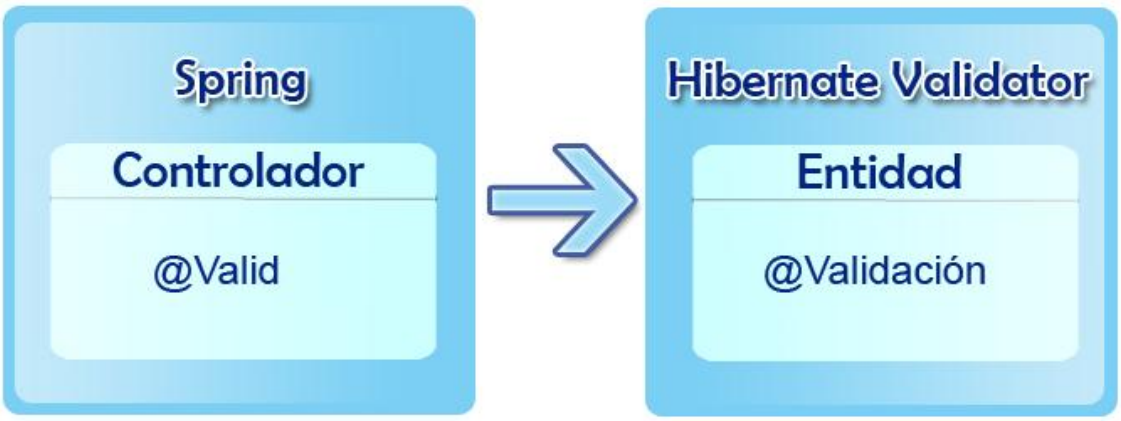

Figura 24. Lógica de validación aplicada en el sistema.

3.6 Principales Interfaces de los Módulos

A continuación se muestran las principales interfaces de los caso de uso Gestionar Actividad Realizada y Gestionar Listado de Actividades Realizadas, correspondientes al módulo Actividades del Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela.

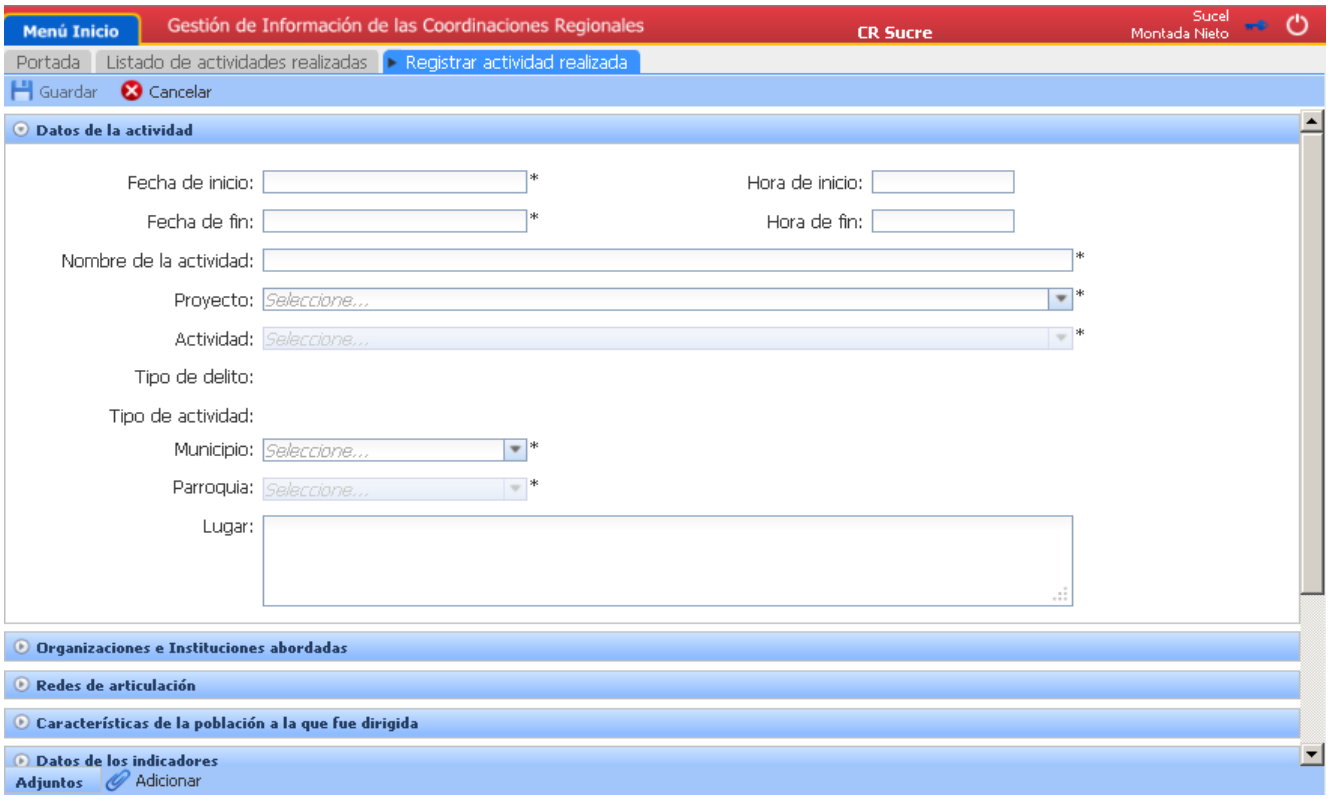

Figura 25. Interfaz registrar actividad realizada.

Figura 26. Interfaz actualizar actividad realizada.

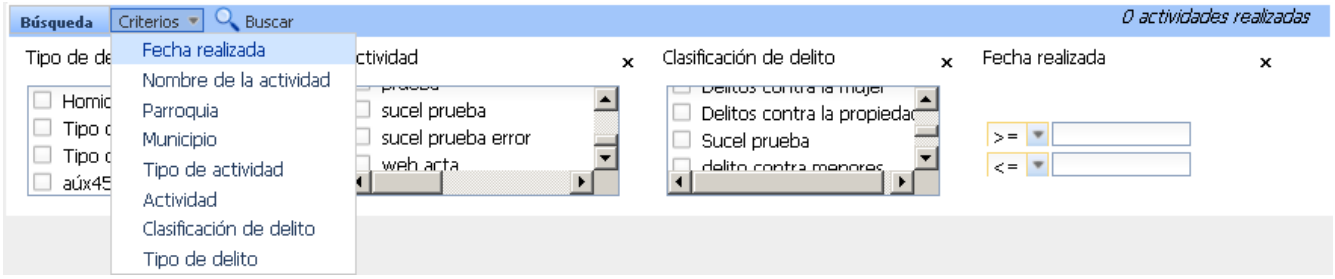

Figura 27. Interfaz buscar actividad realizada.

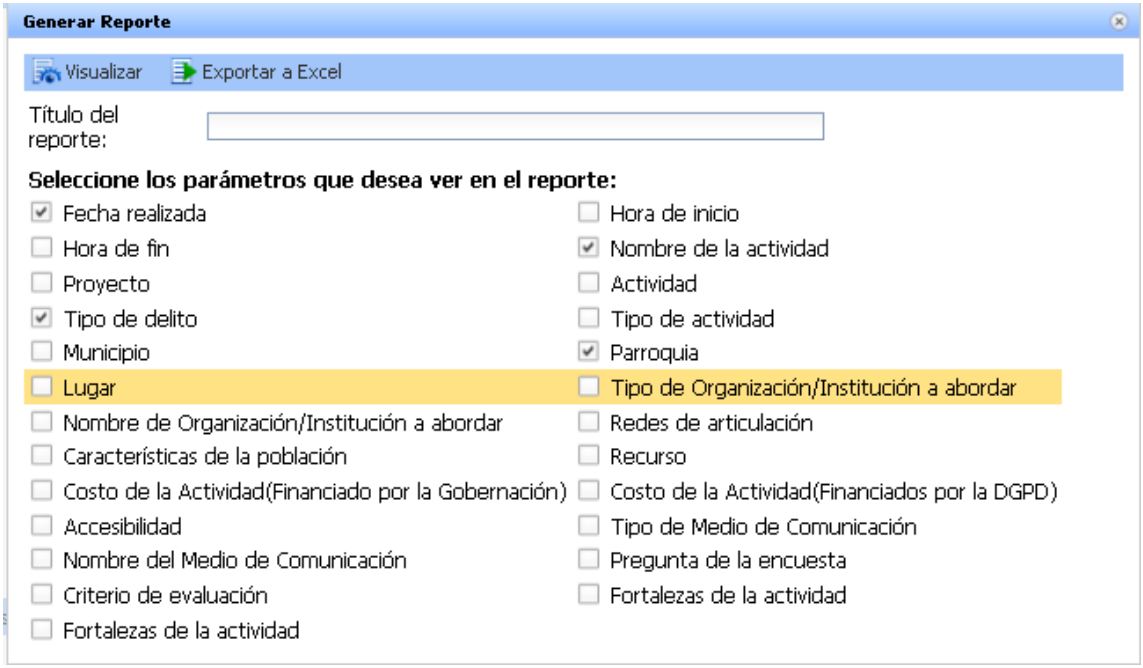

Figura 28. Interfaz generar reporte de actividades realizadas.

Figura 29. Listado de actividades realizadas.

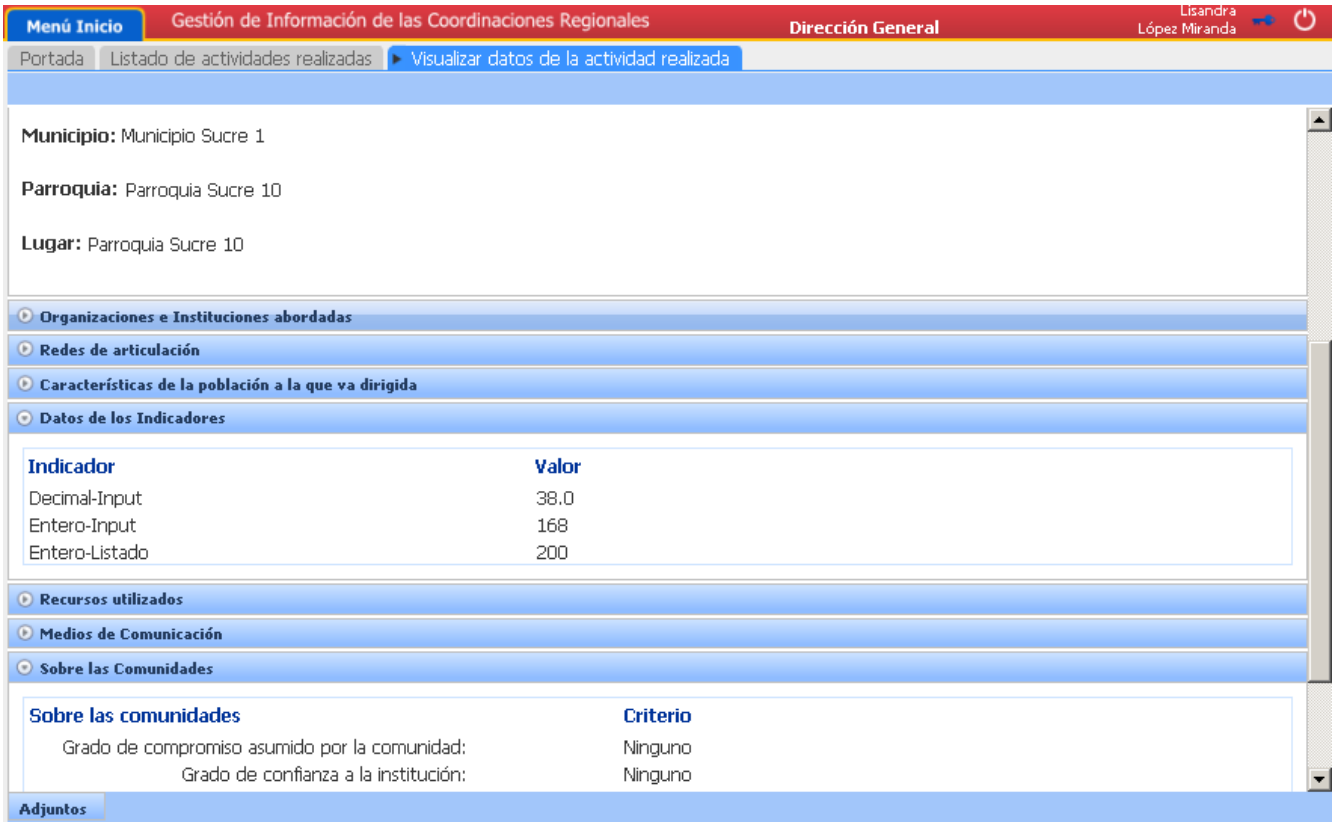

Figura 30. Interfaz visualizar actividad realizada.

3.7 Conclusiones

En este capítulo se presentó el diagrama de componentes correspondiente al caso de uso Gestionar Actividad Realizada, donde se ven descritos los elementos físicos del sistema y sus relaciones. Se hace referencia a los estándares de codificación utilizados, donde queda reflejada la organización de los ficheros en la aplicación, así como un ejemplo del código fuente de uno de los controladores donde se ve el uso de anotaciones, principal recurso utilizado en cada uno de los módulos para dar el comportamiento adecuado a cada uno de los ficheros. Se describe además en este capítulo la lógica de validación aplicada para la correcta interpretación de los datos de entrada y se muestran las principales interfaces de los caso de uso Gestionar Actividad Realizada y Gestionar listado de Actividades Realizadas del módulo Actividades.

CAPÍTULO 4: PRUEBAS DEL SISTEMA.

4.1 Introducción

En el desarrollo del capítulo de pruebas se hace referencia a los niveles de prueba aplicados a los diferentes escenarios de trabajo, a los tipos de pruebas realizadas a los módulos, se exponen los resultados de las no conformidades detectadas internamente en el proyecto a los módulos implementados así como el resumen de las no conformidades detectadas en el proceso de liberación de Calisoft[2]. Se identifican las herramientas de automatización de las pruebas del sistema. Se monitorea el progreso de las mismas y el resultado en cada ciclo de pruebas.

4.2 Pruebas de Software

Una vez generado el código fuente, es necesario probar el software para descubrir y corregir la mayor cantidad de errores posible antes de entregarlo al cliente. Para lograr contribuir con la calidad del software es recomendable que el producto software 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.

De manera concreta se puede definir pruebas de software como una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones específicas, los resultados son observados, registrados y una evaluación es hecha de algún aspecto del sistema o componente. [16]

4.3 Niveles de Pruebas

Las pruebas se aplican para diferentes tipos de objetivos, en diferentes escenarios o niveles de trabajo. Se distinguen los siguientes niveles de pruebas:

- **Pruebas de Desarrollador**

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

- **Pruebas al Sistema**

[2] **Calisoft:** Centro de Calidad para soluciones tecnológicas. La Dirección de Calidad de Software de la UCI es un centro de referencia de calidad y órgano de certificación de software, conocido y acreditado a nivel nacional. Cuenta con especialistas altamente calificados y un alto por ciento de ellos certificados internacionalmente, con un laboratorio certificado según las normas ISO/IEC 17025 y con todos los procesos definidos e institucionalizados.

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

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

- **Pruebas de Unidad**

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

4.4 Tipos de Pruebas

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

4.4.1 Pruebas de Funcionalidad

Este tipo de pruebas se realiza con el propósito de verificar el cumplimiento de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados.

Pruebas de Caja Negra

Dentro de las pruebas funcionales se realiza el método de Caja Negra, prueba que se lleva a cabo sobre la interfaz del software, y es completamente indiferente el comportamiento interno y la estructura del programa. Una prueba de este tipo examina algún aspecto funcional de un sistema que tiene poca relación con la estructura interna del software. El objetivo es demostrar que las funciones del software son operativas, que las entradas se aceptan de forma adecuada y se produce un resultado correcto, y que la integridad de la información externa se mantiene.

Casos de Prueba

Un caso de prueba se diseña según las funcionalidades descritas en los casos de usos. Este diseño se elabora previo a la realización de las pruebas funcionales de la aplicación. Cada planilla de caso de prueba recoge la especificación de un caso de uso, dividido en secciones y escenarios, para hacer más fructífera la ejecución de las pruebas.

Los casos de prueba de la caja negra pretende demostrar que:

- Las funciones del software son operativas.
- La entrada se acepta de forma adecuada.
- Se produce una salida correcta.
- La integridad de la información externa se mantiene.

Se derivan conjuntos de condiciones de entrada que ejerciten completamente todos los requerimientos funcionales del programa.

La prueba de la caja negra intenta encontrar errores de las siguientes categorías:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

No Conformidades

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

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

Tabla 4.4.1.1 Resumen de NC del proceso de liberación interno.

Módulo	Total NC	Proceden	No Proceden	Resueltas
Actividades	37	37	0	37
Administración de Programas	41	39	2	39

Tabla 4.4.1.2 Resumen de NC del proceso de liberación de Calisoft. Primera iteración.

Iteración 1				
Módulo	**Total NC**	**Proceden**	**No Proceden**	**Resueltas**
Actividades	28	28	0	28
Administración de Programas	12	12	0	12

Tabla 4.4.1.3 Resumen de NC del proceso de liberación de Calisoft. Segunda iteración.

Iteración 2				
Módulo	**Total NC**	**Proceden**	**No Proceden**	**Resueltas**
Actividades	9	9	0	9
Administración de Programas	8	8	0	8

Tabla 4.4.1.4 Resumen de NC del proceso de liberación de Calisoft. Tercera iteración.

Iteración 3				
Módulo	**Total NC**	**Proceden**	**No Proceden**	**Resueltas**
Administración de Programas	9	2	7	2

En la siguiente tabla se expone un resumen donde se clasifican las No Conformidades detectadas por Calisoft según los tipos de errores:

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

Tabla 4.4.1.5 Clasificación de las no conformidades según los tipos de errores. Primera iteración.

MÓDULO	F	V	CA	EI	FO	R	ET	E	O	S
Actividades	20	-	3	2	-	-	1	1	-	1
Programas	3	2	-	2	-	4	1	-	-	-

Tabla 4.4.1.6 Clasificación de las no conformidades según los tipos de errores. Segunda iteración.

MÓDULO	F	V	CA	EI	FO	R	ET	E	O
Actividades	3	3	-	1	-	-	-	-	2
Programas	4	-	-	1	-	1	-	-	2

Tabla 4.4.1.7 Clasificación de las no conformidades según los tipos de errores. Tercera iteración.

MÓDULO	F	V	CA	EI	FO	R	ET	E	O
Programas	1	1	-	-	-	-	-	-	-

Las NC detectadas reflejadas en las tablas anteriores se clasifican según los tipos de errores teniendo en cuenta el impacto en su solución para el equipo de desarrollo.

CLASIFICACIÓN DE NO CONFORMIDADES SEGÚN EQUIPO DE DESARROLLO
Baja (B) -Redacción, ortografía, recomendaciones, formato.
Media (M)- Error de interfaz, seguridad, correspondencia con otro artefacto.
Alta (A)- Funcionalidad, validación, error técnico, excepción.

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

Módulo	Total de no conformidades	# de no conformidades que proceden	# de no conformidades que no proceden	# de no conformidades resueltas
ITERACIÓN 1				
Actividades	28	**(A)-**22 **(M)-**6 **(B)-**0	0	28
Programas	12	**(A)-**6 **(M)-**2 **(B)-**4	0	23
ITERACIÓN 2				
Actividades	9	**(A)-**6 **(M)-**1 **(B)-**2	0	9
Programas	8	**(A)-**4 **(M)-**1 **(B)-**3	0	8
ITERACIÓN 3				
Programas	9	**(A)-**2 **(M)-**0 **(B)-**0	7	2

4.4.2 Pruebas de Control de Acceso al Sistema

Pruebas que se centran en garantizar que los datos del destino de la prueba sólo son accesibles para los actores a los que se dirigen, o sea se realizan para garantizar que los usuarios estén restringidos a funciones específicas o su acceso esté limitado únicamente a los datos que están autorizados a acceder. En el caso del Módulo Actividades los permisos de acceso estarían distribuidos de la siguiente manera:

Tabla 4.4.2.1 Tabla de permisos de acceso al módulo Actividades.

Módulo Actividades	
Funcionalidad	**Rol/es**
➢ Cronograma de actividades planificadas en la CR. ➢ Buscar actividad. ➢ Generar reporte de actividad. ➢ Visualizar reporte de actividad. ➢ Exportar a formato Excel reporte de actividad. ➢ Visualizar actividad planificada. ➢ Imprimir actividad planificada. ➢ Listado de actividades realizadas. ➢ Buscar actividades realizadas. ➢ Generar reporte de actividad realizada. ➢ Visualizar reporte de actividad realizada. ➢ Exportar a formato Excel de actividad realizada. ➢ Listado de adjuntos de actividad realizada. ➢ Descargar actividad realizada. ➢ Visualizar actividad realizada. ➢ Imprimir actividad realizada.	**CR, Secretaria de la CR, ETP[3], Admin CR[4]**
➢ Listado de informes de cumplimiento mensual en la CR. ➢ Buscar informes de cumplimiento mensual en la CR. ➢ Generar informe de cumplimiento mensual en la CR. ➢ Informe de de cumplimiento mensual en la CR con datos. ➢ Publicar informe de cumplimiento mensual en la CR. ➢ Visualizar informe de cumplimiento mensual en la CR. ➢ Exportar a Excel el informe de cumplimiento mensual en la CR. ➢ Listado de informes trimestrales de logro en la CR. ➢ Buscar informes trimestrales de logro en la CR. ➢ Generar informe trimestral de logro en la CR. ➢ Informe trimestral de logro en la CR con datos. ➢ Publicar informe trimestrales de logro en la CR.	**CR**

[3] **ETP**: Equipo Técnico Profesional.
[4] **Admin CR:** Administrador de la Coordinación Regional.

➢ Visualizar informe trimestral de logro en la CR. ➢ Exportar a excel informe de cumplimiento trimestral en la CR.	
➢ Cronograma de actividades planificadas en la Sup. ➢ Buscar actividad planificada en la Sup. ➢ Generar reporte de actividad planificada en la Sup. ➢ Visualizar reporte de actividad planificada en la Sup. ➢ Exportar a formato Excel de actividad planificada Sup. ➢ Visualizar actividad planificada Sup. ➢ Imprimir actividad planificada Sup. ➢ Listado de actividades realizadas Sup. ➢ Buscar actividades realizadas Sup. ➢ Generar reporte de actividad realizada Sup. ➢ Visualizar reporte de actividad realizada Sup. ➢ Exportar a formato Excel de actividad realizada Sup. ➢ Visualizar actividad realizada Sup. ➢ Imprimir actividad realizada Sup.	**Sup[5](solo la de los estados que atiende), DN[6], Secretria DN**
➢ Datos para buscar informes de cumplimiento mensual de la CR en la Sup. ➢ Informe de cumplimiento mensual de la CR en la Sup con datos. ➢ Exportar a Excel el informe de cumplimiento mensual de la CR en la Sup. ➢ Emitir observaciones de Informe de cumplimiento mensual de la CR en la Sup. ➢ Cancelar una observación. ➢ Publicar Informe de cumplimiento mensual de la CR en la Sup. ➢ Exportar a Excel Informe de cumplimiento mensual de la CR en la Sup. ➢ Listado de informe de cumplimiento mensual en la Sup con datos. ➢ Buscar informe de cumplimiento mensual en la Sup con datos. ➢ Publicar informe de cumplimiento mensual en la Sup. ➢ Visualizar informe de cumplimiento mensual en la Sup. ➢ Exportar a Excel el informe de cumplimiento mensual en la Sup. ➢ Emitir observaciones del informe de cumplimiento mensual en la Sup. ➢ Listado de informes trimestrales de cumplimiento en la Sup. ➢ Buscar informes de cumplimiento trimestral en la Sup. ➢ Generar informe trimestral cumplimiento en la Sup.	**Sup**

[5] **SUP:** Supervisor.
[6] **DN:** Dirección nacional.

➢ Informe trimestral cumplimiento en la Sup con datos. ➢ Publicar informe trimestral de cumplimiento en la Sup. ➢ Visualizar informe trimestral de cumplimiento en la Sup. ➢ Exportar a Excel el informe de cumplimiento trimestral en la Sup. ➢ Emitir observaciones del informe de cumplimiento mensual en la Sup. ➢ Listado de informes de cumplimiento anual en la Sup. ➢ Generar informe de cumplimiento anual en la Sup. ➢ Informe de cumplimiento anual en la Sup con datos. ➢ Publicar informe de cumplimiento anual en la Sup. ➢ Visualizar informe de cumplimiento anual en la Sup. ➢ Exportar a excel informe de cumplimiento anual en la Sup. ➢ Emitir observaciones del informe de cumplimiento anual en la Sup. ➢ Informes trimestrales de logros de la CR de la Sup. ➢ Informes trimestrales de logros de la CR de la Sup con datos. ➢ Exportar a Excel Informes trimestrales de logros de la CR de la Sup. ➢ Listado de informes trimestrales de logro en la Sup. ➢ Buscar informes trimestrales de logro en la Sup. ➢ Emitir datos cualitativos del informe trimestral de logro en la Sup. ➢ Publicar informe trimestrales de logro en la Sup. ➢ Visualizar informe trimestral de logro en la Sup. ➢ Exportar a excel informe de cumplimiento trimestral en la Sup. ➢ Listado de informes anuales de logro en la Sup. ➢ Generar informe anual de logro en la Sup. ➢ Emitir datos cualitativos del informe anual de logro en la Sup. ➢ Publicar informe anual de logro en la Sup. ➢ Visualizar informe anual de logro en la Sup. ➢ Imprimir informe de cumplimiento anual en la Sup.	
➢ Datos para buscar informes de cumplimiento mensual de la CR en la DN. ➢ Buscar informes de cumplimiento mensual de la CR en la DN. ➢ Informe de cumplimiento mensual de la CR en la DN con datos. ➢ Exportar a Excel el informe de cumplimiento mensual de la CR en la DN. ➢ Listado de informe de cumplimiento mensual en la DN. ➢ Buscar informe de cumplimiento mensual en la DN con datos. ➢ Visualizar informe de cumplimiento mensual en la DN. ➢ Exportar a excel el informe de cumplimiento mensual en la DN.	**DN, secretaria DN**

➢ Emitir observaciones del informe de cumplimiento mensual en la DN. ➢ Listado de informes trimestrales de cumplimiento en la DN. ➢ Buscar informes de cumplimiento trimestral en la DN. ➢ Visualizar informe trimestral de cumplimiento en la DN. ➢ Exportar a excel el informe de cumplimiento trimestral en la DN. ➢ Emitir observaciones del informe de cumplimiento trimestral en la DN. ➢ Listado de informes de cumplimiento anual en la DN. ➢ Visualizar informe de cumplimiento anual en la DN. ➢ Exportar a Excel el informe de cumplimiento anual en la DN. ➢ Emitir observaciones del informe de cumplimiento anual en la DN. ➢ Datos para buscar Informes trimestrales de logros de la CR de la DN. ➢ Informes trimestrales de logros de la CR de la DN con datos. ➢ Exportar a Excel Informes trimestrales de logros de la CR de la DN. ➢ Listado de informes trimestrales de logro en la DN. ➢ Emitir datos cualitativos del informe trimestral de logro en la DN. ➢ Visualizar informe trimestral de logro en la DN. ➢ Exportar a Excel el informe de cumplimiento trimestral en la DN. ➢ Visualizar leyenda del indicador. ➢ Listado de informes anuales de logro en la DN. ➢ Emitir datos cualitativos del informe anual de logro en la DN. ➢ Visualizar informe anual de logro en la DN. ➢ Exportar a formato Excel el informe de cumplimiento anual en la DN. ➢ Visualizar leyenda del indicador.	

4.4.3 Pruebas de Confiabilidad

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

4.4.4 Prueba de Estrés

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

Se trata de un tipo de prueba de fiabilidad que se centra en la evaluación de cómo responde el sistema en circunstancias anormales. Las tensiones del sistema pueden ser cargas de trabajo extremas, memoria insuficiente, servicios y hardware no disponible o recursos compartidos limitados. Estas pruebas suelen realizarse para saber mejor cómo y en qué áreas fallará el sistema, de forma que se puedan planificar y presupuestar los planes de contingencia y el mantenimiento de las actualizaciones con bastante antelación. [17]

4.4.5 Pruebas de Rendimiento

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

4.4.6 Pruebas Unitarias

El objetivo con el que se realizan estas pruebas es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el trozo de código debe satisfacer. Facilitan el cambio de código para mejorar estructura así asegurarse de que los nuevos cambios no han introducido errores, simplifica la integración, separación de la interfaz y la implementación, se puede cambiar cualquiera de los dos sin afectar al otro y los errores están más acotados y son más fáciles de localizar: dado que tenemos pruebas unitarias que pueden desenmascararlos.

4.4.7 Pruebas Exploratorias

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

4.4.8 Pruebas de Regresión

Éste tipo de pruebas se realiza para asegurar que cuando una NC encontrada en el sistema ha sido corregida ninguna de las funcionalidades liberadas previamente falla como resultado de las correcciones o que las características nuevamente agregadas no han creado conflicto con las versiones anteriores del software, es una manera de darle seguimiento y tratamiento a las NC. Estas se registran en el documento de NC. El objetivo de las pruebas de regresión es no tener que “volver atrás”.

4.4.9 Pruebas de Carga

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

Se trata de un tipo de prueba de rendimiento que se utiliza para validar y evaluar la aceptabilidad de los límites operativos de un sistema bajo cargas de trabajo variables, mientras el sistema que se está probando permanece igual. En algunas variantes, la carga de trabajo permanece igual y se modifica la configuración del sistema que se está probando. Las medidas suelen tomarse en función del rendimiento de la carga de trabajo y el tiempo de respuesta de las transacciones en línea. Las variaciones de la carga de trabajo suelen incluir la emulación del pico y el promedio de cargas de trabajo que se producen dentro de la tolerancia operativa normal. [17]

4.5 Herramientas para automatizar las Pruebas

4.5.1 JUnit

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

Confección de las pruebas:

Las pruebas se realizaron a las funcionalidades que se encuentran en las diferentes capas que se definieron en la arquitectura del sistema. Se probaron las clases del paquete **facade** y **service** pertenecientes a la capa de Lógica de Negocio; y las clases del paquete **dao** perteneciente a la capa Acceso a Datos.

Análisis de los resultados:

Una vez hecho el análisis de los resultados arrojados por las pruebas realizadas con la herramienta JUnit, se demuestra que se reduce significativamente la cantidad y complejidad de líneas de código a medida que se prueba lo que se programa. Fueron detectados a tiempo errores en los diferentes niveles en los que se desarrollaron las pruebas, como es el caso de las clases de acceso a datos, los servicios y la fachada. Se determina que los errores en su mayoría surgían por el mapeo de las relaciones entre clases. A medida que se desarrollaban las pruebas se fueron obteniendo resultados satisfactorios, probando la correcta ejecución de las funcionalidades definidas.

Las imagen que se presenta a continuación es una muestra del Entorno de Desarrollo Integrado (IDE) Eclipse, donde se ve en tiempo de ejecución una prueba realizada con la herramienta Junit.

1: Resultado de la ejecución del método mostrado en la consola.
2: Números de métodos que se pueden ejecutar, la cantidad de errores y la traza de los fallos en caso que ocurra algún error en la ejecución del método.
3: Implementación del método a ejecutar.
4: Clase que contiene los diferentes métodos implementados para la realización de las pruebas.

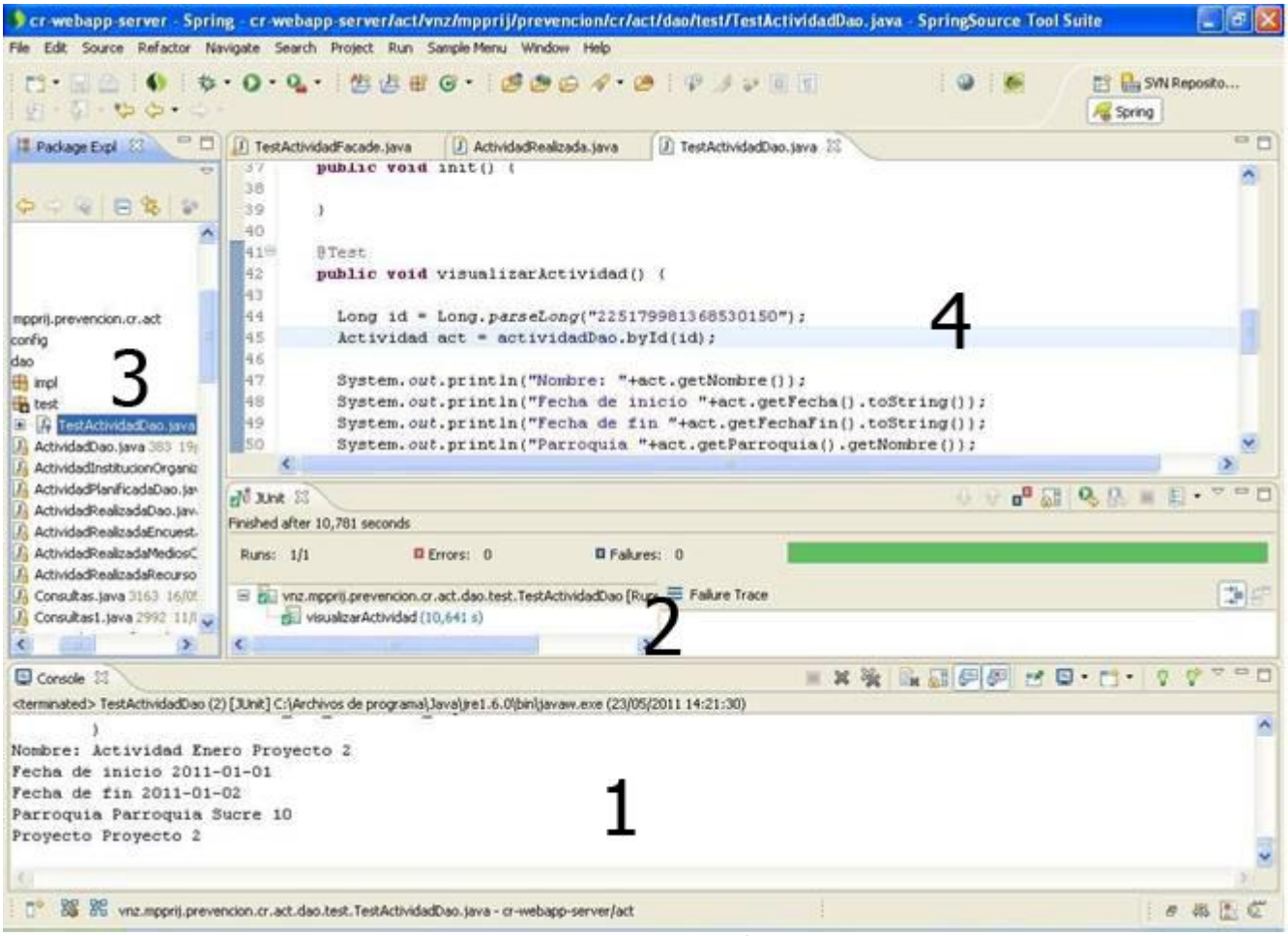

Figura 31. Prueba del dao del módulo Actividades.

4.5.2 JMeter

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

Para la ejecución de las pruebas en la aplicación se hizo una selección de los principales casos de usos del sistema donde se tuviera una representación de todas las funcionalidades del sistema distribuidas por todos sus módulos. Se utilizó script de pruebas complejas debido a las características del sistema, donde se modela la interacción del usuario con la aplicación mediante la entrada y salida de datos.

Confección de las pruebas:

Se realizaron 3 iteraciones con la aplicación corriendo en el sistema operativo Windows, probando todos los escenarios grabados de los módulos. Para una primera iteración se definieron 30 usuarios que es el mínimo de usuarios que van estar conectados a la aplicación. En una segunda iteración se simularon 100 usuarios que es la media de usuarios conectados. Finalmente, se realizó una última iteración para un total de 150 usuarios que es el máximo. En todas las iteraciones se registraron los resultados según la cantidad de usuarios a que se sometió el sistema. Las respuestas de las diferentes peticiones según la cantidad de usuarios se fueron guardando en el Informe Agregado[7]. Lo anteriormente explicado se ejecutó de igual manera sobre el sistema operativo Linux, a modo de comparar el comportamiento del software en ambos sistemas operativos.

Análisis de los resultados:

Antes de comenzar con el análisis cuantitativo de los índices obtenidos es importante especificar las condiciones de hardware en que se desarrollaron las pruebas, así como los requerimientos no funcionales de hardware y software que precisan JMeter y el sistema. Para un eficiente análisis de los resultados se deben tener en cuenta el entorno en que se ejecutaron las pruebas y el ambiente real donde se desplegará el sistema; si ambas condiciones son lo más cercanas posibles, los resultados serán satisfactorios y consecuentes. Es válido señalar que los resultados arrojados por JMeter tienen relación directa con los requisitos de rendimiento, así, con ambos datos se puede hacer una evaluación del software en términos de desempeño y con el tiempo justo, resolver las deficiencias que se encuentren en los resultados de las pruebas.

Jmeter:

- Máquina Virtual Java 1.6

Requisitos No Funcionales del Sistema de Gestión de Información de las Coordinaciones Regionales.

[7] **Informe Agregado**: Informe que se genera en la herramienta JMeter a medida que se van probando los escenarios previamente grabados.

SOFTWARE:

Servidor Web

- Sistema Operativo: Se recomienda Red Hat Enterprise Linux 4.0, aunque pudiera utilizarse otra distribución de Linux para servidores.
- Servidor web: Apache Jakarta Tomcat 6.0.20.

PC cliente

- Navegador: para un funcionamiento óptimo del Sistema de Gestión se recomienda: Mozilla Firefox 3.5 o superior, Google Chrome 3.0 o superior.

HARDWARE:

Servidor Web

- Procesador Intel Pentium Xeon, 4 GHz, de 4 Gb de RAM.
- 30 GB de HD.
- NIC: 10/100/1000 bit Ethernet controller.

PC Cliente

- Procesador Intel Pentium IV, 2 GHz, de 256 Mb de RAM
- 1 GB de HD disponible para caché del navegador (sin incluir el requerido para el sistema operativo).
- Conectividad con el servidor correspondiente

Condiciones de la PC:

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

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

Descripción de los aspectos que se muestran en las tablas con los resultados del Informe Agregado.

Muestras: Cantidad de páginas (Hilos) que simulan la cantidad de usuarios que están interactuando con el sistema desde la misma URL.

Media: Media de tiempo en milisegundos en que las páginas que se cargaron de manera satisfactoria.

Mediana: Tiempo promedio que han tardado en cargarse las páginas.

Min: Tiempo mínimo que ha demorado en cargarse una página.

Max: Tiempo máximo que ha tardado en cargarse una página.

Línea 90 %: 90 por ciento del tiempo en el que las páginas que se cargaron de manera satisfactoria.

%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.

Resumen de los resultados de la ejecución de las pruebas:

Windows

Tabla 4.5.1 Resumen del Informe Agregado obtenido para 30 usuarios.

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

Tabla 4.5.2 Resumen del Informe Agregado obtenido para 100 usuarios.

# Muestras	Media	Mediana	Línea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
7700	80	45	143	0	2513	0.00%	784.8/sec	7710416.9

Tabla 4.5.3 Resumen del Informe Agregado obtenido para 150 usuarios.

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

Linux

Tabla 4.5.4 Resumen del Informe Agregado obtenido para 30 usuarios.

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

Tabla 4.5.5 Resumen del Informe Agregado obtenido para 100 usuarios.

# Muestras	Media	Mediana	Línea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
7700	77	54	136	1	729	0.00%	1036.3/sec	10174360.7

Tabla 4.5.6 Resumen del Informe Agregado obtenido para 150 usuarios.

# Muestras	Media	Mediana	Línea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
11555	153	51	437	0	1509	0.00%	471.3/sec	4627162.7

Como se observa en las tablas anteriormente expuestas luego de la ejecución de las pruebas, se puede llegar a la conclusión que los resultados obtenidos son satisfactorios. A medida que se aumentó la cantidad de usuarios concurrentes se elevó el número de peticiones, lo que se corresponde con un incremento en el tiempo de respuesta. El % de error es muy bajo, lo que significa que la mayoría de las páginas se cargaron satisfactoriamente. Se puede concluir que el rendimiento del software sobre el sistema operativo Linux es mejor que sobre Windows. También se puede verificar que para una sobrecarga de usuarios concurrentes el sistema disminuye su rendimiento, por tanto se recomienda que no se deban conectar simultáneamente al sistema más de 150 usuarios, y en caso de ser así, las condiciones deben ser mejores.

Conclusiones

En este capítulo se abordan los niveles de pruebas, los tipos de pruebas realizadas para validar la completitud de respuesta a los requerimientos de los módulos. Dentro de las pruebas funcionales se expone una muestra de los resultados de las No Conformidades (NC) obtenidos para contabilizar a modo de resumen el total de NC de los módulos: Actividades y Administración de Programas en el proceso de liberación interno y en el proceso de liberación de Calisoft. Se abordan además las principales herramientas utilizadas para automatizar las pruebas del sistema.

CONCLUSIONES

El desarrollo de los módulos Actividades y Administración de Programas del Sistema de Gestión de Información de las Coordinaciones Regionales de Prevención del Delito de la República Bolivariana de Venezuela, a partir del diseño de clases que cumplen con estándares y patrones, la implementación basada en la utilización de frameworks y la realización de pruebas que garantizan el cumplimiento de las funcionalidades definidas, posibilita a los especialistas a partir de la información gestionada en dichos módulos identificar fortalezas y debilidades de las actividades realizadas y proyectar su labor preventiva en las comunidades.

RECOMENDACIONES

1. En la generación de los indicadores dinámicos en la interfaz de “Registrar actividad realizada” se recomienda profundizar en el estudio de los TagLibs de Spring los cuales facilitan la generación de forma dinámica de código html y de esta forma optimizar el procedimiento que genera dichos indicadores.

2. Continuar con el desarrollo de los componentes realizados en Dojo para los listados que se encuentran en las interfaces de los módulos profundizando en el trabajo con el DataGrid de Dojo.

3. Se recomienda el estudio y aplicación de Reflection para facilitar el desarrollo del sistema, disminuyendo la cantidad de código y la reusabilidad del mismo. El API Reflection es una herramienta muy poderosa que nos permite realizar en Java cosas que en otros lenguajes es imposible. Sin embargo, y a pesar de su potencial, es un API bastante desconocido, sobre todo para los principiantes en el mundo Java.

REFERENCIAS BIBLIOGRÁFICAS

1. Programa Integral de Gestión e Investigación para la Prevención del Delito. 2010. [En línea][Citado el: 1 de Febrero del 2011]. Disponible en: http://www.presidencia.gob.mx/programas/seguridad/?contenido=35019

2. La Legislatura de la Ciudad Autónoma de Buenos Aires. 2011. [En línea] [Citado en: 28 enero del 2001]. Disponible en http://www.cedom.gov.ar/es/legislacion/normas/leyes/ley2593.html.

3. Ivar Jacobson, Grady Booch, Jim Rumbaugh. El Proceso Unificado de Desarrollo de Software. [En línea] [Citado el: 26 de octubre del 2010].

4. Grady Booch, Jim Rumbaugh, Ivar Jacobson. UML El Lenguaje Unificado de Modelado. [En línea] [Citado el: 3 de Diciembre del 2010]. Disponible en: http://elvex.ugr.es/decsai/java/pdf/3E-UML.pdf

5. Ciberaula International Training. 2010. [En línea] [Citado el: 15 de Noviembre del 2010]. Disponible en: http://www.ciberaula.com/curso/java2

6. SOA agenda. 2010. [En línea] [Citado el: 16 de Noviembre del 2010]. Disponible en: http://soaagenda.com/journal/articulos/que-son-los-frameworks

7. JBoss Community. Hibernate. [En línea] [Citado el: 15 de Noviembre del 2010].Disponible en: http://hibernate.bluemars.net

8. Raúl Eduardo Chavarría. IDE Eclipse. 2006. [En línea] [Citado el: 30 de Noviembre del 2010]. Disponible en: http://www.slideshare.net/Benedeti/ide-eclipse-breve-gua-201399

9. PostgreSQL-es. Sobre PostgreSQL. 2010. [En línea] [Citado el: 18 de Noviembre del 2010].Disponible en: http://www.postgresql-es.org/sobre_postgresql

10. Introducción a la Arquitectura de Software Marzo de 2004 Carlos Billy Reynoso[Citado el: 5 de Febrero del 2011].Disponible en: http://www.willydev.net/descargas/prev/IntroArq.pdf

11. Osorio, Eneisy y Asdrúbal. Propuesta de arquitectura para el Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito. UCI. Ciudad de la Habana, 2010. Capítulos 2 y 3, Tesis.

12. Piattini Velthuis, Mario. Metodologías de desarrollo de software. 1996. [En línea] [Citado el: 10 de Noviembre del 2010]. Disponible en: http://www.itba.edu.ar/archivos/secciones/c19-icie99-ingenieriasoftwareeducativo.pdf

13. Metodologías Ágiles en el Desarrollo de Software, Taller realizado en el marco de las VIII Jornadas de Ingeniería del Software y Bases de Datos, JISBD. 2003. [En línea] [Citado el: 11 de Noviembre del 2010]. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.4553&rep=rep1&type=pdf#page=9

14. Nicolás Tedeschi. 2010. [Citado el: 6 de febrero de 2011]. Disponible en: http://msdn.microsoft.com/es-es/library/bb972240.aspx

15. Vicente, Raúl, Inyección de dependencias con Spring. http://www.programania.net/diseno-de-software/inyeccion-de-dependencias-con-spring/

16. César Javier Acuña. Pruebas de Software. Ingeniería del Software I Universidad Rey Juan Carlos.

17. Ayuda extendida de Rational Rose Enterprise Edition.

BIBLIOGRAFÍA

1. Visual Paradigm Company. Visual Paradigm. Disponible en: www.visual-paradigm.com
2. Visual Paradigm Company. Installing Visual Paradigm for UML. Disponible en: 206.222.18.10/media/documents/vpuml60ig/html/Ch01_Installing_VP-UML/Ch01_Installing_VP-UML.html
3. Universidad Politécnica de Valencia. Introducción a RUP. Disponible en: pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Introducci%C3%B3n%20a%20RUP.doc
4. Apache Software Foundation. Disponible en: tomcat.apache.org
5. BREIDENBACH, C. W. W. R. Spring in Action. Second Edition. Editado por: Manning. 2008. In Action.
6. MULLER, R. J. W. J. Expert One on one J2EE Development Without EJB. Wiley, 2004.
7. KAISLER, S. H. Software Paradigms. Wiley, 2005.
8. Johson, Rod. Professional Java Development with the Spring Framework. 2005.
9. Ayuda extendida de Rational Rose Enterprise Edition.
10. Hernández Aguilar, V., & González Jorrín, M. Proceso de pruebas de caja negra basado en la descripción de los casos de uso.
11. Lores Sánchez, L., & Monné Roque, D. Aplicación de las pruebas de liberación al Sistema Informático.
12. Juristo, N., M. Moreno, A., & Vegas, S. (17 de octubre de 2006). TÉCNICAS DE EVALUACIÓN DE SOFTWARE.
13. Osorio, Eneisy y Asdrúbal. Propuesta de arquitectura para el Sistema de Gestión de Información para las Coordinaciones Regionales de Prevención del Delito. UCI. Ciudad de la Habana, 2010.
14. Spring-Framework-Reference [PDF]
15. Framework Dojo [En línea] Disponible en: http://www.dojotoolkit.org/
16. hibernate.org [En línea] Disponible en: Disponible en: http://www.hibernate.org/
17. Group, P.G.D. PostgreSQL. 2007 [En línea] Disponible en: http://www.postgresql.org/.
18. jasperforge.org Sitio Web Oficial de JasperReport [En línea] Disponible en: http://jasperforge.org/website/.
19. Hibernate Validator JSR 303 Reference Implementation Reference Guide 4.0.1.GA [PDF]
20. Montada Aguilar, F., & Cordero Domínguez, O. Sistema Informático de Gestión de Información de las Comunidades, los Consejos Comunales, Citas y Denuncias en las Coordinaciones Regionales.

21. Barban Otaño, W., & Alonso Llerena, D. Sistema Informático para la Gestión de Información de los Recursos Materiales y Programas- Proyectos de las Coordinaciones Regionales de Prevención del Delito.