Universidad de las Ciencias Informáticas

Facultad 3

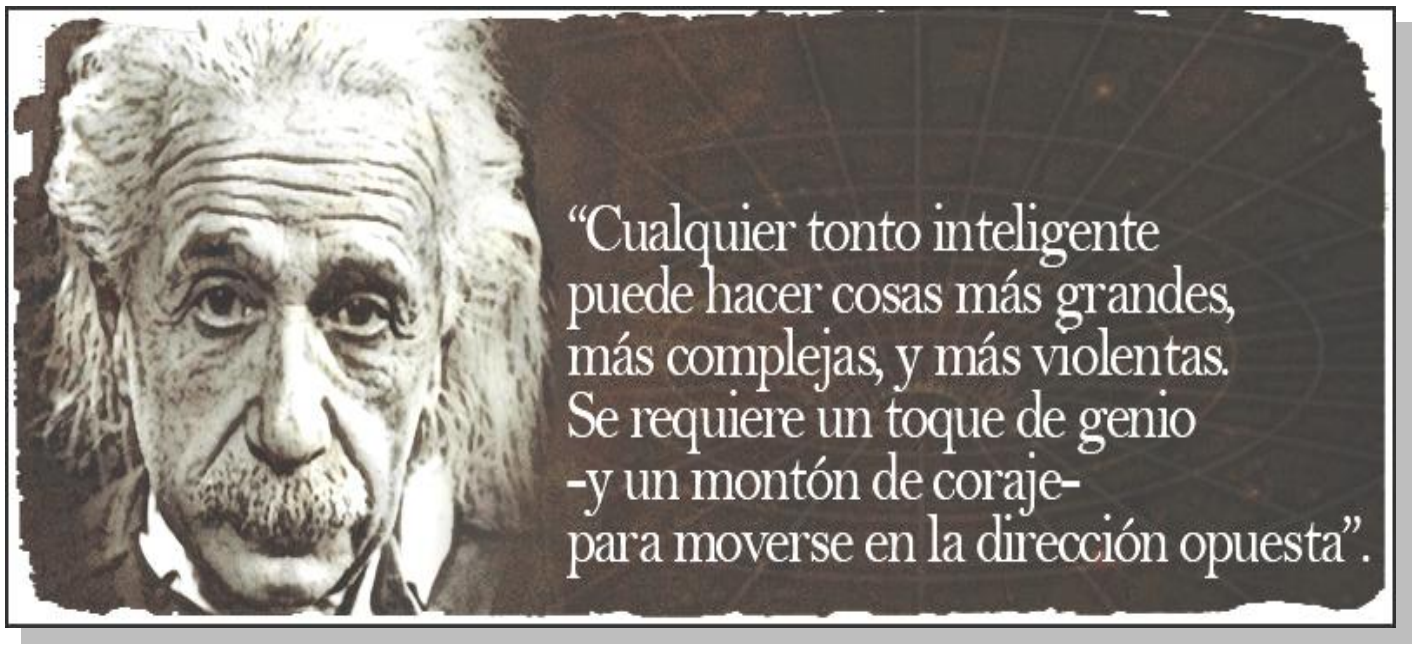
Título: Diseño e Implementación de un módulo de integración para la Solución Tecnológica Integral para la Modernización de la División de Antecedentes Penales.

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

Autor: Carlos Alejandro Suárez García

Tutor: Ing. Julio Cesar Prieto Alvarez

Ciudad de la Habana, Junio 2011



Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Carlos Alejandro Suárez García

_____

Firma del Autor

Ing. Julio César Prieto Álvarez

_____

Firma del Tutor

AGRADECIMIENTOS

Quisiera agradecer a:

- *Mis padres, por sacrificarse toda su vida para convertir a este hijo en Ingeniero.*
- *Mis hermanos, por quererme siempre y apoyarme en todo.*
- *Mis abuelos, por ser mi madre y padre, mi abuela y mi abuelo, y partes fundamentales de mi vida.*
- *Ileana por ser mi otra mamá, a mis tíos y tías que siempre me ayudan en todo, en fin a toda mi familia.*
- *Mis amigos de la infancia que han estado junto a mí en los buenos y malos momentos.*
- *Mi novia por ayudarme en todo. Por ser mi tutora, oponente y mi compañera de tesis.*
- *Mis amigos de la universidad por ayudarme siempre en todo lo que pudieron y estar siempre a mi lado.*
- *Mis profesores de la facultad III por siempre apoyarme y ayudarme en todo. Especialmente a Julio, Lissuan y Eliober.*
- *A todo aquel que confío en mí y me apoyó en algún momento: Gracias.*

Carlos.

DEDICATORIA

A mi mamá, por comprenderme y apoyarme siempre en todo. Mami gracias por enseñarme el verdadero significado de una madre.

A mi papá por inculcarme los principios humanos que rigen mi vida. Gracias por estar siempre a mi lado cuando te necesito.

A mis hermanos, para que sigan mis pasos y se conviertan en intelectuales. Para que sepan que son lo que más quiero en mi vida.

A mis abuelitos queridos, que siempre están al tanto de todo y apoyándome en lo que sea.

A mi novia Claudia, por su amor, dedicación y resistencia. Gracias por ser mi amiga, mi compañera y por haber confiado en mí todo este tiempo.

En fin a toda mi familia, que siempre estuvieron al tanto de todo, apoyándome en todo momento.

RESUMEN

Producto del convenio entre Cuba y la República Bolivariana de Venezuela, en la décima mixta, se firma el contrato Solución Tecnológica Integral para la Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela. Dicha solución comprende dos subsistemas Centro de Digitalización y Sistema de Gestión de Antecedentes Penales, los cuales debido al negocio que presentan poseen elementos en común, por lo que es necesario desarrollar un Módulo de Integración que facilite el desarrollo de los dos subsistemas y sea capaz de gestionar la información que necesita el Centro de Digitalización y que genera la Solución de Gestión de Antecedentes Penales sin tener que depender directamente de este último. Para dar solución a esta problemática se realiza este trabajo de diploma cuyo objetivo principal es lograr el diseño e implementación de un módulo de integración entre los subsistemas Centro de Digitalización y Solución de Gestión de Antecedentes Penales. Su desarrollo estará basado en tecnologías libres, usando como estilos arquitectónicos el Cliente-Servidor, el N-Capas y el Basado en Componentes. Se utilizó Java como lenguaje de programación orientada a objetos, Glassfish como servidor de aplicaciones, PostgreSQL como gestor de base de datos y NetBeans como Entorno de Desarrollo, utilizando Swing como librería de componentes visuales. El proceso de desarrollo estuvo guiado por el Proceso Unificado de Desarrollo. Finalmente la solución propuesta fue sometida a pruebas a través de mecanismos de validación establecidos, obteniéndose resultados satisfactorios.

ÍNDICE

INTRODUCCIÓN

En los últimos años Las Tecnologías de la Información y las Comunicaciones han alcanzado un notable desarrollo en las distintas ramas de la sociedad, facilitando el acceso y almacenamiento de grandes volúmenes de información, la obtención de resultados de forma rápida y la digitalización de documentos, ventajas que hacen de las TIC el motor impulsor de la sociedad actual.

Una de las esferas donde las Tecnologías de la Información y las Comunicaciones se han ido incorporando progresivamente es la gubernamental, introduciendo el término de Gobierno Electrónico o gobierno-e (conocido por sus siglas en inglés como e-government). El objetivo fundamental del gobierno electrónico es conseguir una administración más eficaz del gobierno, mediante la transparencia y el acceso público a la información, así como la mejora de las relaciones entre sectores públicos y los ciudadanos.

Una de las aristas del Gobierno Electrónico que se ha desarrollado es la Informática Jurídica, definida como el conjunto de tecnologías aplicadas a la sistematización y automatización de la información jurídica. Esta a su vez se divide en varias ramas, teniendo entre sus principales la documental y la de gestión. En lo que a la rama documental se refiere, existe un predominio de digitalización de documentos como forma de preservación del fondo documental. La última se ocupa de facilitar mediante la informatización, la labor de las oficinas de atención al público, entre las que se encuentran fiscalías, tribunales y otros entes públicos, permitiendo de éste modo que se efectúe el control de los trámites que a diario se realizan y de la generación automática de consultas o informes relativos a la información que se gestiona.

Debido al impacto que tiene en la sociedad actual el Gobierno Electrónico, se crea en la Universidad de las Ciencias Informáticas el Centro de Gobierno Electrónico (CEGEL), con el objetivo de desarrollar soluciones informáticas vinculadas a esta rama. Entre estas soluciones se encuentran: el Sistema de Gestión de los Tribunales Populares de la República de Cuba, el Sistema de Gestión para Seguimiento y Control de los Proyectos del Convenio Integral de Cooperación Cuba-Venezuela, Automatización y Modernización de Registros y Notarías Públicas de la República Bolivariana de Venezuela y Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela.

INTRODUCCIÓN

El proyecto Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela tiene como objetivo la construcción de un software que permita la informatización y la mejora de los procesos que se llevan a cabo en la División de Antecedentes Penales. Dentro de estos procesos se encuentran la gestión de los servicios de inscripción de sentencias definitivamente firmes, gestión de los datos de los sancionados y la certificación de antecedentes penales, además de la información al estado y a las instituciones delimitadas por la ley. Este software está formado por dos subsistemas principales, Centro de Digitalización y Solución de Gestión de Antecedentes Penales, que trabajan simultáneamente en dicha institución.

La Solución de Gestión se enmarca principalmente en realizar los procesos de inscripción de sentencias definitivamente firmes y emisión de certificaciones de antecedentes penales, los cuales se llevan a cabo diariamente en la División de Antecedentes Penales. Además de esto, el subsistema tiene como objetivo permitir la gestión de los recursos humanos en vistas a la producción, asignarlos a las diferentes áreas o estaciones de trabajo, entre otras.

El Centro de Digitalización por su parte es el encargado de la digitalización del fondo documental de la División desde sus inicios hasta el momento de comenzar la explotación de la Solución de gestión de antecedentes penales. Este cuenta con varias áreas principales que permiten llevar a cabo la digitalización de los expedientes de antecedentes penales existentes en la División de Antecedentes Penales.

Sin embargo estos subsistemas tienen subprocesos comunes que se llevan a cabo de manera simultánea, teniendo en cuenta que trabajan con el mismo tipo de información y responden a una única estructura administrativa. Además no se cuenta con un mecanismo que permita agrupar componentes comunes que fomenten la reusabilidad entre los subsistemas, y que a su vez atienda los subprocesos del Centro de Digitalización que deben realizarse a través del subsistema Solución de Gestión. La situación anteriormente expuesta conlleva al siguiente **problema a resolver**, que sirve de directriz a la presente investigación. ¿Cómo lograr la integración entre los subsistemas Centro de Digitalización y Solución de Gestión de Antecedentes Penales del proyecto Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela?

La ciencia que estudia dicho fenómeno es el Proceso de Desarrollo de Software, por lo que se le considera como el **objeto de estudio** de esta investigación. Elaboración de las fases de Diseño e

Implementación de módulos de software constituye el **campo de acción** por ser esta el área, dentro de dicha ciencia, donde se va a centrar la investigación.

Este trabajo persigue el siguiente **objetivo**: Diseñar e implementar un módulo de integración entre los subsistemas Centro de Digitalización y Solución de Gestión de Antecedentes Penales del proyecto Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela. Los **objetivos específicos** son los siguientes:

- Elaborar el marco teórico de la investigación.
- Construir el módulo de Integración.
- Validar la propuesta a partir de métodos definidos internacionalmente.

Con esta investigación se defiende la siguiente **idea a defender**: Con la existencia de un módulo de integración entre los subsistemas de Centro de Digitalización y Solución de Gestión de Antecedentes Penales, se podrá realizar una mejor gestión de la información que se maneja en la División de Antecedentes Penales de la República Bolivariana de Venezuela.

Para lograr el cumplimiento de los objetivos propuestos se trazaron un grupo de **tareas investigativas**, estas son:

- Estudio del arte sobre mecanismos de integración entre subsistemas.
- Desarrollo del modelo de diseño del módulo.
- Desarrollo del modelo de implementación del módulo.
- Validación del diseño de la solución propuesta.
- Realización de pruebas de caja negra a la solución propuesta.

En el presente trabajo se hace uso de varios métodos científicos de la investigación por la importancia de estos en el desarrollo de la misma. A continuación se hace referencia a ellos:

Métodos teóricos.

- *Inducción-Deducción:* Permitió arribar a conclusiones para la determinación del problema.
- *Analítico-Sintético:* Permitió realizar un estudio exhaustivo del objeto para de esta forma extraer los elementos más importantes y arribar a conclusiones dentro de la investigación que fomentaron el objetivo.

- *Hipotético-Deductivo:* Desempeñó un papel esencial en el proceso de verificación de la idea que se defiende, ya que a partir de esta se llegaron a nuevas conclusiones y predicciones que se fomentaron con la culminación de la investigación.
- *Histórico-Lógico:* Se utilizó para analizar la trayectoria completa del objeto, su condicionamiento a los diferentes períodos de su historia, así como las etapas principales de su desenvolvimiento. Este método fue de vital importancia para la determinación del estado del arte.

Métodos empíricos.

- *Observación:* Es utilizado en todo el proceso de la investigación ya que este es el instrumento universal del científico, se realiza de forma consciente y orientada a un objetivo determinado. Este método se utilizó para obtener una caracterización detallada de las soluciones existentes para sistemas similares.

La presente investigación está estructurada de la siguiente forma:

Capítulo I Fundamentación Teórica: En este capítulo se abordan elementos importantes sobre los métodos para la integración de subsistemas. Se describe la metodología, las tecnologías, las herramientas y el lenguaje a utilizar para el desarrollo de la aplicación.

Capítulo II Características del sistema. En este capítulo se presenta el diseño del sistema que se propone. Se describe y muestra el diagrama de clases del diseño con la descripción de cada una de sus clases para una mejor organización, sus diagramas de interacción y el modelo de datos.

Capítulo III Implementación y Pruebas. En este capítulo quedan establecidos las interfaces del sistema, así como el diagrama de despliegue, los diagramas de componente, que conformarán el Modelo de Implementación, y la descripción detallada de los paquetes de implementación. Así mismo se exponen las pruebas realizadas a la solución para asegurar el correcto funcionamiento de la aplicación.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

En el presente capítulo se detallan los principales elementos teóricos tenidos en cuenta para dar solución al problema anteriormente descrita. Se hace un análisis de las estrategias fundamentales para la integración de sistemas, así como sus principales requisitos. Además se describen las tecnologías, lenguajes, metodologías de desarrollo y herramientas que pudieran ser utilizadas para dar solución a la problemática existente.

1.1 Integración entre sistemas

Producto al gran desarrollo tecnológico del mundo actual la mayoría de las empresas e instituciones han sido informatizadas, automatizando la mayoría de sus procesos mediante uno o varios sistemas informáticos que administran la información generada a través de los mismos. En algunos casos surge la necesidad de intercambiar o reutilizar la información que manejan individualmente cada uno de estos sistemas, pero en su mayoría son interoperables entre sí, por lo que se hace necesario realizar una integración entre ellos.

Cuando de integración se habla es fundamental analizar tres requisitos fundamentales: autonomía, heterogeneidad y distribución, a continuación se describen brevemente. [1]

Autonomía: indica hasta que punto los sistemas de información pueden operar independientemente sobre los datos que maneja. Existen cuatro tipos de autonomía:

- *Autonomía de diseño:* Las bases de datos locales pueden elegir su propio modelo de datos (cómo se estructura la información), lenguaje de interrogación, restricciones, interpretación semántica de los datos, qué operaciones o funciones soporta, etc. Esta es la principal causa de la heterogeneidad entre los distintos sistemas.
- *Autonomía de comunicación:* Las bases de datos locales tienen el poder de decidir cuándo y cómo responder a las peticiones de información procedentes de otras bases de datos.
- *Autonomía de ejecución:* Las bases de datos locales controlan el orden de ejecución de las transacciones u operaciones independientemente del tipo de éstas: locales o externas.

- *Autonomía de asociación:* las bases de datos locales pueden decidir que datos comparten con determinados usuarios.

Heterogeneidad: Desarrollo e implantación de manera independiente y aislada de cada sistema. La heterogeneidad puede ocurrir a cualquier nivel del sistema de base de datos. A un nivel técnico la heterogeneidad es debida a la utilización de diversos tipos de hardware, sistemas de información, lenguajes de programación o sistemas gestores de bases de datos. Por otro lado, en el nivel conceptual la heterogeneidad aparece cuando se utilizan diversos modelos de datos para representar la información o cuando existen discrepancias con respecto al significado de los datos (heterogeneidad semántica), por ejemplo, cuando se utiliza el mismo nombre para representar diversos conceptos o el uso de nombres diferentes para detonar el mismo concepto. Posiblemente en cualquier proyecto de integración de sistemas de información la superación de la heterogeneidad entre los diversos sistemas es la tarea más difícil. Obviamente, cuanto más diferentes sean los sistemas a integrar mayor es la dificultad. Las técnicas empleadas con mayor frecuencia para superar este problema son la utilización de modelos de datos comunes y estándares específicos, que pueden ser útiles para definir el significado de la información cuando esta debe ser compartida entre diversos sistemas.

Distribución: División de las bases de datos en uno o múltiples sistemas informáticos, que a su vez pueden estar en un único o en múltiples lugares (departamentos, organizaciones o incluso países) pero están interconectados por un sistema de comunicación. Los datos pueden estar distribuidos en diversas bases de datos de varias formas. Estas incluyen, desde un punto de vista relacional, particiones verticales y horizontales. Además pueden existir varias copias de la misma información (replicación) [2].

Disímiles son los métodos usados para llevar a cabo la integración entre sistemas, debido a la diversidad tecnológica y autonomía de estos. Uno de los más utilizados es el empleo de los middleware[1], software que se sitúa entre los usuarios finales y los sistemas existentes, que integra la información creada individualmente por estos y la presenta en un formato unificado, ocultando los detalles sobre la localización y el formato de los datos. Otro de los métodos más comunes es la Integración mediante bases de datos. Este esquema de integración se basa en que múltiples aplicaciones accedan a bases de datos comunes. Las aplicaciones intercambian datos accediendo a la base de datos común. Este enfoque es

[1] Middleware: Software intermediario.

viable debido a la existencia de un lenguaje estándar universal para el acceso a base de datos (SQL, Structured Query Lenguaje). Las estrategias más utilizadas basadas en este enfoque son:

- Creación de una base de datos con información de otras bases de datos. Esto facilita el acceso a los datos de múltiples aplicaciones.
- Utilización de procesos que alimentan la base de datos de una aplicación con información de otras bases de datos. Esto permite que una aplicación acceda a información de otras aplicaciones sin necesidad de ser modificada.
- Utilización de una base de datos común diseñada específicamente para ser accedida por múltiples aplicaciones.

En el caso del proyecto Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela (SIGESAP) la integración entre el Centro de Digitalización y la Solución de Gestión se realiza creando un Módulo de Integración encargado de implementar los componentes comunes entre ambos subsistemas. Este módulo a su vez gestiona la información que necesita Centro de Digitalización de la Solución de Gestión, interactuando con la base de datos de este último, de otra forma el Centro de Digitalización debería depender de que la Solución de Gestión este funcionando para poder consultar información de esta. Este módulo se desplegará con ambos subsistemas como parte de la solución de estos evitando el problema de disponibilidad antes expuesto. La figura que a continuación se muestra ilustra claramente la distribución lógica de la integración en el proyecto en cuestión.

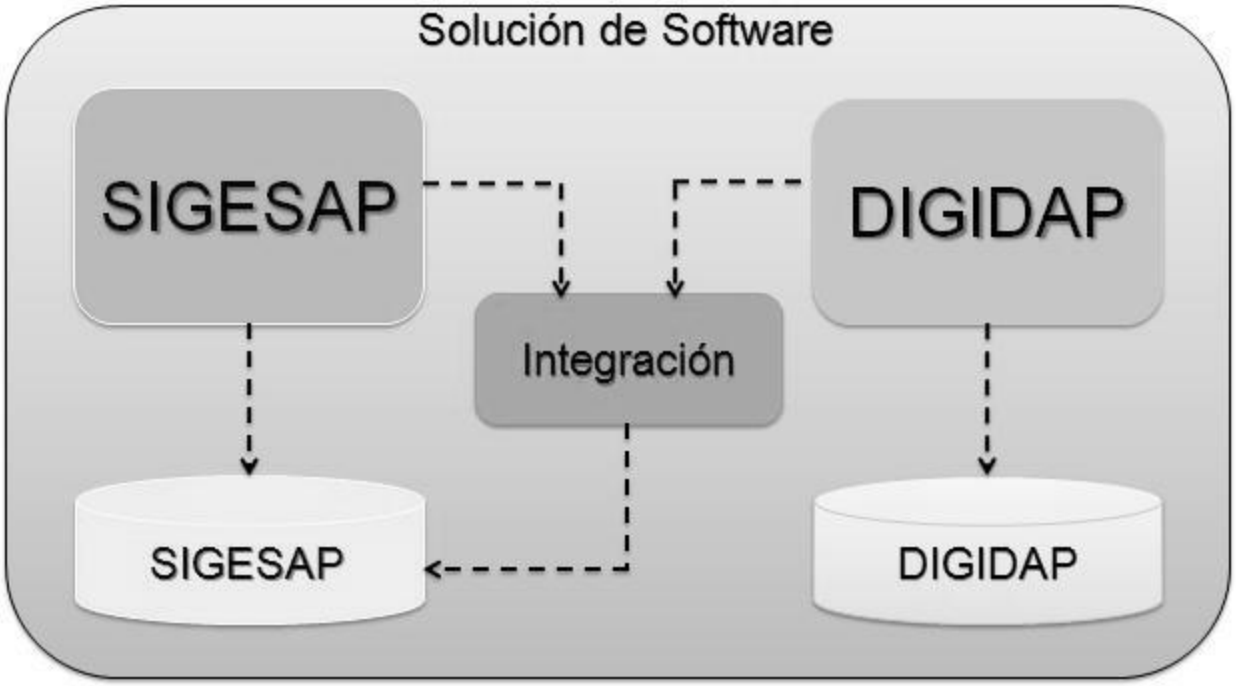


Fig. 1: Distribución Lógica de la Integración.

Por otra parte se debe analizar cómo se afectan las tres características antes mencionadas (autonomía, heterogeneidad, distribución) con la integración propuesta. La autonomía por su parte no sufre alteración, porque a pesar de realizar la integración entre ambos subsistemas estos continúan cumpliendo la funcionalidad para la que fueron creados. La heterogeneidad no es impedimento tampoco ya que ambos subsistemas son homogéneos, puesto que fueron implementados con las mismas herramientas y utilizando las mismas tecnologías. La distribución de los datos por último no constituye una barrera entre ambos subsistemas ya que estos a pesar de tener base de datos independientes manejan el mismo tipo de información. Analizadas estos tres aspectos fundamentales se concluyó que existen las condiciones indispensables para llevar a cabo la integración entre los subsistemas Centro de Digitalización y Solución de Gestión a través del Módulo de Integración.

1.2 Metodologías de desarrollo de software

Las metodologías de desarrollo de software brindan un conjunto de procedimientos, técnicas, herramientas y documentación para ayudar a los desarrolladores a realizar el software. De ahí la

importancia de escoger la metodología que se ajuste a las necesidades del sistema a desarrollar. A continuación se hace una descripción de las metodologías de desarrollo de software más conocidas.

Rational Unified Process (RUP): Es una metodología de desarrollo de software que suministra un enfoque para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad, que satisfaga la necesidad del usuario final dentro de un tiempo y presupuesto previsible. Esta metodología es una de las más utilizadas en la universidad para el desarrollo de software de gran tamaño.

RUP como metodología define quién hace qué, cómo y cuándo, definiendo cuatro elementos: trabajadores (roles), que responden a la pregunta ¿Quién?, las actividades que responden a la pregunta ¿Cómo?, los artefactos (productos), que responden a la pregunta ¿Qué? y los flujos de trabajo de las disciplinas que responden a la pregunta ¿Cuándo? [3]

Los flujos de trabajo agrupan las actividades a desarrollar, esta metodología define 9, los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como de apoyo. Estos son: Modelamiento del negocio, Requerimientos, Análisis y diseño, Implementación, Prueba (Testeo), Instalación, Administración del proyecto, Administración de configuración y cambios y Ambiente. [3]

RUP divide el proceso de desarrollo en 4 fases, en las cuales se realizan varias iteraciones de las actividades. Las fases son: [3]

- *Conceptualización (Concepción o Inicio):* Se describe el negocio y se delimita el proyecto describiendo sus alcances con la identificación de los casos de uso del sistema.
- *Elaboración:* Se define la arquitectura del sistema y se obtiene una aplicación ejecutable que responde a los casos de uso que la comprometen. A pesar de que se desarrolla a profundidad una parte del sistema, las decisiones sobre la arquitectura se hacen sobre la base de la comprensión del sistema completo y los requerimientos (funcionales y no funcionales) identificados de acuerdo al alcance definido.
- *Construcción:* Se obtiene un producto listo para su utilización que está documentado y tiene un manual de usuario. Se obtiene 1 o varios release del producto que han pasado las pruebas. Se ponen estos release a consideración de un subconjunto de usuarios.

- *Transición:* El release ya está listo para su instalación en las condiciones reales. Puede implicar reparación de errores.

Las características principales de RUP son: guiado por casos de uso, centrado en la arquitectura e iterativo e incremental.

Extreme Programing (XP): Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

Los puntos más interesantes de XP son:

- Desarrollo iterativo e incremental.
- Pruebas unitarias continuas, frecuentemente repetidas y automáticas.
- Programación en parejas.
- Frecuentemente interacción del equipo de programación con el cliente o usuario.
- Corrección de todos los errores antes de añadir nueva funcionalidad.
- Hacer entregas frecuentes.
- Refactorización del código.
- Propiedad del código compartida: promueve que todo personal pueda corregir y extender cualquier parte del proyecto.
- Simplicidad en el código

Sus objetivos:

- La satisfacción del cliente. Esta metodología trata de dar al cliente el software que él necesita y cuando lo necesita.

- Potenciar al máximo el trabajo en grupo. Tanto los jefes de proyecto, los clientes y desarrolladores, son parte del equipo y están involucrados en el desarrollo del software. [4]

Microsoft Solution Framework (MSF): Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. MSF es un compendio de las mejores prácticas en cuanto a administración de proyectos se refiere. Más que una metodología rígida de administración de proyectos, MSF es una serie de modelos que puede adaptarse a cualquier proyecto de tecnología de información

MSF tiene las siguientes características:

- *Adaptable:* es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- *Escalable:* puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.
- *Flexible:* es utilizada en el ambiente de desarrollo de cualquier cliente.
- *Tecnología Agnóstica:* porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el modelo de Aplicación.

Modelo de Arquitectura del Proyecto: Diseñado para acortar la planificación del ciclo de vida. Este modelo define las pautas para construir proyectos empresariales a través del lanzamiento de versiones.

Modelo de Equipo: Este modelo ha sido diseñado para mejorar el rendimiento del equipo de desarrollo. Proporciona una estructura flexible para organizar los equipos de un proyecto. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles.

Modelo de Proceso: Diseñado para mejorar el control del proyecto, minimizando el riesgo, y aumentar la calidad acortando el tiempo de entrega. Proporciona una estructura de pautas a seguir en el ciclo de vida

del proyecto, describiendo las fases, las actividades, la liberación de versiones y explicando su relación con el Modelo de equipo.

Modelo de Gestión del Riesgo: Diseñado para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.

Modelo de Diseño del Proceso: Diseñado para distinguir entre los objetivos empresariales y las necesidades del usuario. Proporciona un modelo centrado en el usuario para obtener un diseño eficiente y flexible a través de un enfoque iterativo. Las fases de diseño conceptual, lógico y físico proveen tres perspectivas diferentes para los tres tipos de roles: los usuarios, el equipo y los desarrolladores.

Modelo de Aplicación: Diseñado para mejorar el desarrollo, el mantenimiento y el soporte, proporciona un modelo de tres niveles para diseñar y desarrollar aplicaciones software. Los servicios utilizados en este modelo son escalables, y pueden ser usados en un solo ordenador o incluso en varios servidores. [5]

Todo proyecto está separado en cinco principales fases:

- Visión y alcance
- Planificación
- Desarrollo
- Estabilización
- Implantación

Después de describir algunas de las metodologías de desarrollo de software más relevantes, para el desarrollo de aplicaciones empresariales, se puede concluir que antes de seleccionar la que se empleará para la confección del sistema, se realizó un análisis de las características que tendrá dicho desarrollo y se seleccionó la que más se acopla a las necesidades del proyecto. Para la realización del Sistema Solución Tecnológica Integral para SIGESAP se tomó la metodología RUP. La decisión se debe principalmente a la ausencia que se tiene del cliente en las fases de Análisis y Diseño e Implementación, lo que imposibilita el uso de la metodología XP, además RUP propone un conjunto de entregables

requeridos por el cliente y otros que son de vital importancia para el equipo de proyecto. Esto ligado a la gran documentación que existe de esta metodología y sin obviar la experiencia del personal del proyecto en esta metodología.

1.3 Lenguajes

Los lenguajes son formas de comunicación que tienen los seres humanos. Constituyen un conjunto de signos, orales y escritos, que permiten la expresión y la comunicación humana.

1.3.1 Lenguajes de programación

Un lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. El mismo es utilizado para controlar el comportamiento físico y lógico de una máquina [6]

Dentro de los lenguajes de programación más utilizados en la universidad según el Informe Tecnológico de la Producción del 2010 se encuentran:

Hypertext Pre-processor (PHP): Lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. Es usado principalmente en interpretación del lado del servidor (server-side scripting) pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas incluyendo aplicaciones con interfaz gráfica. [7]

Ventajas de utilizar PHP para el desarrollo de páginas Web:

- Es un lenguaje multi-plataforma.
- Está orientado al desarrollo para aplicaciones Web dinámicas y permite el acceso a la información almacenada en una base de datos.
- No es necesario que el usuario vea el código escrito al momento de ejecutar una orden, puesto que el código fuente escrito en PHP ejecuta la instrucción en el servidor y envía al visitante solo el resultado en el navegador. Esto lo hace confiable y seguro.
- Tiene una capacidad para conectarse con la gran mayoría de los gestores de Bases de datos existentes, el más común es con MySQL.

- Permite utilizar técnicas de programación orientada a objetos, POO por sus siglas en inglés.

Desventajas de usar PHP

- Por medio del código ofuscado se ocultan errores en el código fuente.
- Cuando en PHP se utilizan otros patrones de diseño probados en otros lenguajes, por decir, en Java, quedan inutilizables en PHP, se puede producir una sobrecarga al tener que realizar un ambiente de desarrollo nuevo para cada solicitud de la página, como es autenticación, uso de permisos, etc.
- PHP hace las operaciones matemáticas rápidamente, la desventaja aquí es que tarda en desplegar el resultado al usuario.
- PHP es lento para ejecutar una función comparada con código que este en la misma línea, puesto que consume mayor cantidad de recursos.
- El lenguaje PHP está diseñado hacia una forma de realizar aplicaciones que puede resultar a veces problemático.
- Es difícil de optimizar y no posee un manejo adecuado de Unicode.
- Promueve creación de código que necesita un mantenimiento complejo y desordenado, esto puede ser difícil para quien es novato en la programación PHP, requiere muchas veces respaldo para optimizar su código.[8]

Java: Es probablemente la principal tecnología de desarrollo de software en la actualidad. Entre las principales características que han llevado a tan rápida expansión, se encuentra con que es independiente del sistema operativo y la plataforma (ordenadores personales, teléfonos móviles, PDAs, tarjetas inteligentes, etc.), estandarizado, sencillo, distribuido, robusto, seguro.

- *Independencia de la plataforma:* Java es la tecnología ideal para desarrollar aplicaciones independientes del sistema operativo (Windows, Linux, SunOS, etc.), así como en un amplio espectro de dispositivos: ordenadores personales, PDAs, teléfonos móviles, tarjetas inteligentes.

Es esta característica la que le permite ser la mejor opción en Internet (applets, páginas activas con jsp, servlets, etc.)

- *Estandarización:* A diferencia de C++ y otros lenguajes, Java es un lenguaje completamente estandarizado, de modo que el código es independiente del entorno de desarrollo elegido.
- *Sencillez:* Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos.
- *Distribuido:* Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales.
- *Seguridad:* Java es uno de los primeros lenguajes en considerar la seguridad como parte de su diseño. [9]

C Sharp (C#): La sintaxis y estructuración de C# es muy parecida a la de C++ o Java, puesto que la intención de Microsoft es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Entre las características más reconocidas de C# se encuentran:

- *Sencillez:* Al ser el lenguaje nativo de Microsoft, elimina muchos elementos que son innecesarios para .Net.
- *Modernidad:* C# incorpora en el propio lenguaje elementos que a lo largo de los años ha ido demostrándose que son muy útiles para el desarrollo de aplicaciones como la instrucción foreach.
- *Orientación a objetos:* Como todo lenguaje de programación de propósito general existente en la actualidad. En lo referente a la encapsulación C# añade un cuarto modificador llamado internal, que puede combinarse con protected e indica que al elemento a cuya definición precede sólo puede accederse desde su mismo ensamblado. Respecto a la herencia C# sólo admite herencia simple de clases.

- *Gestión automática de memoria:* C# dispone de un recolector de basura, por lo que no es necesario incluir instrucciones de destrucción de objetos.
- *Sistema de tipos unificado:* En C# todos los tipos de datos que se definan siempre derivarán, aunque sea de manera implícita, de una clase base común llamada System.Object, por lo que dispondrán de todos los miembros definidos en ésta clase.
- *Eficiente:* En principio, en C# todo el código incluye numerosas restricciones para asegurar su seguridad y no permite el uso de punteros aunque se puede violar estas restricciones mediante el modificador unsafe. [10]

C++: Es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de aplicaciones. El C++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original. [11]

Entre las características más notables se pueden mencionar:

- *Programación orientada a objetos:* Esta característica va acorde con el paradigma OO, fue uno de los primeros lenguajes que permitió programar según este paradigma.
- *Portabilidad:* Puede compilar prácticamente el mismo código C++ en casi cualquier tipo de ordenador y sistema operativo sin realizar ningún cambio, esto se debe a que muchos sistemas operativos lo usan para implementar muchas funcionalidades.
- *Programación modular:* En C++ el código puede ser compuesto por diversos archivos de código fuente que se compilan por separado y luego unidos. Permite ahorrar tiempo ya que no es necesario recompilar la aplicación completa al realizar un solo cambio.
- *Velocidad:* El código resultante de una compilación de C++ es muy eficiente, de hecho, debido a su dualidad como lenguajes de alto nivel y de bajo nivel.

Después de hacer un análisis de los lenguajes de programación más utilizados en la universidad, se llegó

a la conclusión de que el más apropiado para realizar la aplicación es Java, por integrarse con las tecnologías y herramientas que se emplearan en el desarrollo de la aplicación. El cual además de constituir el lenguaje de programación solicitado por el cliente, presenta facilidades para el escaneo y uso de firmas digitales, requerimientos básicos de la aplicación.

1.3.2 Lenguajes de modelado

Los lenguajes de modelado son un conjunto estandarizado de símbolos y sus relaciones que permiten diseñar un sistema. A continuación se define el lenguaje a utilizar para el modelado del sistema propuesto.

Lenguaje Unificado de Modelado (UML): Lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema software. Está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. El modelo UML de un sistema es similar a un modelo a escala de un edificio junto con la interpretación del artista del edificio. Es importante destacar que un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementar dicho sistema. [12]

1.4 Herramientas

1.4.1 Entorno de Desarrollo Integrado (IDE)

Eclipse: Es un entorno de desarrollo integrado de código abierto, portable y multiplataforma. Este fue diseñado originalmente por la empresa IBM y actualmente, es desarrollado por la Fundación Eclipse, una organización independiente, sin ánimo de lucro que fomenta una comunidad de código abierto.

Se basa en el uso de módulos (plugins), lo cual hace posible el trabajo en múltiples lenguajes de programación como son Java, C++, PHP, Perl y que se le puedan añadir otras funcionalidades. Mediante el SDE Enterprise edition permite la integración con la herramienta Visual Paradigm, propiciando un mejor entendimiento de todas las partes involucradas en el desarrollo del sistema. Cuenta además con un sistema de control de versiones, el cual usando una combinación de vistas y editores que muestran los diversos aspectos de los recursos del proyecto organizados por el rol o la tarea del desarrollador, hace más fácil y eficiente el trabajo en equipo.

NetBeans: Es un entorno de desarrollo integrado, una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el NetBeans IDE. [13]

Es un producto libre y gratuito sin restricciones de uso. Ofrece servicios comunes a las aplicaciones de escritorio, permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación. Entre las características de la plataforma están:

- Administración de las interfaces de usuario (ej. menús y barras de herramientas)
- Administración de las configuraciones del usuario
- Administración del almacenamiento (guardando y cargando cualquier tipo de dato)
- Administración de ventanas.
- Soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles).

Todas las funciones del IDE son provistas por módulos. Cada módulo provee una función bien definida, tales como el soporte de Java, edición o soporte para el sistema de control de versiones. NetBeans trae incluidas todas las librerías necesarias que le permiten integrarse con las herramientas y tecnologías escogidas para el desarrollo. Además cuenta con Swing, conjunto de componentes visuales, ideal para el desarrollo de aplicaciones de escritorio.

1.4.2 Herramientas CASE

Rational software: Familia de software de International Business Machine, S.A. (IBM) para el despliegue, diseño, construcción, pruebas y administración de proyectos en el proceso de desarrollo de software. Sus productos están centrados en la metodología del Proceso Unificado de Desarrollo (RUP). Entre los productos más conocidos se tienen:

- Rational Application Developer
- Rational Software Architect

- Rational Portfolio Manager
- Rational Requisite Pro
- Rational Rose Enterprise

Rational Rose Enterprise es el producto más completo de la familia Rational Rose, tiene las siguientes características: [14]

- Soporte para análisis de patrones ANSI C++, Rose J y Visual C++.
- Característica de control por separado de componentes modelo que permite una administración más granular y el uso de modelos.
- Soporte de ingeniería Forward y/o reversa para algunos de los conceptos más comunes de Java 1.5.
- La generación de código Ada, ANSI C ++, C++, CORBA, Java y Visual Basic, con capacidad de sincronización modelo- código configurables.
- Soporte Enterprise Java Beans™ 2.0
- Capacidad de análisis de calidad de código.
- El Add-In para modelado Web provee visualización, modelado y las herramientas para desarrollar aplicaciones Web.
- Modelado UML para trabajar en diseños de base de datos, con capacidad de representar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos.
- Capacidad de crear definiciones de tipo de documento XML (DTD) para el uso en la aplicación.
- Integración con otras herramientas de desarrollo de Rational.
- Publicación web y generación de informes para optimizar la comunicación dentro del equipo.

Enterprise Arquitect: Herramienta para modelar, diseñar, visualizar, construir y desplegar software. Proporciona un entorno de modelización de carácter colaborativo y potenciado mediante UML 2.1. Abarca por completo el ciclo de vida de desarrollo de software, proporcionando una trazabilidad completa desde la

fase inicial del diseño a través del despliegue y mantenimiento. También provee soporte para pruebas, mantenimiento y control de cambio.

Presenta las siguientes características:

- Crear elementos del modelo UML para un amplio alcance de objetivos.
- Ubicar esos elementos en diagramas y paquetes
- Crear conectores entre elementos
- Documentar los elementos que ha creado.
- Generar código para el software que está construyendo.
- Realizar ingeniería reversa del código existente en varios lenguajes. [15]

Visual Paradigm para UML: Es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

Lista de características:

- Soporte de UML versión 2.1
- Diagramas de Procesos de Negocio - Proceso, Decisión, Actor de negocio, Documento Modelado colaborativo con CVS y Subversion.
- Interoperabilidad con modelos UML2 (metamodelos UML 2.x para plataforma Eclipse) a través de XMI.
- Ingeniería de ida y vuelta.
- Ingeniería inversa - Código a modelo, código a diagrama.
- Ingeniería inversa Java, C++, Esquemas XML, XML,.NET exe/dll, CORBA IDL.

- Generación de código - Modelo a código, diagrama a código.
- Editor de Detalles de Casos de Uso - Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Diagramas EJB - Visualización de sistemas EJB.
- Generación de código y despliegue de EJB´s - Generación de Beans para el desarrollo y despliegue de aplicaciones.
- Diagramas de flujo de datos.
- Soporte ORM - Generación de objetos Java desde la base de datos.
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación.
- Generador de informes para generación de documentación.
- Distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML.
- Importación y exportación de ficheros XMI.
- Integración con Visio - Dibujo de diagramas UML con plantillas (stencils) de MS Visio.
- Editor de figuras.
- Plataforma Java (Windows/Linux/Mac OS X): SDE para Eclipse, SDE para NetBeans, SDE para Sun ONE,
- DE para Oracle Jdeveloper, SDE para Jbuilder, SDE para IntelliJ IDEA y SDE para WebLogic Workshop.
- Plataforma Windows: SDE para Microsoft Visual Studio [16]

Por todas las características anteriormente planteadas se escogió para la modelación del proyecto la herramienta Visual Paradigm para UML 3.4. Esta herramienta permitirá la creación de distintos diagramas útiles para que el desarrollador pueda implementar de una forma más sencilla y dinámica la propuesta de solución. Así mismo posee todos los recursos para integrarse tanto con el IDE de desarrollo como con el sistema gestor de base de datos a utilizar, permitiendo la ingeniería inversa de la base de datos y del código.

1.4.3 Sistemas de Gestión de Bases de Datos

Los Sistemas de Gestión de Bases de Datos (SGBD), son aplicaciones que permiten a los usuarios definir, crear y mantener la base de datos y proporciona un acceso controlado a la misma. El SGBD es la aplicación que interactúa con los usuarios de los programas de aplicación y la base de datos. [17]

Los principales SGBD utilizados en la universidad, teniendo en cuenta lo reflejado en el informe anteriormente mencionado, para el trabajo con las bases de datos son:

PostgreSQL: Sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones no tiene nada que envidiarle a otras bases de datos comerciales.

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.

Algunos límites de PostgreSQL son:

- Límite Valor Máximo tamaño base de dato: Ilimitado (Depende de tu sistema de almacenamiento)
- Máximo tamaño de tabla: 32 TB
- Máximo tamaño de fila: 1.6 TB
- Máximo tamaño de campo: 1 GB
- Máximo número de filas por tabla: Ilimitado

- Máximo número de columnas por tabla: 250 - 1600 (dependiendo del tipo)
- Máximo número de índices por tabla: Ilimitado [18]

Características PostgreSQL

- Objeto-Relacional.
- Alta extensibilidad. PostgreSQL soporta operadores definidos por el usuario.
- Integridad Referencial. La cual se usa para asegurar la validez de los datos de las bases de datos.
- Flexibilidad API. Esta flexibilidad ha permitido a los vendedores proveer de un soporte de desarrollo fácil para PostgreSQL RDBMS. Estas interfaces incluyen Objetos Pascal, Python, Perl, PHP, ODBC, Java/JDBC, Ruby, TCL, C/C++, y Pike.
- Lenguaje de procedimiento. PosgreSQL tiene soporte para lenguaje interno de procedimiento, que incluye un lenguaje nativo llamado PL/pgSQL. Este lenguaje es comparable al lenguaje de procedimiento de Oracle, PL/SQL. Otra ventaja de PostgreSQL es su habilidad para usar Perl, Python, o TCL como lenguaje de procedimiento insertado.
- MVCC o Control de concurrencia Multiversion. Es la tecnología que PosgreSQL usa para evitar bloqueos innecesarios. Si se ha usado otro SQL con capacidad DBMS, tal como MySQL o Access, probablemente has notado que algunas veces el lector tiene que esperar para acceder a la información en la base de datos. La espera es provocada por personas que están escribiendo en la base de datos.
- Cliente/Servidor. PosgreSQL usa una arquitectura proceso-por-cliente usuario/servidor. Esto es similar al método Apache 1.3.x para el manejo de proceso. Hay un proceso maestro que bifurca para dar concesiones adicionales a cada cliente que procura conectarse a PosgreSQL.

Ventajas y desventajas:

- Limitaciones al escribir funciones y procedimientos en comparación con Oracle's PL/SQL o Sybase's T-SQL.
- Las tablas espaciosas, tablas particionadas, y con bloqueo altamente complicado siguen siendo ofrecidas por los vendedores propietarios de bases de datos.

- Carencia de herramientas de desarrollo propia. [19]
- El costo es la principal ventajas de PostgreSQL
- La habilidad para poder mirar el código fuente y entender que está sucediendo

Oracle: Es un Sistema Gestor de Bases de Datos con características objeto-relacionales, que pertenece al modelo evolutivo de SGBD. Oracle es desarrollado y comercializado por Oracle Corporation. Sus características principales son las siguientes:

- Entorno cliente/servidor.
- Gestión de grandes bases de datos.
- Usuarios concurrentes.
- Alto rendimiento en transacciones.
- Sistemas de alta disponibilidad.
- Disponibilidad controlada de los datos de las aplicaciones.
- Adaptación a estándares de la industria, como SQL-92.
- Gestión de la seguridad.
- Autogestión de la integridad de los datos.
- Opción distribuida.
- Portabilidad.
- Compatibilidad.
- Conectabilidad.
- Replicación de entornos.

Ventajas de Oracle:

- Oracle es el motor de base de datos relacional más usado a nivel mundial.

- Puede ejecutarse en todas las plataformas, desde una PC hasta un supercomputador.
- Permite el uso de particiones para la mejora de la eficiencia, de replicación e incluso ciertas versiones admiten la administración de bases de datos distribuidas.
- Oracle es la base de datos con mas orientación hacía INTERNET.

Desventajas de Oracle:

- El mayor inconveniente de Oracle es su precio. Incluso las licencias de Personal Oracle son excesivamente caras, en mi opinión. Otro problema es la necesidad de ajustes. Un error frecuente consiste en pensar que basta instalar el Oracle en un servidor y enchufar directamente las aplicaciones clientes. Un Oracle mal configurado puede ser muy lento.
- También es elevado el coste de la formación, y sólo últimamente han comenzado a aparecer buenos libros sobre asuntos técnicos distintos de la simple instalación y administración.

El SGBD escogido para el desarrollo de la aplicación es PostgreSQL 8.4, debido a que su código fuente es distribuido libremente, presenta una buena escalabilidad siendo capaz de soportar gran cantidad de peticiones simultáneas correctamente, ideal para grandes bases de datos. Implementa el uso de rollback's, subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz. Condiciones que hacen de este sistema gestor de base de datos el apropiado para cumplir con las necesidades del cliente.

1.5 Tecnologías

A continuación se describen las tecnologías empleadas para el desarrollo del sistema propuesto.

JPA: Es la Interfaz de Programación de Aplicaciones (API) estándar para la persistencia y el mapeo objeto/relacional para la plataforma Java EE, y permite utilizar un modelo de dominio Java para administrar bases de datos relacionales.

Enterprise Java Beans (EJB): Es una arquitectura componente del lado del servidor para la plataforma Java. Permite realizar la administración automática de transacciones, seguridad, escalabilidad, concurrencia, distribución, acceso a ambientes portables y persistencia de datos. Incorpora el estándar

JPA como el principal API de persistencia para aplicaciones EJB3. Proporcionan un modelo de componentes distribuido estándar del lado del servidor. El objetivo de los EJB es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (concurrencia, transacciones, persistencia, seguridad, etc.) para centrarse en el desarrollo de la lógica de negocio en sí. El hecho de estar basado en componentes permite que éstos sean flexibles y sobre todo reutilizables.

En el caso específico del proyecto SIGESAP es usado EJB como una tecnología. Lo que permite tener bien delimitadas cuáles son las funciones del servidor, y publicar en un objeto EJB los gestores remotos que permiten el acceso a los mismos. Un aspecto a destacar sobre el uso y las prestaciones de EJB, es que constituye un elemento de seguridad que permite restringir el uso de clases y métodos específicos dentro de estas a usuarios en particular, logrando así que ningún de estos pueda tener accesos a funcionalidades que no le corresponden.

Swing: Es un conjunto de clases de Java que simplifica la construcción de aplicaciones de escritorio. Permite tener un sistema de ventanas y componentes gráficos, independiente del sistema operativo y la librería de dibujo que se tengan disponibles en la máquina clientes. Swing es un extenso conjunto de componentes que van desde los más simples, como etiquetas, hasta los más complejos, como tablas, árboles, y documentos de texto con estilo. Casi todos los componentes Swing descienden de un mismo padre llamado JComponent que desciende de la clase de AWTContainer. La característica más notable de los componentes Swing es que están escritos al 100% en Java y no dependen de componentes nativos.

Glassfish: Servidor de aplicaciones de software libre que implementa la plataforma JavaEE5, por lo que soporta las últimas versiones de tecnologías como: JSP, JSF, Servlets, EJBs, Java API para Servicios Web (JAX-WS), Arquitectura Java para Enlaces XML (JAXB), Metadatos de Servicios Web para la Plataforma Java 1.0, y muchas otras tecnologías. Es gratuito y de código libre, se distribuye bajo un licenciamiento dual a través de la licencia CDDL y la GNU GPL.

La fundamentación teórica es esencial en el desarrollo de toda investigación, ya que en esta se definen los principales conceptos que permiten el mejor entendimiento de la solución, así como también se

definen las herramientas, tecnologías, lenguajes y metodologías, temas importantes cuando se habla de desarrollo de software en la actualidad. A partir de lo antes expuesto y teniendo en cuenta las necesidades del sistema, se seleccionó como metodología de desarrollo a RUP, debido a que esta permite un mayor control durante el desarrollo de proyectos de software de gran alcance. Como herramienta CASE se utiliza Visual Paradigm 3.4 que a su vez utiliza UML como lenguaje de modelado, el cual permite la integración con el NetBeans 6.9.1, que tiene como lenguaje de programación Java, seleccionando este IDE por la gran variedad de librerías que facilitan el trabajo con las tecnologías.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA.

En este capítulo se abordan las características de la solución propuesta para el sistema. Se describen los requisitos funcionales y no funcionales, apoyados en los diagramas de casos de uso y clases del diseño. Además describe la arquitectura que soportará el desarrollo de la aplicación.

2.1 Requisitos del sistema

En un proceso de desarrollo de software es de gran importancia la descripción de los requisitos funcionales, los que son brindados por el analista de sistema y facilitan un mejor entendimiento de los procesos a desarrollar, permitiendo comprender en profundidad el problema en cuestión y facilitando una mejor identificación de las clases y funcionalidades que serán implementadas. La especificación de requerimientos es la base que permite verificar si se alcanzaron o no los objetivos establecidos en el proyecto, debido a que son un reflejo detallado de las necesidades de los clientes o usuarios del sistema.

2.1.1 Requisitos funcionales

Un requisito funcional define el comportamiento interno del software. Es una condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo y tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente.

RF1. Asociar Unidad Documental a Expediente
RF2. Anexar documento a expediente
RF3. Añadir unidades documentales a un expediente
RF4. Eliminar unidades documentales de un expediente
RF5. Buscar Funcionario
RF6. Mostrar datos de funcionario
RF7. Mostrar vista previa de oficios
RF8. Emitir oficio del trámite de inscripción en curso
RF9. Emitir oficio de certificación de Antecedentes Penales

RF10. Emitir oficio de cancelación
RF11. Emitir oficio de error en datos
RF12. Mostrar delitos de reo
RF13. Adicionar delitos a persona
RF14. Eliminar delito de reo
RF15. Mostrar detalles de delito
RF16. Gestionar Persona a Unidad Documental
RF17. Buscar persona
RF18. Mostrar listado de personas involucradas en una unidad documental
RF19. Adicionar persona involucrada en unidad documental
RF20. Eliminar persona involucrada en unidad documental
RF21. Mostrar Detalles de Unidad Documental
RF22. Mostrar datos generales de la unidad documental
RF23. Mostrar Ficha de Antecedentes Penales
RF24. Mostrar datos procesales del reo
RF25. Mostrar unidad documental digital
RF26. Ver Historial de Notas

2.1.2 Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Estas propiedades se deben concebir para obtener un sistema atractivo, usable, rápido y confiable. Por lo que ayudarán a definir si un producto es bien aceptado o poco aceptado. A continuación se especifican los requisitos no funcionales que se tendrán en cuenta para elaborar el sistema:

RNF1. Usabilidad

- Cumplir con las pautas de diseño de las interfaces.
- Agrupar vínculos y botones por grupos funcionales.
- Mostrar los mensajes, títulos y demás textos que aparezcan en la interfaz del sistema en idioma español.
- Utilizar campos de selección en la interfaz en los casos que sea posible.
- Usar combinaciones de teclas para agilizar la interacción del usuario con el sistema.

RNF2. Fiabilidad

- Asegurar la disponibilidad del sistema.
- Prever contingencias para caída del sistema.

RNF3. Eficiencia

- Responder en tiempos aceptables las peticiones que se realicen en el sistema.
- Garantizar acceso concurrente para todos los usuarios del sistema.

RNF4. Restricciones de diseño

- Diseñar el sistema a través de subsistemas.
- Desarrollar el sistema para un entorno de escritorio.
- Diseñar el sistema siguiendo las pautas definidas por el Ministerio del Poder Popular para las Telecomunicaciones y la Informática.

RNF5. Requisitos para la documentación de usuarios y ayuda del sistema

- Integrar ayuda al sistema.
- Confeccionar manual de usuario.

RNF6. Interfaz

Interfaces Hardware

- Restringir el acceso a la información contenida en los servidores de Bases de Datos desde las estaciones de trabajo.

Interfaces Software

- Acceder a la información que presta el sistema a través de Servicios Web.

RNF7. Requisitos Legales, de Derecho de Autor y otros

- Entregar el sistema a la División de Antecedentes Penales.

RNF8. Requisitos de seguridad

- Restringir el acceso a los servicios alojados en el local tecnológico del la División de Antecedentes Penales.
- Acceder de forma segura a los servicios del sistema.
- Almacenar de forma segura las credenciales de los usuarios.
- Almacenar acciones de los usuarios sobre el sistema.

RNF9. Requisitos de Hardware

- Proporcionar características mínimas de hardware a las estaciones de trabajo.

PC Cliente

- 2 GB de RAM, DDR2 667 MHz
- Procesador de doble núcleo a 2.0 GHz
- 160 GB de disco duro, SATA 5400 RPM.
- Adaptador de red Ethernet 1 Gbps.
- Monitor LCD 17''.
- El sistema tiene que interactuar con dispositivos de impresión.
- El sistema tiene que interactuar con dispositivos de escaneo.

Proporcionar características mínimas de hardware a los servidores.

PC Servidor

- Procesador de doble núcleo de 64 bits a 2.0 GHz de velocidad. Cuatro procesadores en los servidores de aplicación y dos en los servidores de bases de datos.
- 2 GB de RAM por cada instancia de java que correrá en el servidor de aplicaciones.
- 5 TB de espacio para el almacenamiento de la información de la solución.

RNF10. Requisitos de Software

PC Cliente

- Sistema Operativo: Ubuntu 10.10 o Windows XP.
- Máquina Virtual de Java: 1.6.
- Visor de Documentos PDF.
- Windows: Acrobat Reader.
- Ubuntu: Evince
- Herramientas Ofimáticas.

- OpenOffice 3.0.

PC Servidor

Servidor de Aplicación.

- Máquina Virtual de Java: 1.6.
- Servidor de Aplicación: Glassfish 2.1 (Versión community).
- IpTable.

Servidor de Bases de Datos.

- Gestor de Bases de Datos: PostgreSQL 8.4.
- Sistema para Réplica de Datos: Chronos.

2.2 Modelo de casos de uso del sistema

El modelo de casos de uso describe la funcionalidad propuesta del nuevo sistema. Un caso de uso representa una unidad discreta de interacción entre un usuario (humano o máquina) y el sistema. Un caso de uso es una unidad simple de trabajo significativo.

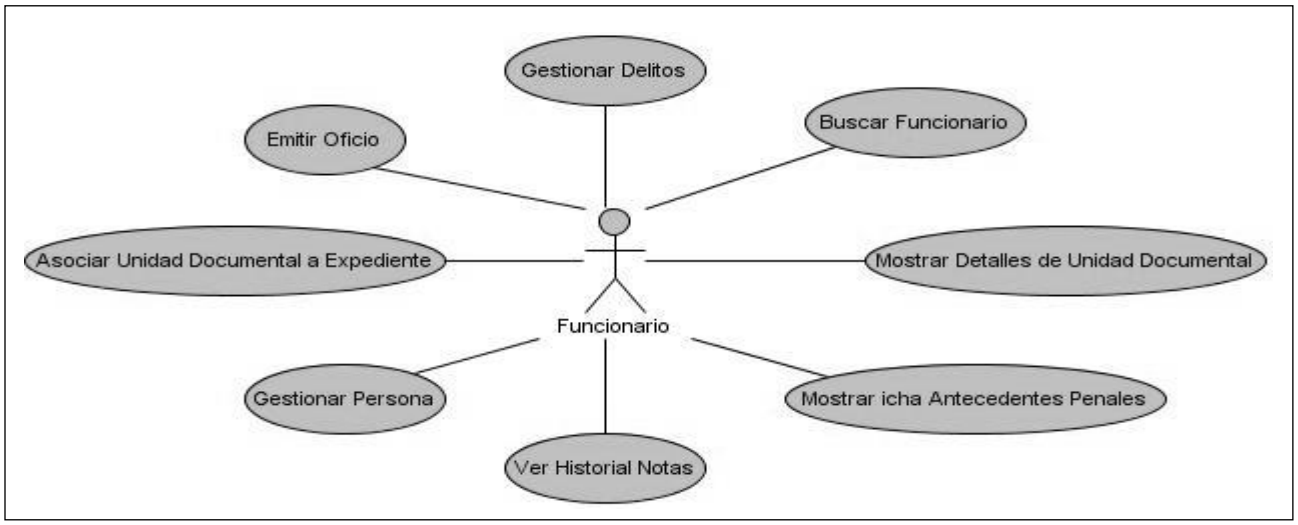


Fig. 2: Modelo de casos de uso del sistema

En este modelo de casos de uso del sistema aparece un único actor, Funcionario, esto se debe a que la mayoría de las funcionalidades son extendidas o incluidas de otros casos de uso, representándose estos mediante dicho actor.

2.2.1 Descripción de los casos de uso

Asociar Unidad Documental a Expediente: Consiste en realizar las búsquedas necesarias para asegurar la consistencia de las unidades documentales que se van a archivar en la División de Antecedentes Penales en el trámite actual. Esta consistirá en la búsqueda de los posibles expedientes a los cuales puede pertenecer la unidad documental, por estar relacionada al mismo proceso penal, con el fin de agregarle la información necesaria al trámite que la contiene, para que el funcionario de archivo pueda ubicar y asociar físicamente la unidad documental a un expediente específico una vez otorgado el trámite.

Buscar Funcionario: Consiste en realizar las búsquedas necesarias para encontrar información referente a un funcionario de la División de Antecedentes Penales.

Emitir Oficio: Consiste en seleccionar el oficio a emitir y llenar los campos necesarios para su emisión. Finaliza con la impresión del oficio emitido.

Gestionar Delitos: Consiste en adicionar delitos a la persona, eliminar delitos de la persona, modificar un delito, ver los detalles de un delito o listar todos los delitos de una persona.

Gestionar Persona a Unidad Documental: Consiste en adicionar personas a la unidad documental, eliminar personas de unidad documental, cambiar la persona por otra existente y mostrar el listado de personas.

Mostrar Detalles de Unidad Documental: Consiste en mostrar los datos generales de la unidad documental asociada. El usuario puede ver los detalles de las personas asociadas a la unidad documental, los datos procesales asociados a la misma y la imagen digital multipágina correspondiente.

Mostrar Ficha de Antecedentes Penales: Consiste en mostrar el resumen de beneficios procesales, Sentencias Definitivamente Firmes y Notas Aclaratorias contenidas en la Ficha de Antecedentes Penales.

Ver Historial de Notas: Consiste en mostrar todas las notas referentes al trámite especificado.

2.3 Arquitectura a utilizar

La arquitectura de software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente y las relaciones entre ellos. [20]

Los estilos arquitectónicos a emplear para estructurar el sistema propuesto son los siguientes:

Arquitectura Cliente-Servidor: en este estilo arquitectónico el procesamiento requerido para ejecutar una aplicación o conjunto de aplicaciones relacionadas se divide entre dos o más procesos que cooperan entre sí. Usualmente la mayoría del trabajo pesado se hace en el proceso llamado servidor y el proceso cliente sólo se ocupa de la interacción con el usuario. Éste es un modelo que intenta proveer usabilidad, flexibilidad, interoperabilidad y escalabilidad en las comunicaciones. Comenzó a ser aceptado a finales de los años 80. Tiene como ventajas:

- La posibilidad de utilizar componentes, tanto de hardware como de software, de varios fabricantes, lo cual contribuye considerablemente a la reducción de costos y favorece la flexibilidad en la implantación y actualización de soluciones.
- Facilita la integración entre sistemas diferentes, permitiéndoles compartir información.
- El mantenimiento y el desarrollo de aplicaciones es rápido ya que se pueden emplear las herramientas existentes.
- No es siempre necesario transmitir información gráfica por la red, pues esta puede residir en el cliente, lo cual permite aprovechar mejor el ancho de banda de la red.
- La estructura inherentemente modular facilita además la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional, favoreciendo así la escalabilidad de las soluciones.

Arquitectura N capas: estilo arquitectónico donde cada capa puede ser modificada tanto como sea necesario sin provocar cambios en las demás. Una capa no tiene dependencias con la capa superior, cada capa depende solamente de la fachada que le permite comunicarse con la capa inmediata inferior. Esta dependencia entre capas es normalmente a través de fachadas, asegurando que el acoplamiento sea el más bajo posible y la abstracción del funcionamiento de la capa inferior, sea casi total. Esto permite

el desarrollo de aplicaciones más robustas debido al encapsulamiento. Es factible un mantenimiento y soporte más sencillo así como una mayor flexibilidad. [21]

Arquitectura basada en componentes: Una arquitectura basada en componentes describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema. Esta arquitectura se enfoca en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas. Esto provee un nivel de abstracción mayor que los principios de orientación por objetos y no se enfoca en asuntos específicos de los objetos como los protocolos de comunicación y la forma como se comparte el estado.

El estilo de arquitectura basado en componentes tiene las siguientes características:

- Es un estilo de diseño para aplicaciones compuestas de componentes individuales.
- Pone énfasis en la descomposición del sistema en componentes lógicos o funcionales que tienen interfaces bien definidas.
- Define una aproximación de diseño que usa componentes discretos, los que se comunican a través de interfaces que contienen métodos, eventos y propiedades.

Principios Fundamentales

Un componente es un objeto de software específicamente diseñado para cumplir con cierto propósito. Los principios fundamentales cuando se diseña un componente es que estos deben ser:

- *Reusable*. Los componentes son usualmente diseñados para ser utilizados en escenarios diferentes por diferentes aplicaciones, sin embargo, algunos componentes pueden ser diseñados para tareas específicas.
- *Sin contexto especifico*. Los componentes son diseñados para operar en diferentes ambientes y contextos. Información específica como el estado de los datos deben ser pasadas al componente en vez de incluirlos o permitir al componente acceder a ellos.
- *Extensible*. Un componente puede ser extendido desde un componente existente para crear un nuevo comportamiento.

- *Encapsulado*. Los componentes exponen interfaces que permiten al programa usar su funcionalidad. Sin revelar detalles internos, detalles del proceso o estado.
- *Independiente*. Los Componentes están diseñados para tener una dependencia mínima de otros componentes. Por lo tanto los componentes pueden ser instalados en el ambiente adecuado sin afectar otros componentes o sistemas.

Beneficios

Los siguientes son los principales beneficios del estilo de arquitectura basado en componentes:

- Facilidad de Instalación. Cuando una nueva versión esté disponible, usted podrá reemplazar la versión existente sin impacto en otros componentes o el sistema como un todo.
- Costos reducidos. El uso de componentes de terceros permite distribuir el costo del desarrollo y del mantenimiento.
- Facilidad de desarrollo. Los componentes implementan un interface bien definida para proveer la funcionalidad definida permitiendo el desarrollo sin impactar otras partes del sistema.
- Reusable. El uso de componentes reutilizables significa que ellos pueden ser usados para distribuir el desarrollo y el mantenimiento entre múltiples aplicaciones y sistemas.
- Mitigación de complejidad técnica. Los componentes mitigan la complejidad por medio del uso de contenedores de componentes y sus servicios. Ejemplos de servicios de componentes incluyen activación de componentes, gestión de la vida de los componentes, gestión de colas de mensajes para métodos del componente y transacciones. [26]

2.4 Patrones para el desarrollo de software

En el ámbito de la informática, los patrones proponen una forma de reutilizar la experiencia de los desarrolladores, clasificando y describiendo formas de solucionar problemas que ocurren en el desarrollo de software, con el objetivo fundamental de evitar errores durante el mismo. Existen diferentes categorías de patrones para el desarrollo de software, los más utilizados son los que se caracterizan a continuación:

- ***Patrones Generales de Software para Asignación de Responsabilidades (GRASP):*** Aquellos que expresan los principios fundamentales de la asignación de responsabilidades a objetos durante la programación orientada a objetos.
- ***Patrones de diseño:*** Aquellos que expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas de software.
- ***Patrón de acceso a datos:*** Aquellos que expresan variantes para la gestión de diversas fuentes de datos.

A continuación se exponen los patrones GRASP, de diseño y de acceso a datos empleados para el desarrollo del sistema.

2.4.1 Patrones de diseño de software

Los patrones de diseño o patrones GoF, están agrupados en tres grupos fundamentales:

- ***Creacionales:*** describen las formas de crear instancias de objetos. El objetivo de estos patrones es el de abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.
- ***Estructurales:*** describen cómo las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.
- ***Comportamiento:*** describen la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos.

A continuación de describen los patrones a utilizar en cada categoría:

Creacionales

- *Singleton:* utilizado para la creación de una instancia de las clases que sean necesarias, como los gestores del negocio, fachada de gestores remotos, entre otras.

Estructurales

- *Fachada (Facade):* proporciona una interfaz unificada a un conjunto de interfaces de un sistema. Facade define una interfaz de alto nivel, posibilitando que el sistema sea más fácil de usar. Entre

las ventajas que tiene están que facilita el uso del sistema, promueve un acoplamiento débil entre el subsistema y sus clientes, al ser eliminadas o reducidas las dependencias, y no existen obstáculos para que las aplicaciones usen las clases del subsistema que necesiten, de esta forma se puede elegir entre facilidad de uso y generalidad. Utilizado para la creación de una interfaz, como fachada de gestores remotos, interfaces remotas, interfaces locales, entre otras.

Comportamiento

- *Mediador (Mediator):* define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto. Evita las referencias explícitas entre los objetos y permite, por tanto, que su interacción se modifique de forma independiente. Ente las ventajas que lo caracterizan están: que al reducir el acoplamiento entre los objetos que interactúan es más fácil variar o reutilizar dichos objetos, y que al hacer de la mediación un concepto independiente, encapsulado en un objeto a parte, se favorece la concentración de la atención en la interacción entre objetos, al margen del comportamiento individual de cada uno. Utilizado en las interfaces, como fachada de gestores remotos, interfaces remotas, interfaces locales, entre otras.

2.4.2 Patrones GRASP

Estos patrones describen los principios fundamentales de la asignación de responsabilidades a objetos. Entre las responsabilidades más importantes están:

Hacer:

- Iniciar una acción en otros objetos.
- Controlar y coordinar actividades en otros objetos.

Conocer:

- Estar enterado de los datos privados encapsulados.
- Estar enterado de los objetos conexos.
- Estar enterado de las cosas que puede derivar o calcular.

A continuación se describen los patrones de este tipo a emplear para el desarrollo del sistema.

Experto: este patrón permite asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. La responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantendrá encapsulada (disminución del acoplamiento). Utilizado en los gestores de negocio del cliente y del servidor, puesto que ambos gestores solamente trabajan con la información de las entidades de dominio que representan.

Ventajas: [22]

- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento. [22]
- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clase “sencillas” y más cohesivas que son más fáciles de comprender y de mantener. Así se brinda soporte a una alta cohesión. [22]

Creador: guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. [22]

Ventajas:

- Brinda soporte a un bajo acoplamiento, lo cual supone poca dependencia respecto al mantenimiento y mejores oportunidades de reutilización.

Bajo Acoplamiento: su principal característica es mantener las clases más independientes entre sí y con la menor cantidad de relaciones; la cual posibilita que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y disminuyendo la dependencia entre las clases, asignándoles una responsabilidad para mantener un bajo acoplamiento. Utilizado para fomentar la mínima dependencia de las clases con respecto al resto del sistema.

Ventajas: [22]

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

Alta Cohesión: la principal característica de este patrón es asignar responsabilidades de modo que la cohesión siga siendo alta. La información que almacena una clase debe de ser coherente y debe estar en la medida de lo posible relacionada con la clase. Utilizado para hacer de los componentes de mi sistema un conjunto bien acoplado.

Ventajas: [22]

- Mejoran la claridad y la facilidad del diseño.
- Simplifican el mantenimiento y las mejoras en funcionalidad.
- A menudo, se genera un bajo acoplamiento.
- Soporta una mayor capacidad de reutilización.

Controlador: posibilita la asignación de responsabilidades para controlar el flujo de eventos del sistema a clases específicas. Sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario enviándolo a las distintas clases según el método llamado.

Ventajas: [22]

- Mayor potencial de los componentes reutilizables. Garantiza que la empresa o los procesos de dominio sean manejados por la capa de los objetos del dominio y no por la de la interfaz.
- Reflexionar sobre el estado del caso de uso. A veces es necesario asegurarse de que las operaciones del sistema sigan una secuencia legal o poder razonar sobre el estado actual de la actividad y las operaciones en el caso de uso subyacente.
- Un corolario importante del patrón Controlador es que los objetos de la interfaz (por ejemplo, objetos de ventanas, applets) y la capa de presentación no deberían encargarse de manejar los eventos del sistema.

2.4.3 Patrón de acceso a datos (DAO)

Centraliza todo el acceso a datos en una capa independiente, aislándolo del resto de la aplicación, desacoplando la lógica de negocio de la lógica de acceso a datos, de manera que se pueda cambiar la fuente de datos fácilmente. Sus principales beneficios son que reduce la complejidad de los objetos de negocio al abstraerlos de la implementación real de la comunicación con la fuente de datos. Cada DAO tiene que implementar los métodos declarados en la interfaz DAO correspondiente. Utilizado en los gestores de acceso a dato del sistema. [23]

2.5 Diagrama de paquetes del diseño

El diagrama de paquetes del diseño refleja cómo el sistema está dividido en agrupaciones lógicas y las relaciones entre estas. Los paquetes están normalmente organizados para maximizar la coherencia interna entre paquetes y minimizar el acoplamiento externo entre los mismos. A continuación se muestra el diagrama de paquetes del diseño del módulo Integración.

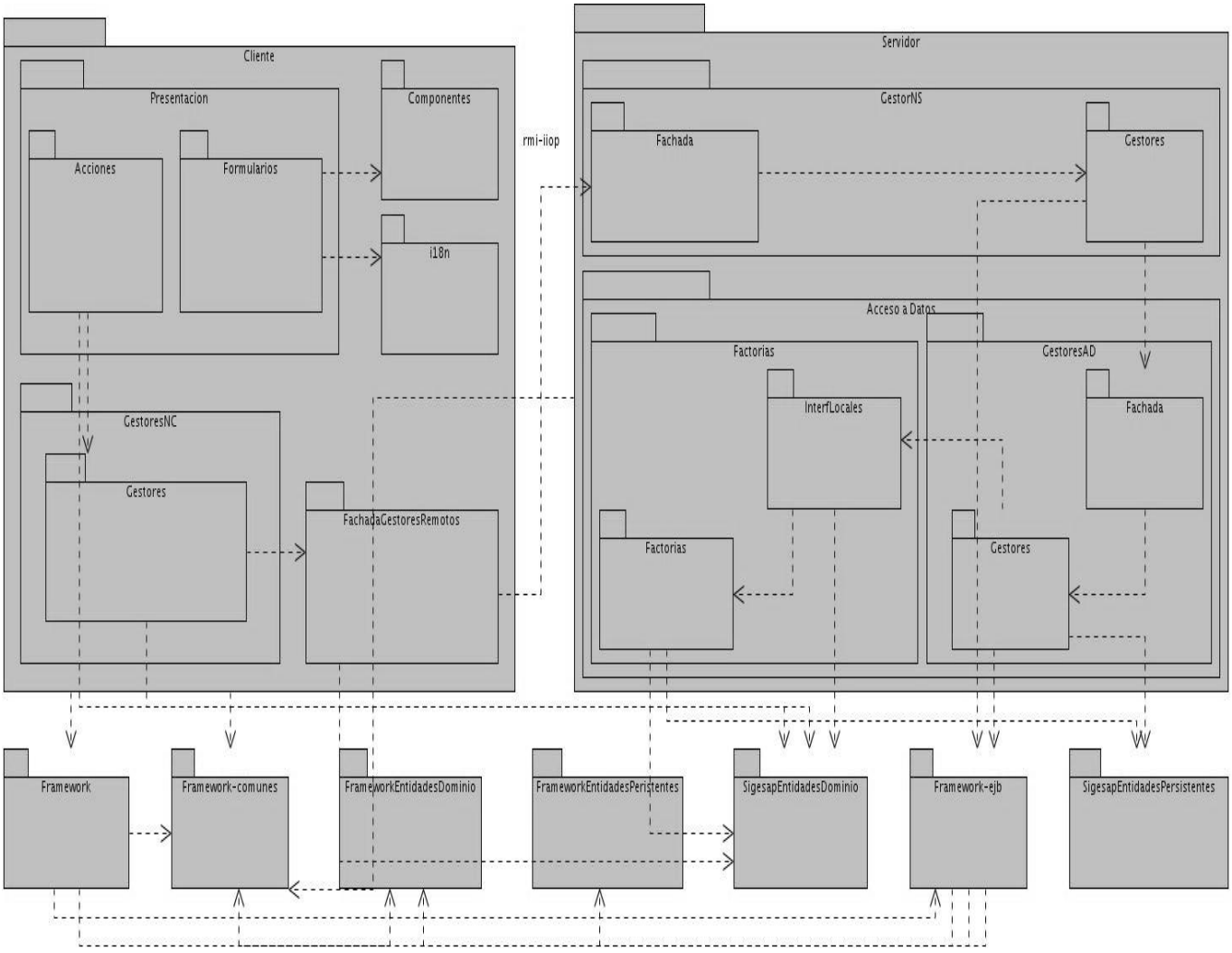


Fig. 3: Modelo de paquetes del diseño

2.6 Diagrama de clases del diseño

En un diagrama de clases de diseño (DCD) se muestran los atributos y métodos de cada clase y se representa de forma sencilla la colaboración y las responsabilidades de las distintas clases que forman el sistema. A continuación se muestra el diagrama de clases del diseño del caso de uso Gestionar Delito y la descripción de cada uno de sus componentes. El resto de los diagramas y sus descripciones,

pertenecientes al módulo Integración, se mostrarán como parte de los anexos del presente trabajo. (Ver Anexo 1)

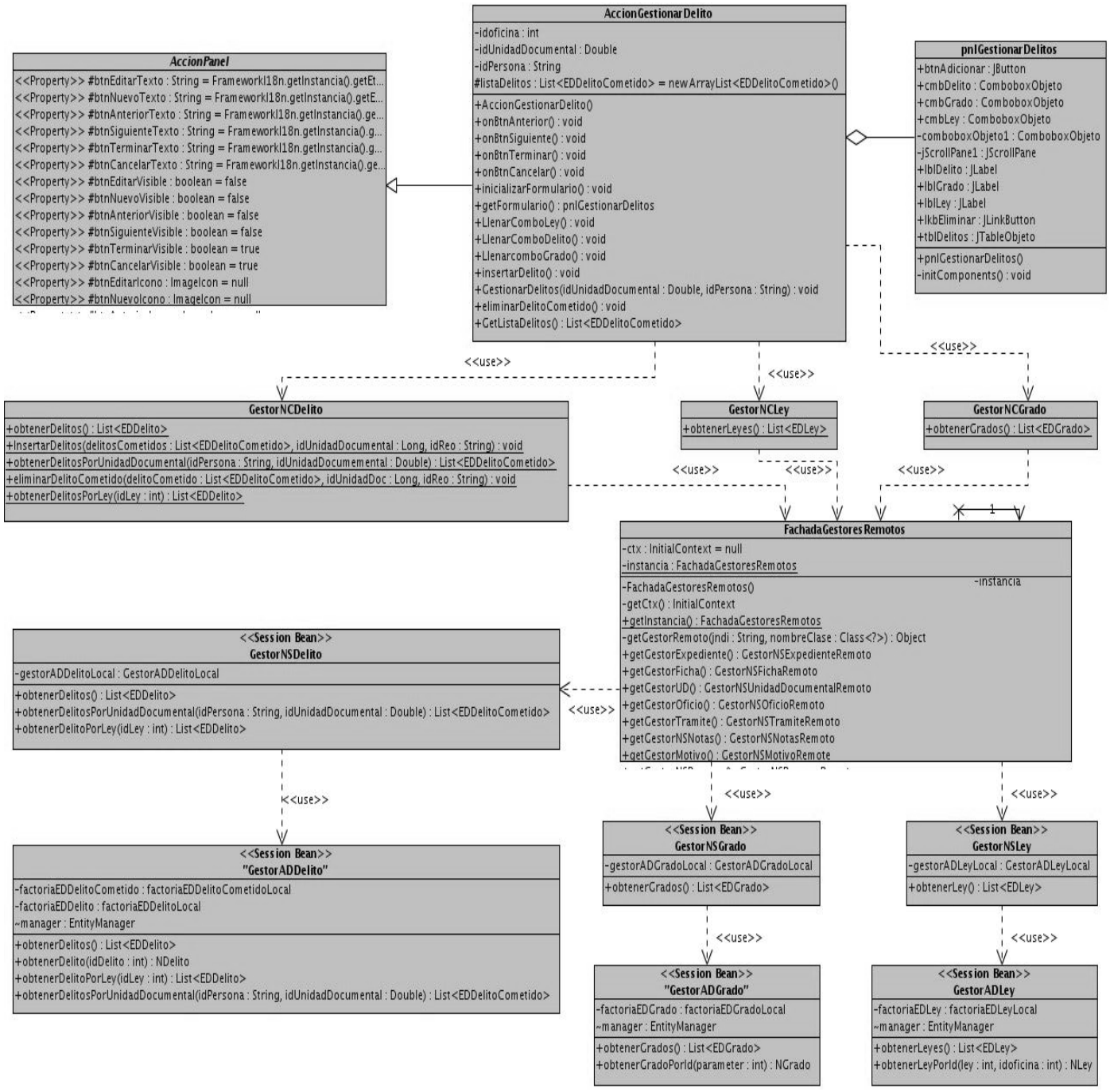


Fig.4: Diagrama de clases del diseño DCD_GestionarDelito

Nombre de la clase: pnlGestionarDelitos	
Tipo de clase: Interfaz	
Atributos	**Tipo**
btnAdicionar	JButton
cmbDelito	ComboboxObjeto
cmbGrado	ComboboxObjeto
cmbLey	ComboboxObjeto
comboboxObjeto1	ComboboxObjeto
JScrollPanel1	JScrollPanel
lblDelito	JLabel
lblGrado	JLabel
lblLey	JLabel
lkbEliminar	JLinkButton
tblDelitos	JTableObjeto

Tabla 1: Descripción de la clase pnlGestionarDelitos

Nombre de la clase: AccionGestionarDelitos	
Tipo de clase: Controladora	
Atributos	**Tipo**
expedientesRecepcionados	List<EDExpediente>
Reponsabilidades de la clase:	
Nombre:	onbtnTerminar
Descripción:	Termina con el gestionar delitos
Nombre:	onbtnCancelar
Descripción:	Cancela el gestionar delitos
Nombre:	onbtnSiguiente
Descripción:	Actualiza la lista de datos procesales
Nombre:	onbtnAnterior
Descripción:	Va al paso anterior

Nombre:	LlenarComboLey
Descripción:	Llena el componente cmbLey con los nombres de las leyes.
Nombre:	LlenarComboGrado
Descripción:	Llena el componente cmbGrado con los nombres de los diferentes Grados
Nombre:	LlenarComboDelito
Descripción:	Llena el componente cmbDelito con los nombres de los diferentes Delitos
Nombre:	InsertarDelito
Descripción:	Agrega un delito a la lista de datos procesales
Nombre:	EliminarDelito
Descripción:	Elimina un delito de la lista de datos procesales
Nombre:	GetListaDelitos
Descripción:	Obtiene una lista de Delitos cometidos
Nombre:	GestionarDelitos
Descripción:	Maneja las distintas funcionalidades

Tabla 2: Descripción de la clase AccionGestionarDelitos

Nombre de la clase: GestorNCDelito	
Tipo de clase: Controladora	
Reponsabilidades de la clase:	
Nombre:	obtenerDelito
Descripción:	Invoca el método obtenerDelito del gestor servidor delito.
Nombre:	InsertarDelito
Descripción:	Invoca el método InsertarDelito del gestor servidor delito.
Nombre:	ObtenerDelitoPorUnidadDocumental

Descripción:	Invoca el método ObtenerDelitoPorUnidadDocumental del gestor servidor delito.
Nombre:	ObtenerDelitoPorLey
Descripción:	Invoca el método ObtenerDelitoPorLey del gestor servidor delito.
Nombre:	EliminarDelitoCometido
Descripción:	Invoca el método EliminarDelitoCometido del gestor servidor delito.

Tabla 3: Descripción de la clase GestorNCDelito

Nombre de la clase: GestorNSDelito	
Tipo de clase: Controladora	
Reponsabilidades de la clase:	
Nombre:	obtenerDelito
Descripción:	Invoca el método obtenerDelito del gestor servidor trámite.
Nombre:	ObtenerDelitoPorLey
Descripción:	Invoca el método ObtenerDelitoPorLey del gestor servidor trámite.
Nombre:	ObtenerDelitoPorUnidadDocumental
Descripción:	Invoca el método ObtenerDelitoPorUnidadDocumental del gestor servidor trámite.

Tabla 4: Descripción de la clase GestorNSDelito

Nombre de la clase: GestorADDelito
Tipo de clase: Controladora

Reponsabilidades de la clase:	
Nombre:	obtenerDelito
Descripción:	Devuelve un delito por el id
Nombre:	obtenerDelitoPorLey
Descripción:	Devuelve un delito por el id de Ley
Nombre:	obtenerDelitoPorUnidadDocumental
Descripción:	Devuelve un delito por el id de Unidad Documental

Tabla 5: Descripción de la clase GestorADDelito

2.7 Diagrama de interacción

Los Diagramas de interacción modelan los aspectos dinámicos de un sistema, debido a que muestran gráficamente cómo los objetos se comunican entre sí para cumplir con los requerimientos. Dentro del Diagramas de interacción se encuentran los diagramas de secuencia (DS) y de colaboración. Por lo general los diagramas de colaboración se realizan en el análisis y los de secuencia en el diseño. Estos últimos muestran la ordenación temporal de los mensajes. A continuación se muestra el diagrama de secuencia del caso de uso Gestionar Delito. El resto de los diagramas de secuencia pertenecientes al módulo Integración, se mostrarán como parte de los anexos del presente trabajo. (Ver Anexo 2)

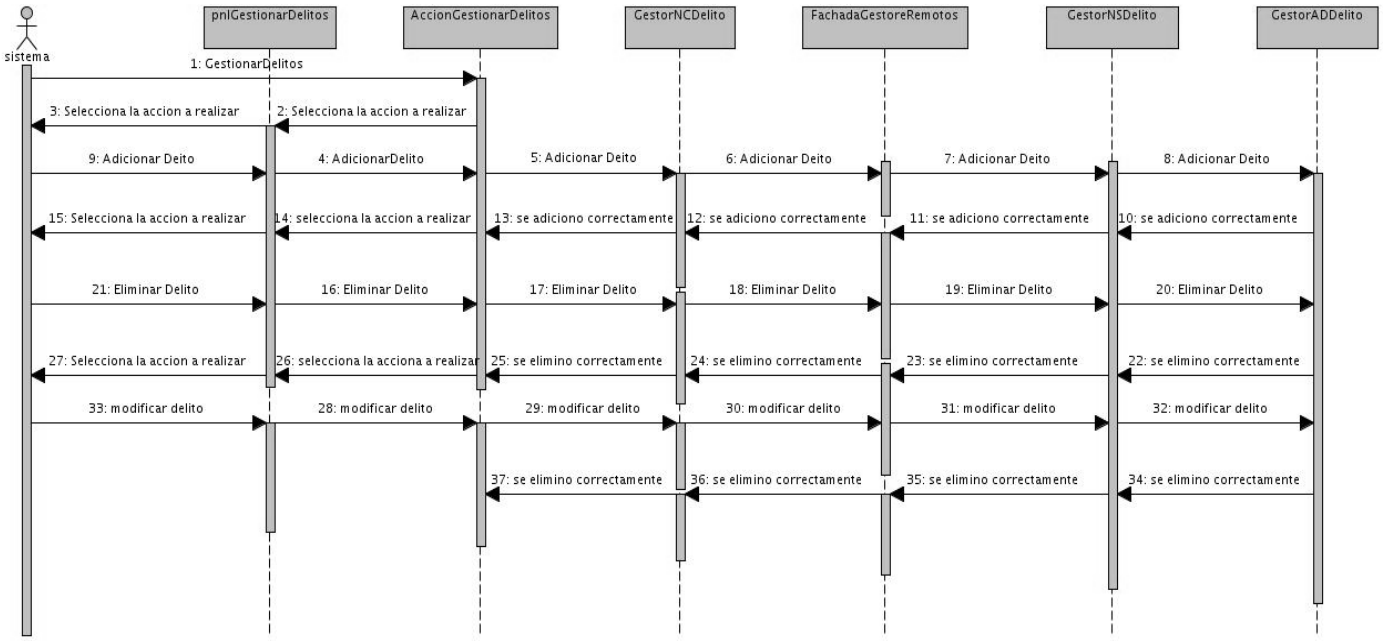


Fig.5: Diagrama de secuencia DS_GestionarDelito

2.8 Pautas de diseño de interfaz de usuario

La Interfaz de Usuario (IU) de un programa es un conjunto de elementos de software de una computadora, que presentan información al usuario y le permiten interactuar con esta y con la computadora. También se puede considerar parte de la IU la documentación (manuales, ayuda, referencia) que acompaña al software.

Si la IU está bien diseñada, el usuario encontrará la respuesta que espera a su acción, para lograr esto, se definieron las siguientes pautas:

- La separación de los componentes respecto a los bordes del área de trabajo (margen izquierdo, derecho, superior e inferior) debe ser de 10 píxeles como mínimo.
- En un formulario, los datos asociados a una misma cosa se agruparán utilizando paneles con título. El título del panel se escribe en formato de oración, en negrita y sin dos puntos al final. Ej. La dirección, contiene ciudad, calle, etc. Se pondría un panel con título Dirección y dentro todos los campos que describen la dirección.

- Cuando el formulario tiene varios componentes. El nombre de las etiquetas se escribe encima de los componentes. El texto en las etiquetas se escribe en formato de oración, sin utilizar negrita y terminado en dos puntos. Los componentes irán debajo, a una distancia de 8 píxeles, cada componente de entrada de datos debe tener una altura de 23 píxeles. La separación horizontal entre componentes debe ser de 20 píxeles y vertical de 15 píxeles. Todo alineado a la izquierda, y en caso de que aparezcan unos debajo de los otros se debe tratar de que todos los componentes tengan un mismo tamaño (el del mayor componente).
- En caso de que el formulario tenga pocos componentes, se pondrá primero la etiqueta y al lado el componente, tanto los componentes como las etiquetas con las mismas pautas anteriores. Además se alinean las etiquetas a la izquierda y los componentes también a la izquierda.
- Cuando la etiqueta no es para una entrada de datos, sino para mostrar los mismos, entonces irá en negrita. Ej. Nombre y apellidos: Rusel Sablón Fernández.
- El texto de las etiquetas, títulos de columnas de tablas, botones, etc. debe ser siempre en formato de oración.
- Los combos deben mostrar como texto inicial –Seleccione--. Los ítems para seleccionar deben escribirse en formato de oración y deben aparecer en orden alfabético, a no ser que representen un flujo, en ese caso aparecerían de acuerdo al orden correspondiente. Cuando haya una opción Otros, estará ubicada al final.
- Cuando haya que introducir una fecha debe ponerse un componente para seleccionar la misma, evitar las entradas manuales del usuario.
- Los Botones siempre estarán situados en la parte inferior derecha de los formularios, con una altura de 25 píxeles y el ancho depende del Caption, siempre dejando un espacio de 8 píxeles delante y detrás de la palabra o el ícono. Sólo en el caso de los mensajes de confirmación, avisos y error los botones irán en el centro. El texto que contienen debe estar en formato de oración y sin utilizar negrita. Cuando aparezcan varios botones uno al lado del otro, de ser posible todos deben tener el mismo ancho y la separación entre ellos debe ser de 10 píxeles. El ancho mínimo de los botones debe ser de 75 píxeles.
- Los botones no llevan íconos, el Anterior y Siguiente llevan los caracteres <, > respectivamente.
- Los botones de “Si” y “No” en los mensajes de confirmación deben llevar íconos y tendrán un ancho de 50 píxeles.

2.9 Modelo de datos

Es el modelo que describe de manera abstracta cómo se representan los datos de una aplicación o sistema de información. Consiste en una descripción de algo conocido como contenedor de datos, así como de los métodos para almacenar y recuperar información de esos contenedores. El Modelo de Datos tiene gran importancia en el ciclo de desarrollo de software, y de manera particular para la fase de implementación, pues define formalmente las estructuras permitidas y las restricciones que se aplican con el fin de representar los datos del dominio de la aplicación. Está compuesto por objetos: entidades que existen y se manipulan; atributos: características básicas de dichos objetos y relaciones: forma en que se enlazan los objetos entre sí. [20]

El modelo de datos fue confeccionado agrupando las entidades, teniendo en cuenta para esto los diferentes procesos que se tienen, a continuación se muestran dichas agrupaciones. El modelo de datos general se muestra como parte de los anexos del presente trabajo. (Ver Anexo 3)

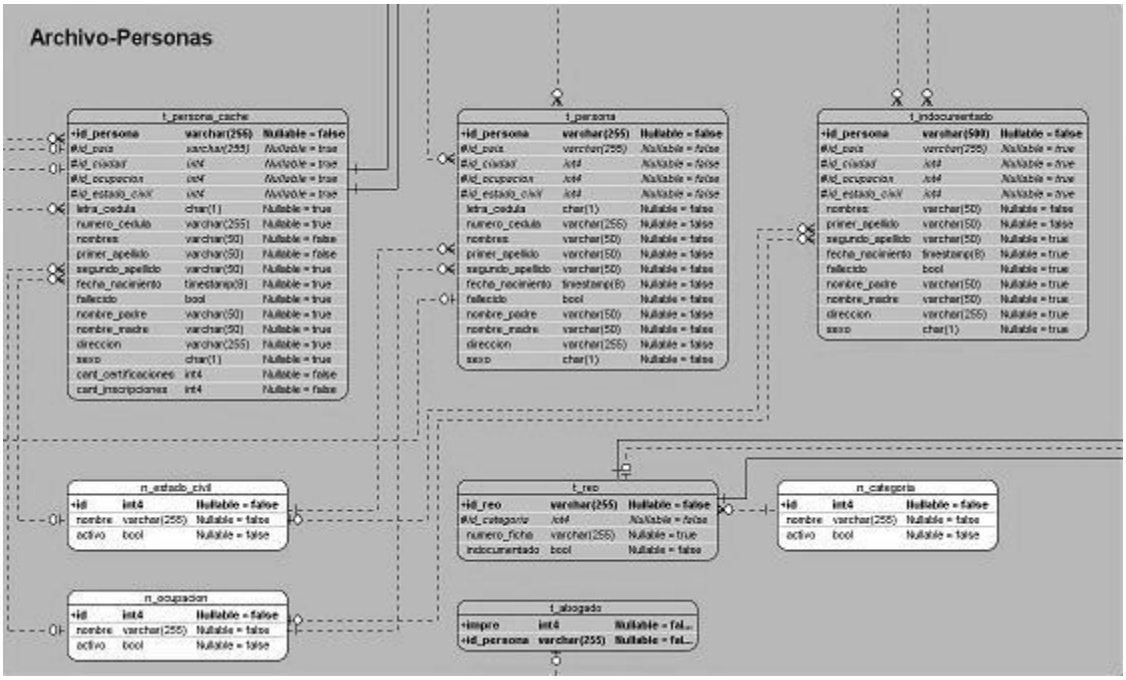


Fig. 6: Modelo de datos. Agrupación Archivo-Personales

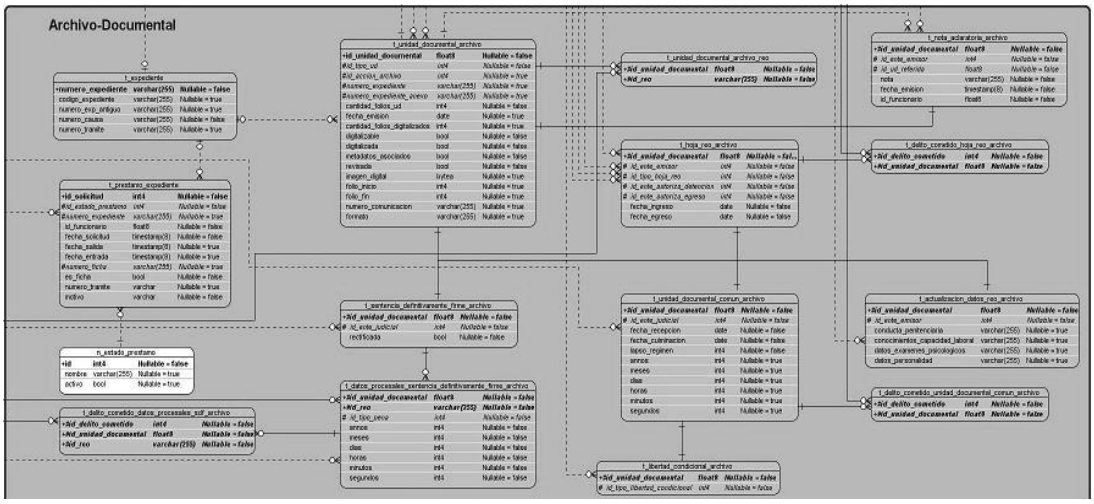


Fig. 7: Modelo de datos. Agrupación Archivo-Documental

Dentro de las entidades más importantes de este modelo se encuentran:

- **t_tramite:** Almacena los trámites que se realizan en el sistema.
- **t_usuario:** Almacena los datos generales de los usuarios del sistema.

- **t_nota:** Almacena las notas emitidas.
- **t_reo:** Almacena los reos presentes en el sistema.
- **t_persona:** Almacena los datos asociados a la persona.
- **t_indocumentado:** Almacena los datos asociados a la persona.
- **n_delito:** Almacena los delitos cometidos, definidos en el sistema.
- **n_ley:** Almacena las leyes o ley, definidas en el sistema.
- **n_grado:** Almacena el grado de un delito definido para el sistema.
- **t_expediente:** Almacena los datos relacionados al expediente.
- **n_tipo_unidad_documental:** Almacena cada tipo de unidad documental, como un nomenclador.
- **t_tramite_archivado:** Almacena los trámites que se realizan en el archivo.
- **t_unidad_documental_comun:** Almacena las unidades documentales Suspensión Condicional de la Pena, Suspensión Condicional de la Ejecución de la Pena, Destacamento de Trabajo, Conmutación de la Pena o Libertad Condicional.
- **t_unidad_documental:** Almacena las unidades documentales del sistema.
- **t_actualizacion_datos_reo:** Almacena las actualizaciones sobre los datos del reo.
- **t_hoja_reo:** Almacena los datos pertenecientes a un reo, definiéndolos en la hoja del reo.
- **t_libertad_condicional:** Almacena los datos procesales de la libertad condicional asociados a una unidad documental.
- **t_sentencia_definitivamente_firme:** Almacena la unidad documental sentencia definitivamente firme y sentencia definitivamente firme rectificada.
- **t_oficio:** Almacena los oficios emitidos en archivo

En este capítulo se abordaron los diferentes elementos que ilustran cómo está construido el sistema, en términos del diseño, los que posibilitó comprender la lógica del sistema en general. Se realizaron diferentes diagramas de clases del diseño donde se representaron las clases y las relaciones entre estas. Se definieron los patrones para el desarrollo del software a emplear, que evitaron la búsqueda de soluciones a problemas ya conocidos y solucionados, y por último se presentó el diseño de la base de datos, mediante el modelo de datos.

CAPITULO 3: IMPLEMENTACIÓN Y PRUEBA

Este capítulo está orientado a la implementación y prueba de la solución. Se obtendrá el modelo de implementación, el cual está formado por los diagramas de componentes y el de despliegue. Además, se verificará la calidad del resultado de la implementación, se hará uso de los artefactos generados durante el flujo de trabajo de pruebas, exponiendo los resultados obtenidos por diferentes tipos de pruebas realizadas al módulo.

3.1 Modelo de Implementación

Está conformado por los Diagramas de Componentes y de Despliegue, describiendo cómo los elementos del Modelo de Diseño se implementan en términos de componentes, ficheros de código fuente y ejecutables. El Modelo de Implementación describe además cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización, disponibles en el entorno de implementación y en el lenguaje de programación utilizado, y cómo dependen los componentes unos de otros. [20]

3.1.1 Diagrama de componentes

Los diagramas de componentes son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las elaciones entre los elementos de implementación. [20]
A continuación se muestran el diagrama de componentes de la capa de presentación, de la capa de negocio y la capa de acceso a datos del caso de uso Gestionar Delito. El diagrama de componentes de estas capas para el módulo Integración se muestra en los anexos del presente trabajo. (Ver Anexo 4)

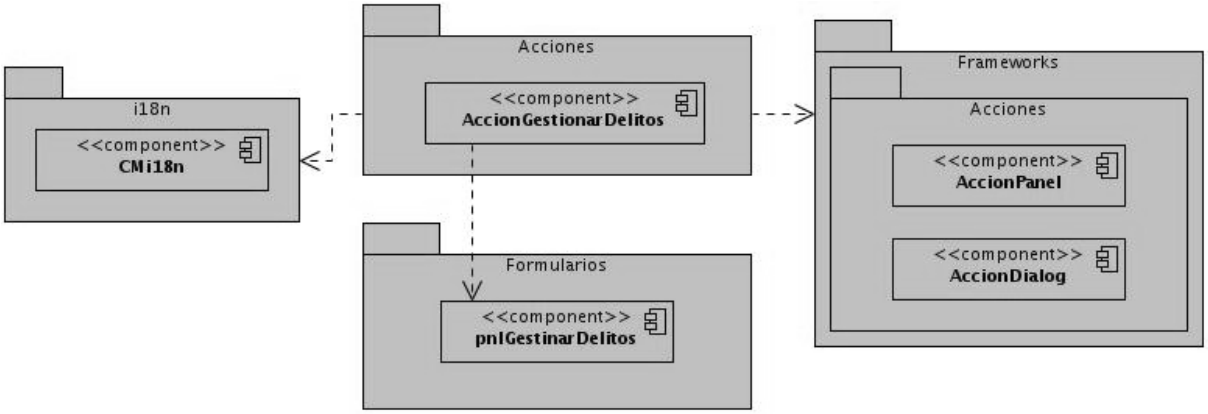


Fig. 8: Diagrama de componentes de la capa presentación. Caso de uso Gestionar Delito

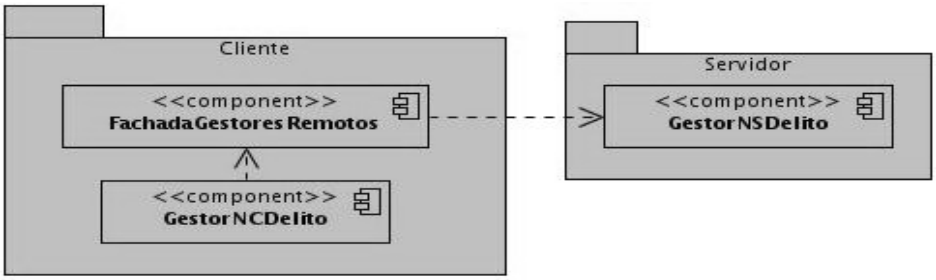


Fig. 9: Diagrama de componentes de la capa negocio. Caso de uso Gestionar Delito

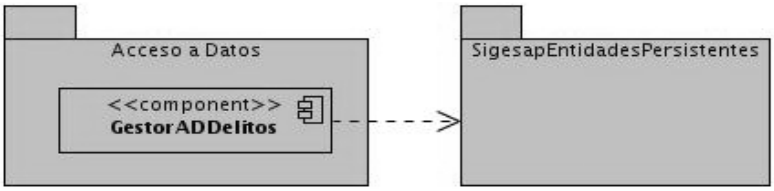


Fig. 10: Diagrama de componentes de la capa acceso a datos. Caso de uso Gestionar Delito

3.1.2 Diagrama de despliegue

El modelo de despliegue muestra la configuración de los nodos de procesamiento en tiempo de ejecución, los links de comunicación entre ellos, y las instancias de los componentes y objetos que residen en ellos. El propósito del modelo de despliegue es capturar la configuración de los elementos de procesamiento, y las conexiones entre estos elementos en el sistema. El modelo consiste en uno o más nodos (elementos de procesamiento con al menos un procesador, memoria, y posiblemente otros dispositivos), dispositivos (nodos estereotipados con una capacidad de procesamiento en el nivel modelado de abstracción), y conectores, entre nodos, y entre nodos y dispositivos. El modelo de despliegue también mapea procesos dentro de estos elementos de procesamiento, permitiendo la distribución del comportamiento a través de los nodos que son representados. [20]

A continuación se muestra el diagrama de despliegue de la solución propuesta.

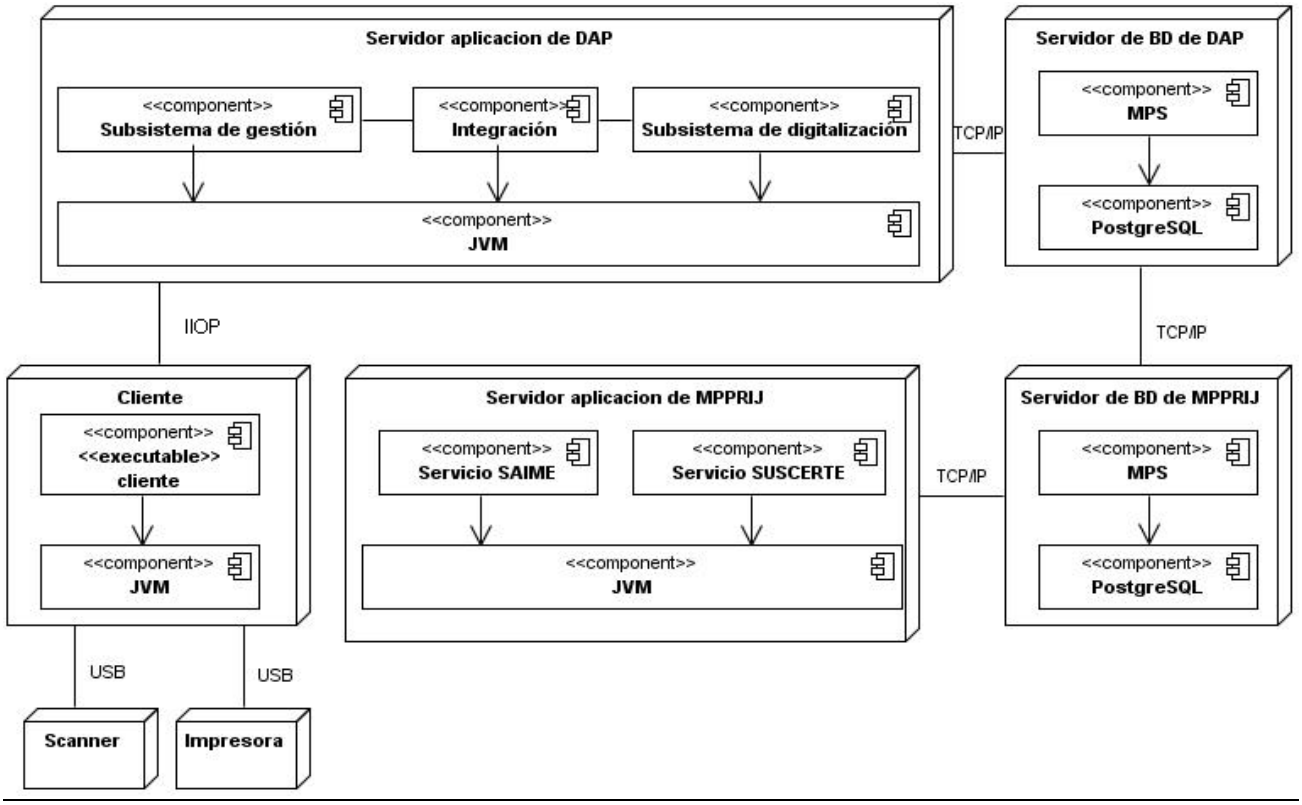


Fig. 11: Diagrama de despliegue

3.2 Estándares de codificación

Los estándares de codificación son un grupo de reglas que deben seguir los desarrolladores para obtener un código fácil de comprender y de alta calidad, fomentando las buenas prácticas definidas por la Ingeniería de Software. A continuación cómo nombrar cada uno de los elementos del módulo Integración:

- Todas las nomenclaturas a utilizar son en español.
- El identificador para las variables, los parámetros y los métodos se definen escribiendo, las palabras simples con la primera letra en minúsculas, y en caso de ser un nombre compuesto usando la notación Camello, para el cual existen dos variantes:

  UpperCamelCase: las palabras que forman el nombre se escriben juntas y la primera letra de cada una de ellas en mayúscula.

  LowerCamelCase: las palabras que forman el nombre se escriben juntas, la primera letra de la primera palabra en minúscula y del resto de las palabras, la primera letra en mayúscula.

  En el caso del SIGESAP se utilizará la segunda variante para definir los atributos y variables privadas y en el caso contrario se utilizará la primera variante, aplicando la técnica verbo-sustantivo.
- Los paquetes (o package) deberán comenzar con el nombre del sistema (SIGESAP) seguido por el subsistema, luego el módulo y por último la capa lógica a la que pertenece, quedando de la siguiente forma Sigesap.Subsistema.Modulo.Capa. Para el caso de los paquetes se usará la misma nomenclatura definida para los métodos de clases y clases.
- Camello se utilizará en los nombres de las Entidades persistentes, Entidades de Dominio, Propiedades y Funcionalidades del sistema.
- Las clases persistentes comenzarán con las siglas definidas en la BD para cada tabla.
- Los formularios de tipo popup comienzan con las siglas Frm, y los de tipo panel con las siglas Pnl.
- Los gestores de negocio del cliente comienzan con las siglas GestorNC.
- Las clases de la lógica de negocio de acceso a datos comienzan con las siglas GestorAD.
- Las clases de la lógica de negocio del servidor comienzan con las siglas GestorNS.
- Las interfaces publicadas en el servidor terminan con el prefijo Remoto.
- Las excepciones terminan con el sufijo Exception.
- Las relaciones entre clases usarán el estereotipo <<use>>.
- Las relaciones entre clases e interfaces usarán el estereotipo <<realize>>.

- Las relaciones entre clases y paquetes usarán el estereotipo <<access>>.
- Nomenclatura para los componentes de formularios:
- Los botones (JButton) comienzan con las siglas btn.
- Los campos de texto (JTextField) comienzan con las siglas tfd.
- Los campos fecha (Date) comienzan con las siglas fch.
- Para el componente JDateChooser ------ dtc
- Para el componente JDayChooser ------ dyc
- Los componentes de tipo Jscrool usarán las siglas sc.
- Las cajas de texto (JComboBox) comienzan con las siglas cmb.
- Las áreas de texto (JTextArea) comienzan con las siglas txt.
- Los labels (JLabel) comienzan con las siglas lbl.
- Los cuadros de chequeo (JCheckBox) comienzan con las siglas chb.
- Los radiobutton (JRadioButton) comienzan con las siglas rbt.
- Las tablas (JTable) comienzan con las siglas tbl.

3.3 Pruebas de software

Las pruebas de software son un conjunto de herramientas, técnicas y métodos que evalúan el desempeño de un programa. Involucran las operaciones del sistema bajo condiciones controladas y evaluando los resultados, es por eso que la realización de las mismas a los software es un factor de vital importancia. Antes de liberar el producto es necesario tener la certeza de que este se encuentra libre de errores, aunque se considera que no se puede estar 100% seguro de esto. Pero una cosa si es cierta, mientras más pruebas se le realicen al software más cerca se podrá estar de lograr un producto con la calidad esperada por los usuarios. De las pruebas se plantean varias normas que pueden servir acertadamente como objetivos de las mismas. Probar es un proceso de ejecución de un programa con la intención de descubrir un error. Una prueba tiene éxito si se descubre un error no detectado hasta entonces. [24]

La Prueba es aplicada para diferentes tipos de objetivos, en diferentes escenarios o niveles de trabajo.
Se distinguen los siguientes niveles de pruebas:

- *Prueba de desarrollador:* prueba diseñada e implementada por el equipo de desarrollo.

- *Prueba independiente:* prueba que es diseñada e implementada por alguien independiente del grupo de desarrolladores. Una vista de la prueba independiente es la prueba independiente de los stakeholder, que son pruebas basadas en las necesidades y preocupaciones de los stakeholders.
- *Prueba de unidad:* prueba enfocada a los elementos testeables más pequeño del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera.
- *Prueba de integración:* prueba ejecutada para asegurar que los componentes en el modelo de implementación operen correctamente cuando son combinados para ejecutar un caso de uso.
- *Prueba de sistema:* pruebas que se hacen cuando el software está funcionando como un todo.
- *Prueba de aceptación:* prueba de aceptación del usuario es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales. [25]

Existen varios métodos de pruebas, los más usados son:

- *Caja Blanca:* se basa en el examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que examinen que están correctas todas las condiciones y/o bucles para determinar si el estado real coincide con el esperado o afirmado. Esto genera gran cantidad de caminos posibles por lo que hay que dedicar esfuerzos a la determinación de las condiciones de prueba que se van a verificar.
- *Caja Negra:* pruebas que se llevan a cabo sobre la interfaz del software, por lo que los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta, así como que la integridad de la información externa se mantiene. Esta prueba examina algunos aspectos del modelo fundamentalmente del sistema sin tener mucho en cuenta la estructura interna del software.

El método a utilizar para probar el sistema propuesto es el de Caja Negra, el cual consta de varias técnicas:

- *Técnica de la Partición de Equivalencia:* esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- *Técnica del Análisis de Valores Límites:* esta técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.

- *Técnica de Grafos de Causa-Efecto:* es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

La técnica a emplear es la de Partición de Equivalencia, que se basa en una evaluación de las clases de equivalencia para una condición de entrada. Donde una clase de equivalencia representa un conjunto de estados válidos, inválidos o que no aplican, para determinadas condiciones de entrada. Las condiciones de entrada son valores numéricos específico, un rango de valores, un conjunto de valores relacionados o una condición lógica. Un ejemplo de la condición de entrada para el caso de uso Gestionar Delito es:

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Ley	Combo box	No	Nombre de una Ley
2	Delito	Combo box	No	Nombre del Delito
3	Grado	Combo box	No	Grado del Delito

Tabla 6: Condición de entrada. Caso de uso Gestionar Delito

Arrojando como resultado de la prueba:

Escenario	**Ley**	**Delito**	**Grado**	**Respuesta del Sistema**	**Resultado de la Prueba**	**Flujo Central**
EC 1.1 Adicionar Delito Correctamente	V Ley de protección a los derechos	V Abuso sexual en 1er	V Premeditación y alevosía.	El Sistema adiciona el Delito Correctamente	Satisfactorio.	Menú principal • Inscribir documento, selecciona un trámite y da

Escenario	Ley	Delito	Grado	Respuesta del Sistema	Resultado de la Prueba	Flujo Central
EC 1.2: Existen Datos Incompletos	I	V Abuso sexual en 1er Grado.	NA	El sistema muestra un mensaje de error y pedirá que se complete el dato faltante	Prueba satisfactoria el sistema mostró el siguiente mensaje ”Datos Incompletos”	Menú principal • Inscribir documento, selecciona un trámite y da siguiente. • Seleccionar una persona y Registrar Datos procesales.
	V Ley de protección a los derechos de la mujer.	I	NA	El sistema muestra un mensaje de error y pedirá que se complete el dato faltante	Prueba satisfactoria el sistema mostró el siguiente mensaje ”Datos Incompletos”	

Tabla 7: Resultado de la prueba. Caso de uso Gestionar Delito

Las condiciones de entrada y el resultado de las pruebas de los restantes casos de uso, se muestran en los anexos del presente trabajo. (Ver Anexo 5)

3.4 Validación del diseño

El desarrollo de software es algo muy complejo y la medida de su calidad real no es automatizable. Por esta razón la aplicación de métricas de calidad para evaluar el diseño orientado a objeto posee gran importancia. A continuación se presenta un estudio que brinda un esquema sencillo de implementar y que a la vez cubre los principales atributos de calidad de software, siendo esto la principal razón de la concepción de las métricas.

Atributos de calidad que se abarcan:

- *Responsabilidad:* Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- *Complejidad del diseño:* Consiste en la complejidad que posee una estructura de diseño de clases.
- *Complejidad de implementación:* Consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
- *Reutilización:* Consiste en el grado de reutilización de presente en una clase o estructura de clase, dentro de un diseño de software.
- *Acoplamiento:* Consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización. Complejidad del mantenimiento. Consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.
- *Cantidad de pruebas:* Consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (Componente, modulo, clase, conjunto de clases, etc.) diseñado.
- *Nivel de Cohesión:* Consiste en el grado de especialización de las clases concebidas para modelar un dominio o concepto específico.

Tamaño operacional de clase (TOC): Esta métrica está relacionada con el número de métodos asignados a una clase. Teniendo en cuenta los siguientes atributos:

Atributo que afecta	**Modo en que lo afecta**
Responsabilidad	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de Implementación	Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC implica una disminución en el grado de reutilización de la clase.

Tabla 8: Atributos para la métrica TOC

Luego de realizado un análisis profundo en cada una de las clases utilizadas, la evaluación de la métrica reflejó lo siguiente:

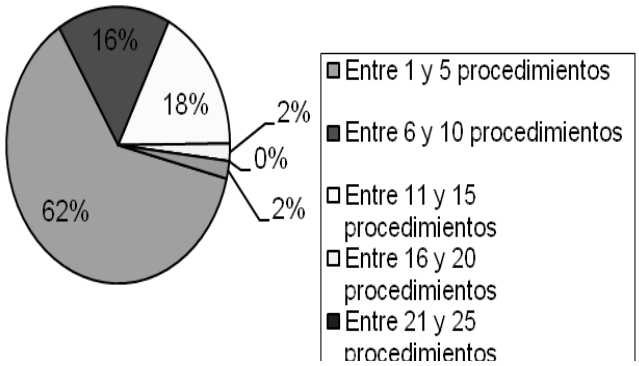


Fig. 12: Representación de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

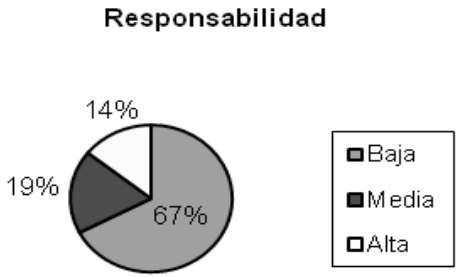


Fig. 13: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad.

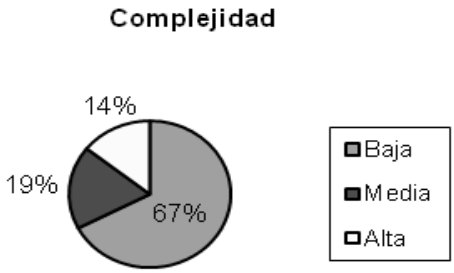


Fig. 14: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de Implementación.

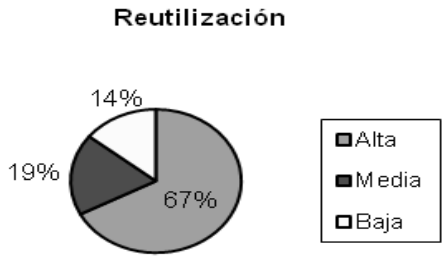


Fig. 15: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica TOC, se puede concluir que el diseño del módulo de Integración tiene una buena calidad pudiéndose observar que el 62% de las clases posee menos cantidad de operaciones que la media registrada en las mediciones. Además el 67% de las clases posee evaluaciones positivas en los atributos (Responsabilidad, Complejidad de Implementación y Reutilización).

Relaciones entre clases (RC): Está dado por el número de relaciones de uso de una clase con otras.

Atributo que afecta	**Modo en que lo afecta**
Acoplamiento	Un aumento del RC implica un aumento del Acoplamiento de la clase.
Complejidad del mantenimiento	Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
Reutilización	Un aumento del RC implica una disminución en el grado de reutilización de la clase.
Cantidad de pruebas	Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

Tabla 9: Atributos para la métrica RC

Luego de realizado un análisis profundo en cada una de las clases utilizadas, la evaluación de la métrica reflejó lo siguiente:

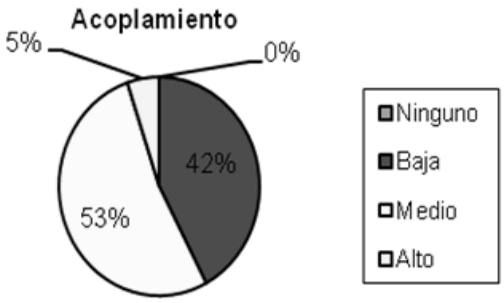


Fig. 16: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento.

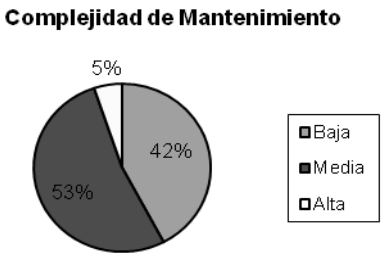


Fig. 17: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento.

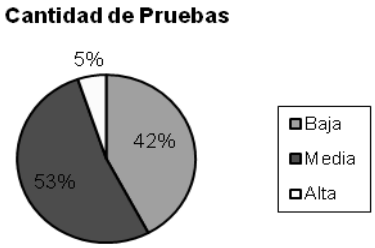


Fig. 18: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.

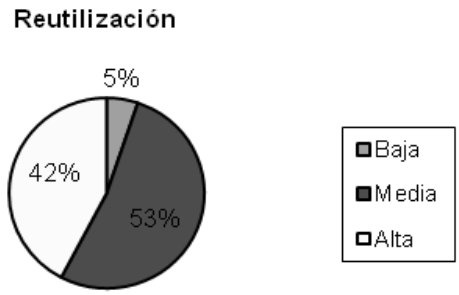


Fig. 19: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica RC, se puede concluir que el diseño del módulo de Integración tiene una buena calidad ya que el 53% posee índices aceptables en cuanto a Acoplamiento. Así mismo los atributos de calidad Complejidad de Mantenimiento, Cantidad de Pruebas y Reutilización se comportan satisfactoriamente en un 53% de las clases.

En el desarrollo de este capítulo se analizaron diferentes aspectos como el significado de las pruebas de software, sus objetivos y alcance. Esta fase añade valor al producto, todos los programas poseen errores y la fase de pruebas los descubre. Se realizó una descripción de los diferentes test, dentro de los cuales se analizó la prueba de Caja Negra, obteniéndose resultados satisfactorios para la entrada de los datos al sistema. Con la validación de la implementación se determinó que el diseño de la misma estuvo correcto, posibilitando una futura reutilización del código debido al bajo acoplamiento y la alta cohesión, tal como fue mostrado por las métricas utilizadas.

CONCLUSIONES

Una vez finalizado el trabajo se llega a las siguientes conclusiones:

- Se obtuvieron conocimientos teóricos sobre la integración entre sistemas en general, conceptos fundamentales que permitieron interpretar la propuesta de solución.
- Se realizó un análisis acerca de los lenguajes, herramientas y metodologías a utilizar, facilitando el proceso de desarrollo de software.
- Con la realización del diseño se demostró que la arquitectura definida soporta el desarrollo del sistema.
- Se implementaron las funcionalidades del módulo Integración, logrando así la integración entre los subsistemas Solución de Gestión y Centro de Digitalización del proyecto Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela.
- Las pruebas realizadas al sistema propuesto arrojaron que este cuenta con la calidad requerida, demostrando su validez y el cumplimiento de los objetivos propuestos.

RECOMENDACIONES

Se recomienda:

- Realizarle las pruebas de calidad para la liberación al producto.
- Realizar un estudio en vistas a agregarle nuevas funcionalidades al módulo.

REFERENCIAS BIBLIOGRÁFICAS

1. Sheth, A.P., Larson, J.A.(1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Computing Surveys, 22 (3), 183-235.
2. Stead W.W. et al. (2000). Integration and beyond: liking information from disparate sources and into workflow. Journal of the American Medical Informatics Association, 7-2, 135-145.
3. Entorno Virtual de Aprendizaje. [En línea] Septiembre de 2010. http://eva.uci.cu/file.php/102/Curso_2010-2011/Clases/Semana_02/Conferencia_3/Materiales_complementarios/Introduccion_a_RUP_y_UML.pdf.
4. Metodologías de desarrollo de software. [En línea] Enero de 2011. http://www.qualitrain.com.mx/index2.php?option=com_content&do_pdf=1&id=160.
5. Silva Paredes, R.J. Universidad Estatal de Milagros. Metodología SMF. [Online] http://es.scribd.com/doc/43507455/msf.
6. Lenguajes de programacion. [Online] 2008/2009. http://guimi.net/descargas/Monograficos/G-Lenguajes_de_programacion.pdf.
7. Itexa. [Online] Itexa, Argentina, 2011. http://www.itexa.com.ar/web/servicios/tecnologias.html.
8. OK HOSTING . [Online] 2010. http://www.miempresaenlinea.com/web-hosting/php-empresas-con-php-hosting.aspx.
9. Curso de Programación Orientada a Objetos para Ingenieros en Java. [Online] 2009. http://webs.uvigo.es/redes/cursoJava/.
10. Organización Autónoma solo de Ingenieros en Sistema. [Online] 2010. http://oasis.cisc-ug.org/letzhune/cisc/tutoriales/tercero/lenguaje%20prog%20c%23.pdf.
11. García de Jalón, J. and Ignacio Rodríguez, J**.** Escuela Superior de Ingenieros Industriales de San Sebastián. [Online] http://mat21.etsii.upm.es/ayudainf/aprendainf/Cpp/manualcpp.pdf.
12. Shmuller, J. Aprendiendo UML *En 24 Horas.* 2001.

13. Netbeans. [Online] 2011. http://netbeans.org/index_es.html.

14. Grupo soluciones GSINova . [Online] 2011. http://www.rational.com.ar/herramientas/roseenterprise.html.

15. Sparx Systems. [Online] 2011. http://www.sparxsystems.com.ar/EAUserGuide/ea.html.

16. Freedownloadmanager. [Online] 2007. http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/.

17. Gil, F., Albrigo, J. and Do Rosario, J. Universidad de Carabobo. [Online] 2009. https://www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r61632.PDF.

18. PostgresSQL. [Online] 2011. http://www.postgresql.org.es/sobre_postgresql.

19. Jiménez Guadalupe, H. Universidad Autónoma del Estado de Morelos. [Online] www.uaem.mx/posgrado/mcruz/cursos/miic/postgress3.ppt.

20. Pressman. 2001. Ingeniería del software Un enfoque práctico. s.l.: Mcgraw-hill , 2001. 8448132149.

21. Expósito Álvarez, A. y Escobar Requejo, Y. 2008. Diseño e implementación de las capas de Negocio y Acceso a datos de los módulos Salidas Transitorias y Traslados Interpenal del SIGEP. [Digital] La Habana: Universidad de las Ciencias Informáticas, 2008.

22. Astudillo, H. Universidad Técnica Federico Santa María. [Online] 2009. http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.

23. Larman, G**.** UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado. s.l. : Prentice Hal, 2004.

24. Myers, G. 1979. The Art of Software Testing. 1979. ISBN-10: 0471078786.

25. Entorno Virtual de Aprendizaje. [Online] 2011. http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_9/Conferencia_7/Materiales_Basicos/Sobre_la_disciplina_de_Prueba.pdf.

26. Peláez, J C. La Guía de Arquitectura Versión 2.0a. s.l. : Patterns and Practices de Microsoft.

BIBLIOGRAFÍAS

1. Piñeiro, Pedro Y and Arcia Carrazana, Maikel. *Informe Tecnológico de la Producción.* La Habana : UCI, 2010.
2. Carlos Olivares, J. Sistema Nacional de Educacion Superior Tecnologica. [Online] http://antares.itmorelia.edu.mx/~jcolivar/courses/dp07a/patrones.pdf.
3. Méndez G., M.C. L. C. Universidad Tecnológica de Tulancingo. [Online] 2011. http://es.scribd.com/doc/51838546/TIPOS-DE-PRUEBAS-DE-SOFTWARE.
4. Raja Prado, E. Sistedes. [Online] 2007. http://www.sistedes.es/TJISBD/Vol-1/No-4/articles/pris-07-raja-ctps.pdf.
5. Informatizate. [Online] http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html.
6. Milagros Díaz Flores, M. Escuela de Ingeniería de Sistemas. [Online] 2009. http://www.usmp.edu.pe/publicaciones/boletin/fia/info49/articulos/RUP%20vs.%20XP.pdf.
7. Limachi Laricano , H.J. Administración y Control de Equipos Informaticos y Tecnológicos de la Universidad Nacional del Altiplano de Puno. [Online] 2007. http://www.vision.ime.usp.br/~hlimachi/images/Informe.pdf
8. González, C.D. . Usabilidad Web. [Online] Mayo 2011. http://www.usabilidadweb.com.ar/cpp.php.
9. Chavarría, R.E. Universidad de Antioquia. [Online] 2006. http://www.slideshare.net/Benedeti/ide-eclipse-breve-gua-201399.
10. IBM. [Online] 2011. http://www-01.ibm.com/software/awdtools/swarchitect/standard.
11. Muller, H. Swing Application Framework. [Online] 2007. http://swinglabs.org/docs/presentations/2007/DesktopMatters/app-framework-talk.pdf.
12. Merseguer , Prof J. Universidad de Zaragosa, España. [Online] 2011. http://webdiis.unizar.es/~jmerse/IS-2/Patrones.pdf.
13. Carrera , C. Universidad Politécnica de Madrid. [Online] 2011. -http://www-lsi.die.upm.es/~carreras/ISSE/patrones_diseno.x2.pdf.

GLOSARIO DE TÉRMINOS

1. **Base de Datos**

   Es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

2. **Componentes**

   El componente es la unidad de construcción elemental del diseño físico. Las características de un componente son:

   - Se define según cómo interactúa con otros
   - Encapsula sus funciones y sus datos
   - Es reusable a través de las aplicaciones
   - Puede verse como una caja negra
   - Puede contener otros componentes

3. **Implementación**

   Proceso por el cual se escribe (en un lenguaje de programación), se prueba, se depura y se mantiene el código fuente de un programa informático.

4. **Métricas**

   Las métricas son un buen medio para entender, monitorizar, controlar, predecir y probar el desarrollo software y los proyectos de mantenimiento.

5. **Módulo**

   A efectos prácticos, hablar de *programas* es lo mismo que hablar de *productos de software* o hablar de *módulos de software*. Cada *módulo* es una parte del sistema, que se instala y funciona por separado, entrelazándose con otros módulos con los que intercambia información.

6. **Nomencladores**

   Tabla de valores definidos por el usuario, que luego no pueden ser modificados.

7. **Objeto**

Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad. Corresponden a los objetos reales del mundo que nos rodea, o a objetos internos del sistema.

8. **Plataformas**

   Entorno de software común en el cual se desenvuelve la programación de un grupo definido de aplicaciones.

9. **SGBD**

   Programas que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada.

10. **Software**

    Se refiere a los programas y datos almacenados en un ordenador.

    Los programas dan instrucciones para realizar tareas al hardware o sirven de conexión con otro software.

    Los datos solamente existen para su uso eventual por un programa.

11. **Software Libre**

    Es la denominación del software que brinda libertad a los usuarios sobre su producto adquirido y por tanto, una vez obtenido, puede ser usado, copiado, estudiado, modificado y redistribuido libremente.