Universidad de las Ciencias Informáticas

Facultad 3

Diseño e implementación del Módulo de Administración y Configuración Local para SIGESAP

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

Autores:

Lisbet Sánchez Moreno.
Ever Luis Hernández Ayala.

Tutor:

Ing. Julio Cesar Prieto Alvarez.

Ciudad de La Habana, junio 2011

“No conozco ningún hecho más alentador que la incuestionable capacidad del hombre para dignificar su vida por medio del esfuerzo consciente”.

Henri David Thoreau

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de junio del año 2011

_____

Firma del autor

Lisbet Sánchez Moreno

_____

Firma del autor

Ever Luis Hernández Ayala

_____

Firma del Tutor

Ing. Julio César Prieto Álvarez

DEDICATORIA

Dedico este trabajo de diploma a mis padres, por su cariño y dedicación.

Lisbet

Dedico este trabajo de diploma a mi familia en general por ser lo más valioso que tengo en la vida.

Ever

AGRADECIMIENTOS

Agradezco especialmente a mis padres quienes han hecho de mí la persona que hoy soy.

A mi mamá que es lo más grande que tengo en la vida, por estar a mi lado y poder contar siempre con su comprensión.

A mi papá por siempre haber confiado en mí, apoyarme y ayudarme cuando más lo he necesitado. Por sus consejos y cada minuto que dedicó a mis estudios, para que este sueño se hiciera realidad.

A mi hermanita linda que ha sido mi ejemplo y siempre ha estado presente en los momentos difíciles.

A mi abuelita que ocupa un lugar especial en mi corazón y que ha hecho muchísimo por mí.

A Yadir por ser esa persona maravillosa en la que siempre puedes confiar, gracias por estar ahí, quererme y ayudarme en todo momento. A su familia que de una forma u otra me han ayudado muchísimo (Alexitín, Yadira, Alexis y Teresa) .Un profundo agradecimiento a Iralda por todos estos años de preocupación y dedicación, por siempre estar pendiente de mí y de todas mis cosas.

A Mima, Pipo y Madaly que siempre que los he necesitado he podido contar con ellos.

A mi compañero de tesis un especial agradecimiento por su apoyo incondicional, a mi tutor muchas gracias por brindarnos su colaboración, a los muchachos y profesores del proyecto que me ayudaron a la realización de este trabajo, especialmente (Lily, Lisday, Kariné, Aldo, Randy y Carlos).

A todos mis amistades en general por tantos momentos agradables, en especial (Arisday, Liennys, Oda, Ropy, Nany, Lisy y Ainadys).

A todos muchas gracias!!!

Lisbet

AGRADECIMIENTOS

Me gustaría empezar agradeciendo a las personas más importantes en mi vida, esas que siempre han estado ahí en los buenos y malos momentos, que me han dado todo su apoyo sin esperar nada a cambio, por eso te agradezco mimi (mamá) por estar siempre ahí haciendo la función de madre y padre no tengo palabras para decir cuánto te lo agradezco, a ti mami (mi abuela) que es la persona más especial en mi vida, a mi abuelo, a mi tía Dany que también ha sido como una madre para mí, a ñeña mi otra tía con la que siempre he podido contar y sé que está muy feliz de que esto haya pasado, a mi abuelo Roy que ha sido un padre para mí y lo quiero mucho, a mis abuelos ,a todos mis tíos, a mi hermanita que la quiero con el alma y es la niña más linda y buena que conozco.

Agradecer a mis compañeros de estudio a los que conozco desde hace años que hemos crecido juntos y he aprendido mucho de ellos (El oju, Paio, Rocky, El Pera, Carlos, Randy, el Rene, Aurelio), a la gente del ponche (Osvaldo, Manuel, Oggit, Arturo…), a aquellos que ya no están en la escuela pero que recuerdo con mucho cariño como mi hermano Michel ,a Coco que es mi hermanito menor aquí, también a los que conozco no hace tanto pero son muy buenos amigos, a la gente del Green Party (Pedro el gordo, Pedritín, Jorgito, Angel, el Wicho, Pupo, Papito), a las tiesas que son muchachitas especiales para mí(Annia, Evelyn, Diana, Milena) y a todas las que han compartido un pedacito de su vida conmigo.

Muy especial agradecimiento a mi compañera de tesis que ha estado ahí conmigo cada minuto para lograr este trabajo, a la gente del proyecto que tanto han ayudado que de verdad son gente buena y los aprecio mucho, al tutor que con sus resabios y malos entendidos aún así ha sabido guiarnos para lograr lo que hoy tenemos.

A todos a los que mencioné y algunos que no, pero no por no recordarlos, gracias por ser como son y por poder contar con ustedes muchas gracias… Desde el Cariño

Ever

RESUMEN

La División de Antecedentes Penales de la República Bolivariana de Venezuela es una institución que controla los procesos de gestión de los residentes en este país y brinda los servicios necesarios al estado acerca de estos ciudadanos. Actualmente en estas oficinas se trabaja con la aplicación SIRCAP, pero la misma no cumple con los requisitos necesarios para el total desempeño de sus funcionalidades, ya que presenta ciertas limitaciones. En el marco de las relaciones entre Cuba y Venezuela se ha concebido el proyecto Solución Tecnológica Integral para la Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela y para su realización se ha creado el software SIGESAP (Sistema de Gestión de Antecedentes Penales), con el objetivo de brindar mayor eficiencia, ritmo y calidad al trabajo que ejecuta la División, protegiendo así la perdurabilidad de la información que se gestiona y preservando la imagen de la seguridad jurídica del Estado Venezolano. Como solución al problema planteado se desarrolló una aplicación de escritorio con tecnología Java, haciéndose uso del framework Swing, usando como guía la metodología de desarrollo de software Rational Unified Process (RUP) y como entorno de desarrollo NetBeans, obteniéndose una aplicación que cumple con los requisitos identificados durante el proceso de Ingeniería de Requerimientos. Específicamente este trabajo de diploma está centrado en el diseño y la implementación del Módulo de Administración y Configuración Local para este proyecto.

Índice de Contenido

INTRODUCCIÓN **1**

CAPÍTULO 1 : FUNDAMENTACIÓN TEÓRICA **4**

1.1 Introducción 4

1.2 Administración 4

1.2.1 Control de acceso *4*

1.3 Configuración 6

1.3.1 Configuración predeterminada *6*

1.3.2 Configuración personalizada *7*

1.3.3 Errores de configuración *7*

1.4 Módulos de administración y configuración local 7

1.5 Metodologías de desarrollo 9

1.5.1 Microsoft Solutions Framework (MSF por sus siglas en inglés) *9*

1.5.2 Proceso Unificado Racional (RUP por sus siglas en inglés) *10*

1.6 Lenguaje de modelado UML 12

1.7 Lenguaje de Programación 12

1.7.1 Python *13*

C++ *13*

1.7.2 C# *14*

1.7.3 Java *15*

1.8 Entorno de Desarrollo Integrado (IDE) 16

1.8.1 Eclipse *16*

1.8.2 NetBeans *17*

1.9 Herramientas CASE 17

1.9.1 Rational Rose *18*

1.9.2 Visual Paradigm *18*

1.10 Sistema Operativo 19

1.11 Frameworks 19

1.12 Tecnologías 20

1.13 Servidor de Aplicaciones 22

1.13.1 Apache Tomcat y GlassFish *23*

1.14 Gestor de Base de Datos 23

1.15 ARQUITECTURA 24
1.16 CONCLUSIONES 26

CAPÍTULO 2 : DISEÑO PROPUESTO 27

2.1 INTRODUCCIÓN 27
2.2 PROPUESTA DEL MÓDULO 27
2.3 MODELO DE LA PROPUESTA DE SOLUCIÓN 27
2.3.1 Requisitos funcionales 27
2.3.2 Requisitos no funcionales 29
2.4 DIAGRAMA DE CASOS DE USO DEL SISTEMA 31
2.5 ARQUITECTURA DE LA SOLUCIÓN 33
2.6 PATRONES DE DISEÑO 33
2.7 DIAGRAMA DE PAQUETES 36
2.8 MODELO DE DISEÑO 36
2.9 ESTÁNDARES DEL DISEÑO 43
2.10 MODELO DE DESPLIEGUE 44
2.10.1 Diagrama de Despliegue 45
2.11 MODELO DATOS 46
2.11.1 Diagrama de diseño del modelo de datos 46
2.11.2 Descripción de las tablas 47
2.12 CONCLUSIONES 48

CAPÍTULO 3 : IMPLEMENTACIÓN Y PRUEBA 49

3.1 INTRODUCCIÓN 49
3.2 ARQUITECTURA DE INFORMACIÓN 49
3.3 ESTÁNDARES DE CODIFICACIÓN 50
3.4 MODELO DE IMPLEMENTACIÓN 52
3.4.1 Diagrama de componentes del sistema 52
3.5 TRATAMIENTOS DE ERRORES 55
3.6 VALIDACIÓN DE LA PROPUESTA DE SOLUCIÓN 55
3.6.1 Métricas para el modelo de diseño 55
3.6.2 Pruebas de caja negra 60
3.7 CONCLUSIONES 63
CONCLUSIONES 64
RECOMENDACIONES 65

BIBLIOGRAFÍA 66
GLOSARIO DE TÉRMINOS 69

Índice de Figuras

Figura 1: Diagrama de Actores del módulo de Administración y Configuración Local. 32
Figura 2: Diagramas de Casos de Uso del módulo de Administración y Configuración Local. 32
Figura 3: Diagrama de paquetes del módulo de Administración y configuración. 36
Figura - 4: Diagrama de clases del diseño Gestionar Nomencladores. 38
Figura 5: Modelo de Despliegue. 45
Figura 6: Modelo de datos del framework. 46
Figura 7: Formulario Gestionar Nomencladores 50
Figura 8: Ejemplo del uso de comentarios. 51
Figura 9*: Ejemplo del tamaño y espacio.* 51
Figura 10: Capa de Presentación. 53
Figura 11: Capa de Lógica de Negocio. 54
Figura 12: Capa de Acceso a Datos 54
Figura 13: Representación de los resultados obtenidos en el instrumento agrupados en los intervalos definidos 57
Figura 14: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad 57
Figura 15: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de Implementación 57
Figura 16: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización 58
Figura 17: Representación en por ciento de los resultados obtenidos en el instrumento agrupados en los intervalos definidos. 59
Figura 18: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento. 59
Figura 19: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de mantenimiento. 59
Figura 20: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas. 60
Figura 21: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización 60

Índice de Tablas

Tabla 1. Descripción de la clase PnlGestionarNomencladores 39
Tabla 2. Descripción de la clase frmAdicionarNomenclador 39
Tabla 3: Descripción de la clase AccionGestionarNomenclador 40
Tabla 4: Descripción de la clase AccionNomenclador 40
Tabla 5: Descripción de la clase AccionEditarNomenclador 41
Tabla 6: Descripción de la clase AccionAdicionarNomenclador. 41
Tabla 7: Descripción de la clase GestorNCNomencladores. 42
Tabla 8: Descripción de la clase GestorNSNomencladores. 42
Tabla 9: Descripción de la clase GestorADNomenclador. 43
Tabla 10: Descripción de la métrica TOC 56
Tabla 11: Descripción de la métrica RC 58

Introducción

El 30 de octubre del 2000 ocurrió un hecho trascendente con la firma del Convenio Integral de Cooperación por los Presidentes de la República Bolivariana de Venezuela y la República de Cuba, que mostró al mundo una nueva forma de relaciones y marcó el camino a seguir por los pueblos de Latinoamérica como una oportunidad de complementar las relaciones de nuestros pueblos. Esto ha dado lugar a la creación de numerosos proyectos en distintas esferas de la Economía y la Ciencia, particularmente la implementación de **Sistemas Informáticos.**

Una activa participación en los proyectos relacionados con la Informática ha tenido la Universidad de las Ciencias Informáticas (**UCI**), materializando su inserción en el Mercado Internacional con la creación del nuevo software a través de la empresa comercializadora (**ALBET.SA**). Las distintas facultades de la universidad dirigen los Centros de Desarrollo de software, como es el **Centro de Gobierno Electrónico** adscrito a la facultad 3, donde se realiza la formación del ciclo profesional de los estudiantes guiada por profesores que laboran en estos proyectos de desarrollo. La misión del mismo es satisfacer las necesidades de los clientes mediante el desarrollo de productos, servicios y soluciones integrales de alta **confiabilidad, calidad y eficiencia**, a partir de un personal altamente calificado, lo cual ha permitido que sea un centro de referencia nacional en el desarrollo de proyectos. Así en este contexto de prestación de servicios, se trabaja actualmente en la **Solución Tecnológica Integral para la Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela**.

La División de Antecedentes Penales, es una institución que tiene el control de los antecedentes penales de las personas que residen en ese país, abarca los procesos de gestión de los datos de los sancionados, servicios de inscripción, certificación e información al estado y a otras instituciones delimitadas por la ley. Este proyecto se centra en la construcción de un software que permita la informatización y perfeccionamiento del trabajo de la División de Antecedentes Penales de Venezuela, que sustituirá además el software que se utiliza actualmente.

El proyecto está dividido en varios módulos, entre ellos el **Módulo de Administración y Configuración Local**. Este contará con funcionalidades que permitan gestionar la autenticación y autorización de los usuarios del sistema, así como el ajuste y adaptación de los parámetros configurables del mismo, maximizando la flexibilidad y escalabilidad. Módulo de vital importancia para la realización de este proyecto.

Teniendo en cuenta lo antes mencionado se plantea el siguiente **problema a resolver**: ¿Cómo diseñar e implementar el Módulo de Administración y Configuración Local de la Solución Tecnológica Integral para la Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela?***.***

Para abordar la temática planteada, a continuación se define:
El **objeto de estudio** de la investigación es el proceso de desarrollo de software.

El **objetivo general** es diseñar e implementar el Módulo de Administración y Configuración Local de la Solución Tecnológica Integral para la Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela.
Se define como **campo de acción** módulos de Administración y Configuración Local**.**
El desarrollo de la investigación es guiado por la siguiente **idea a defender:** Con el desarrollo de un producto informático para la administración y configuración local con las particularidades del proyecto, se logrará una mejor interacción con el sistema.

Para una mejor organización de la investigación se desglosa el objetivo general de la investigación en los siguientes **objetivos específicos:**

- Elaborar el marco teórico de la investigación para un mejor entendimiento de la misma.
- Realizar el diseño del Módulo de Administración y Configuración Local para el proyecto.
- Implementar el Módulo de Administración y Configuración Local para el proyecto.
- Validar el correcto funcionamiento del Módulo de Administración y Configuración Local para el proyecto.

Se llevarán a cabo las siguientes **tareas de investigación:**

- Estudio de los patrones de diseño.
- Diseño de los diagramas de clases y del modelo de diseño.
- Implementación de los casos de uso.
- Realización de las pruebas de caja negra y aplicación de métricas.

Se utilizaron algunos Métodos de investigación científica tales como:

- **Métodos teóricos:**
  - ✓ **Histórico-lógico**: Se realizó un estudio del estado del arte haciendo un análisis de los fenómenos relacionados con la Administración y Configuración en sistemas informáticos implementados en el país y en el mundo, teniendo en cuenta su trascendencia a lo largo de su existencia.
  - ✓ **Analítico-Sintético:** Se realizó un estudio profundo de las metodologías y herramientas a utilizar en el desarrollo de la investigación basándose en la revisión de documentos, artículos e informes. Además de analizar las dimensiones y los hechos asociados.
- **Métodos empíricos:**
  - ✓ **Entrevistas**: Este método se desarrolló con el personal que participó en Venezuela en las entrevistas, lo cual nos permitió acceder a la información necesaria. Además de encuentros que se tuvieron con personas de otros proyectos para hacer el estudio del estado del arte.

Estructura capitular:

Capítulo I: Fundamentación teórica. En este capítulo se realizará un estudio del estado del arte del trabajo, así como el análisis de las metodologías, herramientas y tecnologías, que se utilizarán para desarrollo del sistema.

Capítulo II: Diseño de la solución propuesta. Se analizarán los elementos fundamentales a tener en cuenta en el desarrollo del producto, realizando una valoración crítica del diseño para el módulo. Principalmente los diagramas de clases.

Capítulo III: Implementación y Pruebas. En este capítulo se describe la implementación del módulo, así como las garantías de su funcionamiento de acuerdo con lo previsto.

Capítulo 1. Fundamentación teórica

Capítulo 1: Fundamentación Teórica

1.1 Introducción

Este capítulo ofrece el marco teórico en que se desarrolla el trabajo, abordando temas como la administración, el control de acceso y la configuración, lo que permitirá un mejor entendimiento de los conceptos que existen sobre el tema en el cual se enmarca el problema a resolver. Además, se efectuará el estudio del ambiente de desarrollo, ofreciendo una comparación entre las metodologías, lenguajes y herramientas apropiadas para utilizar en la implementación de la solución, así como sus principales características.

1.2 Administración

Es la ciencia social y técnica que se ocupa de la planificación, organización, dirección y control de los recursos de la empresa, en el caso específico de la informática, esta se centra en controlar los accesos para proteger la seguridad del sistema. Además de encargase de la gestión del usuario.

1.2.1 Control de acceso

Es el proceso de conceder permisos a usuarios o grupos de acceder a objetos. Está basado en tres conceptos fundamentales: **identificación, autenticación y autorización**. Además incluye identificar y autenticar la identidad de los usuarios o grupos y autorizar el acceso a datos o recursos. Los controles de accesos son necesarios para proteger la **confidencialidad, integridad y disponibilidad** de los objetos, y por extensión de la información que contienen, pues permiten que los usuarios autorizados accedan solo a los recursos que ellos requieren para realizar sus tareas.

Identificación: Acción por parte de un usuario de presentar su identidad a un sistema, usualmente se utiliza un identificador de usuario. Establece además que el usuario es responsable de las acciones que lleve a cabo en el sistema, lo que se relaciona con los registros de auditorías que permiten guardar las acciones realizadas dentro del sistema y rastrearlas hasta el usuario autenticado.

Autenticación: Es la verificación de que el usuario que trata de identificarse es válido, usualmente se implementa con una contraseña en el momento de iniciar una sesión. Existen técnicas que permiten realizar la autenticación de la identidad del usuario, las cuales pueden ser utilizadas individualmente o combinadas.

Autorización: Proceso por el cual la red de datos autoriza al usuario identificado a acceder a determinados recursos de la misma.

1.2.1.1 El Control de Acceso Basado en Roles

Mejor conocido como RBAC (por sus siglas en inglés) es una tecnología que surge para satisfacer las principales necesidades en cuanto a Control de Accesos se refiere. Uno de los problemas principales al ejecutar grandes sistemas en red es la seguridad, ya que la administración es costosa y compleja, y es por ello que surge RBAC, para tratar de minimizar todos estos problemas.

Al inicio, los sistemas manejaron dos tipos diferentes de Control de Accesos, uno llamado Control de Acceso Discrecional (DAC, por sus siglas en inglés), en el que el usuario es quién decide como proteger el sistema, mediante Controles de Acceso impuestos por el sistema, y el otro Control de Acceso Obligatorio (MAC, por sus siglas en inglés), en este tipo de Control de Acceso, el sistema es quién protege los recursos. Sin embargo, actualmente este tipo de problemas han disminuido, gracias a que la seguridad está siendo llevada a cabo por RBAC.

La administración de la seguridad, consiste en que los roles deben asignarse adecuadamente a las diferentes tipos de personas, según sus capacidades y puestos de cada una de ellas.

- Características
  - ✓ Es fácil administrar un sistema asignando roles a los usuarios para así llevar una administración confiable y de bajo coste.
  - ✓ El propietario o administrador del sistema es quien maneja los datos, para así satisfacer las necesidades de la empresa u organización.
- Usuarios, Roles y Permisos

Para que un sistema sea confiable se debe de tener en cuenta cuáles podrían ser los caminos para llegar a la confiabilidad, los niveles que existen en los usuarios (por ejemplo: usuario y súper usuario), y los permisos o privilegios que dichos usuarios puedan llegar a tener.

La seguridad es muy difícil de mantener, ya que muchas veces administrar sistemas se vuelve complicado y por lo tanto de alto coste, además de que no todas las aplicaciones funcionan, sin embargo es muy probable que el rendimiento final sea mejor.

Cuando en una organización se cambia a un usuario a otro puesto, éste tiene que cambiar de permisos y responsabilidades, esto es difícil y llega a ser de alto coste. Estos problemas pueden ser evitados por RBAC debido a que son los roles del usuario los que hacen que se tenga acceso al sistema, en vez de que sea la identificación del usuario. Por lo tanto, identificamos entonces que un usuario es la persona que está en una organización, cuya labor básicamente es cumplir una función, en este caso un rol o múltiples roles.

Cada Rol realiza la función que tiene que desempeñar, y puede tener a uno o más usuarios asignados a su cargo. Los permisos determinan los datos y aplicaciones que pueden ser accedidas, a cada rol se le asignan diferentes privilegios, los cuales se le son asignados dependiendo de las capacidades que se tengan para ejecutar el rol.

En el proyecto se ha tenido en cuenta lo antes expuesto por lo que se va a utilizar RBAC para el control de acceso.

1.3 Configuración

Es un conjunto de datos que determina el valor de algunas variables de un programa o sistema operativo, estas opciones generalmente son cargadas en su inicio y en algunos casos se deberá reiniciar para poder ver los cambios, ya que el programa no podrá cargarlos mientras se esté ejecutando, si la configuración aún no ha sido definida por el usuario (personalizada), el programa o sistema cargará la configuración por defecto (predeterminada).

1.3.1 *Configuración predeterminada*

La configuración predeterminada es la que no se ha definido aún, generalmente no es la más recomendada, ya que por ese mismo motivo se le da la posibilidad al usuario de modificarla, una configuración predeterminada tiene que estar preparada para:

- Usuarios de todas las edades y ambos sexos.
- Generalmente en inglés.
- Nivel gráfico medio.
- Seguridad media.

Esta configuración pretende ser lo más adaptable posible, pero siempre es mejor poseer una configuración personalizarla para adaptarla a las necesidades.

1.3.2 *Configuración personalizada*

Una configuración personalizada es la definida especialmente por el usuario, esta es guardada generalmente en un archivo o en una base de datos, puede estar cifrada para que solo se pueda modificar por el programa a configurar, o puede ser texto plano para que también se pueda modificar sin depender del programa.

1.3.3 *Errores de configuración*

Un error de configuración es generado por una escritura incorrecta de las líneas del archivo de configuración o que el hardware este limitado a una configuración que no requiera de tantos recursos como esta, esto conlleva a una ejecución defectuosa del programa o sistema operativo o a la imposibilidad de ejecutarse. Para evitar errores de configuración, es importante leer los requerimientos mínimos de una configuración y que estos sean iguales o estén por debajo de los del hardware.

1.4 Módulos de administración y configuración local

La **administración** en sistemas informáticos surgió a partir de la necesidad de establecer determinados niveles de accesibilidad y seguridad a la información que se manejan en estos. La seguridad es uno de los principales elementos que se tienen en cuenta a la hora de diseñar un módulo de administración, pero este es solo una parte del mismo, lo cual se complementa con otros elementos como pueden ser los requerimientos funcionales de la solución y las decisiones arquitectónicas que se consideren. Por otro lado la **configuración local** en las soluciones se ve como la necesidad de configurar determinados valores de la solución que pudieran variar durante la vida útil de estos, permitiendo así que los sistemas informáticos tengan mayor tiempo de explotación.

Unidos estos conceptos conforman un **Módulo de Administración y Configuración Local**, el cual estaría aplicado al proyecto para dar cumplimiento a las funcionalidades necesarias para llevar a cabo el mismo.

En la actualidad se ven los módulos de administración y configuración acordes a un proceso de negocio específicos o muy globales. En el caso de los específicos responden a procesos de negocio de un sistema informático muy propios, los cuales no resolverían los problemas del

sistema, como pudieran ser: los módulos de seguridad de los proyectos: Sistema Automatizado de Gestión Bancaria (SAGEB), Gestión Integral Aduanera (GINA) y Sistema de Gestión Penitenciaria (SIGEP) realizados estos en la UCI. En caso contrario responden a procesos muy amplios que se enajenan del negocio, tal es el caso de: Liferay, que no es específicamente un módulo, pero si contiene elementos que pueden ser de nuestro interés. La complejidad de conformar un componente de software que proporcione la mayor cantidad de funcionalidades necesarias es muy alta, por lo que es común encontrar herramientas que implementen determinados aspectos de seguridad, que se puedan integrar a este en lugar de un módulo concreto.

SAGEB se desarrolló bajo la tecnología Java Enterprise Edition (J2EE) utilizando como frameworks: Dojo, Spring, Springwebflow. Como base de datos utilizó sqlserver2005 y como framework para la persistencia de datos utilizado fue Hibernate.

SIGEP se desarrolló bajo la tecnología Java Enterprise Edition utilizando como framework arquitectónico base a Spring Framework. Aunque como base de datos se utilizó Oracle, el framework para la persistencia de datos utilizado fue Hibernate. En la capa de presentación se utilizó como tecnología Spring MVC.

GINA se desarrolló bajo la tecnología Hypertext Pre-processor (PHP) utilizando como framework arquitectónico base Symfony. Para la base de datos se utilizó Oracle, el framework para la persistencia de datos utilizado fue Propel.

Liferay es un portal de gestión de contenidos de código abierto escrito en Java. Se creó en el 2000 en principio como solución para las organizaciones sin ánimo de lucro. Este posee administración de Usuarios y Herramientas para la administración de organizaciones. Además tiene numerosas ventajas tales como:

- Corre en la mayoría de los servidores de aplicaciones y contenedores de servlets, base de datos y sistemas operativos, con más de 700 combinaciones posibles.
- Compatible con JSR-268.
- Disponibilidad Out-of-the-Box de más de 60 Portlets pre-construidos.
- Construido dentro del Sistema de manejo de contenido (CMS) y Suite Colaborativa.
- Páginas personalizadas para todos los usuarios.

Creado para empresas, Liferay portal provee espacios virtuales donde se puede centralizar, compartir y colaborar.

1.5 Metodologías de desarrollo

Para llevar a cabo un proyecto se requiere de la aplicación de una metodología de desarrollo de software, que aborda el conjunto de pasos y procedimientos a seguir durante la ejecución del mismo.

1.5.1 Microsoft Solutions Framework

Es una flexible e interrelacionada serie de conceptos, modelos y prácticas de uso que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. Microsoft Solutions Framework (MSF) se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. Originalmente creado en 1994 para conseguir resolver los problemas a los que se enfrentaban las empresas en sus respectivos proyectos, se ha convertido posteriormente en un modelo práctico que facilita el éxito de los proyectos tecnológico.

Fases del proceso:

Visión: El equipo y el cliente definen los requerimientos del negocio y los objetivos generales del proyecto. La fase culmina con el hito Visión y Alcance aprobados.

Planeación. Durante la fase de planeación el equipo crea un borrador del plan maestro del proyecto, además de un cronograma del proyecto y de la especificación funcional del proyecto. Esta fase culmina con el hito plan del proyecto aprobado.

Desarrollo. Esta fase involucra una serie de releases internos del producto, desarrollados por partes para medir su progreso y para asegurarse que todos sus módulos o partes están sincronizados y pueden integrarse. La fase culmina con el hito Alcance completo.

Estabilización: Esta fase se centra en probar el producto. El proceso de prueba hace énfasis en el uso y el funcionamiento del producto en las condiciones del ambiente real. La fase culmina con el hito Release Readiness aprobado.

Implantación: En esta fase el equipo implanta la tecnología y los componentes utilizados por la solución, estabiliza la implantación, apoya el funcionamiento y la transición del proyecto, y obtiene la aprobación final del cliente.

Características principales:

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 o 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.

1.5.2 *Proceso Unificado Racional*

Proceso Unificado Racional (RUP por sus siglas en inglés) es la forma disciplinada de asignar tareas y responsabilidades en una empresa de desarrollo (quién hace qué, cuándo y cómo), considerando una infraestructura flexible de desarrollo de software que proporciona prácticas recomendadas probadas y una arquitectura configurable. Hoy en día es el proceso de desarrollo más general que existe, que permite seleccionar fácilmente el conjunto de componentes de proceso que se ajustan a las necesidades específicas del proyecto.

Se podrán alcanzar resultados predecibles unificando el equipo con procesos comunes que optimicen la comunicación y creen un entendimiento para todas las tareas, responsabilidades y artefactos.

También esta metodología busca detectar defectos en las fases iniciales, reducir el número de cambios y realizar el análisis y diseño completo tanto como sea posible. Para conseguir lo que se propone se basa fundamentalmente en el orden y la documentación *(Jacobson, y otros, 2000).*

Los procesos de RUP estiman tareas y horario del plan, midiendo la velocidad de las iteraciones concernientes a sus estimaciones originales. Estas iteraciones tempranas de proyectos se enfocan fuertemente sobre la arquitectura del software. RUP es una de las metodologías robustas más generales, está organizado por 9 flujos de trabajo, 6 principales y 3 de apoyo, a su vez está dividido en 4 fases (Figueroa, y otros, 2007).

Fases del proceso

Inicio: Se describe el negocio y se delimita el proyecto, describiendo sus alcances con la identificación de los casos de uso del sistema. *(Jacobson, y otros, 2000)*.

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 con el alcance definido. *(Jacobson, y otros, 2000)*.

Construcción: Se logra un producto listo para su utilización que está documentado y tiene un manual de usuario. Se obtiene una o varias versiones del producto que han pasado las pruebas. Se ponen estos entregables a consideración de un subconjunto de usuarios.

Transición: La versión ya está lista para su instalación en las condiciones reales. Puede implicar reparación de errores. *(Jacobson, y otros, 2000)*:

Características Principales

Dirigido por casos de uso: Los casos de uso definen lo que el usuario desea a partir de la captura de requisitos y la modelación del negocio por eso ellos guían el proceso de desarrollo, pues los modelos que se obtienen representan la realización de los mismos. *(Jacobson, y otros, 2000).*

Centrado en la arquitectura: La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. El modelo de arquitectura se representa a través de vistas (4+1), o sea, perspectivas del sistema, que logran una abstracción particular en cada uno de los casos. *(Jacobson, y otros, 2000).*

Iterativo e incremental: Propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. Las iteraciones hacen referencia a pasos en los flujos de trabajo y los incrementos, al crecimiento del producto. *(Jacobson, y otros, 2000).*

Se seleccionó RUP como metodología de desarrollo del software **SIGESAP (Sistema Gestión Antecedentes Penales)** debido a las ventajas que presenta, ya que posee una mayor

abundancia de detalles y contenido relacionado con un proceso de desarrollo, mientras que MSF trata conceptos similares de una manera menos detallada. RUP además de ofrecer una amplia orientación al mostrar la técnica de priorización de casos de uso y la estabilización de la arquitectura, es una metodología muy utilizada por la universidad por lo que los integrantes del proyecto poseen gran experiencia en la desarrollo con la misma.

1.6 Lenguaje de modelado UML

Considerado como el lenguaje estándar en el análisis y diseño de sistemas de cómputo, UML es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. Ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. Fusiona notaciones de diversas técnicas para formar una herramienta compartida entre todos los ingenieros de software que trabajan en el desarrollo orientado a objetos.

Se puede aplicar en el desarrollo de software el lenguaje UML entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software como es el caso de RUP, pero no especifica en sí mismo qué metodología o proceso usar.

En la actualidad UML se ha convertido en una práctica que da seguridad a la especificación de los procesos en el desarrollo del software y de gran utilidad para la comunicación y documentación.

UML cuenta con varios diagramas que muestran diferentes aspectos de las entidades representadas como son: los diagramas de estructura, de comportamiento y de interacción, de los de estructura se utilizan en el presente trabajo los siguientes: diagrama de clases, diagrama de componentes, diagrama de despliegue, diagrama de paquetes y de los de comportamiento se utiliza el diagrama de caso de uso.

1.7 Lenguaje de Programación

Los lenguajes de programación son herramientas que nos permiten crear programas, entre ellos se encuentran: Python, C#, C++, PHP, Delphi, Pascal, Java, etc. Una computadora funciona bajo el control de un programa el cual debe estar almacenado en la unidad de memoria; tales como el disco duro. Los lenguajes de programación de una computadora en particular se conocen como código de máquina o lenguaje de máquina.

Estos lenguajes facilitan la tarea de programación, ya que disponen de formas adecuadas que representan los códigos en forma simbólica y en manera de un texto lo que permite ser leído y escrito por personas, a su vez resultan independientes del modelo de computador a utilizar.

1.7.1 *Python*

Ha sido diseñado por Guido van Rossum y está en un proceso de continuo desarrollo por una gran comunidad de desarrolladores. Aproximadamente cada seis meses se hace pública una nueva versión de Python. Esos cambios no son radicales, sino que se le hacen mejoras, tratando de mantener la compatibilidad con versiones anteriormente desplegadas.

Son muchas las características de este lenguaje que lo hacen tan potente, entre las que se pueden mencionar:

- Lenguaje de programación multiparadigma. Permite varios estilos de programación como la Orientada a Objetos y la estructural.
- Python se ejecuta en muchos sistemas operativos como Windows, GNU/Linux y Mac *(Dayley, 2006).*
- Este puede integrarse con muchas plataformas. Además facilita que los programadores de este lenguaje puedan acceder a las librerías de .NET, a través de IronPython *(Dayley, 2006).*
- Es fácil el aprendizaje. La sencillez de las sintaxis brinda facilidad para crear y debuguear programas *(Dayley, 2006).*

A pesar de las facilidades que este lenguaje proporciona, es criticado por algunos desarrolladores. Uno de los aspectos más criticados es que este es débilmente tipado, es decir, una variable que se declara puede tomar lo mismo un valor numérico que una cadena de caracteres.

Otros de los puntos debatidos por los desarrolladores es la importancia que se le atribuye a la identación del código en los programas escritos en este lenguaje. La identación puede causar molestias a la hora de desarrollar pero sin embargo pudiera ser muy importante a la hora de aprender a programar.

C++

El lenguaje C++ es una mejora al lenguaje C. En este se le agregan funcionalidades que C no tenía, tales como: POO, excepciones, sobrecarga de operadores, templetes o plantillas. Existen

ciertas características que lo posiciona entre uno de los más potente para el desarrollo de software. 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 *(Souli,* 2000).

A pesar de estas características este lenguaje tiene muchos problemas que pueden provocar molestias para los programadores. Una de las más notables es que el uso de la memoria queda completamente en manos del programador. El manejo de punteros y memoria permite un mejor control de la memoria y una buena administración de recursos de computadora, pero también puede llevar a colapsar la aplicación ya que todo el manejo de los mismos queda en manos del programador.

1.7.2 *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. Un ejemplo de ello es que no utiliza marcos, tampoco la herencia múltiple, elimina el uso de operadores tales como (::).
- 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*, que permite recorrer colecciones con facilidad**.**
- 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. *(Seco, 2006).*

1.7.3 *Java*

Es uno de los lenguajes más usados para el desarrollo de software. Este ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de lenguajes, como los punteros de C++. Dentro de las características más importantes de este se puede mencionar:

- **Orientado a Objetos**: Como la mayoría de los lenguajes, este cumple con el paradigma de la orientación a objetos. Posee una arquitectura neutral: La compilación del software hecho en java es completamente independiente de la arquitectura de la máquina donde se ejecute. Para java lo único que es verdaderamente dependiente del sistema es la Máquina Virtual de Java (JVM) y las librerías fundamentales que sean usadas. La neutralidad de la arquitectura mide en gran medida el grado de portabilidad de este lenguaje.
- **Seguro**: Se elimina el uso de los punteros con el objetivo de eliminar el uso indebido de recursos en memoria. Además la máquina virtual se encarga de verificar que el software no posea fragmentos de código ilegal (código que falsea punteros, viola derechos de acceso sobre objetos o intenta cambiar el tipo o clase de un objeto) antes de ejecutarlo.
- **Interpretado y compilado a la vez**: Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina (bytecodes), semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, ya que los bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se haya instalado la

máquina virtual de java.

El diseño de Java, su robustez, el respaldo de la industria y su fácil portabilidad han hecho de este uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática. A pesar de las facilidades de estos lenguajes existen otros factores que se debe tener en cuenta para seleccionar un lenguaje de programación para un proyecto de software.

La experiencia acumulada por los especialistas de un proyecto, las licencias para el desarrollo de software, disponibilidad de documentación y herramientas que permitan un desarrollo rápido y con mejor calidad son algunos de estos factores que no se pueden obviar.

Analizando las características de todos estos posibles lenguajes y teniendo en cuenta los factores que anteriormente se mencionan se ha decidido usar Java como el lenguaje de programación para el desarrollo de este sistema. Esto está condicionado, además de las características mencionadas, por la experiencia del personal del proyecto en este lenguaje y su plataforma en general. Las herramientas existentes para esta plataforma, los marcos de trabajo existentes y la documentación de los mismos ratifican la decisión.

1.8 Entorno de Desarrollo Integrado (IDE)

Es un programa compuesto por un conjunto de herramientas para un programador, el cual forma un entorno de programación que ha sido empaquetado como un programa de aplicación; es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. *(Universidad de Oviedo 2003)*

1.8.1 *Eclipse*

Es un IDE multiplataforma con soporte para varios lenguajes de programación, aunque originalmente es para Java, gracias a que presenta una arquitectura que permite la incorporación de plugins.

Eclipse brinda soporte para Subversion, gracias a varios plugins, los más usados son el Subclipse y el Subversive. Quizás el mayor defecto que se le puede señalar al Eclipse es su alto consumo de memoria, porque para que corra depende de la máquina virtual de Java.

1.8.2 *NetBeans*

NetBeans IDE es un proyecto exitoso de código abierto con una comunidad en constante crecimiento en todo el mundo. Además de ser un producto libre y gratuito, sin restricciones de uso.

Proporciona integración entre varias tecnologías java. NetBeans IDE integra un conjunto de tecnologías muy populares en el desarrollo de aplicaciones en Java. La integración entre varias de estas tecnologías es prácticamente transparente para el desarrollador, debido a que este IDE se encarga de realizar esta tarea, o proporciona interfaces gráficas de forma tal que el desarrollo pueda configurar esta integración de forma sencilla.

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.

NetBeans IDE soporta el desarrollo de todos los tipos de aplicación Java (J2SE, J2EE, JME). Entre sus características se encuentra un sistema de proyectos basado en Ant, control de versiones y refactoring. 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 contiene todos los módulos necesarios para el desarrollo de aplicaciones Java en una sola descarga, permitiéndole al usuario comenzar a trabajar inmediatamente. Por todo lo antes mencionado acerca de NetBeans fue seleccionado este como entorno de desarrollo para la aplicación SIGESAP.

1.9 Herramientas CASE

La tecnología CASE (Ingeniería de Software Asistida por Computadora) es utilizada con el fin de automatizar los aspectos clave de todo el proceso de desarrollo de un sistema. Esta proporciona un conjunto de herramientas automatizadas y semiautomatizadas que están desarrollando la ingeniería de software en el mundo. En la actualidad esta tecnología ha permitido evolucionar la actividad de desarrollar software hacia un proceso automatizado,

ayudando a perfeccionar la calidad y la productividad en el desarrollo de Sistemas de Gestión. Entre sus principales objetivos tiene: *(Collado Cabeza, y otros, 2003).*

- Permitir aplicar en la práctica metodologías, agilizando el trabajo.
- Facilitar la realización de prototipos y el desarrollo conjunto de aplicaciones.
- Simplificar el mantenimiento del software.
- Mejorar y estandarizar la documentación.
- Facilitar la reutilización de componentes software.
- Permitir un desarrollo y un refinamiento visual de las aplicaciones, mediante la utilización de gráficos.

1.9.1 *Rational Rose*

Permite completar los flujos fundamentales de RUP y se ha convertido en una de las mejores opciones por la notación estándar que brinda, para especificar, visualizar y construir productos software y sistemas, por lo que está en la avanzada en cuanto al desarrollo de UML. Rose es una herramienta con plataforma independiente que ayuda a la comunicación entre los miembros del equipo, a monitorear el tiempo de desarrollo y a entender el entorno de los sistemas. También Rose permite que los diseñadores puedan modelar sus componentes e interfaces en forma individual y luego unirlos con otros componentes del proyecto. Pero aún con todos estos beneficios presenta desventajas significativas ya que para su utilización requiere de una licencia, de alta capacidad de procesamiento y de un alto costo para su adquisición *(Menéndez, Rosa, 2009).*

1.9.2 *Visual Paradigm*

Es una herramienta de modelado con una interfaz gráfica muy cómoda para el usuario ayudándoles al modelado con el uso de estándares UML. Visual Paradigm está diseñada para desarrollar software con programación orientada a objetos. Busca reducir la duración del ciclo de desarrollo, brindando ayuda a arquitectos, analistas, diseñadores y desarrolladores.

Una de las características más importantes de su uso es que brinda la posibilidad de sincronización del modelo de diseño y el código en todo el ciclo de desarrollo; disponibilidad de múltiples versiones para cada necesidad y es multiplataforma. Visual Paradigm proporciona la facilidad de programar directamente sobre el código fuente generado y a su vez actualizar el diseño con cambios que se realicen en la programación.

Esta herramienta ofrece un entorno de creación de diagramas para UML, un diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad; utiliza un lenguaje estándar común para todo el equipo de desarrollo que facilita la comunicación; presenta capacidades de ingeniería directa e inversa. Se caracteriza además por ser un producto de calidad, soportar aplicaciones web y varios idiomas, generación de código para Java, fácil de instalar, actualizar y presenta compatibilidad entre ediciones.

Teniendo en cuenta todo lo antes expuesto, la experiencia de la UCI en la utilización de esta herramienta y su excelente entorno de trabajo se ha seleccionado la herramienta Visual Paradigm en su versión 3.4 para ser utilizada en el diseño de este proyecto.

1.10 Sistema Operativo

Para la correcta selección del sistema operativo donde se ejecutará la aplicación es necesario analizar algunos aspectos que pueden repercutir en el funcionamiento del sistema, tanto en servidores como estaciones de trabajo. En el caso de las estaciones de trabajo es importante tener en cuenta la interacción con impresora (impresión por lotes y en ambas caras), scanner (corrección automática de la imagen y tracto masivo), Dispositivos de firma digital y lectores de código de barra, por los problemas que estos pudieran traer a la ejecución del sistema.

Independencia tecnológica que se refleja en la legislación de Venezuela. **Canaima** es un sistema operativo Open Source de Venezuela pero tiene problemas de actualización de los paquetes de software, actualización de los drivers y otros elementos imprescindible.

Ubuntu es una distribución Linux que ofrece un sistema operativo predominantemente enfocado a ordenadores de escritorio, aunque también proporciona soporte para servidores. Basada en Debian GNU/Linux, Ubuntu concentra su objetivo en la facilidad y la libertad en la restricción de uso, los lanzamientos regulares (cada 6 meses) y la facilidad en la instalación, por lo que este fue seleccionado por la dirección del proyecto para la realización.

1.11 Frameworks

- **Swing**

Desde hace más de una década, el sistema de componentes gráfico Swing ha acompañado a Java. Basado en trabajos previos de Sun y James Gosling, permite tener un sistema de ventanas, componentes gráficos, independiente del sistema operativo y la librería de dibujo que se tengan disponibles en la máquina destino.

Los componentes son dibujados usando Java, en los últimos años ha tenido una competencia fuerte de SWT de la gente de Eclipse e IBM, que tiene otro "approach": aprovechar los controles del sistema anfitrión.

1.12 Tecnologías

Las tecnologías existentes para el desarrollo de software no son perfectas y están concebidas para las necesidades de sistemas específicos que se pretendan construir. Estas a su vez están representadas por lenguajes de programación, aunque la tendencia de los últimos años demuestra que estas proporcionan gran interoperabilidad con otros lenguajes de programación y tecnologías en general.

Para Java existen tres categorías que encapsulan las tecnologías existentes. Estas categorías son las siguientes:

- **Java Stadard Edition** *(JSE):* Dentro de esta categoría se aglomeran todas las tecnologías existentes en Java para sistemas comunes o estándares.
- **Java Enterprise Edition** *(JEE):* Esta categoría agrupa todas las tecnologías que proporciona Java para la implementación de aplicaciones empresariales.
- **Java Mobile Edition** *(JME):* Esta tecnología contiene las tecnologías que proporciona Java para aplicaciones móviles.

El diseño de la solución estará centrado en el uso de la categoría JEE, especialmente en la tecnología Enterprise Java Beans

- **Enterprise Java Beans (EJB)**

Es una tecnología que permite instalar una solución que tiene un conjunto de componentes en varios sistemas de cómputos, gestionados por varias JVM, facilitando así que los componentes se distribuyan sin restricción alguna. Con la tecnología Enterprise JavaBeans es posible desarrollar componentes que se pueden reutilizar y ensamblar en distintas aplicaciones dentro y fuera de la empresa. Los componentes EJB son manejados por un contenedor EJB, comportándose como una aplicación que gestiona estos componentes.

El contenedor EJB proporciona dos formas de interacción entre los componentes, una es a través de Interfaces Remotas a métodos (RMI, por sus siglas en inglés) y la otra es a través de Servicios Web (WS, por sus siglas en inglés).

Este contenedor también implementa un conjunto de servicios que serán de gran ayuda para solución. A continuación algunos de los servicios implementados por el contenedor EJB:

Seguridad: comprobación de permisos de acceso a los métodos del bean.

Manejo de transacciones: apertura y cierre de transacciones asociadas a las llamadas a los métodos del bean.

Concurrencia: llamada simultánea a un mismo bean desde múltiples clientes.

Servicios de red: comunicación entre el cliente y el bean en máquinas distintas.

Gestión de recursos: gestión automática de múltiples recursos, como colas de mensajes, bases de datos o fuentes de datos en aplicaciones heredadas que no han sido traducidas a nuevos lenguajes/entornos y siguen usándose en la empresa.

Persistencia: sincronización entre los datos del bean y tablas de una base de datos.

Gestión de mensajes: manejo de Java Message Service (JMS).

Escalabilidad: Posibilidad de constituir clúster de servidores de aplicaciones con múltiples hosts para poder dar respuesta a aumentos repentinos de carga de la aplicación con sólo añadir hosts adicionales.

Adaptación en tiempo de despliegue: posibilidad de modificación de todas estas características en el momento del despliegue del bean.

- **APIs**

Java Message Service (JMS)

Es la solución creada por Sun Microsystems para el uso de colas de mensajes. Este es un estándar de mensajería que permite a los componentes de aplicaciones, basados en la plataforma JEE, crear, enviar, recibir y leer mensajes. También hace posible la comunicación de manera síncrona y asíncrona en ambos casos. Existen dos modelos de la API JMS, los cuales son:

Modelo Punto a Punto: Este modelo cuenta con solo dos clientes, uno que envía el mensaje y otro que lo recibe. Este modelo asegura la llegada del mensaje ya que si el receptor no está disponible para aceptar el mensaje o atenderlo, de cualquier forma se

le envía el mensaje y este se encola en una pila del tipo FIFO para luego ser recibido según haya entrado.

Modelo Publicador/Suscriptor: Este modelo cuenta con varios clientes, unos que publican temas(tópicos) o eventos, y los que ven estos tópicos, a diferencia del modelo punto a punto este modelo tiende a tener más de un consumidor.

Java Persistence API (JPA)

Es la API de persistencia desarrollada para la plataforma Java EE. Esta API busca unificar la manera en que funcionan las utilidades que proveen un mapeo objeto-relacional. El objetivo que persigue el diseño de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos y permitir usar objetos regulares (conocidos como POJOs). El principal factor de decisión de esta API es la integración que tiene con la tecnología que se utilizará para el desarrollo de la solución.

API Reflection

Es una herramienta muy poderosa que permite realizar en Java cosas que en otros lenguajes es imposible. Sin embargo, y a pesar de su potencial, es un API bastante desconocido, sobre todo para los principiantes en el mundo Java. Un buen diseño es muy importante en cualquier programa, para ello estamos acostumbrados a utilizar herencia, interfaces, etc. Sin embargo, la infinita variedad de problemas a los que nos podemos tener que enfrentar, hace que estas técnicas no siempre sean suficientes para nuestros propósitos.

1.13 Servidor de Aplicaciones

Un servidor de aplicaciones no es más que un software que se ejecuta en un sistema de cómputo, el cual se encarga de ejecutar ciertas aplicaciones informáticas. Este tipo de software es muy común cuando de aplicaciones Web se trata, lo que no quiere decir que sea el único tipo de aplicaciones que usen estos servidores de aplicación. Un servidor de aplicaciones generalmente gestiona la mayor parte de las funciones de lógica de negocio y de acceso a los datos de un sistema informático. Los principales beneficios de la aplicación de la tecnología de servidores de aplicación son la centralización y la disminución de la complejidad en el desarrollo y el mantenimiento de las aplicaciones. Dentro de los servidores de Aplicación que se pueden encontrar, con gran uso en el mercado de las soluciones informáticas se pueden mencionar Apache, Apache Tomcat, GlassFish e Internet Information Server (IIS) con un papel

protagónico. Apache es muy utilizado en aplicaciones Web, fundamentalmente para aplicaciones desarrolladas en PHP. De igual forma sucede con IIS para la plataforma .Net.

1.13.1 *Apache Tomcat y GlassFish*

Son los servidores de aplicación más comunes para aplicaciones desarrolladas en Java, en ambos casos desarrollados por Sun Microsystems. La diferencia más significativa que existe entre estos es que GalssFish es una colección de contenedores de aplicaciones JEE, entre los que incluye contenedor Web y un contenedor EJB, mientras que Apache Tomcat es solamente un contenedor Web.

Existen alternativas para utilizar Apache Tomcat como contenedor EJB, tal es el caso de Apache OpenEJB. OpenEJB se puede utilizar integrado a Apache Tomcat u otros de manera integración de dos servidores de aplicaciones para complementar las prestaciones de Glassfish, lo que permite seleccionar a Glassfish como servidor de aplicaciones para la solución.

1.14 Gestor de Base de Datos

La necesidad de persistir grandes volúmenes de información referente a los procesos, y unidades documentales asociadas, que se ejecutan en la División de Antecedentes Penales requiere el uso de gestores de Bases de Datos que permita gestionar de manera ágil la información necesaria. La selección debe estar acorde con las políticas que establecen los organismos rectores de la República Bolivariana de Venezuela.

PostgreSQL es un gestor de Base de Datos, Open Source, muy útil cuando se trata de la gestión de grandes volúmenes de datos. Otras razones que consolidaron la selección son las siguientes:

- Permite herencia entre tablas, las operaciones pueden ser polimórficas.
- Las transacciones en PostgreSQL protege los datos de la concurrencia múltiple a través del full transaction processing. El modelo de la transacción usado por PostgreSQL es basado en el Multiversion Concurrency Control (MVCC). Es una tecnología que PostgreSQL usa para evitar bloqueos innecesarios a los usuarios.
- PostgreSQL implementa integridad referencial apoyando las relaciones de llave primaria y foránea así como los disparadores, más conocidos por su nomenclatura en inglés (triggers).

- Permite un gran número de conexiones concurrentes. Por defecto permite 100 pero esta puede ser configurable, solo depende de la RAM del servidor.
- Recorridos de Mapas de Bits. Los índices son convertidos a mapas de bits en memoria cuando es apropiado, otorgando hasta veinte veces más rendimiento en consultas complejas para tablas muy grandes.
- Particionamiento de Tablas. El optimizador de consultas es capaz de evitar recorrer secciones completas de tablas grandes, a través de una técnica conocida como Exclusión por Restricciones. Esta característica mejora tanto el rendimiento como la gestión de datos para tablas de varios gigabytes.
- Bloqueos Compartidos de Registros: El modelo de bloqueos a través de la adición de candados compartidos a nivel de registro para llaves foráneas. Estos candados compartidos mejoran el rendimiento de inserción y actualización para muchas aplicaciones Procesamiento de transacciones en línea (OLTP, por sus siglas en inglés) de gran concurrencia.

Otras alternativas fueron **SQL Server, Oracle y MySQL**.

En los casos de SQL Server y Oracle, a pesar de ser unos gestores muy potentes, son herramientas privativas y su adquisición es muy costosa y provoca un gasto innecesario.

MySQL es muy rápido y robusto para aplicaciones pequeñas y medianas. El objetivo fundamental de MySQL es la velocidad por lo que le resta importancia a algunas características para el manejo de los datos, como para transacciones, rollback's y subconsultas, por lo que se vuelve menos eficiente que PostgreSQL cuando el volumen de datos es considerable y complejo. Las versiones más actuales son propietarias. Por estas razones se determinó que se utilizaría PostgreSQL como gestor de base de datos.

1.15 Arquitectura

En *An Introduction to Software Architecture*, David Garlan y Mary Shaw sugieren que la arquitectura es un nivel de diseño que se centra en aspectos cuando plantea: *"Más allá de los algoritmos y estructuras de datos de la computación; el diseño y la especificación de la estructura general del sistema emergen como una clase nueva de problema. Los aspectos estructurales incluyen la estructura global de control y la organización general; protocolos de comunicación, sincronización y acceso de datos; asignación de funciones para diseñar*

elementos; distribución física, composición de elementos de diseño; ajuste y rendimiento; y selección entre otras alternativas de diseño (Garlan, 1994).”

La Arquitectura **Cliente-Servidor** se puede definir como una arquitectura distribuida en la cual un cliente realiza peticiones a un servidor, el cual devuelve una respuesta a dichas peticiones.

El Cliente, normalmente, es el que maneja todas las funciones relacionadas con la manipulación y despliegue, estas son desarrolladas sobre plataformas que permiten construir interfaces gráficas de usuario (GUI). Otras de las funcionalidades es acceder a los servicios distribuidos en cualquier parte de una red.

El servidor es el encargado de atender las peticiones de los clientes sobre los recursos que administra. Generalmente, este maneja todas las funcionalidades relacionadas con las reglas del negocio y los recursos de datos.

Las características de esta arquitectura traen consigo un gran número de ventajas como son:

- Bajos costos en hardware para su implementación.
- Permite la integración de sistemas operativos de cualquier índole.
- Al presentar interfaces gráficas orientadas al cliente existe una mayor interacción con el mismo.
- Contribuye además, a proporcionar soluciones locales a las organizaciones, pero permitiendo la integración de la información relevante a nivel global.

En esta arquitectura es necesario tener en cuenta la persistencia de los datos así como la seguridad de los mismos. Esto viene dado por la compartimentación de los recursos en la red.

Garlan y Shaw definen el Estilo Arquitectura en Capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior *(Garlan y otros 1994).*

La principal ventaja de la **arquitectura en n capas** consiste en facilitar la gestión de los cambios en un sistema. Solo basta con trabajar con el nivel que se necesite realizar el cambio sin tener que tocar el resto de los niveles. Este estilo facilita que cualquier cambio que se realice sobre alguna capa sea transparente respecto a las otras capas que se implementan en el sistema. La abstracción de los desarrolladores de los otros niveles es además otra de las ventajas, estos solo necesitan conocer la interfaz que conecta cada una de las capas. Este

estilo arquitectónico es muy usado, ya que permite una gran escalabilidad en los sistemas que se desarrollan, se debe a que facilita la ampliación de los mismos.

La **Arquitectura basada en el desarrollo de componentes** es una de las más usadas para la producción de software comercial. Por lo general, los componentes terminan siendo subsistemas que tienen una funcionalidad específica, generalmente, aparecen en forma de librerías (DLL), pero no quiere decir que esta sea la única forma de verlos. La mayoría de las definiciones que existen giran sobre la idea de que es la implementación de una funcionalidad de un sistema de software, que puede ejecutar varias funcionalidades liberadas. Estos proveen una interfaces para visualizar las funcionalidades que implementan. La reutilización de los componentes es una de las características más importantes de este estilo. Pero estos deben tener características bien definidas como:

- **Identificable**: un componente debe tener una identificación clara y consistente que facilite su catalogación y búsqueda en repositorios de componentes.
- **Accesible sólo a través de su interfaz**: el componente debe exponer al público únicamente el conjunto de operaciones que lo caracteriza (interfaz) y ocultar sus detalles de implementación. Esta característica permite que un componente sea remplazado por otro que implemente la misma interfaz.
- **Servicios invariantes**: las operaciones que ofrece un componente, a través de su interfaz, no deben variar. La implementación de estos servicios puede ser modificada, pero no deben afectar la interfaz.
- **Documentado:** un componente debe tener una documentación adecuada que facilite su búsqueda en repositorios de componentes, evaluación, adaptación a nuevos entornos, integración con otros componentes y acceso a información de soporte.

1.16 Conclusiones

A partir del estudio del estado del arte se comprendió que no existe un módulo de Administración y Configuración Local que cumpla con los aspectos requeridos por el proyecto, por lo tanto se debe crear un módulo que brinde las funcionalidades necesarias. Además en este capítulo después de un análisis se puntualizaron herramientas, tecnologías y metodología a utilizar, el por qué de su elección y sus principales características, así quedando sentadas las bases del desarrollo del producto.

Capítulo 2. Diseño Propuesto

Capítulo 2: Diseño Propuesto

2.1 Introducción

En el presente capítulo se tratarán los temas relacionados con el diseño de la propuesta de solución y se describirán los elementos fundamentales a tener en cuenta en el desarrollo del producto.

2.2 Propuesta del módulo

El módulo a implementar debe garantizar el acceso a la aplicación para proteger la seguridad de la misma y la configuración de los parámetros necesarios. Se encargará de la gestión de roles de usuarios, para que estos tengan acceso únicamente a la responsabilidades asignadas al rol que desempeñan, además de la gestión de las trazas, nomencladores, excepciones, sesiones y otros.

2.3 Modelo de la propuesta de solución

Para modelar la propuesta de solución se describen sus requisitos, tanto funcionales como no funcionales. Además, los requisitos funcionales se modelarán en términos de casos de uso del sistema.

2.3.1 *Requisitos funcionales*

Los requerimientos funcionales de un sistema describen lo que el sistema debe hacer *(Sommerville, 2005).*

- Son declaraciones de los servicios que debe proporcionar el sistema.
- Especifica la manera en que éste debe reaccionar a determinadas entradas.
- Especifica cómo debe comportarse el sistema en situaciones particulares.
- Pueden declarar explícitamente lo que el sistema no debe hacer.

Para el módulo de Administración y Configuración Local se definieron 33 requisitos funcionales, pero para la investigación de este trabajo de diploma solamente se definieron diez los cuales son mencionados en el epígrafe siguiente.

2.3.1.1 Requisitos funcionales del módulo de Administración y Configuración Local

- RF1- Gestionar excepciones del sistema.
- RF2- Almacenar trazas de acciones de usuario.
- RF3- Crear rol.
- RF4- Editar rol.
- RF5- Eliminar rol.
- RF6- Mostrar listado de roles.
- RF7- Buscar Usuario.
- RF8- Adicionar rol a usuario.
- RF9- Eliminar rol a usuario.
- RF10- Buscar sesiones activas del sistema.
- RF11- Desconectar sesión activa.
- RF12- Buscar trazas del sistema.
- RF13- Mostrar detalles de trazas.
- RF14- Buscar excepciones del sistema.
- RF15- Eliminar excepciones del sistema.
- RF16- Mostrar detalles de las excepciones.
- RF17- Mostrar políticas del sistema.
- RF18- Configurar Políticas del sistema.
- RF19- Configurar Políticas de tiempo.
- RF20- Mostrar políticas de tiempo.
- RF21- Configurar política de la fortaleza de la contraseña.
- RF22- Mostrar política de la fortaleza de la contraseña.
- RF23- Configurar política del número de intentos de autenticación fallidos.
- RF24- Mostrar política de número de intentos de autenticación fallidos.
- RF25- Crear puntos de trabajo.
- RF26- Mostrar puntos de trabajo disponibles.
- RF27- Configurar cambios de estatus.
- RF28- Activar aplicación en estación de trabajo.
- RF29- Buscar nomencladores del sistema.
- RF30- Adicionar nomenclador simple.
- RF31- Adicionar nomenclador complejo.
- RF32- Buscar valores de nomenclador.

- RF33- Modificar diseño de la interfaz del sistema.

2.3.2 *Requisitos no funcionales*

Los requerimientos no funcionales son aquellos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste *(Sommerville, 2005).*

- Son restricciones de los servicios o funciones ofrecidas por el sistema (fiabilidad, tiempo de respuestas, capacidad de almacenamiento, etc.).
- Definen propiedades y restricciones del sistema.

2.3.2.1 Requisitos no funcionales del Módulo de Administración y Configuración Local

Usabilidad: El software brindará una ayuda para cualquier tipo de usuario, lo que le permitirá un avance considerable en la explotación de la aplicación en todas sus funcionalidades. Existirán servidores locales con capacidad necesaria para el procesamiento de las solicitudes del conjunto de aplicaciones de las diferentes oficinas los cuales tendrán conexión con los servidores centrales para mantener la actualización de los datos en ambos sentidos.

Fiabilidad: El sistema estará disponible 24 horas al día, 7 días a la semana.

Eficiencia: Tiempo de respuesta promedio de las peticiones que se realizan al servidor no deberá ser mayor de 3 segundos.

Soporte: Soporte para grandes volúmenes de datos y velocidad de procesamiento. Tiempo de respuesta rápido en accesos concurrentes. El sistema debe ser multiplataforma.

Restricciones de Diseño

- El sistema se desarrollará en plataforma Linux.
- El lenguaje de programación es Java.
- La herramienta IDE de desarrollo utilizada será Netbeans.
- La herramienta case utilizada es el Visual Paradigm.
- La herramienta gestor de base de datos es el PostgreSQL.

Requisitos de Apariencia o Interfaz Externa

- El sistema tiene que ofrecer una interfaz amigable y fácil de operar.
- La interfaz contará con teclas de función y menús desplegables que faciliten y aceleren su utilización.
- La entrada de datos incorrecta será detectada claramente e informada al usuario.
- Todos los textos y mensajes en pantalla aparecerán en idioma español.

Requisitos de Seguridad

- Protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.
- La seguridad se establecerá por roles que se le asignarán a los usuarios que interactúen con el sistema brindando solamente aquellas funcionalidades que le competen.
- El sistema mantendrá en todo momento las trazas que se corresponden con las diferentes situaciones críticas que se puedan ocurrir.

Confiabilidad: La herramienta de implementación a utilizar tiene soporte para recuperación ante fallos y errores.

Hardware

- 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.
- Servidores
  - ✓ 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.
  - ✓ 2GB 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.

Software

- 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.
- Servidor
  - ✓ Servidores de Aplicación.

  Máquina Virtual de Java: 1.6.

  Servidor de Aplicación: Glassfish 2.1 (Versión community).

IpTable.

- ✓ Servidores de Bases de Datos.

Gestor de Bases de Datos: PostgreSQL 8.4.

Sistema para Replica de Datos: Chronos.

2.4 Diagrama de casos de uso del Sistema.

Usuario: Usuario que accede al sistema para realizar las funcionalidades a las que tiene permiso. Debe autenticarse en el sistema y puede configurar su perfil de usuario.

Administrador: Usuario del sistema que tiene privilegios de administración y configuración del mismo. Puede gestionar información referente al sistema, funcionalidades, restricciones de acceso y cuentas de usuario.

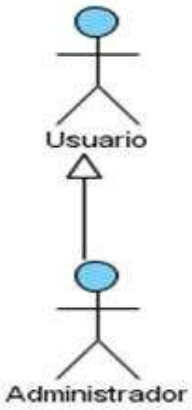

Figura 1: Diagrama de Actores del módulo de Administración y Configuración Local.

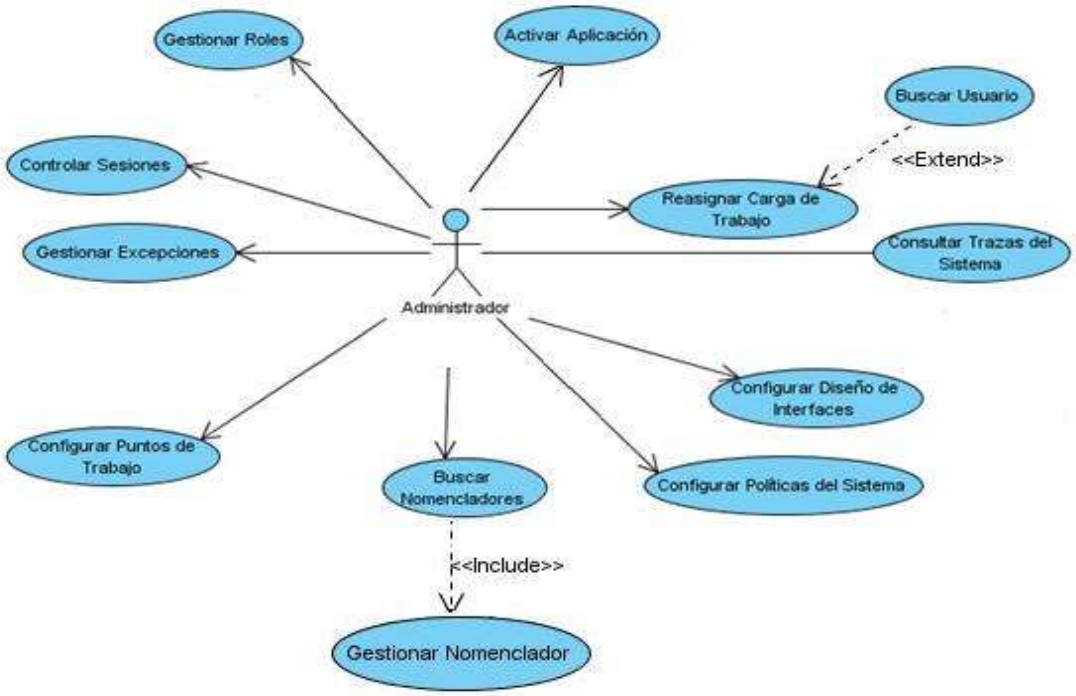

Figura 2: Diagramas de Casos de Uso del módulo de Administración y Configuración Local.

2.5 Arquitectura de la solución

Para la solución se definió utilizar una Arquitectura **cliente–servidor**, distribuidas **n capas** lógicas y **basada en el desarrollo de componentes de software**.

Se dice que es una arquitectura cliente-servidor distribuida en n capas porque contamos con una aplicación cliente que contendría la capa de presentación, una capa de negocio local la cual se encarga de desarrollar la lógica que responde la interacción con la capa de presentación y de hacer las llamadas a la lógica de negocio servidora. Por el lado del servidor tenemos una capa de lógica de negocio EJB compuesta, principalmente, por las interfaces remotas con las cuales el cliente invoca los métodos que necesita de los objetos de negocio publicados, a través del protocolo RMI-IIOP, esta capa de lógica de negocio es la encargada de acceder a la capa de lógica de acceso a datos local desarrollada a partir de EJB locales y esta a su vez realiza las peticiones a la capa de datos ubicada en los servidores de datos. El sentido de la interacción entre las capas es uno de los principios fundamentales de las arquitecturas n capas y nunca debe ser violada. Cada capa se comunica con ella misma y con su siguiente, y devuelve el resultado de sus llamadas a la capa anterior o siguiente. A partir de esta interrelación entre las capas, la necesidad de reutilizar parte de funcionalidades, la necesidad de modular el sistema y encapsular sus contenidos, se dice que es basada al desarrollo de componentes de software.

2.6 Patrones de diseño

Brad Appleton define un patrón de diseño de la siguiente manera: “Un patrón es una semilla de conocimiento, la cual tiene un nombre y transporta la esencia de una solución probada a un problema recurrente dentro de cierto contexto en medio de intereses y competencias”. Dicho de otro modo un patrón de diseño describe una estructura de diseño que resuelve un problema de diseño en particular dentro de un contexto específico y en medio de “fuerzas” que pueden tener un impacto en la manera que se aplica y utiliza el patrón *(Pressman,Roger S).*

La finalidad de cada patrón de diseño es proporcionar una descripción que le permita determinar al diseñador:

- ✓ Si el patrón es aplicable al trabajo actual.
- ✓ Si el patrón se puede reutilizar.
- ✓ Si el patrón puede servir como guía para desarrollar un patrón similar, pero diferente en cuanto a la funcionalidad o estructura.

Los patrones más importantes a usar en el desarrollo.

Según el libro GOF (Gang-Of-Four Book) existen 3 tipos de patrones:

De Creación: abstraen el proceso de creación de instancias.

- ✓ **Singleton** para la creación de una única instancia de las clases que sean necesarias, como los gestores del negocio y en las clases Internacionalización.

Estructurales: se ocupan de cómo clases y objetos son utilizados para componer estructuras de mayor tamaño.

- ✓ **Facade**, para proporcionar una interfaz para la creación de familias de objetos interdependientes o interrelacionados, sin especificar sus clases concretas. El patrón se utilizó para la creación de la clase FachadaGestoresRemotos para facilitar la conexión del los gestores cliente y servidor.

De Comportamiento: atañen a los algoritmos y a la asignación de responsabilidades entre objetos.

- ✓ **Iterator** para proporcionar una forma coherente de acceder secuencialmente a los elementos de una colección, independientemente del tipo de colección subyacente. El patrón se utilizó para recorrer las estructuras de datos utilizadas.

- ✓ **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.

Los **patrones GRASP**

Patrones Generales de Software para Asignación de Responsabilidades (General Responsibility Assignment Software Patterns). Describen los principios fundamentales de la asignación de responsabilidades a objetos. Entre las responsabilidades más importantes están:

- **Hacer**:
  - ✓ Hacer algo uno mismo.
  - ✓ 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.

Ellos son los siguientes:

- **Bajo Acoplamiento** para estimular la asignación de responsabilidades de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento. El Bajo Acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios y también más reutilizables.

- **Experto** para que cada objeto realice la funcionalidad de acuerdo a la información que domina, la cuestión a la hora de diseñar es asignar responsabilidades a la clase que mayor información posee para cumplir con dicha tarea.

- **Creador** para encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Brinda soporte a un bajo acoplamiento lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Este patrón se utiliza en las factorías.

- **Alta cohesión** para brindar una alta cohesión funcional cuando los elementos de un componente (clase, por ejemplo) colaboran para producir algún comportamiento bien definido.

- **Controlador** para asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones:
  - ✓ El sistema global (controlador de fachada).
  - ✓ La empresa u organización global (controlador de fachada).
  - ✓ Algo en el mundo real que es activo (por ejemplo, el papel de una persona) y que pueda participar en la tarea (controlador de tareas).
  - ✓ Un manejador artificial de todos los eventos del sistema de un caso de uso, generalmente denominados "Manejador<NombreCasodeUso>" (controlador de casos de uso).

Este patrón se utilizó en los gestores, cliente, servidor y acceso a datos.

2.7 Diagrama de Paquetes

Los paquetes son usados para organizar y manipular la complejidad de los modelos largos. Un grupo de paquetes modelan elementos y los diagramas semejantes como el uso de casos, clases, actividades, procesos, estados, etc., y sus diagramas asociados; en tal camino que eso puede ser remitido como uno entero. Los paquetes pueden ser representados en un diagrama, remitido como Diagrama de Paquete.

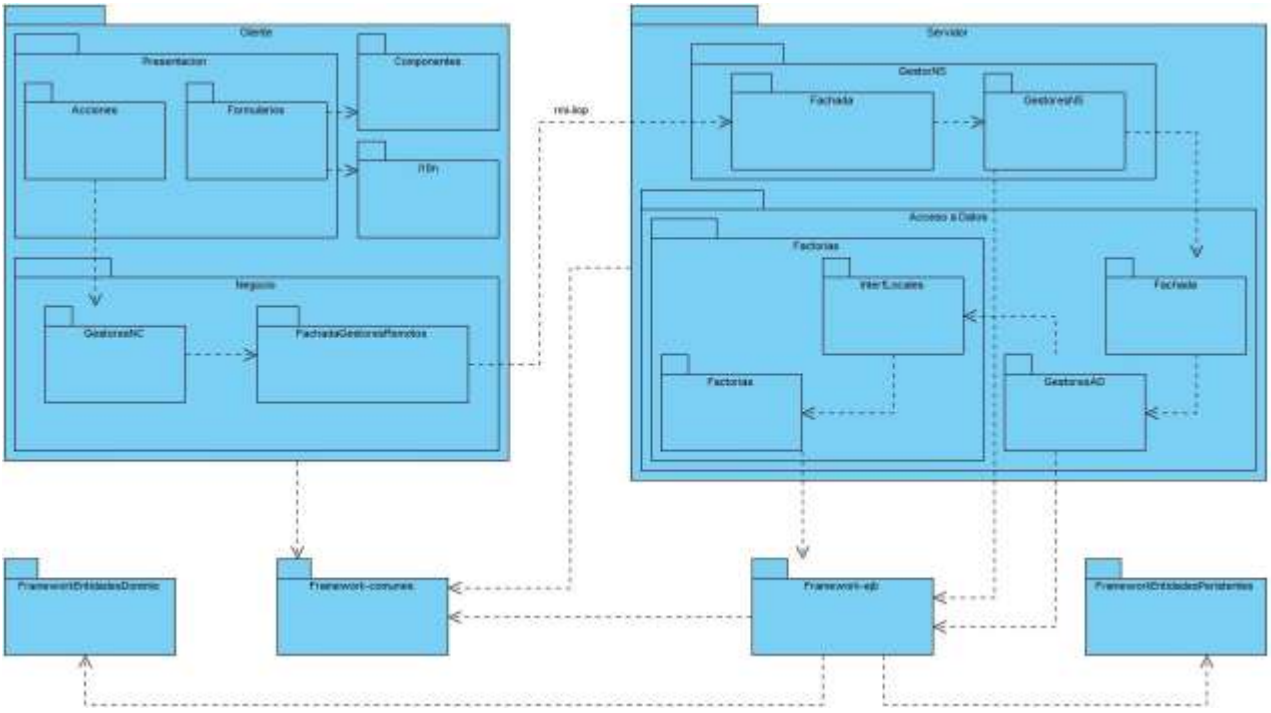

Figura 3: Diagrama de paquetes del módulo de Administración y configuración.

2.8 Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación tienen impacto en el sistema a considerar. Además, sirve como abstracción de la implementación del sistema y es, de este modo, utilizada como una entrada fundamental de las actividades de implementación *(Jacobson, y otros, 2000).*

Como parte del modelo de diseño se realizaron los diagramas de clases y descripciones por caso de uso.

Ver Anexo 1. A continuación se muestra el caso de uso **Gestionar Nomencladores**.

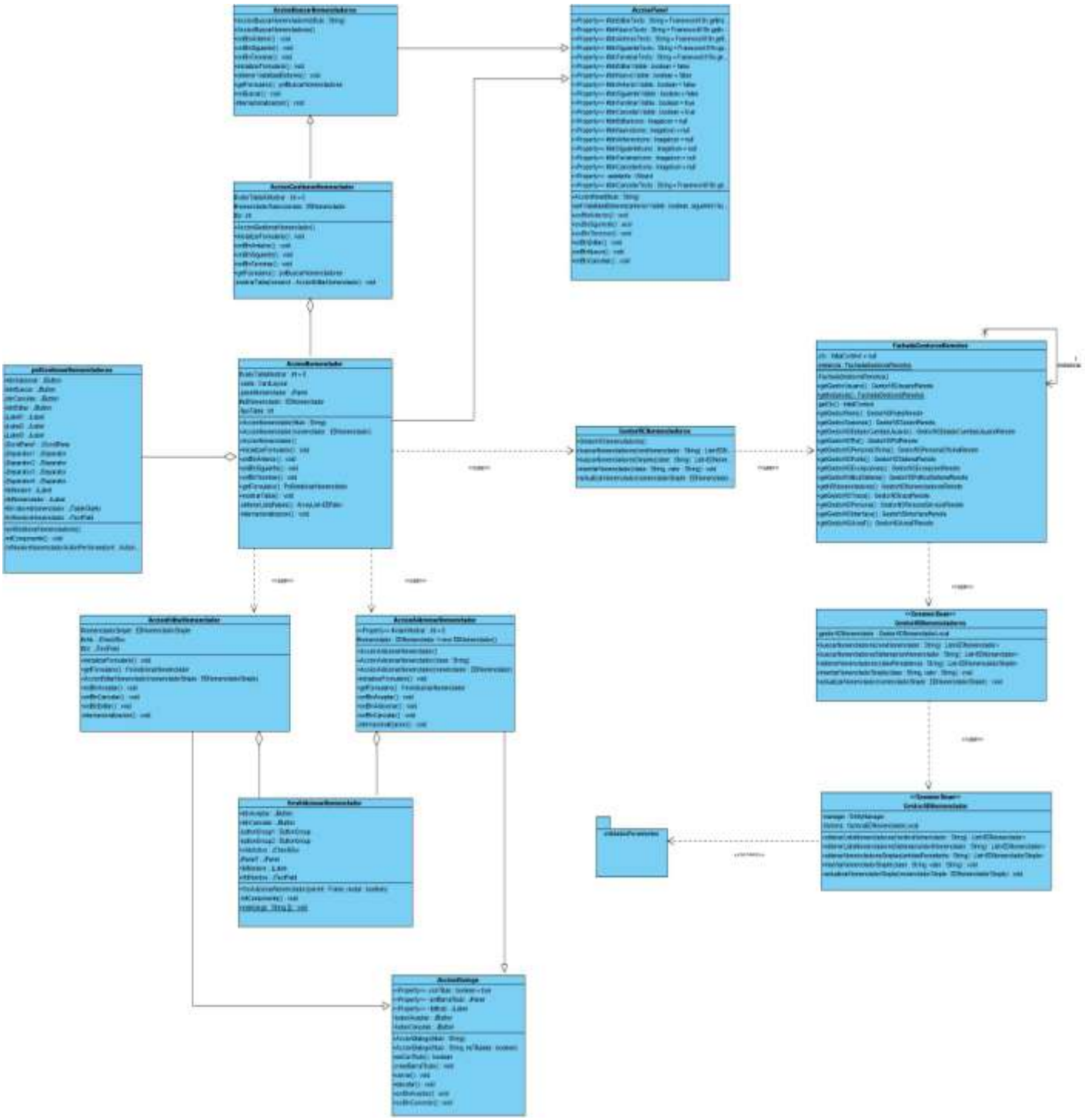

Figura - 4: Diagrama de clases del diseño Gestionar Nomencladores.

Descripción de clases

Tabla 1. Descripción de la clase PnlGestionarNomencladores	
Nombre de la clase: PnlGestionarNomenclador	
Tipo de clase: Interfaz	
Descripción: Clase que representa el formulario Gestionar *Nomencladores* en el que se encuentran los datos necesarios para gestionar los Nomencladores.	
Atributos	**Tipo**
lblNombreNomenclador	JLabel
lblValorNomenclador	JLabel
pnlBuscarNomenclador	JPanel
pnlListaNomencladores	JPanel
tblNomencladorEnte	JTableObjeto
tblNomencladorEnteJudicial	JTableObjeto
tblNomencladorPais	JTableObjeto
tblNomencladorTipoTramite	JTableObjeto

Tabla 2. Descripción de la clase frmAdicionarNomenclador	
Nombre de la clase: frmAdicionarNomenclador	
Tipo de clase: Interfaz	
Descripción: Clase que representa el formulario *Adicionar Nomencladores* en el que se encuentran los datos necesarios para adicionar los Nomencladores.	
Atributos	**Tipo**
btnAceptar	JButton
btnCancelar	JButton
buttonGroup2	ButtonGroup
buttonGroup1	ButtonGroup
chbActivo	JCheckBox
jPanel1	JPanel
lblNombre	JLabel
tfdNombre	JTextField

Tabla 3: Descripción de la clase AccionGestionarNomenclador.	
Nombre de la clase: AccionGestionarNomenclador	
Tipo de clase: Controladora	
Descripción: Clase acción que controla las funcionalidades del formulario Buscar Nomencladores.	
Responsabilidades de la clase:	
Nombre:	mostrarTabla()
Descripción:	Realiza la funcionalidad de mostrar cada tabla en el formulario en dependencia del tipo de nomenclador.
Nombre:	onBtnSiguiente ()
Descripción:	Realiza la acción de invocar a la acción AccionNomenclador

Tabla 4: Descripción de la clase AccionNomenclador	
Nombre de la clase: AccionNomenclador	
Tipo de clase: Controladora	
Descripción: Clase acción que controla las funcionalidades del formulario Gestionar Nomencladores.	
Responsabilidades de la clase:	
Nombre:	mostrarTabla()
Descripción:	Realiza la funcionalidad de mostrar cada tabla en el formulario en dependencia del tipo de nomenclador.
Nombre:	onBtnSiguiente ()
Descripción:	Realiza la acción de invocar a la acción AccionEditarNomenclador
Nombre:	onBtnTerminar()
Descripción:	Realiza la acción de invocar a la acción AccionAdicionarNomenclador
Nombre:	internacionalizacion()
Descripción:	Método que garantiza la internacionalización del formulario.

Tabla 5: Descripción de la clase AccionEditarNomenclador	
Nombre de la clase: AccionEditarNomenclador	
Tipo de clase: Controladora	
Descripción: Clase acción que controla las funcionalidades del formulario Editar Nomencladores.	
Responsabilidades de la clase:	
Nombre:	onBtnEditar()
Descripción:	Realiza la funcionalidad de invocar al método actualizarNomenclador() del gestor GestorNCNomencladores.
Nombre:	onBtnCancelar ()
Descripción:	Cancela la acción de edición.
Nombre:	internacionalizacion()
Descripción:	Método que garantiza la internacionalización del formulario.

Tabla 6: Descripción de la clase AccionAdicionarNomenclador.	
Nombre de la clase: AccionAdicionarNomenclador	
Tipo de clase: Controladora	
Descripción: Clase acción que controla las funcionalidades del formulario Adicionar Nomencladores.	
Responsabilidades de la clase:	
Nombre:	onBtnAdicionar()
Descripción:	Realiza la funcionalidad de invocar al método insertarNomenclador() del gestor GestorNCNomencladores.
Nombre:	onBtnCancelar ()
Descripción:	Cancela la acción de Adicionar nomenclador.
Nombre:	Internacionalización ()
Descripción:	Método que garantiza la internacionalización del formulario.

Tabla 7: Descripción de la clase GestorNCNomencladores.	
Nombre de la clase: GestorNCNomencladores	
Tipo de clase: Controladora	
Descripción: Clase que gestiona la Lógica de Negocio del Cliente para la entidad Nomenclador. Realiza las llamadas correspondientes a la Lógica de Negocio del Servidor para completar las funcionalidades.	
Responsabilidades de la clase:	
Nombre:	buscarNomencladores()
Descripción:	Invoca al método buscarNomencladores() del gestor Nomencladores del servidor
Nombre:	buscarNomencladoresSimples()
Descripción:	Invoca el método buscarNomencladoresSimples() del gestor Nomencladores del servidor
Nombre:	insertarNomenclador()
Descripción:	Invoca el método insertarNomenclador() del gestor Nomencladores del servidor
Nombre:	actualizarNomenclador()
Descripción:	Invoca el método actualizarNomenclador() del gestor Nomencladores del servidor

Tabla 8: Descripción de la clase GestorNSNomencladores.	
Nombre de la clase: GestorNSNomencladores	
Tipo de clase: Controladora	
Descripción: Clase que se utiliza para gestionar la lógica de negocio del servidor para la entidad Nomenclador.	
Responsabilidades de la clase:	
Nombre:	buscarNomencladores()
Descripción:	Invoca al método obtenerListaNomencladores() del gestor Nomencladores del acceso a datos.
Nombre:	buscarNomencladoresSimples()
Descripción:	Invoca al método obtenerNomencladoresSimples() del gestor Nomencladores del acceso a datos
Nombre:	insertarNomenclador()
Descripción:	Invoca al método insertarNomencladorSimple() del gestor Nomencladores del acceso a datos
Nombre:	actualizarNomenclador()
Descripción:	Invoca al método actualizarNomencladorSimple() del gestor Nomencladores del acceso a datos
Nombre:	buscarNomencladoresSistema()
Descripción:	Invoca al método obtenerListaNomencladoresSistema() del gestor Nomencladores del acceso a datos

Tabla 9: Descripción de la clase GestorADNomenclador.	
Nombre de la clase: GestorADNomenclador	
Tipo de clase: Controladora	
Descripción: Clase que se utiliza para gestionar la lógica de acceso a datos para la entidad Nomenclador.	
Responsabilidades de la clase:	
Nombre:	obtenerListaNomencladores()
Descripción:	Método que obtiene todos los nomencladores, o los nomencladores por un criterio de búsqueda.
Nombre:	obtenerNomencladoresSimples()
Descripción:	Método que Obtiene nomencladores simples.
Nombre:	insertarNomencladorSimple()
Descripción:	Método que tiene la funcionalidad de insertar un nomenclador a las entidades persistentes.
Nombre:	actualizarNomencladorSimple()
Descripción:	Método mediante el cual se actualizan los nomencladores que persisten.
Nombre:	obtenerListaNomencladoresSistema()
Descripción:	Método que obtiene la lista de los nomencladores del sistema.

2.9 Estándares del Diseño

El esquema de nombres es una de las ayudas más importantes para entender el flujo lógico de una aplicación. A continuación cómo nombrar cada uno de los elementos del proyecto, teniendo en cuenta que la nomenclatura se debe definir en español.

En los casos que los nomencladores sean compuestos por varias palabras se usará una notación de Camello. La notación Camello consiste en escribir los identificadores con la primera letra de cada palabra en mayúsculas y el resto en minúscula: EndOfFile. Existen dos variantes:

UpperCamelCase: en esta variante la primera letra también es mayúscula.

lowerCamelCase: la primera letra es minúscula.

Se utilizará la primera variante, aplicando la técnica verbo-sustantivo.

- 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 base de datos para cada tabla.
- Los formularios de tipo Diálogo 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 y los del servidor con las siglas GestorNS.
- 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 clases interceptoras de validación terminan con el prefijo Interceptor.
- Las relaciones entre clases usarán el estereotipo <<use>>.

Nomenclatura para los componentes de formularios:

- Los botones (JButton) comienzan con las siglas btn.
- Los campos de texto (JTextField) comienzan con las siglas txb.
- Los campos fecha (Date) comienzan con las siglas fch.
- Para el componente JDateChooser ------ dtc
- Las cajas de texto (JComboBox) comienzan con las siglas cmb.
- Las áreas de texto (JTextArea) comienzan con las siglas txa.
- Los labels (JLabel) comienzan con las siglas lbl.
- Los cuadros de chequeo (JCheckBox) comienzan con las siglas chb.
- Las casillas de selección (JRadioButton) comienzan con las siglas rbt.
- Las tablas (JTable) comienzan con las siglas tbl.

2.10 Modelo de Despliegue

El modelo de despliegue es un modelo de objeto que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. El modelo de despliegue se utiliza como entrada fundamental en las actividades de diseño e implementación debido a que la distribución del sistema tiene una influencia principal en su diseño. *(Jacobson, y otros, 2000).*

Del modelo de despliegue se puede decir que *(Jacobson, y otros, 2000).*

- Cada nodo representa un recurso de cómputo, normalmente un procesador o un dispositivo hardware similar.
- Los nodos poseen relaciones que representan medios de comunicación entre ellos.
- El modelo de despliegue puede describir diferentes configuraciones de red.

2.10.1 *Diagrama de Despliegue*

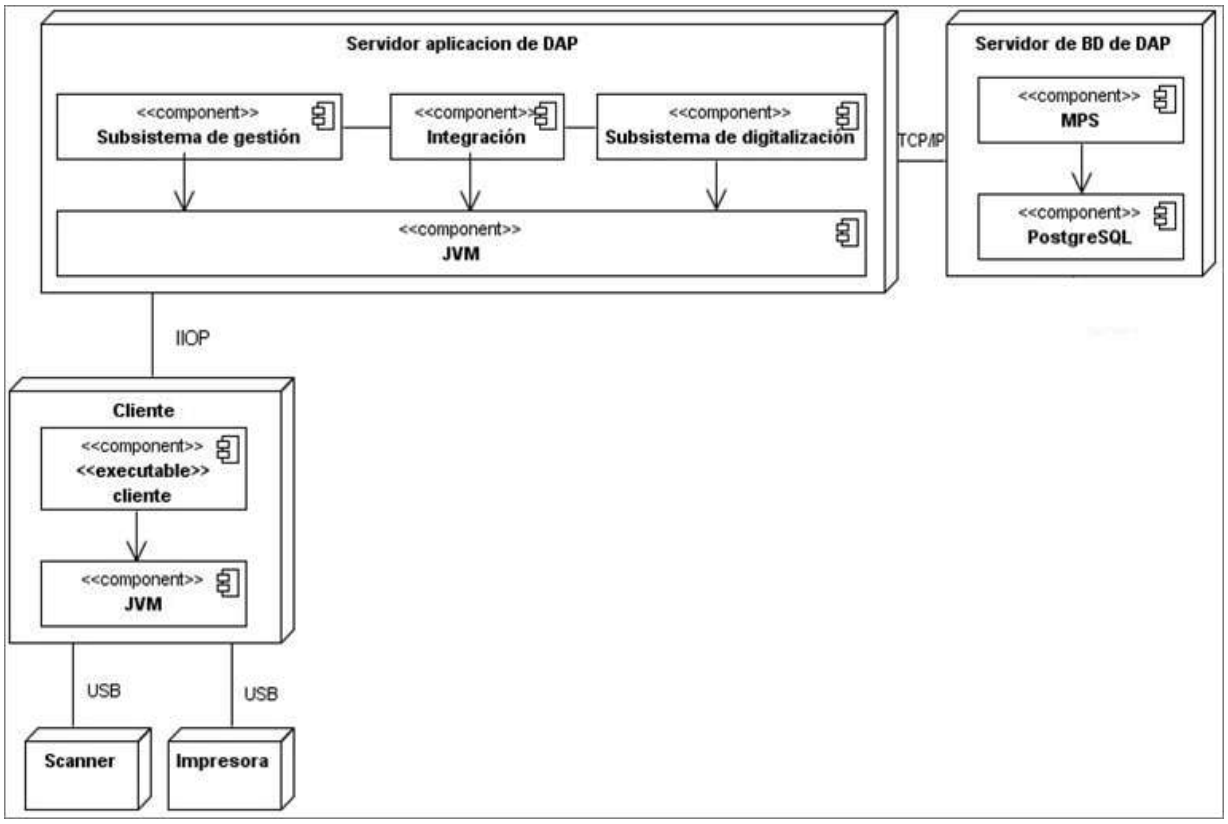

Figura 5: Modelo de Despliegue.

2.11 Modelo Datos

El modelo de datos que muestra la estructura física de las tablas de la base de datos, obtenido a partir del diagrama de clases persistentes.

2.11.1 *Diagrama de diseño del modelo de datos*

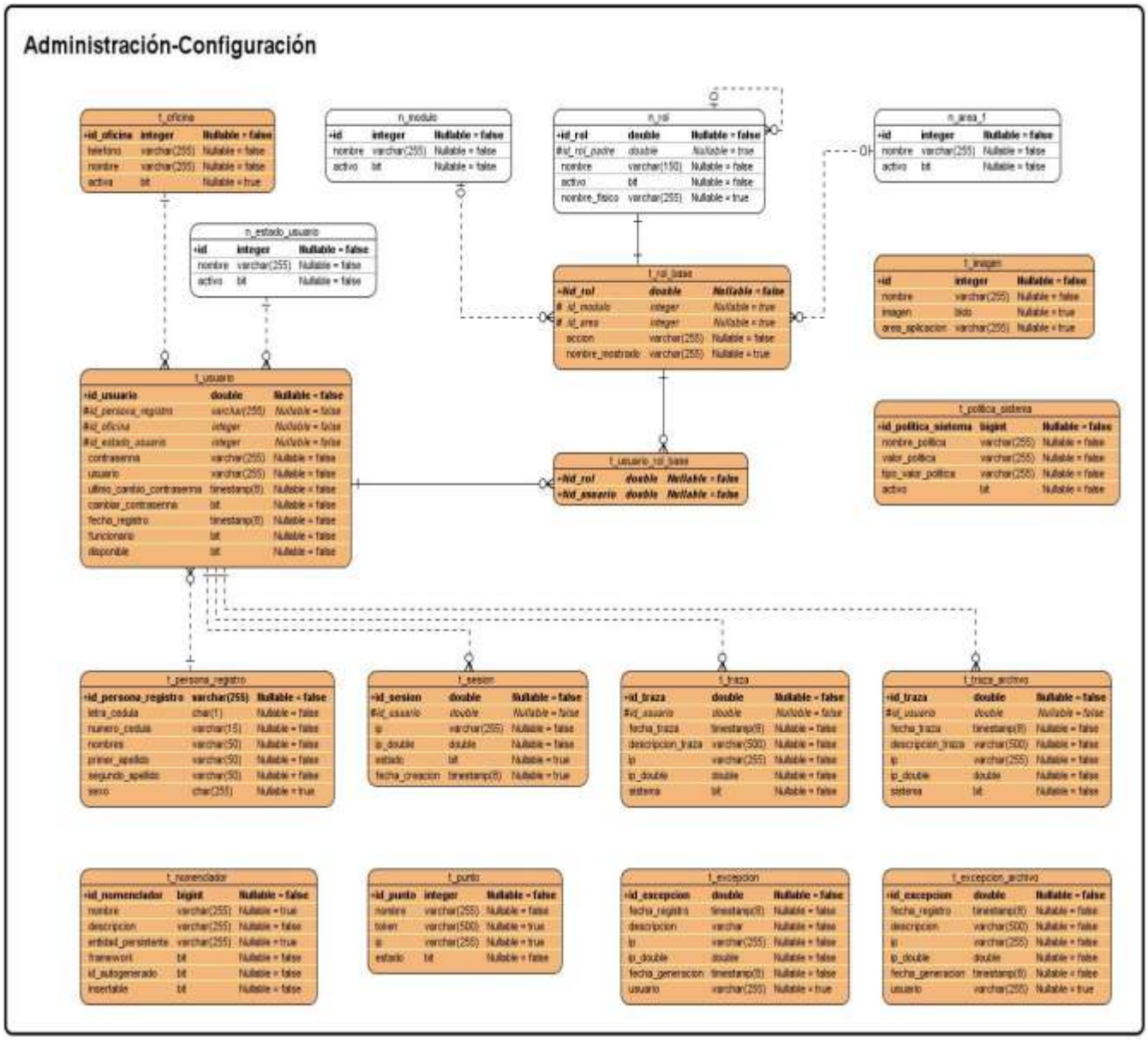

Figura 6: Modelo de datos del framework.

2.11.2 *Descripción de las tablas*

Nomencladores

- **n_area_f.** Almacena las diferentes áreas por la que puede pasar un trámite.
- **n_rol.** Representa todos los roles que existen en el sistema.
- **n_estado_usuario.** Almacena todos los estados de los usuarios.
- **n_modulo.** Representa todos los módulos presentes en el sistema.

Tablas

- **t_oficina.** Entidad que almacena los datos relacionados con cada oficina del local de la división.
- **t_excepcion.** Almacena de forma general las excepciones o errores que puedan ocurrir o dar el sistema.
- **t_excepcion_archivo.** Almacena de forma general las excepciones o errores que puedan ocurrir o dar el sistema pero que lleven tiempo almacenadas y sea raro que se consulten.
- **t_nomenclador.** Entidad que almacena de forma general los nomencladores del sistema.
- **t_punto.** Almacena todos los puntos de los puestos de trabajo en el sistema.
- **t_sesion.** Almacena las sesiones presentes en el sistema.
- **t_traza.** Almacena las trazas de forma general que surgen en el sistema.
- **t_traza_archivo.** Almacena todas las trazas existentes en el sistema pero que lleven tiempo almacenadas y sea raro que se consulten.
- **t_usuario.** Almacena los datos generales de los usuarios del sistema.
- **t_rol_base.** Almacena los datos generales que representan los roles desempeñados por los usuarios del sistema.
- **t_usuario_rol_base.** Representa la relación de entre la tabla usuario y rol_base, especificando que los usuarios pueden tener muchos roles base y los roles base, pueden ser desempeñados por varios usuarios.
- **t_politica_sistema.** Tabla que almacena los datos de configuración de las políticas del sistema.
- **t_persona_registro.** Representa los datos de todas las personas que se encuentran registradas en el sistema que pueden ser usuarios del sistema.
- **t_imagen.** Almacena los datos generales de las imágenes para el diseño de las interfaces.

2.12 Conclusiones

En este capítulo se le dio cumplimiento a los requisitos funcionales y no funcionales, además se obtuvo como artefacto el modelo de diseño y una vez realizado correctamente este, se concluye que, puede desarrollarse sin problemas el modelo de implementación de la solución propuesta.

Capítulo 3. Implementación y Prueba

Capítulo 3: Implementación y Prueba

3.1 Introducción

Este capítulo es orientado a la implementación de la propuesta solución. Se obtendrá el modelo de implementación, este expone los diferentes los componentes que los forman. Además, se verificará el resultado de la implementación, se hará uso de los artefactos generados durante el flujo de trabajo de pruebas, exponiendo resultados obtenidos por diferentes tipos de pruebas realizadas al módulo.

3.2 Arquitectura de información

En el expediente de proyecto se encuentra el documento Pautas del diseño de interfaces, donde se explican normas y estándares definidas por el equipo de desarrollo para diseñar los formularios de la aplicación. A continuación se muestra algunos aspectos recogidos en el documento.

En un formulario, los datos asociados a un mismo criterio se agruparán utilizando paneles con título. El título del panel se escribe en formato de oración, en negrita y sin dos puntos al final.

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).

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.

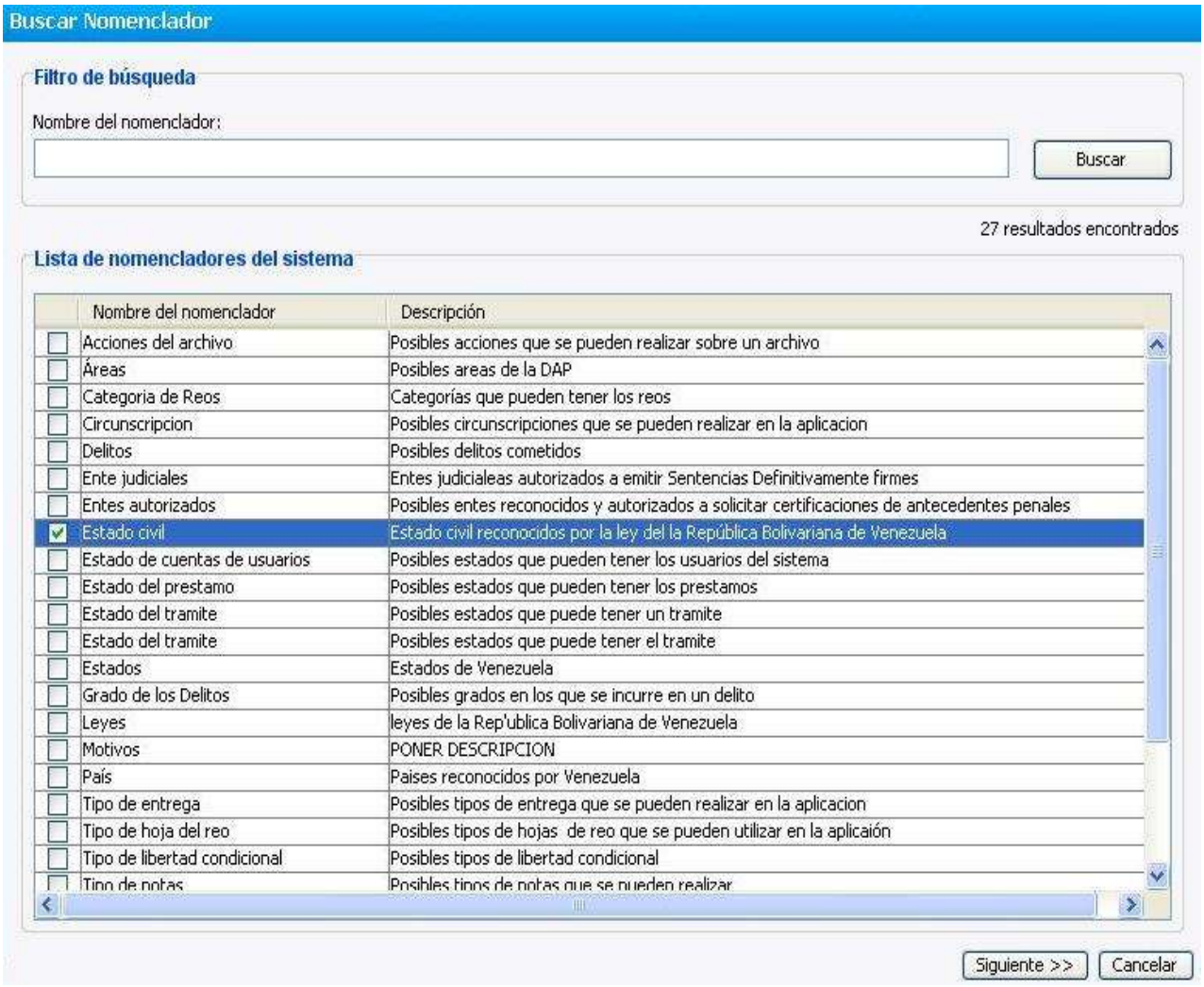

Figura 7: Formulario Gestionar Nomencladores

3.3 Estándares de codificación

En el expediente de proyecto se encuentra el documento que contiene los estándares de codificación utilizados y definidos por el equipo de desarrollo para la implementación. A continuación se muestra algunos aspectos recogidos en el documento.

El uso de comentarios en el código permitirá lograr un mayor entendimiento del mismo en tiempo de mantenimiento y de consulta por otros desarrolladores. Estos comentarios deberán redactarse en español y al principio de cada clase y de cada método.

```
 *
 * @author Ever
 */
public class AccionGestionarNomenclador extends AccionBuscarNomenclador

    protected int valorTablaAMostrar = 0;
    protected EDNomenclador nomencladorSeleccionado;
    protected int id;
```

Figura 8: Ejemplo del uso de comentarios.

Por otro lado no se debe dejar espacios entre los nomencladores de los métodos y los paréntesis que encapsulan los parámetros de dicho método, ya sean globales o instantáneos. De igual forma no se darán espacios entre el paréntesis "(" y el primer parámetro y entre el ultimo parámetro y el ")".

Cuando una expresión no entre en una línea, romperla de acuerdo con estos principios:

- Romper después de una coma.
- Romper antes de un operador.
- Preferir roturas de alto nivel (más a la derecha que el "padre") que de bajo nivel (más a la izquierda que el "padre").
- Alinear la nueva línea con el comienzo de la expresión al mismo nivel de la línea anterior.

Si las reglas anteriores llevan a código confuso o a código que se aglutina en el margen derecho, identar justo 8 espacios en su lugar.

```
public AccionAdicionarNomenclador(String clase) throws Exception{
    super(FrameworkI18n.getInstancia().getEtiqueta("AccionAdicionarNomenclador.titulo"));
    this.nomenclador.setTabla(clase);
}

public AccionAdicionarNomenclador(EDNomenclador nomenclador) throws Exception{
    super(FrameworkI18n.getInstancia().getEtiqueta("AccionAdicionarNomenclador.titulo"));
    this.nomenclador = nomenclador;
}
```

Figura 9*: Ejemplo del tamaño y espacio.*

Buenas prácticas

Las buenas prácticas guiarán el desarrollo hacia buenos principios en la codificación de sistemas empresariales, evitando caer en problemas que atenten con el buen comportamiento del sistema y que favorezcan al cumplimiento de los requerimientos del mismo. A continuación algunas de estas buenas prácticas que han sido identificadas.

- La nomenclatura de las variables booleanas deben describir su estado, siendo este el verdadero. Un ejemplo de estas nomenclaturas pueden ser puedeEliminarse, esGrande; en cambio no podría ser noPuedeEliminarse.
- No se deberá hacer uso de abreviaturas a menos que sea extremadamente necesario. En caso de existir la necesidad es necesario tener en cuenta que esta solo le corresponde a la palabra abreviada, es decir que no existan ambigüedades. Por ejemplo, si utiliza "min." para abreviar "mínimo", hágalo siempre así, y no use también "min." para abreviar "minuto".
- No se debe declarar varias variables en una misma línea, incluso siendo del mismo tipo. Por ejemplo no se debe declarar int a, b;

3.4 Modelo de Implementación

El modelo de implementación describe cómo los elementos del modelo de diseño y las clases se implementan en términos de componentes, como ficheros de código fuente, ejecutables, etc. El modelo de implementación describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y cómo dependen los componentes unos de otros. *(Jacobson, y otros, 2000).*

3.4.1 *Diagrama de componentes del sistema*

“Un componente es el empaquetamiento físico de los elementos de un modelo, como son las clases en el modelo de diseño” *(Jacobson, y otros, 2000)* De ahí que los componentes tengan relaciones de trazas con los elementos del modelo que implementan. *(Jacobson, y otros, 2000).*

Los diagramas de componentes muestran la organización y dependencias entre un conjunto de componentes. *(Ferré Grau, y otros, 2004).*

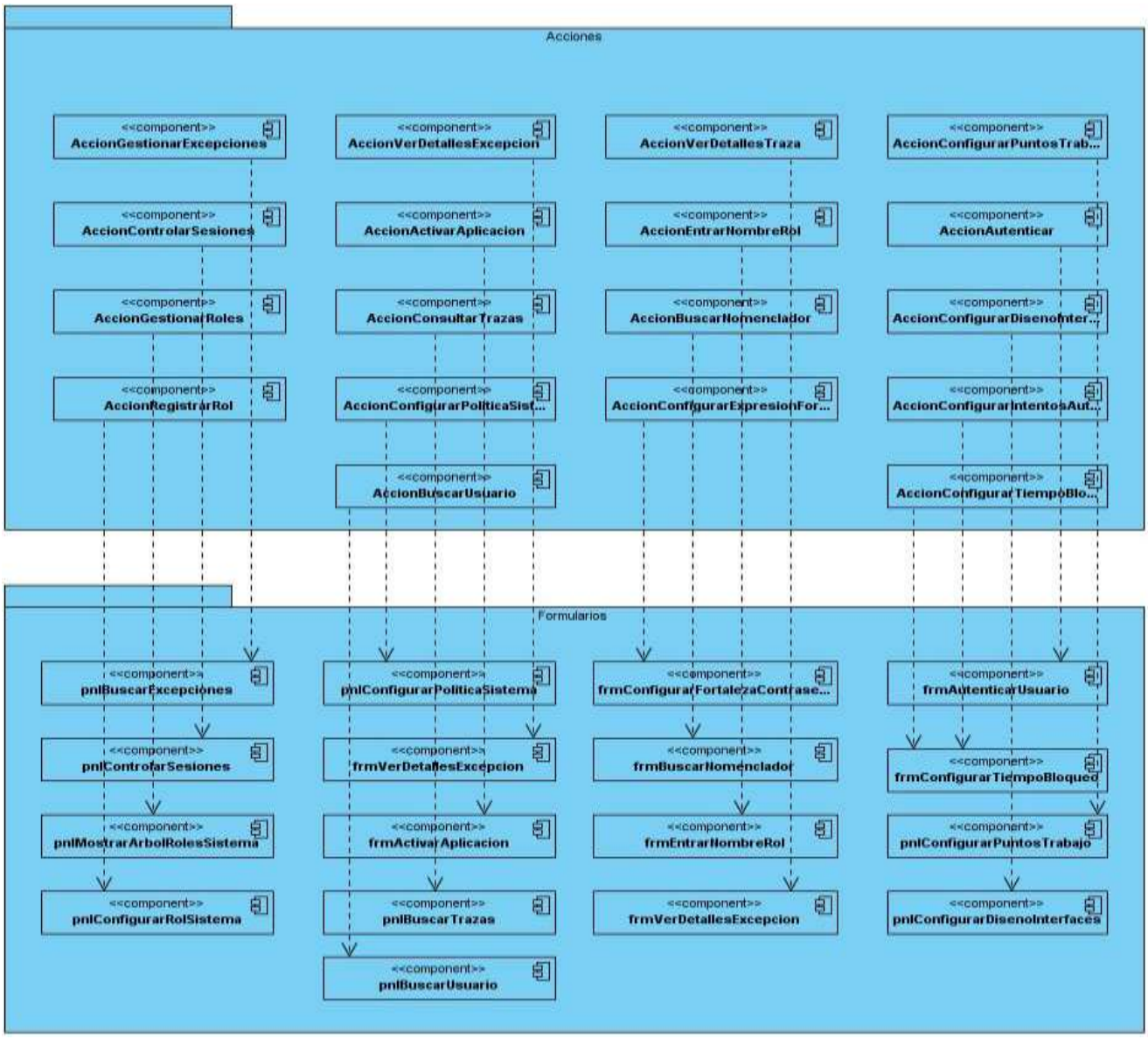

Figura 10: Capa de Presentación.

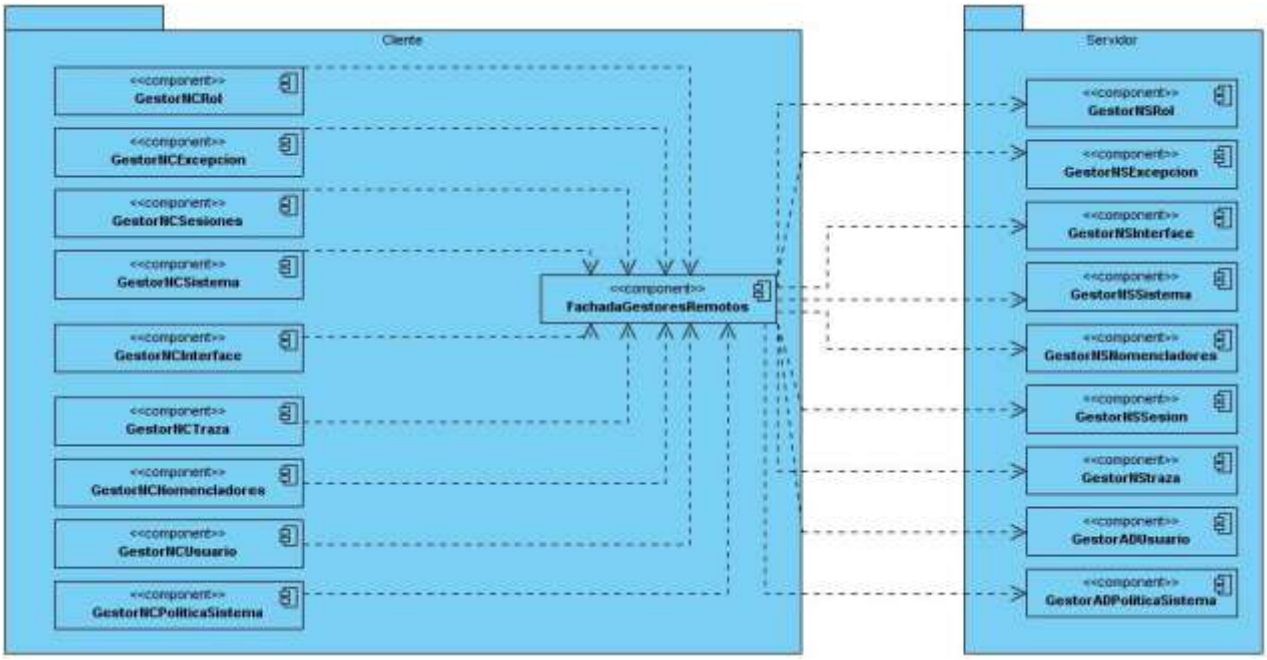

Figura 11: Capa de Lógica de Negocio.

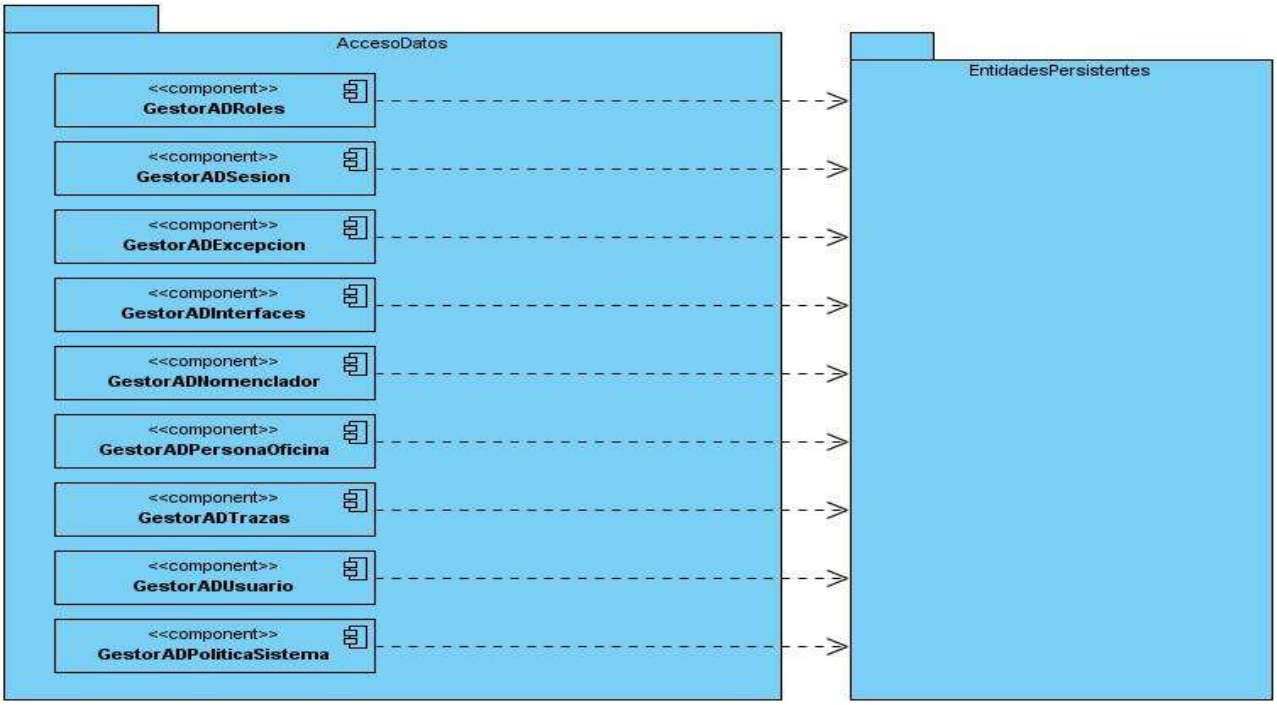

Figura 12: Capa de Acceso a Datos

3.5 Tratamientos de errores

Para garantizar el correcto funcionamiento del sistema se debe tener en cuenta un tratamiento de errores, el sistema captura todas aquellas excepciones que son lanzadas y se le facilita un tratamiento para que no colapse. En el sistema se identifican cada uno de los puntos en los que puede dar el error, los mismos son capturados y gestionados en los métodos implementados, en el cual se definen los posibles errores y cómo deben ser mostrados los mismos de manera entendible para el usuario, además este manejo de errores se realiza en dependencia del tipo de error lanzado. Se utilizó principalmente el Try {} Catch (Exception e) {}. Otros elementos generales del tratamiento de errores son los siguientes:

- La corrección de errores en la introducción de datos será clara y fácil de realizar.
- La entrada de datos incorrecta será detectada claramente e informada al usuario.
- Todos los textos y mensajes en pantalla aparecerán en idioma español.

3.6 Validación de la Propuesta de Solución

3.6.1 *Métricas para el modelo de 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 presentaremos 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 que presente 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 indirectamente 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, módulo, 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. *(Yzquierdo Herrera, y otros, 2007)*

Métrica Tamaño operacional de clase (TOC):

Está dado por el número de métodos asignados una clase.

Tabla 10: Descripción de la métrica TOC

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.

Ver Instrumento y tabla de resultados en (Anexo 2 Instrumento de medición de la métrica Tamaño operacional de clase (TOC)). Luego de realizado un análisis profundo en cada una de las clases utilizadas, la evaluación de la métrica reflejó lo siguiente:

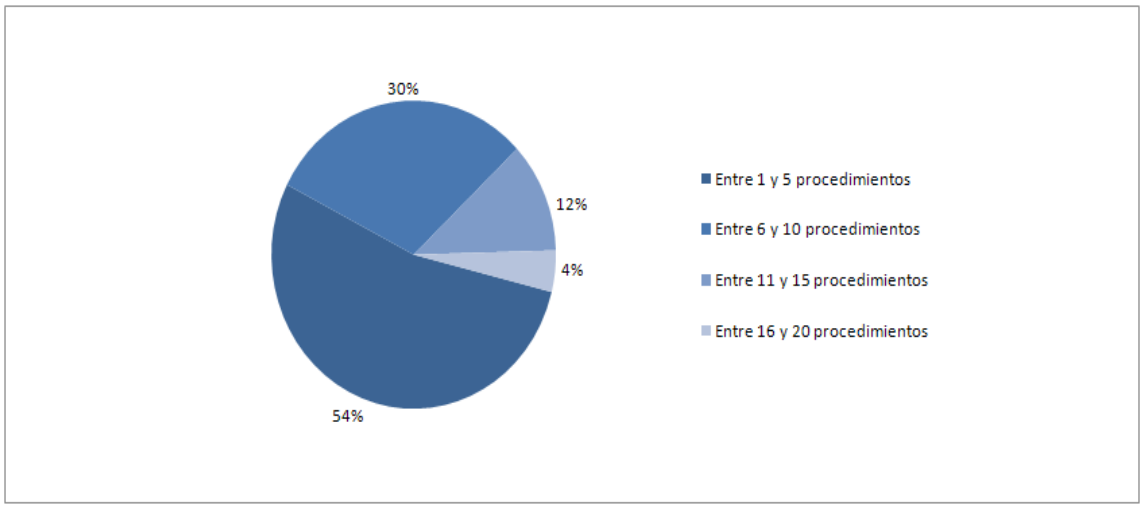

Figura 13: Representación de los resultados obtenidos en el instrumento agrupados en los intervalos definidos

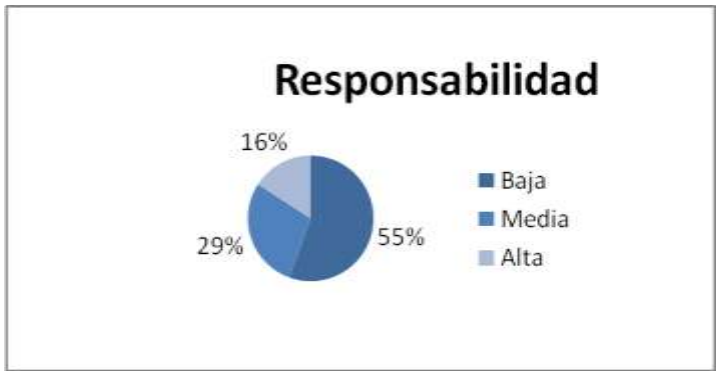

Figura 14: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad

Figura 15: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de Implementación

Figura 16: 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 propuesto está entre los límites aceptables de calidad, pudiéndose observar que el 84% de las clases poseen menos cantidad de operaciones que la media registrada en las mediciones. Además el 55% de las clases posee evaluaciones positivas en los atributos Responsabilidad y Complejidad de Implementación así como la Reutilización con 56%.

Relaciones entre clases (RC):

Está dado por el número de relaciones de uso de una clase con otras.

Tabla 11: Descripción de la métrica RC

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 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.

Ver tabla de resultados en (Anexo 2 Instrumento de medición de la métrica Relaciones entre clases (RC)). Luego de realizado un análisis profundo en cada una de las clases utilizadas, la evaluación de la métrica reflejó lo siguiente:

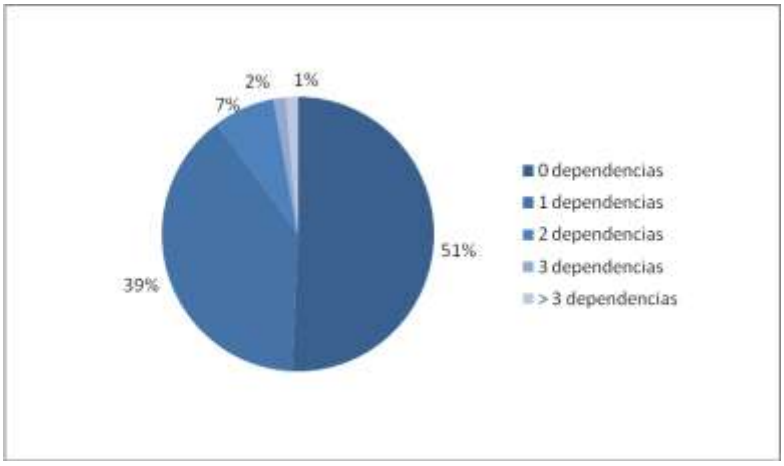

Figura 17: Representación en por ciento de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

Figura 18: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento.

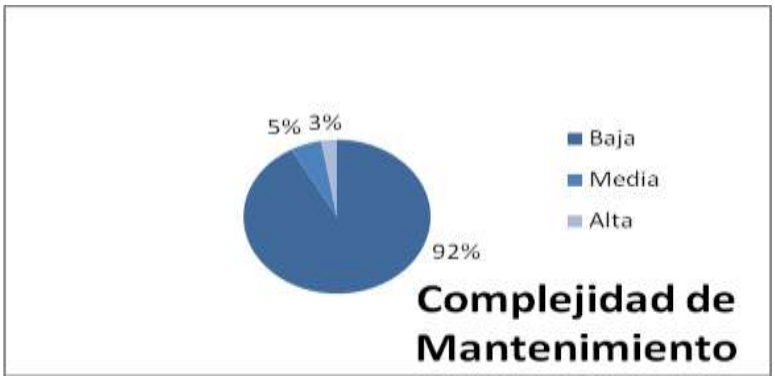

Figura 19: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de mantenimiento.

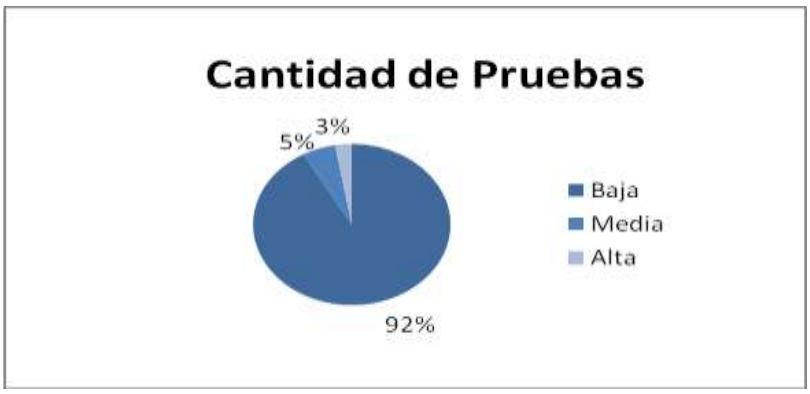

Figura 20: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.

Figura 21: 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 propuesto está entre los límites aceptables de calidad, pudiéndose observar que el 97% de las clases poseen menos de 3 dependencias de otras clases. Además el 17% de las clases no poseen acoplamiento con otras y el 44% posee índices aceptables en cuanto a Acoplamiento. Así mismo los atributos de calidad Complejidad de Mantenimiento y Cantidad de Pruebas se comportan satisfactoriamente en un 92% de las clases y Reutilización en un 79%.

3.6.2 *Pruebas de caja negra*

Las pruebas de caja negra, también denominada prueba de comportamiento, se centran en los requisitos funcionales, permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un

programa. Este tipo de prueba intenta descubrir diferentes errores que los métodos de caja blanca no logran identificar. Las categorías de los errores encontrados por este tipo de prueba son: funciones incorrectas o ausentes, errores de interfaz, estructuras de datos, rendimiento, inicialización y terminación. *(Pressman, 1997)*

La realización de los casos de pruebas tiene como objetivo demostrar al cliente la reacción que corresponderá por parte del sistema luego de realizar alguna acción en el mismo. Para un mejor entendimiento de las respuestas o posibles funcionalidades que brindará el sistema, según la necesidad del usuario.

La descripción de los casos de prueba del sistema se realizó por escenarios y casos de uso. Las mismas son actividades en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación. *(Ingeniería de Software I Conferencia #5)*

Ver Casos de pruebas de los casos de uso críticos del módulo Anexo 3.

Caso de Prueba Gestionar Nomencladores

Secciones a probar en el caso de uso

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC 1: Adicionar nuevo valor de nomenclador	EC 1.1: Adicionar nuevo valor de nomenclador correctamente.	Se introducen los datos del nuevo valor correctamente.
	EC 1.2: Adicionar nuevo valor de nomenclador incorrectamente.	Se introducen campos no permitidos al valor del nomenclador o no se le establece valor alguno.
SC 2: Editar valor de nomenclador	EC 2.1: Editar valor de nomenclador correctamente.	Se introducen los datos del nuevo valor correctamente.
	EC 2.2: Editar valor de nomenclador incorrectamente.	Se introducen campos no permitidos al valor del nomenclador o no se le establece valor alguno.

Descripción de variables

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
[1]	Nombre del delito	Campo de Texto	No	Permite cualquier valor
[2]	Ley	Lista Desplegable.	No	Seleccionar al menos uno
[3]	Activo	Campo de Selección	Si	Puede o no ser seleccionado
[4]	Nombre	Campo de Texto	No	Puede o no ser seleccionado

Matriz de datos

SC 1: Adicionar nuevo valor de nomenclador

Escenario	Nombre del delito	ley	Activo	Respuesta del Sistema	Resultado de la Prueba	Flujo Central
EC 1.1: Adicionar nuevo valor de nomenclador correctamente.	V delito1	Ley 1	Marcado	Mostrará que se registrará la información y se a adicionará.	Satisfactorio	1. Autenticarse. 2. Administración local. 3. Gestionar nomencladores.
EC 1.2: Adicionar nuevo valor de nomenclador incorrectamente.	I Vacío	Ley 1	NA	El sistema mostrará un mensaje de error con los datos del error	Satisfactorio	
	V delito 2	Vacío	NA	El sistema mostrará un mensaje de error con los datos del error.	Satisfactorio	

SC 1: Editar valor de nomenclador

Escenario	Nombre	Activo	Respuesta del Sistema	Resultado de la Prueba	Flujo Central
EC 1.1: Editar valor de nomenclador correctamente.	V Casado	Marcado	Mostrará que se actualiza la información y es editado el nomenclador	Satisfactorio	1. Autenticarse. 2. Administración local. 3. Gestionar

Escenario	Nombre	Activo	Respuesta del Sistema	Resultado de la Prueba	Flujo Central
EC 1.2: Editar valor de nomenclador incorrectamente.	I Vacio	NA	El sistema mostrará un mensaje de error con los datos del error.	Satisfactorio	nomencladores.

Otro elemento que confirmó el alto nivel de calidad que posee el presente trabajo fue la liberación del Módulo de Administración y Configuración Local emitida por el Equipo de Calidad de la Facultad.

3.7 Conclusiones

En este capítulo se realizó el diagrama de componentes estructurado en capas, logrando un mejor entendimiento del sistema desde el punto de vista de implementación. Además, los resultados obtenidos en las pruebas realizadas, ejemplifican la calidad del trabajo efectuado en la implementación del Módulo de Administración y Configuración Local y también se muestra pleno cumplimiento de los requisitos funcionales y no funcionales.

Conclusiones

Con la realización del presente trabajo se logró satisfacer los requisitos del Módulo de Administración y Configuración Local del proyecto Solución Tecnológica Integral para la Automatización y Modernización de la División de Antecedentes Penales de la República Bolivariana de Venezuela y se le dio cumplimiento a los objetivos trazados, obteniendo los resultados siguientes:

- Se alcanzaron conocimientos teóricos sobre la administración y la configuración, conceptos fundamentales que permiten interpretar la propuesta de solución.
- Se obtuvieron los artefactos necesarios para el diseño de acuerdo a la metodología de desarrollo aplicada en el proyecto.
- Se obtuvo la implementación de todos los casos de uso de acuerdo al diseño elaborado previamente.
- Se lograron resultados satisfactorios en cuanto al nivel de calidad que presenta el diseño a partir de la aplicación de métricas para el modelo de diseño orientado a objetos.
- Se alcanzó un alto nivel de calidad en cuanto al funcionamiento de la aplicación, comprobado mediante las pruebas de Caja Negra.

Recomendaciones

Realizar un estudio para determinar si pueden adicionar nuevas funcionalidades al Módulo de Administración y Configuración Local. Identificar qué proyectos similares podrían reutilizar los componentes y la solución.

Bibliografía

- **Booch, Grady, Rumbaugh, Jim y Jacobson, Ivar. 1999.** El Lenguaje Unificado de Modelado. S.l.: Addison Wesley, 1999. ISBN: 84-7829-028-1. [En línea] [Citado el: 25 de noviembre 2010].
- **Dayley, Brad. 2006.** Python Phrasebook. s.l.: Sams, 2006. 0-672-32910-7. [En línea] [Citado el: 26 de enero 2011].
- **Collado Cabeza, Eduardo y Díaz Berenguer, Angi. 2003.** Madritel. http://web.madritel.es/personales3/edcollado/index.html. [En línea] [Citado el: 26 de febrero 2011].
- **Conferencia 7 Prueba**. http://eva.uci.cu. [En línea] [Citado el: 4 de mayo 2011].
- **Ferré Grau, Xavier y Sánchez Segura, María Isabel. 2004.** Clikear.com. Desarrollo Orientado a Objetos con UMLhttp://www.clikear.com/manuales/uml/index.aspx. .. [En línea] [Citado el: 7 de abril 2011]
- **Figueroa, Roberth G. y Solis, Camilo J.2007.**Metodologías Tradicionales Vs. Metodologías Ágiles. [En línea] [Citado el 5 de enero 2011].
- **Garlan, David and Shaw, Mary.** *An introduction to software architecture.* s.l. : CMU Software Engineering Institute Technical Report, 1994. [En línea] [Citado el 10de enero 2011].
- **Glassfish** http://www.marlonj.com/blog/2010/10/glassfish-comparando-versiones-2-1-x-y-3-0-x/. [En línea] [Citado el: 20 de enero 2011].
- **Ingeniería de Software I Conferencia #5**Flujo de Trabajo Implementación. http://eva.uci.cu/... [En línea] [Citado el: 21 de abril 2011].
- **Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000**. El Proceso Unificado de Desarrollo de Software. Madrid: Addison Wesley, 2000. 84-7829-036-2. [En línea] [Citado el: 15 de marzo 2011].
- **Jacobson, Ivar, Rumbaugh James y Booch, Grady. 2007.** El lenguaje Unificado de Modelado. Manual de Referencia. Madrid: Addison Wesley, 2007. Segunda Edición. [En línea] [Citado el: 18 de enero 2011].
- **Larman C .UML y Patrones**: “Introducción al análisis y diseño de la programación Orientada a Objetos”. http://bibliodoc.uci.cu/pdf/reg00061.pdf. [En línea] [Citado el: 23 de marzo 2011].

- **Linux**
  http://www.linuxespanol.com/viewtopic.php?p=96480&sid=c1e2f61c2c9f25c5ca0e492a6c87c62e. [En línea] [Citado el: 2 de diciembre 2010].
- **Menéndez, Rosa. Proyectos Disponibles en**:
  http://www.usmp.edu.pe/publicaciones/boletin/fia/info36/proyectos.html#a. [En línea] [Citado el: 3 de marzo 2011].
- **MSF**
  http://www.ibm.com/developerworks/rational/library/apr07/santos/index.html. http://www.getec.etsit.upm.es.[En línea] [Citado el: 16 de junio 2011].
- **PostgreSQL. 2008**.
  http://www.postgresql.org/about/.[En línea] [Citado el: 17de febrero 2011].
- **Pressman, Roger S. 1997**. Ingeniería de Software un enfoque práctico. Quinta Edición.[En línea] [Citado el: 24 de abril 2011].
- **Pressman. 2001**. Ingeniería del software Un enfoque práctico. s.l.: McGraw-Hill. [En línea] [Citado el: 29 de abril 2011].
- **RUP vs XP**
  http://www.usmp.edu.pe/publicaciones/boletin/fia/info49/articulos/RUP%20vs.%20XP.pdf.[En línea] [Citado el: 18 de noviembre 2010].
- **Seco, José Antonio González. 2006.** www.devjoker.com. www.devjoker.com. www.devjoker.com. http://www.devjoker.com/asp/ver_contenidos.aspx?co_contenido=125. . [En línea] [Citado el: 20 de enero 2011].
- **Somerville, Ian. 2005.** Ingeniería de Software - 7ma Edición**.** Madrid: Addison Wesley, 2005. ISBN-84-7829-074-5. [En línea] [Citado el: 10 de marzo 2011].
- **Souli, Juan 2009.**cplusplus.com. http://www.cplusplus.com/info/description.html. [En línea] [Citado el: 20 de enero 2011].
- **Universidad de Oviedo. 2003.** Sitio Web de la E.U de Ingeniería Técnica Informática de Oviedo. http://petra.euitio.uniovi.es/. [En línea] [Citado el: 8 de febrero 2011].
- **Visual Paradigm Organización. 2009.** Sitio Web oficial Visual-Paradigm. Sitio Web oficial Visual-Paradigm. [En línea] [Citado el: 9 de diciembre 2010].
- **Visual Paradigm** http://www.visual-paradigm.com/product/vpuml/. [En línea] [Citado el: 12 de enero 2011].

- **Yzquierdo Herrera, Raykenler y Lazo Ochoa, René. 2007.** El modelo de diseño del sistema HyperWeb. Módulos de Tratamiento Farmacológico y Configuración. [En línea] [Citado el: 28 de marzo 2011].

Glosario de Términos

- **POO:** La programación orientada a objetos o POO (OOP según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos.
- **Microsoft:** (acrónico de Microcomputer Software): Es una empresa de Estados Unidos, fundada por Bill Gates y Paul Allen. Dueña y productora de los sistemas operativos: Microsoft DOS y Microsoft Windows, que se utilizan en la mayoría de las computadoras del planeta.
- **Bytecode:** es el código intermedio entre el código fuente y el código máquina. Suele tratársele como un fichero binario que contiene un programa ejecutable similar a un módulo objeto. (ALEGSA, 2010).
- **Plugin:** Es un módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande. (Masadelante.com, 1999).
- **SWT:** (siglas en inglés de Standard Widget Toolkit) es un conjunto de componentes para construir interfaces gráficas en Java, (widgets) desarrollados por el proyecto Eclipse. Recupera la idea original de la biblioteca AWT de utilizar componentes nativos, con lo que adopta un estilo más consistente en todas las plataformas, pero evita caer en las limitaciones de ésta.
- **IBM:** International Business Machines o IBM (conocida coloquialmente como el Gigante Azul) es una empresa multinacional que fabrica y comercializa herramientas, programas y servicios relacionados con la informática.
- **JVM:** (Java Virtual Machine o Máquina Virtual Java). Es un programa nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial.
- **FIFO:** acrónimo inglés de ***F****irst* ***I****n,* ***F****irst* ***O****ut* (primero en entrar, primero en salir), es un concepto utilizado en estructuras de datos, contabilidad de costes y teoría de colas.

- **Mapeo Objeto_Relacional:** Más conocido por su nombre en inglés, Object-Relational Mapping, o sus siglas ORM, es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia.
- **POJO:** (acrónimo de Plain Old Java Object) es una sigla creada por Martin Fowler, Rebecca Parsons y Josh MacKenzie, utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un framework en especial.
- **Triggers:** Disparadores.
- **Mapa de bits:** Es una estructura de datos que representa una rejilla rectangular de píxeles o puntos de color, que puede ser visualizada en un monitor de computadora, en un papel o en otros dispositivos de representación.
- **Open Source**: Representa el software de dominio público, esto significa sin licencia, cuyo código fuente está disponible y se le permite usar y modificar.
- **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.
- **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.