Universidad de las Ciencias Informáticas

"Facultad 3"

Diseño e implementación del módulo de Importación de Metadatos del Sistema DigiPyrus de los Registros y Notarías de la República Bolivariana de Venezuela

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

Autor: José Carlos Pupo Acosta

Tutor(es): MsC. Yaniesis Lorenzo Costa
Ing. Yunier Pérez Barroso

La Habana, Cuba
Junio, 2011

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 _____ del año _____.

_____
José Carlos Pupo Acosta
Autor

_____
MsC. Yaniesis Lorenzo Costa
Tutor

_____
Ing. Yunier Pérez Barroso
Tutor

AGRADECIMIENTOS

Primero agradezco inmensamente a Yani, un día mi jefa, otro mi amiga y hoy mi tutora, sin tu ayuda este trabajo de diploma hubiera sido solo eso, “Un Trabajo”, gracias por dedicarme tiempo y por tu paciencia.

A Yunier, Héctor, Juan, Aliuska, Carlos, Yulieski, Juan Pablo, todo el equipo de proyecto de Registros y Notarías, a Lourdes, gracias porque siempre tuve en quién apoyarme cuando necesité un consejo o dirección.

A Yailín porque siempre que me ponía vago pensaba en cuando estabas también haciendo tu tesis y cómo te alentaba a trabajar, gracias por estar pendiente de mí a pesar que ya no estás en la UCI.

A mis compañeros de aula, por estar junto a mí siempre, en estos cinco años pasé más tiempo con ustedes que con mi propia familia.

A mis padres que aunque nunca los tuve a mi lado en la universidad siempre fue pensando en ellos que logré todo lo que hoy estoy orgulloso de ser.

A mis segundos y terceros padres, Juani y Ana, Luisi y Mery, por darme estos cinco años todo cuanto estuvo en sus manos, sin ustedes estos cinco años no tendrían ni la mitad de los momentos buenos que recuerdo, gracias por su apoyo y su cariño.

A Yulier porque sé que siempre que pudiste me diste buenos consejos y en muchos casos los seguí, por eso entré a esta universidad, ojalá y hubieras venido a estudiar aquí también, aunque sé que como doctor eres más necesario.

A Yuliet, gracias porque en ti encontré el apoyo que no podía buscar en otras personas, porque escuchabas mis problemas cuando también te preocupaban los tuyos, gracias por tu tiempo y amor, te quiero mucho.

DEDICATORIA

A mis padres y mis abuelos.

A mi familia.

A mis amigos y hermanos Henry, José Abel y Yoandris.

RESUMEN

Se presenta el diseño e implementación de un módulo para la importación de metadatos del sistema DigiPyrus, que permita su uso de forma sencilla y sin necesidad de un entrenamiento complejo. La aplicación debe constituir una herramienta para importar de manera rápida y segura, los objetos digitales obtenidos de la digitalización del fondo documental histórico de los Registros y Notarías Públicas de la República Bolivariana de Venezuela, contribuyendo de esta manera a elevar la calidad de los servicios del Sistema Registral y Notarial.

Se desarrolló la solución propuesta y se realizaron las pruebas establecidas por la norma internacional ISO (*International Standard Organization)* asociadas a la seguridad y fidelidad de los datos. El módulo fue probado con las pruebas de unidad y las realizadas durante la Fase II del proyecto de Desarrollo de una Solución Tecnológica Integral para la Automatización y Modernización de los Registros y Notarías Públicas, las cantidades importadas de objetos digitales se realizaron durante 10 horas ininterrumpidas, no se apreciaron diferencias entre los metadatos exportados y los importados. En la actualidad este módulo se encuentra en uso, y se han importado más de 20000 objetos digitales con resultados satisfactorios.

PALABRAS CLAVE

Importación de metadatos, Objetos digitales.

ÍNDICE

INTRODUCCIÓN

Debido al gran avance que se ha logrado alcanzar en cuanto a tecnologías en el mundo, todos los países están llevando a cabo una amplia informatización en todos los sectores de su sociedad. La informática juega un papel importante en esta misión ya que se ha convertido en una potente herramienta para todos los seres humanos debido a los vínculos que tiene en casi todas las tareas que el mismo realiza, fundamentalmente en el ámbito laboral. El hombre para demostrar sus conocimientos y lograr una gran mejora en todos los procesos que desarrolla se ha propuesto crear soluciones informáticas de todos tipos con el fin de lograr automatizar casi todos los sectores de la sociedad.

La República Bolivariana de Venezuela desde hace algún tiempo ha estado inmersa en una ardua tarea en cuanto a la modernización de su Sistema Registral y Notarial, para poder estar a la altura de los países en busca del desarrollo. La Dirección General de Registros y Notarías de la República Bolivariana de Venezuela es creada a partir del Decreto N° 371 de fecha 7 de octubre de 1999, publicado en la Gaceta Oficial N° 5.389 de fecha 21 de octubre de 1999, contentivo del Reglamento Orgánico del Ministerio del Poder Popular para Relaciones Interiores y Justicia (MPPRIJ)[1].

A finales del año 2006 con el objetivo de regular la organización, el funcionamiento, la administración y las competencias de los Registros Principales, Mercantiles, Públicos y las Notarías, se crea la nueva Ley de Registro y del Notariado de la República de Venezuela publicada en la Gaceta Oficial Nº 5.833 extraordinaria del 22 de diciembre de 2006. Esta plantea en el Artículo 10, la creación del Servicio Autónomo de Registros y Notarías (SAREN).

De igual forma aborda sobre el manejo electrónico en su Artículo 23, donde establece que “Todos los soportes físicos del Sistema Registral y Notarial actual se digitalizarán y se transferirán a las bases de datos correspondientes”. El proceso registral y notarial podrá ser llevado a cabo íntegramente a partir de un documento electrónico.

Además, se plantea sobre el Principio de Publicidad Registral en su Artículo 9, el cual establece “La fe pública registral protege la verosimilitud y certeza jurídica que muestran sus asientos. La información contenida en los asientos de los Registros es pública y puede ser consultada por cualquier persona”.

1 Ministerio del Poder Popular para Relaciones Interiores y Justica (MPPRIJ): Ministerio encargado de hacer cumplir la constitución y otros instrumentos jurídicos vigentes, de carácter nacional e internacional que ayuden a enaltecer los derechos humanos y propicien el mejoramiento de la calidad de vida de todos los ciudadanos que habitan en la República Bolivariana de Venezuela.

Lo antes expuesto conduce a la necesidad de mantener y conservar el patrimonio registral de cada oficina en formato digital, pues la información asentada en estas debe ser pública para que cualquier persona natural o jurídica pueda consultarla cumpliendo así con el Principio de Publicidad Registral.

Como parte de los acuerdos de cooperación entre la República de Cuba y la República Bolivariana de Venezuela, nace la idea de un proyecto de *software* para automatizar los procesos de gestión del SAREN, con el fin de que se pueda ofrecer garantía y seguridad jurídica de las operaciones que se realicen.

Cuando surge este proyecto la información que se maneja en los Registros y Notarías se realiza de manera manual, lo que imposibilita agilizar los trámites legales y obstaculiza las prestaciones de los servicios jurídicos brindados a la población venezolana, generando un amplio grado de descontento en todos los ciudadanos o clientes que solicitan estos servicios.

Con el fin de llevar a cabo de manera eficiente la automatización de los procesos registrales y notariales, se requirió la digitalización del fondo documental histórico preservado en dichas oficinas. De ahí que se acordó la implementación del sistema DigiPyrus que consiste en una aplicación informática para la digitalización de la documentación salvaguardada en los Registros y Notarías. Esta aplicación consta de un conjunto de módulos a través de los cuales se lleva a cabo la recepción, digitalización, revisión de calidad, preparación y encuadernación de los documentos. El sistema también permite la asociación de los metadatos, exportación de los objetos digitales y los metadatos obtenidos firmados digitalmente y validación de los mismos, configuración de los parámetros generales de la aplicación y generación de un conjunto de reportes que posibilita la evaluación y supervisión del flujo de trabajo.

Además, es necesario contribuir a la integridad y confidencialidad de los datos, así como lograr de forma rápida y eficiente sin duplicidad y pérdida de la información su movimiento hacia la Base de Datos (BD) del SAREN. Debido a que la aplicación informática desarrollada en la Fase I del proyecto sólo satisfacía parcialmente la digitalización del fondo documental preservado en las oficinas del SAREN y que parte de esta información se encontraba en la BD antes mencionada, surge la idea de tener un mecanismo fiable de importación de los datos, que permita ingresar los objetos a la BD correspondientes en un ambiente seguro y confiable.

Se trata mediante esta investigación de brindar respuesta a la siguiente pregunta: ¿Cómo automatizar el proceso de importación de metadatos del sistema DigiPyrus de los Registros y Notarías Públicas

de manera que se contribuya a la integridad y confidencialidad de los datos, y elimine la pérdida y duplicidad de la información?

Al analizar la problemática planteada surgió la idea del diseño y desarrollo del módulo de Importación de Metadatos del sistema DigiPyrus, de forma sencilla y sin necesidad de un entrenamiento complejo. La aplicación permitirá importar de manera rápida y segura, toda la información digitalizada del fondo documental histórico de los Registros y Notarías Públicas.

Esta solución se enmarca en el área de la informática jurídica, al permitir cumplir con el Principio de Publicidad Registral, al realizar la transferencia progresiva de los objetos digitales y metadatos obtenidos hacia la BD del SAREN.

Siendo el objeto de estudio el proceso de desarrollo de *software* y centrando el campo de acción en el diseño e implementación del proceso de importación de metadatos para el sistema DigiPyrus del Centro de Digitalización.

Para resolver el problema planteado se define, como objetivo general realizar el diseño e implementación del módulo de Importación de Metadatos del sistema DigiPyrus de los Registros y Notarías Públicas de la República Bolivariana de Venezuela de manera que se contribuya a la integridad y confidencialidad de los datos, y se elimine la pérdida y duplicidad de la información.

La hipótesis consiste, en que el módulo de Importación de Metadatos del sistema DigiPyrus, permitirá eliminar la pérdida y duplicidad de la información, así como contribuirá a la integridad y confidencialidad de los datos.

Se definieron como objetivos específicos los que se muestran a continuación:

- Elaborar el marco teórico de la investigación.
- Realizar el diseño de la solución informática para la importación de metadatos del sistema DigiPyrus.
- Realizar la implementación de la solución informática diseñada.
- Evaluar y aplicar los resultados obtenidos.

Para realizar esta solución informática, es necesario trazar las tareas de investigación que guíen el camino a seguir para dar cumplimiento a los objetivos definidos y obtener los resultados esperados, estas son las siguientes:

- Análisis de la situación actual de temas afines con la investigación.
- Análisis de las metodologías disponibles para el diseño e implementación del software.
- Estudio de la arquitectura definida para la implementación de la solución informática.
- Realización de los diagramas de clases y los diagramas de secuencia de los casos de uso, para el proceso de Importación de Metadatos.
- Implementación de la solución informática diseñada.
- Validación de los diagramas de clases propuestos mediante métricas de diseño.
- Validación del código fuente mediante pruebas de unidad.

La importancia de esta investigación radica en que el módulo de Importación de Metadatos del sistema DigiPyrus, contribuye a mitigar los problemas actuales de gestión de la información en el Sistema Registral y Notarial de la República Bolivariana de Venezuela.

El presente documento está conformado por la introducción, tres capítulos a través de los cuales se desarrolla esta investigación, al finalizar se dan las conclusiones, se hacen recomendaciones, se enuncian las referencias bibliográficas y se presentan los anexos.

La introducción ubica al lector en los antecedentes del problema que se tratará, diseño teórico, su importancia y los objetivos de la tesis.

En el primer capítulo se comienza con un estudio teórico en el que se enuncian los conceptos básicos sobre los metadatos, la programación orientada a objetos y aspectos, y sobre la firma electrónica. Se continúa con la definición de las metodologías de desarrollo de *software*, se enuncian las herramientas CASE (*Computer Aided Software Engineering* por sus siglas en inglés), se presentan las características de la plataforma y *frameworks* de desarrollo, el sistema gestor de BD y se culmina con un análisis crítico de las fuentes y bibliografías revisadas.

En el segundo capítulo se presenta la propuesta de solución del sistema, exponiendo el diseño y la arquitectura con la que se desarrollará el mismo, también se hace referencia a los patrones empleados así como los artefactos obtenidos durante el diseño y la implementación.

En el tercer capítulo se muestra la validación del módulo. Se detallan los resultados obtenidos de la medición del código por las principales métricas utilizadas para determinar la calidad del diseño, así como se exponen además las pruebas de unidad realizadas al código del módulo de Importación de Metadatos.

Por último, en las conclusiones se resumen los logros de la labor realizada y en las recomendaciones se proponen los aspectos en que se debe continuar trabajando.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

En este capítulo se presentarán los conceptos básicos con respecto a los metadatos y sobre la programación orientada a objetos y a aspectos. Además se hace un estudio de las metodologías de desarrollo de *software*, herramientas CASE, la plataforma de desarrollo, *frameworks* de desarrollo, y el sistema gestor de BD para luego concluir con las utilizadas en la solución expuesta. Finalmente se hace un análisis crítico de las fuentes y bibliografías revisadas.

1.1 Conceptos básicos de Metadatos

Desde hace varios años, los profesionales de la información tienen que afrontar el reto de solucionar los problemas de preservación y recuperación de información provocados por la aparición de las bibliotecas digitales y el aumento vertiginoso de la información en formato electrónico. Un documento digital tiene como principal desafío obtener, mantener y preservar documentos, así como permitir la accesibilidad, autenticidad y confiabilidad de la información.

El término metadatos surgió en la década de los sesenta para describir conjuntos de datos. La primera acepción que se le dio y actualmente la más difundida fue la de dato sobre el dato, ya que proporcionaban la información mínima necesaria para identificar un recurso. Otra definición trata de precisar el término como datos estructurados y codificados que describen características de instancias conteniendo informaciones que permiten identificar, descubrir, preservar y administrar las instancias descritas a lo largo del tiempo. Esta clase de definición hace mayor hincapié en los metadatos en relación con la recuperación de información (CAPLAN, 1995).

Los metadatos pueden describir colecciones de objetos y también los procesos en los que están involucrados, describiendo cada uno de los eventos, sus componentes y cada una de las restricciones que se les aplican. Los mismos definen las relaciones entre los objetos, como las tuplas en una BD o clases en orientación a objetos, generando estructuras (Heery, 1997).

Por eso la representación del conocimiento usa metadatos para categorizar informaciones, resumir el significado de los datos, permitir búsquedas, indicar relaciones con otros recursos, controlar la gestión. Además, estos facilitan el flujo de trabajo convirtiendo datos automáticamente de un formato a otro (CHILVERS, et al., 1998).

De todo lo expuesto hasta ahora se puede extraer varios puntos cruciales (dato sobre el dato, objeto, recuperación de información) que pueden ser útiles para presentar una definición que aglutine a todas las publicadas hasta la fecha, de tal forma que resulte posible concluir que metadato es toda aquella información descriptiva sobre el contexto, calidad, condición o características de un recurso, dato u objeto que tiene la finalidad de facilitar su recuperación, autentificación, evaluación y preservación (Senso, 2003) (Metadata, 1997).

1.2 Solución Informática Existente

En la actualidad en el SAREN existe una aplicación informática que permite la importación de los metadatos obtenidos de los Registros Públicos. Esta aplicación sólo automatiza el proceso aislado y no tiene relación con las demás oficinas, por lo que no cuentan en el SAREN con un sistema único que permita realizar la integración del proceso importación para todos los tipos de oficinas. El principal inconveniente es que la solución existente es independiente y no se subordina a un ente controlador, por lo que los Registros Principales, Registros Mercantiles y Notarías no cuentan con este servicio.

Debido al problema que presenta la aplicación existente hoy en el SAREN es necesario hacer un análisis riguroso para que la solución a desarrollar erradique estos problemas y a la vez facilite un mejor servicio para todas las oficinas, contribuyendo a la integridad y confidencialidad de los datos, y eliminando la pérdida y duplicidad de la información.

1.3 Paradigmas de Programación

Un paradigma de programación representa un enfoque particular o filosofía para la construcción del *software*. Diferentes paradigmas resultan en diferentes estilos de programación y en diferentes formas de pensar la solución de problemas.

1.3.1 Programación Orientada a Objetos

La programación orientada a objetos (POO) es un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento. En la actualidad, existe variedad de lenguajes de programación que soportan la orientación a objetos.

1.3.1.1 Conceptos Fundamentales

Los conceptos de la programación orientada a objetos tienen origen en Simula 67, un lenguaje diseñado para hacer simulaciones, creado por Ole-Johan Dahl y Kristen Nygaard del Centro de Cómputo Noruego en Oslo. En este centro, se trabajaba en simulaciones de naves, que fueron confundidas por la explosión combinatoria de cómo las diversas cualidades de diferentes naves podían afectar unas a las otras. La idea ocurrió para agrupar los diversos tipos de naves en diversas clases de objetos, siendo responsable cada clase de objetos de definir sus propios datos y comportamientos (Olaf, y otros, 2004)(BOOCH, 1996).

La programación orientada a objetos tomó posición como el estilo de programación dominante a mediados de los años ochenta, en gran parte debido a la influencia de C++, una extensión del lenguaje de programación C. Su dominio fue consolidado gracias al auge de las Interfaces, para las cuales la programación orientada a objetos está particularmente bien adaptada. En este caso, se habla también de programación dirigida por eventos, ver Epígrafe 1.3.2 (NUÑO, et al., 2002).

Las características de orientación a objetos fueron agregadas a muchos lenguajes existentes durante ese tiempo, incluyendo BASIC, Pascal y otros. La adición de estas características a los lenguajes que no fueron diseñados inicialmente para ellas condujo a menudo a problemas de compatibilidad y en la capacidad de mantenimiento del código. Los lenguajes orientados a objetos puros, por su parte, carecían de las características de las cuales muchos programadores habían venido a depender. Para saltar este obstáculo, se hicieron muchas tentativas para crear nuevos lenguajes basados en métodos orientados a objetos, pero permitiendo algunas características imperativas de maneras seguras.

La programación orientada a objetos es una forma de programar que trata de encontrar una solución a estos problemas. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entre ellos de destacan los siguientes (Abadi, y otros, 1996) (NUÑO, et al., 2002):

- Clase: Contiene definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ellas.
- Herencia: Es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D. Por lo tanto, puede usar los mismos métodos y variables públicas declaradas en C. Los componentes registrados como privados o *private* también se heredan, pero como no pertenecen a la clase, se mantienen escondidos al programador y sólo pueden ser accedidos a través de otros métodos públicos.

- Objeto: Entidad provista de un conjunto de propiedades o atributos y de comportamiento o funcionalidad los mismos que consecuentemente reaccionan a eventos. Se corresponde con los objetos reales del mundo que nos rodea, o a objetos internos del sistema. Es una instancia a una clase.
- Método: Algoritmo asociado a un objeto o a una clase de objetos, cuya ejecución se desencadena tras la recepción de un mensaje. Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un evento con un nuevo mensaje para otro objeto del sistema.
- Evento: Es un suceso en el sistema. El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento, a la reacción que puede desencadenar un objeto, es decir la acción que genera.
- Mensaje: Una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.
- Propiedad o atributo: Contenedor de un tipo de datos asociados a un objeto o a una clase de objetos, que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.
- Componentes de un objeto: Atributos, identidad, relaciones y métodos.
- Identificación de un objeto: Un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y funciones correspondientes.

En comparación con un lenguaje imperativo, una variable, no es más que un contenedor interno del atributo del objeto o de un estado interno, así como la función es un procedimiento interno del método del objeto.

1.3.1.2 Características de la POO

Hay un cierto desacuerdo sobre exactamente qué características de un método de programación o lenguaje lo definen como orientado a objetos, pero actualmente existe un consenso general en que las características que se definen a continuación se plantean como las más importantes (Schach, 2006):

- Abstracción: Denota las características esenciales de un objeto, donde se capturan sus comportamientos. Cada objeto en el sistema sirve como modelo de un agente abstracto que puede

realizar trabajo, informar y cambiar su estado, y comunicarse con otros objetos en el sistema sin revelar cómo se implementan estas características. Permite seleccionar las características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades en el mundo real.

- Encapsulamiento: Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema.
- Modularidad: Se denomina modularidad a la propiedad que permite subdividir una aplicación en partes más pequeñas llamadas módulos, cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes. Estos módulos que se puedan compilar por separado, pero que tienen conexiones con otros módulos.
- Principio de ocultación: Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que específica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas.
- Polimorfismo: Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado.
- Herencia: Las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. Organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes.
- Recolección de basura: La recolección de basura o *garbage collector* es la técnica por la cual el entorno de objetos se encarga de destruir automáticamente, y por tanto desvincular la memoria asociada, los objetos que hayan quedado sin ninguna referencia a ellos.

1.3.2 Programación Orientada a Aspectos (POA)

A pesar de las grandes ventajas que ofrece la POO, esta no evita que sigan quedando funciones dispersas en el código del programa, teniendo el inconveniente de tener que modificar varias clases para realizar la integración de varias funciones por lo que hace engorroso la funcionalidad del sistema.

La programación orientada a aspectos (POA) viene a resolver esa problemática aspirando a soportar la separación de competencias para los aspectos de la aplicación. El concepto de POA fue introducido por Gregor Kiczales, y esta se podría decir se encarga principalmente de permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de incumbencias en un *software* (KICZALES, y otros, 1997) (QUINTERO, 2000).

Un aspecto según Kiczales y su grupo es una unidad modular que se disemina por la estructura de otras unidades funcionales. Los aspectos existen tanto en la etapa de diseño como en la de implementación. Un aspecto de diseño es una unidad modular del diseño que se entremezcla en la estructura de otras partes del mismo. Un aspecto de programa o de código es una unidad modular del programa que aparece en otras unidades modulares de este (KICILLOF, 2007).

Entre las ventajas de la POA se encuentra que:

- El código que se tiene es más natural y reducido.
- Elimina la dispersión del código, lo que posibilita que las implementaciones se vuelvan más comprensibles y adaptables.
- Mayor facilidad para depurar y hacer modificaciones en el código.
- Permite que grandes cambios en la definición de una materia tenga un pequeño impacto en las demás.
- Se tiene un código más reusable que se puede acoplar y desacoplar cuando sea necesario.

Sobre sus limitaciones se podría decir que:

Hasta ahora los lenguajes orientados a aspectos específicos y de propósito general han elegido utilizar un lenguaje base existente. Aparte de las ventajas de esta elección, esto trae consigo una serie de problemas. En el caso de los lenguajes de dominio específico no se es completamente libre para diseñar los puntos de enlace sino que se restringe a aquellos que pueden ser identificados en el lenguaje base (2009).

También los entornos de programación orientada a aspectos que han habido hasta ahora no soportan una separación de funcionalidades suficientemente amplia, o bien el lenguaje de aspecto soporta el dominio de aspecto parcialmente, o no se sostiene bien la restricción del lenguaje base.

1.4 Patrones de Diseño Orientado a Objetos

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de *software* y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

“Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma” (ALEXANDER, y otros, 1977)

Los patrones de diseño ayudan a los diseñadores a reutilizar con éxito diseños para obtener nuevos diseños. El objetivo de los patrones es guardar la experiencia en diseños de programas orientados a objetos, haciendo más fácil reutilizar con éxito los diseños y arquitecturas, ayudan a elegir entre diseños alternativos, hacen a un sistema reutilizable y evitan alternativas que comprometen la reutilización.

Un patrón de diseño puede considerarse como un documento que define una estructura de clases que aborda una situación particular. Entre los patrones de diseño se pueden mencionar dos grandes grupos, los patrones de asignación de responsabilidades GRASP y los patrones GOF, siglas de *Gang of Four*, aunque también existen otros patrones e incluso cada diseñador puede crear su propio patrón de diseño. Estos dos grupos antes mencionados son los más utilizados en la práctica (LARMAN, 1999).

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Dentro de este grupo de patrones se encuentran los siguientes: experto, creador, bajo acoplamiento, alta cohesión, controlador, fabricación pura, indirección, variaciones protegidas, no hables con extraños y polimorfismo.

Por otra parte los patrones de diseño GOF son 23 (ver Anexo 1) y se clasifican según su propósito en creacionales, estructurales, y de comportamiento. Y según su ámbito en objeto y de clase (ADOLPH, 2001).

1.5 Metodologías de desarrollo del software

Durante el ciclo de vida del *software* se deben realizar una serie de tareas para obtener un producto final. Se dice que los distintos componentes de *software* deben pasar por distintas fases o etapas durante el ciclo de vida de desarrollo del *software*. Para dar seguimiento a estas tareas y controlar y actualizar cada una de ellas en su paso por cada una de estas etapas o fases se necesita una guía, un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda para el desarrollo del producto de *software* que queremos realizar y a esta serie de aspectos se le llama Metodología. Una metodología puede seguir uno o varios modelos de ciclo de vida, es decir, el ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto pero no cómo hacerlo. La metodología indica cómo hay que obtener los distintos productos parciales y finales (SANCHEZ, 2004).

La finalidad de una metodología de desarrollo es garantizar la eficacia y la eficiencia en el proceso de desarrollo de *software*.

1.5.1 Metodologías Tradicionales

Las metodologías tradicionales o pesadas son aquellas que implantan una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de desarrollar software más eficiente. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada.

Este tipo de metodologías son más eficaces y necesarias cuanto:

- •El proyecto es de gran envergadura respecto a tiempo y recursos que son necesarios emplear.
- •El equipo de desarrollo es grande.
- •Los requisitos están bien definidos.

1.5.1.1 Rational Unified Process. (Proceso Racional Unificado - RUP)

El Proceso Racional Unificado (RUP) es un proceso de desarrollo de *software* y junto con el Lenguaje Unificado de Modelado (UML), constituye una metodología robusta, la más estándar y utilizada para el

análisis, implementación y documentación de sistemas orientados a objetos. Su primera versión fue puesta en el mercado en 1998, como resultado de una convergencia de ideas que se venía dando entre Boehm, Hartman e Ivar Jacobson (JACOBSON, et al., 2000).

RUP es una metodología para la ingeniería de *software* que va más allá del mero análisis y diseño orientado a objetos para proporcionar una familia de técnicas que soportan el ciclo completo de desarrollo de *software*. El resultado es un proceso basado en componentes, dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental.

Seguidamente se presentan las características principales de la metodología RUP (SPARKS, 2005):

- Centrado en los modelos: Los diagramas son un vehículo de comunicación más expresivo que las descripciones en lenguaje natural. Se trata de minimizar el uso de descripciones y especificaciones textuales del sistema.
- Guiado por los Casos de Uso: Los Casos de Uso son el instrumento para validar la arquitectura del *software* y extraer los casos de prueba.
- Centrado en la arquitectura: Los modelos son proyecciones del análisis y el diseño constituye la arquitectura del producto a desarrollar.
- Iterativo e incremental: Durante todo el proceso de desarrollo se producen versiones incrementales del producto en desarrollo.

El ciclo de vida de RUP es una implementación del Desarrollo en espiral. Fue creado ensamblando los elementos en secuencias semi-ordenadas. El ciclo de vida organiza las tareas en fases e iteraciones.

RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades. En la Figura 1.1 se muestra cómo varía el esfuerzo asociado a las disciplinas según la fase en la que se encuentre el proyecto RUP (ZAVALA, 2002).

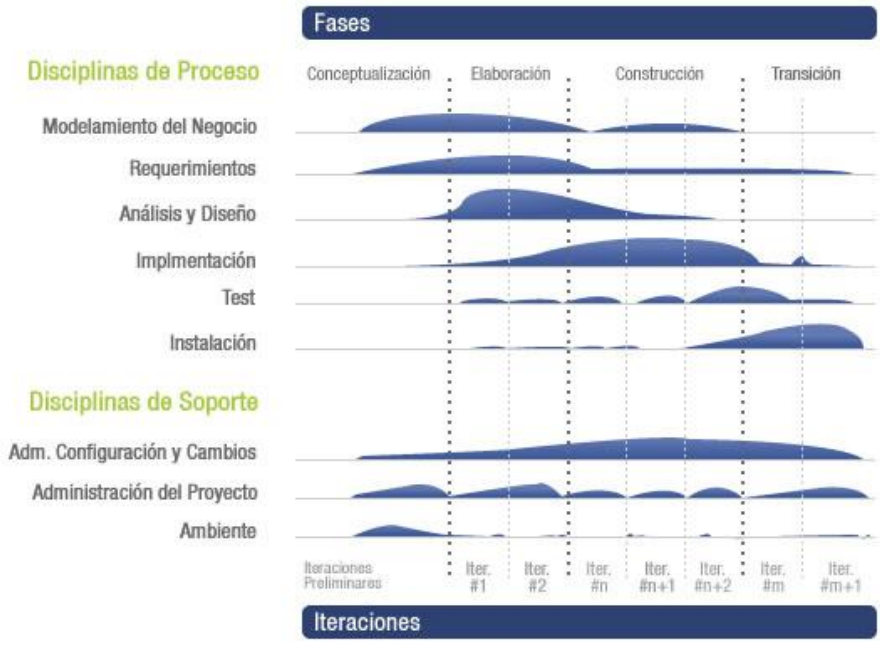

Figura 1.1 RUP Fases y Disciplinas.

Las primeras iteraciones en las fases de Inicio y Elaboración se enfocan hacia la comprensión del problema y la tecnología, la delimitación del ámbito del proyecto, la eliminación de los riesgos críticos, y al establecimiento de una línea base de la arquitectura.

Durante la fase de inicio las iteraciones hacen mayor énfasis en actividades de modelado del negocio y de requerimientos. En la fase de elaboración, las iteraciones se orientan al desarrollo de la línea base de la arquitectura, abarcan más los flujos de trabajo de requerimientos, modelo de negocios, análisis, diseño y una parte de implementación orientado a la línea base de la arquitectura (TORRES, 2005).

En la fase de construcción, se lleva a cabo la construcción del producto por medio de una serie de iteraciones. Para cada iteración se selecciona algunos Casos de Uso, se refina su análisis y diseño y se procede a su implementación y pruebas. Se realiza una pequeña cascada para cada ciclo. Se realizan tantas iteraciones hasta que se termine la implementación de la nueva versión del producto. En la fase de transición se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios.

Como se puede observar en la Figura 1.1 en cada fase participan todas las disciplinas, pero que dependiendo de la fase el esfuerzo dedicado a una disciplina varía.

Ventajas de la metodología de desarrollo RUP (ZAVALA, 2002):

- Permite desarrollar aplicaciones sacando el máximo provecho de las nuevas tecnologías, mejorando la calidad, el rendimiento, la reutilización, la seguridad y el mantenimiento del *software* mediante una gestión sistemática de los riesgos.
- Permite la producción de *software* que cumpla con las necesidades de los usuarios, a través de la especificación de los requisitos, con una agenda y costo predecible.
- Enriquece la productividad en equipo y proporciona prácticas óptimas de *software* a todos sus miembros.
- Permite llevar a cabo el proceso de desarrollo práctico, brindando amplias guías, plantillas y ejemplos para todas las actividades críticas.
- Unifica todo el equipo de desarrollo de *software* y mejora la comunicación al brindar a cada miembro del mismo una base de conocimientos, un lenguaje de modelado y un punto de vista de cómo desarrollar *software*.
- Optimiza la productividad de cada miembro del equipo al poner al alcance la experiencia derivada de miles de proyectos y muchos líderes de la industria.
- No solo garantiza que los proyectos abordados serán ejecutados íntegramente sino que además evita desviaciones importantes respecto a los plazos.
- Permite una definición acertada del sistema en un inicio para hacer innecesarias las reconstrucciones parciales posteriores.

Se ha seleccionado la metodología RUP porque presenta varios aspectos favorables para el desarrollo del módulo de Importación de Metadatos del sistema DigiPyrus:

- Es una metodología sumamente configurable que ha proporcionado resultados satisfactorios en aplicaciones similares.
- El equipo de desarrollo está familiarizado con esta metodología, por lo que la introducción de una nueva atrasaría el cronograma trazado.
- Genera documentación que es exigida por el cliente y se necesaria por la dinámica del proyecto, tener documentación de lo que se ha realizado.

1.5.2 Metodologías Ágiles

Las metodologías ágiles están especialmente orientadas para proyectos pequeños. Constituyen una solución con una elevada simplificación, pero a pesar de ello no renuncian a las prácticas esenciales para asegurar la calidad del producto. Las características de los proyectos para los cuales las metodologías ágiles han sido especialmente pensadas se ajustan a un amplio rango de proyectos industriales de desarrollo de software; aquellos en los cuales los equipos de desarrollo son pequeños, con plazos reducidos, requisitos volátiles y nuevas tecnologías.

1.5.2.1 Programación Extrema (XP)

Es el más destacado de los procesos ágiles de desarrollo de *software*, surge ideada por Kent Beck, como proceso de creación de *software* diferente al convencional. En palabras de Beck: “XP es una metodología ligera, eficiente, con bajo riesgo, flexible, predecible y divertida para desarrollar *software*” (Beck, 1999).

Los objetivos de XP son muy simples: lograr la satisfacción del cliente y potenciar al máximo el trabajo en grupo. Esta metodología trata de dar al cliente el *software* que él necesita y cuando lo necesita. Por tanto, se debe responder muy rápido a las necesidades del cliente, incluso cuando los cambios sean al final de ciclo de la programación. Tanto los jefes de proyecto, los clientes y desarrolladores, son parte del equipo y están involucrados en el desarrollo del *software* (CÁNOS, et al., 2002).

La programación extrema se basa en la simplicidad, la comunicación y el reciclado continuo de código. Lo que buscan en definitiva es la reducción de costos.

En el ciclo de vida del desarrollo de un proyecto de *software* los cambios generalmente pueden ocurrir, cambiarán los requisitos, las reglas de negocio, el personal, la tecnología, por esto XP propone valores que deben poseer los miembros del equipo de desarrollo, para implementar el trabajo y conseguir los planteamientos iniciales. Estos cuatro valores son comunicación, sencillez, retroalimentación y valentía. Además, define cuatro variables para proyectos de *software*: costo, tiempo, calidad, y ámbito.

Las prácticas básicas de XP se resumen a continuación:

- El juego de la planificación.
- Pequeñas versiones.
- Metáfora.
- Diseño sencillo.

- Hacer pruebas.
- Recodificación.
- Programación por parejas.
- Propiedad colectiva.
- Integración continúa.
- 40 Horas semanales.
- Cliente In-situ.
- Estándares de codificación.

1.5.3 Selección de la metodología de desarrollo

Después de realizar un estudio por parte del equipo de proyecto se decidió seleccionar la metodología RUP para guiar el proceso de desarrollo, pues a pesar de ser una metodología tradicional o pesada, es una metodología muy exitosa y utilizada actualmente en proyectos de gran envergadura, además posee muchas características que ayudaron a su selección, entre ellas se pueden citar:

- Posee tres características que lo hacen único: Dirigido por casos de uso, centrado en la arquitectura e iterativo e incremental.
- Establece rigurosamente las actividades involucradas, los roles, las herramientas y notaciones que se usarán, incluyendo modelado y documentación detallada, así como los artefactos que se deben producir, esto último la convierte en una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.
- Aumenta la productividad del equipo, proporciona un acceso común a todos los miembros del proyecto al mismo conocimiento base.
- Se emplea fundamentalmente en proyectos con desarrollo a largo plazo.
- Es orientada al proceso.

1.6 Herramientas CASE

Las herramientas CASE o Ingeniería de *Software* Asistida por Ordenador son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de *software* reduciendo el costo de las mismas en términos de tiempo y de dinero. Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del *software* en tareas como el proceso de realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

La tecnología CASE supone la automatización del desarrollo del *software*, contribuyendo a mejorar la calidad y la productividad en el desarrollo de sistemas de información y se plantean los siguientes objetivos (JACOBSON, 2002):

- Permitir la aplicación práctica de metodologías estructuradas, las cuales al ser realizadas con una herramienta se consigue agilizar el trabajo.
- Facilitar la realización de prototipos y el desarrollo conjunto de aplicaciones.
- Simplificar el mantenimiento de los programas.
- Mejorar y estandarizar la documentación.
- Aumentar la portabilidad de las aplicaciones.
- Facilitar la reutilización de componentes de *software*.
- Permitir un desarrollo y un refinamiento visual de las aplicaciones mediante la utilización de gráficos.

De una forma esquemática se puede decir que una herramienta CASE se compone de los siguientes elementos:

- Repositorio: Donde se almacenan los elementos definidos o creados por la herramienta, y cuya gestión se realiza mediante el apoyo de un sistema de gestión de base de datos (SGBD) o de un sistema de gestión de ficheros.
- Meta modelo: Constituye el marco para la definición de las técnicas y metodologías soportadas por la herramienta.
- Carga o descarga de datos: Son las facilidades que permiten cargar el repertorio de las herramientas CASE con datos provenientes de otros sistemas, o bien generar a partir de la propia herramienta esquemas de base de datos, programas, que pueden a su vez, alimentar otros sistemas. Este elemento proporciona así un medio de comunicación con otras herramientas.
- Comprobación de errores: Facilidades que permiten llevar a cabo un análisis de la exactitud, integridad y consistencia de los esquemas generados por la herramienta.
- Interfaz de usuario: Consta de editores de texto y herramientas de diseño gráfico que permiten, mediante la utilización de un sistema de ventanas, iconos y menús, con la ayuda del ratón, definir los diagramas, matrices, que incluyen las distintas metodologías.

1.6.1 Enterprise Architect for UML v2.1

Enterprise Architect es una herramienta de modelado UML 2.1 flexible, completa y poderosa para plataformas Windows. Es una herramienta CASE orientada a objetos que provee el límite competitivo para el desarrollo de sistemas, administración de proyectos y análisis de negocios.

Es una herramienta completa que maneja totalmente el ciclo de vida y el análisis y diseño UML, cubriendo el levantamiento de requerimientos de desarrollo, a través del análisis, diseño, pruebas y mantenimiento. Un impresionante rango de lenguajes de desarrollo son soportados, incluyendo ActionScript, C, C++, C# y VB.NET, Java, Visual Basic 6, Python, PHP, XSD, WSDL y más (LARMAN, 1999).

Representa una plataforma líder de modelado de *software* que tiene una fuerte base de usuarios en más de 60 países del mundo. Es utilizado por varios tipos de desarrolladores de programas de sistemas para una amplia gama de industrias, incluyendo: Banca, Desarrollo Web, Ingeniería, Finanzas, Medicina, Investigación, Academia, Transporte, Detalle, Servicios Básicos, Ingeniería Eléctrica y muchas más.

Es también utilizado efectivamente para propósitos de entrenamiento en UML y arquitecturas de negocios en muchos prominentes colegios, compañías de entrenamiento y universidades alrededor del mundo.

Entre sus ventajas se pueden encontrar las mencionadas a continuación:

- Es una herramienta progresiva que cubre todos los aspectos del ciclo de desarrollo, proporcionando una trazabilidad completa desde la fase inicial del diseño a través del despliegue y mantenimiento.
- Provee soporte para pruebas, mantenimiento y control de cambio.
- Sus modelos se pueden exportar en formato RTF para una personalización y presentación final.
- Realiza ingeniería directa e inversa de código fuente en Action Script, C++, C#, Delphi, Java, Python, PHP, VB.NET, Visual Basic.
- Tiene una alta capacidad de sincronización de código.
- Posee plug-ins para vincular EA a Visual Studio.NET y Eclipse.
- Posee una interfaz de usuario intuitiva.
- Posibilita el desarrollo distribuido, es decir, es multiusuario.
- Soporta diferentes repositorios basados en DBMS, incluyendo Oracle, SQL Server, My SQL, PostGreSQL.

- Soporta importar/exportar archivos XMI para manejar la distribución y actualización de frameworks y otros paquetes basados en la estructura del modelo.
- Soporta control de versiones de repositorios.
- Soporta importar archivos .JAR en java y ensamblados .NET.
- Posee un perfil incorporado para XSD para simplificar el desarrollo de esquemas XML usando UML.
- Ingeniería Directa de esquema XML desde Modelos UML
- Posee gran velocidad, estabilidad y buen rendimiento.

1.6.2 ER/Estudio

ER/Studio es una herramienta profesional de modelado de datos fácil de usar para el diseño y construcción de bases de datos a nivel lógico y físico. Esta herramienta permite la sincronización bidireccional entre los diseños lógicos y físicos, lo que posibilita que si se hace algún cambio en uno de los dos diseños antes mencionados, se refleje de forma automática en el otro.

ER/Studio está equipado para crear y manejar diseños de bases de datos funcionales y confiables. Entre las características del ER/Studio se encuentran:

- Permite realizar diagramas con un desempeño claro y rápido.
- Mantiene automáticamente todas las dependencias entre sub-vistas y el diagrama completo.
- Da la posibilidad de hacer reingeniería inversa de bases de datos.
- Soporta un gran número de gestores de base de datos.
- Permite la creación de reportes de forma sencilla.
- Genera vistas, procedimientos almacenados, reglas, y tipos de datos de usuario, lo cual ayuda a la auto-ordenación de tipos de objetos para eliminar errores de dependencia al construir la base de datos.

1.6.3 Selección de las herramientas CASE

Por las características antes mencionadas, las grandes ventajas que brindan Enterprise Architect y ER/Studio, así como la vasta experiencia existente por años de utilización de estas herramientas en el Proyecto Registros y Notarias se decidió su selección para el desarrollo del módulo de Importación de Metadatos del Sistema DigiPyrus.

1.7 Plataformas de Desarrollo de Software

Para implementar un sistema informático y lograr que funcione correctamente es necesario tener un amplio conocimiento de las plataformas de desarrollo existentes en el mundo, sus características y principales funcionalidades para así escoger la adecuada de acuerdo a las necesidades o instituciones que se beneficiarán con el uso del sistema implementado.

Una plataforma de desarrollo es el entorno de *software* común en el cual se desenvuelve la programación de un grupo definido de aplicaciones. Comúnmente se encuentra relacionada directamente a un sistema operativo; sin embargo, también es posible encontrarla ligada a una familia de lenguajes de programación o a una Interfaz de programación de aplicaciones (API) (KATRIB, et al., 2008).

Su objetivo es ofrecer una interfaz más amigable para el programador y abstraerlo de las funciones más elementales, la creación de aplicaciones con sus prestaciones para diferentes campos de acción como lo constituyen, la Web y las aplicaciones de escritorio. A continuación, se expone la plataforma utilizada para el desarrollo de la aplicación de escritorio DigiPyrus.

1.7.1 Plataforma .Net (Visual Studio 2008)

.NET es el modelo de desarrollo de *Microsoft* que hace que el *software* sea independiente de la plataforma y de los dispositivos, y hace que los datos estén disponibles a través de Internet. El .NET *Framework* es la infraestructura básica subyacente de .NET. Ha sido implementado desde el principio pensando en una arquitectura abierta. Es una plataforma que puede utilizarse para generar y ejecutar la siguiente generación de aplicaciones Windows y aplicaciones Web.

Así mismo, .NET *Framework* proporciona la base sobre la que se desarrollan y ejecutan las aplicaciones y los servicios Web XML. La naturaleza unificada del .NET *Framework* significa que todas las aplicaciones, tanto si son aplicaciones Windows, aplicaciones Web o servicios Web XML, se desarrollan utilizando un conjunto de herramientas y código comunes, y se integran fácilmente entre sí (KATRIB, et al., 2008).

El .NET *Framework* está formado por (2011):

- El CRL (*Common Language Runtime*) el cual gestiona los servicios en tiempo de ejecución, incluyendo la integración de lenguajes, la seguridad y la gestión de memoria. Durante el desarrollo, el CLR proporciona funcionalidades necesarias para simplificar el desarrollo.

- Las bibliotecas de clases proporcionan código reutilizable para las tareas más habituales, incluyendo el acceso a datos, el desarrollo de servicios Web XML, Web Forms y *Windows Forms*.

Las ventajas de utilizar el .NET *Framework* para desarrollar aplicaciones incluyen:

- Basado en estándares y prácticas Web: El .NET *Framework* soporta completamente las tecnologías existentes de Internet, incluyendo HTML (Hypertext Markup Language), HTTP, XML, SOAP (*Simple Object Access Protocol*), XSLT (*Extensible Stylesheet Language Transformation*), XPath (*XML Path Language*) y otros estándares Web.
- Diseñado utilizando modelos de aplicación unificados: La funcionalidad de una clase .NET está disponible desde cualquier lenguaje compatible con .NET o modelo de programación. Por tanto, la misma pieza de código puede ser utilizada por aplicaciones Windows, aplicaciones Web y Servicios Web XML.
- Fácil de utilizar para los desarrolladores: En el .NET *Framework*, el código está organizado en espacios de nombres jerárquicos y en clases. El .NET *Framework* proporciona un sistema de tipos comunes, conocido también como sistema de tipos unificados, que puede ser utilizado por cualquier leguaje compatible con .NET. En el sistema de tipos unificados, todos los elementos del lenguaje son objetos. Estos objetos pueden ser utilizados por cualquier aplicación .NET escrita en cualquier lenguaje basado en .NET.
- Clases extensibles: La jerarquía del .NET *Framework* no queda oculta al desarrollador. Podemos acceder y extender las clases .NET (a menos que estén protegidas) mediante la herencia. También podemos implementar la herencia entre múltiples lenguajes.
- Seguridad de acceso al código: Se puede especificar que una pieza de código tenga permisos de lectura de archivos pero no de escritura. Es posible aplicar distintos niveles de seguridad al código, de forma que se puede ejecutar código procedente del Web sin tener que preocuparse si esto va a estropear el sistema.
- Despliegue: Por medio de los ensamblados resulta mucho más fácil el desarrollo de aplicaciones distribuidas y el mantenimiento de las mismas. El *Framework* realiza esta tarea de forma automática mejorando el rendimiento y asegurando el funcionamiento correcto de todas las aplicaciones.

Algunas desventajas que presenta esta plataforma es el mantenimiento en múltiples lenguajes. Mantener un proyecto en múltiples lenguajes es costoso. Si una aplicación está realizada en varios lenguajes se necesitan expertos en varios lenguajes para entenderla y mantenerla, aumentando los costos. Otra de las desventajas fundamentales es que no es multiplataforma, o sea ella solo está disponible para la familia de Windows. El tema de las licencias es también otras de las desventajas por lo que es un código cerrado y no hay licencias libres. Sin embargo, es una infraestructura de desarrollo robusta y fue la solicitada por el cliente SAREN en el alcance del proyecto.

1.7.1.1 Crystal Reports .Net

Crystal Reports para Visual Studio .NET es la herramienta de elaboración de informes estándar para Visual Studio .NET. Permite crear contenido interactivo con calidad de presentación en la plataforma .NET, lo que ha supuesto una ventaja fundamental para Crystal Reports durante años (CrystalReports, 2010).

De esta forma se asegura que Crystal Reports permite:

- Transformar rápidamente cualquier fuente de datos en contenido interactivo.
- Integrar estrechamente capacidades de diseño, modificación y visualización en aplicaciones .NET, Java o COM.
- Permitir a los usuarios finales acceder e interactuar con los reportes a través de portales Web, dispositivos móviles y documentos de Microsoft Office®

1.8 Frameworks de Desarrollo

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

Framework define, en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular, que sirve como referencia para enfrentar y resolver nuevos problemas de índole similar.

Representa una arquitectura de *software* que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

Dado a que la solución presentada es un sistema de gestión, que se necesita sea seguro, con transacciones rápidas de información, flexible y de fácil mantenimiento, por lo antes expuesto se utilizó Spring.Net como *framework* de desarrollo.

1.8.1 Spring.Net

Spring.NET es un *framework* de aplicación que proporciona un soporte de infraestructura completa para el desarrollo de aplicaciones .NET. Le permite eliminar la complejidad incidental cuando se utiliza las bibliotecas de clases base, utilizando las mejores prácticas, tales como *Test Driven Development*. Spring.NET es creado, soportado y sostenido por SpringSource.

El diseño de Spring.NET se basa en la versión de *Java* de *Spring Framework*, que ha demostrado en el mundo real beneficios y se utiliza en miles de aplicaciones empresariales de todo el mundo. Spring.NET no es un puerto rápido de la versión de Java, sino más bien un puerto espiritual basado en los patrones de arquitectura y diseño que han sido probados y no se vinculan a una plataforma determinada. Con Sring.NET no es un todo o nada la solución, se puede utilizar la funcionalidad de sus módulos de forma independiente.

La plataforma. NET proporciona una gran cantidad de funcionalidades para la arquitectura y la construcción de aplicaciones, que abarcan todas las formas desde los bloques básicos de construcción con tipos primitivos y clases (y los medios para definir nuevas clases), hasta servidores de aplicaciones con todas las funciones y *frameworks* de programación Web (KATRIB, y otros, 2008).

El *framework* Spring tiene las mejores prácticas que han demostrado a lo largo de los años en numerosas aplicaciones y formalizarse como patrones de diseño, y de hecho codifica estos patrones como objetos de primera clase que se pueden quitar o integrar a la aplicación. Esta es una de las mayores ventajas, atestiguada por las numerosas organizaciones e instituciones que han utilizado el *framework* de Spring, brindando aplicaciones robustas y fáciles de mantener (POLLACK, Mark; EVANS, Rick; SEOVIC, Aleksandar, The Spring Java Team, 2010).

1.8.2 NUnit

Es una herramienta encargada de analizar ensamblados generados por .NET, interpretar las pruebas inmersas en ellos y ejecutarlas. Utiliza atributos personalizados para interpretar las pruebas y provee además métodos para implementarlas. Compara valores esperados y valores generados, si estos son diferentes la prueba no pasa, caso contrario la prueba es exitosa.

NUnit carga en su entorno un ensamblado y cada vez que ejecuta las pruebas que contiene, lo recarga. Lo que permite que se puedan tener ciclos de codificación y ejecución de pruebas simultáneamente, así cada vez que se compile no se tiene que volver a cargar el ensamblado al entorno de NUnit si no que este siempre obtiene la última versión del mismo (HUNT, y otros, 2004).

1.9 Sistema Gestor de Base de Datos

En estos sistemas se hace necesario gestionar la información de manera que persista, que se garantice su seguridad y esté siempre disponible para acceder a ella. Para que los usuarios puedan procesar, describir, administrar y recuperar los datos almacenados es necesario un sistema gestor de bases de datos o SGBD.

Un Sistema Gestor o Manejador de Bases de Datos (SGBD) es un conjunto de programas que permite a los usuarios crear y mantener una BD, por lo tanto, el SGBD es un *software* de propósito general que facilita el proceso de definir, construir y manipular la BD para diversas aplicaciones. El éxito del SGBD reside en mantener la seguridad e integridad de los datos (KUMAR, y otros, 2005).

1.9.1 Oracle

Oracle es un Sistema de Gestión de Bases de Datos Relacionales (SGBDR), desarrollado por la empresa *Oracle Corporation* surgido a finales de la década de 1970. Permite el almacenamiento de datos en tablas formadas por filas y columnas, y su posterior consulta y mantenimiento mediante un sencillo y potente lenguaje de consulta estructurado (SQL) (RAMIREZ, y otros, 2006).

Existen numerosos SGBDR´s y entre ellos el más popular a nivel profesional es el de Oracle.

El SGBDR de Oracle se instala en prácticamente cualquier equipo informático, pudiendo empezar por un ordenador personal basado en *Windows* e ir escalando a arquitecturas más complejas a medida que las necesidades de la carga de proceso lo requieran. Oracle en este sentido es un SGBDR muy amigable,

pues permite trasladar las estructuras de datos y desarrollos a arquitecturas más complejas sin necesidad de realizar cambios significativos.

Oracle está concebido con el fin de manejar grandes cantidades de información, además de admitir conexiones concurrentes de muchos usuarios hacia los mismos datos. La versión de Oracle propuesta implementa un buen proceso de réplica, lo que permite la persistencia y seguridad de la información.

Oracle 10g R2 Enterprise Edition RAC: Se requiere este potente gestor dada la gran cantidad de información que se maneja y la seguridad que requiere esta información. Dentro de sus principales funcionalidades aportadas se encuentran (KNOX, 2004):

- Soporte y tratamiento de una gran cantidad de datos (TBytes).
- Soporte de una gran cantidad de usuarios accediendo concurrentemente a los datos.
- Seguridad de acceso a los datos, restringiendo dicho acceso según las necesidades de cada usuario.
- Integridad referencial en su estructura de base de datos.
- Conectividad entre las aplicaciones de los clientes en sus puestos de trabajo y el servidor de datos Oracle (estructura cliente/servidor).
- Conectividad entre las bases de datos remotas (estructura de bases de datos distribuidas).
- Portabilidad.
- Compatibilidad.
- Altamente escalable.
- Gran capacidad de almacenamiento.
- Gran capacidad de réplica.
- Máxima seguridad.
- Administración simplificada.
- Alto rendimiento en transacciones.
- Sistemas de alta disponibilidad.
- Disponibilidad controlada de los datos de las aplicaciones.
- Gestión de la seguridad.

1.10 Normalización de Base de Datos

La teoría de la normalización se ha desarrollado para obtener estructuras de datos eficientes que eviten las anomalías de actualización. El concepto de normalización fue introducido por Edgar F. Codd y pensado para aplicarse a sistemas relacionales. Sin embargo, tiene aplicaciones más amplias (CODD, 2000). Es por lo tanto el proceso durante el cual los esquemas de relación insatisfactorios se descomponen, repartiendo sus atributos entre esquemas de relación más pequeños que poseen propiedades deseables.

La normalización es la expresión formal del modo de realizar un buen diseño y provee los medios necesarios para describir la estructura lógica de los datos en un sistema de información.

“Es un proceso complejo formado por muchas reglas específicas y distintos niveles de intensidad. La definición completa de normalización es el proceso de descartar la repetición de grupos, minimizar la redundancia, eliminar claves compuestas para la dependencia parcial y separar los atributos que no sean de la clave. En términos generales, las reglas de normalización se pueden resumir en una sola frase: Cada atributo (…) debe ser una realidad de la clave, toda la clave y nada más que la clave. Cada tabla debe describir sólo un tipo de entidad (…)” (RAMIREZ, y otros, 2006)(2011).

Algunas de las ventajas de la normalización de BD por la que se hizo necesaria su aplicación durante la realización del modelo de datos se mencionan a continuación:

- Integridad de datos.
- Consultas optimizadas.
- Creación y ordenación de índices más rápidas.
- Ejecución más rápida de la instrucción UPDATE.
- Resolución de concurrencias mejorada.

Para obtener un diseño de calidad, en el que estén presentes y se apliquen cada una de las ventajas anteriormente mencionas se debe lograr que la totalidad de las tablas o al menos la gran mayoría de ellas, estén en 3ra forma normal, por lo que deben cumplir con los criterios de normalización definidos a continuación.

Formas Normales (FN):

- 1ra FN: todos los atributos tienen que ser atómicos, no pueden existir atributos multievaluados, compuestos o derivados.
- 2da FN: no pueden existir dependencias parciales de la llave primaria.
- 3ra FN: no pueden existir dependencias transitivas.

1.11 Seguridad de la información

La norma ISO 17799:2005 establece los estándares utilizados para realizar la gestión de la seguridad de la información dirigidas a los responsables de iniciar, implantar o mantener la seguridad de una organización (2005).

La seguridad de la información se define como la preservación de:

- Confidencialidad. Aseguramiento de que la información es accesible sólo para aquellos autorizados a tener acceso.
- Integridad. Garantía de la exactitud y completitud de la información y de los métodos de su procesamiento.
- Disponibilidad. Aseguramiento de que los usuarios autorizados tienen acceso cuando lo requieran a la información y sus activos asociados.

Esta norma establece diez dominios de control que cubren por completo la Gestión de la Seguridad de la Información:

- Política de seguridad.
- Aspectos organizativos para la seguridad.
- Clasificación y control de activos.
- Seguridad ligada al personal.
- Seguridad física y del entorno.
- Gestión de comunicaciones y operaciones.
- Control de accesos.
- Desarrollo y mantenimiento de sistemas.
- Gestión de continuidad del negocio.
- Conformidad con la legislación

1.11.1 Firma Electrónica

La firma electrónica o digital puede ser definida como una secuencia de datos electrónicos que se obtienen mediante la aplicación a un mensaje determinado de un algoritmo de cifrado asimétrico o de clave pública, y que equivale funcionalmente a la firma autógrafa en orden a la identificación del autor del que procede el mensaje. Desde un punto de vista material, la firma digital es una simple cadena o secuencia de caracteres que se adjunta al final del cuerpo del mensaje firmado digitalmente (MARTÍNEZ, 2002) (RODRIGUEZ, 2004).

Según la Ley de la Comisión de las Naciones Unidas para el Derecho Mercantil Internacional (CNUDMI o UNCITRAL) en su Artículo 2 define la firma electrónica en general, como un conjunto de datos en forma electrónica consignados en un mensaje de datos o adjuntados o lógicamente vinculados al mismo, que pueden ser utilizados para identificar al firmante de ese mensaje e indicar que este aprueba la información allí contenida (URDANETA, 2010)

En Venezuela a partir del Decreto Ley sobre Mensajes de Datos y Firma Electrónica (DLMDFE), y como se establece en su Artículo 6, cuando se trate de actos o negocios jurídicos para los cuales la Ley exija la firma autógrafa, este requisito quedará satisfecho en los casos en que los mensajes de datos tengan asociado una firma electrónica. Asimismo, en el Artículo 16 se establece que la firma electrónica que permita vincular al signatario con el mensaje de datos y atribuir la autoría de éste, tendrá la misma validez y eficacia probatoria que la ley otorga a la firma autógrafa, y enuncia tres aspectos con los cuales debe cumplir la firma electrónica (PRADA, 2000) (URDANETA, 2010):

- Garantizar que los datos utilizados para su generación puedan producirse una sola vez, y asegurar, razonablemente, su confidencialidad.
- Ofrecer seguridad suficiente que no pueda ser falsificada con la tecnología existente en cada momento.
- No alterar la integridad del mensaje de datos.

La firma electrónica otorgada por los Proveedores de Servicios de Certificación Electrónica (PSC) de acuerdo al DLMDFE ofrece grandes beneficios para los usuarios, tanto desde el punto de vista tecnológico como jurídico, al garantizar la seguridad y la confidencialidad de que gozarán los documentos electrónicos, así como las transacciones firmadas que se realicen a través de medios telemáticos. Entre las principales ventajas que ofrece la firma electrónica se destacan las siguientes:

- Integridad de la Información.
- Autenticidad del Origen del Mensaje.
- No repudio del Origen.
- Imposibilidad de Suplantación.
- Auditabilidad.

Por esta razón se decidió incluir una firma electrónica como mecanismo de seguridad de los objetos digitales multipáginas que serán importados en el Centro de Datos del SAREN.

1.12 Calidad de Software

La Organización Internacional para la Estandarización es la agencia especializada en estandarización más grande a nivel mundial, establecida en 1947 con la finalidad de promover la estandarización internacional para facilitar el intercambio de bienes y servicios, así como de desarrollo científico y tecnológico. ISO establece los estándares de calidad en respuesta las necesidades claramente expresadas por los sectores y *stakeholders* sobre determinado producto.

Según ISO 8402, “Calidad son todas las características que permiten que un producto satisfaga necesidades explicitas o implícitas a un costo aceptable” (D’ANGELO, y otros, 2005). De esta manera podemos decir que la calidad de los productos puede ser medida a través de la comparación de sus características y atributos.

1.12.1 Métricas de Diseño

La medición es un elemento clave en cualquier proceso de ingeniería. Las medidas se emplean para comprender mejor los atributos de los modelos que se crean y evaluar la calidad de los productos de la ingeniería o de los sistemas que se construyan.

Aunque las métricas no suelen ser absolutas, proporcionan una manera sistemática de evaluar la calidad a partir de un conjunto de reglas definidas con claridad. También proporcionan al ingeniero información inmediata y en el sitio, no posterior al hecho, teniendo la ventaja de describir y corregir problemas potenciales antes de que se conviertan en efectos catastróficos (HAMILTON, 2004)(MILANÉS, 2007).

Las métricas para el diseño cuantifican los atributos del diseño para poder evaluar la calidad del mismo. Entre las mismas se incluyen (HUNT, y otros, 2004):

- Métricas arquitectónicas: consideran los aspectos estructurales del modelo de diseño.
- Métricas al nivel de componentes: indican la calidad del módulo al establecer medidas indirectas para la cohesión, el acoplamiento y la complejidad.
- Métricas de diseño de la interfaz: proporcionan un indicio de la facilidad con la que se usa la interfaz gráfica del usuario.
- Métricas especializadas en diseño orientado a objeto: se concentran en la medición que puede aplicarse a las características de clase y diseño (localización, encapsulamiento, ocultamiento de información, herencia y técnicas de abstracción de objetos) que convierten a una clase en única.

1.12.2 Pruebas de Caja Blanca

La prueba de caja blanca del *software* se basa en un examen cercano al detalle procedimental. Se prueban las rutas lógicas del *software* y la colaboración entre componentes, al proporcionar casos de pruebas que ejerciten conjuntos específicos de condiciones, bucles o ambos (PRESSMAN, 2005).

Estas requieren del conocimiento de la estructura interna del programa, deben garantizar como mínimo que:

- Se ejerciten por lo menos una vez todos los caminos independientes para cada módulo.
- Se ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsa.
- Ejecuten todos los bucles en sus límites y con sus límites operacionales.
- Se ejerciten las estructuras internas de datos para asegurar su validez.

1.13 Análisis crítico de las fuentes y bibliografías utilizadas

En el entorno del desarrollo de *software*, se han publicado numerosos artículos en revistas especializadas, libros y tesis que describen con un alto grado científico las herramientas y etapas en el proceso de desarrollo de aplicaciones informáticas (QUINTANA, 2006) (HERNÁNDEZ R.S., 2003).

Durante la realización de esta investigación, se efectúo primeramente una búsqueda exhaustiva de información sobre el tema que se pretende abordar y la fiabilidad de esta bibliografía es extremadamente importante en el resultado final, ya que esta enriquece el nivel científico del problema planteado.

Las bibliografías consultadas se pueden clasificar de la siguiente forma:

- Libros y revistas científicas especializadas.
- Trabajos de tesis de maestría.
- Sitios de Internet especializados.

El conjunto de documentos consultados, se puede afirmar que es actual porque aproximadamente el 60% ha sido publicado en los últimos cinco años. La información obtenida de sitios de Internet fue seleccionada cuidadosamente para garantizar su veracidad.

1.14 Conclusiones del capítulo

Es un capítulo que se realizó con carácter teórico, con el objetivo de efectuar una revisión bibliográfica minuciosa en los temas relacionados con la investigación como los metadatos y su vinculación con la programación orientada objetos y el proceso de desarrollo de *software*, esta conllevó a un análisis coherente, preciso y actualizado, del cual se obtuvieron los siguientes resultados:

Se expusieron las ventajas de las metodologías de desarrollo de *software* para seleccionar la empleada en el sistema. La metodología utilizada por ser simple, configurable, generar gran volumen de información y satisfacer los requisitos de la aplicación fue RUP.

Como herramienta para el modelado del diseño se usó el *Enterprise Architect*, por ser esta una herramienta profesional y que presenta las características necesarias para el modelado la solución propuesta y para el modelado de los datos se utilizó ER/Estudio.

Otro aspecto importante, fue que se ha realizado un análisis de la plataforma y *framework* de desarrollo a utilizar como Visual Studio .NET y Spring.Net respectivamente, que permitirán desarrollar una aplicación segura y robusta. Como gestor de BD se empleó Oracle por su seguridad, alto rendimiento en transacciones y escalabilidad. Para las pruebas de unidad se empleó el *framework* NUnit.

CAPÍTULO 2. SOLUCIÓN DEL MÓDULO PROPUESTO

En el presente capítulo se muestra la propuesta de solución del problema planteado, realizada con la utilización de las herramientas CASE, metodología y tendencias analizadas en el capítulo anterior. Se presenta la arquitectura del sistema con el fin de describir el mismo y como parte de la propuesta de solución se exponen los patrones utilizados, la descripción del proceso de Importación de Metadatos y los modelos de diseño, datos e implementación.

2.1 Arquitectura del sistema

El módulo de Importación de Metadatos está estructurado siguiendo el patrón arquitectónico en Capas ya que este facilita tanto la comprensión como la construcción del subsistema, se seleccionó teniendo en cuenta las características del *software* a desarrollar.

A continuación, se muestra en la Figura 2.1 la vista de la arquitectura del módulo de Importación de Metadatos.

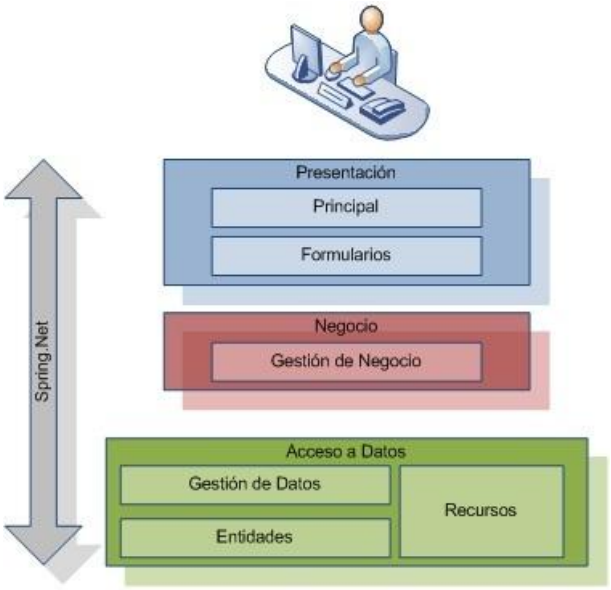

Figura 2.1 Vista de la arquitectura del módulo de Importación de Metadatos.

La vista arquitectónica abarca la distribución de los componentes que dan vida a la arquitectura para cada módulo de manera individual. Presenta un modelo multicapa, donde las capas tienen un orden lógico de procedencia, es decir las funcionalidades y recursos están encapsulados en cada nivel de acuerdo a la responsabilidad que se establece para cada uno de ellos. La arquitectura está constituida por las capas Presentación, Negocio y Acceso a Datos, de las cuales se da una breve descripción a continuación.

2.1.1 Nivel Presentación

Es la capa que contiene los componentes con los que interactuará el usuario. Esta tiene acceso a las funcionalidades que brinda el negocio y puede mostrar o capturar la información a través de los diferentes elementos que incluye. En este nivel se ubican los mecanismos de inicialización y configuración de algunos componentes radicados en la Arquitectura Base, los formularios de interfaz de usuario, los cuales responden a un diseño y comportamiento estándar en todos los módulos con vista a facilitar el trabajo con ellos y lograr la estandarización de la aplicación, así como las clases diseñadas para realizar las pruebas de unidad u otras a los diferentes bloques de código de este nivel.

2.1.2 Nivel de Negocio

Esta capa se encarga de recibir una petición de la capa superior, gestionar o procesar la misma, de ser necesario solicitándola a la capa de Acceso a Dato y finalmente envía la respuesta a quien realizó la petición. Contiene clases encargadas de resolver determinadas funcionalidades correspondientes a los Casos de Usos de un módulo, las interfaces para ofrecer funcionalidades que van a representar un nivel de abstracción y clases diseñadas para realizar pruebas de unidad u otras a los diferentes bloques de código de este nivel.

2.1.3 Nivel de Acceso a Datos

Es la capa inferior, por lo que los componentes que contiene desconocen los niveles superiores. Este nivel se limita al manejo de la información, ya sea para persistirla u obtenerla para su procesamiento y propagación por la aplicación. Contiene las clases que implementan la totalidad de las operaciones de persistencia y obtención de datos (DAO´s[2]), las interfaces que representan las funcionalidades que brinda

[2]Siglas en inglés de Objetos de Acceso a Datos.

esta capa, la entidades, así como las configuraciones y estructuras que presenta la capa para el trabajo correcto de determinadas funcionalidades para la persistencia.

En la Figura 2.2 se representa la vista lógica general para mayor entendimiento de la arquitectura. En ella se muestran las relaciones existentes entre las diferentes capas, la Arquitectura Base y Spring.Net.

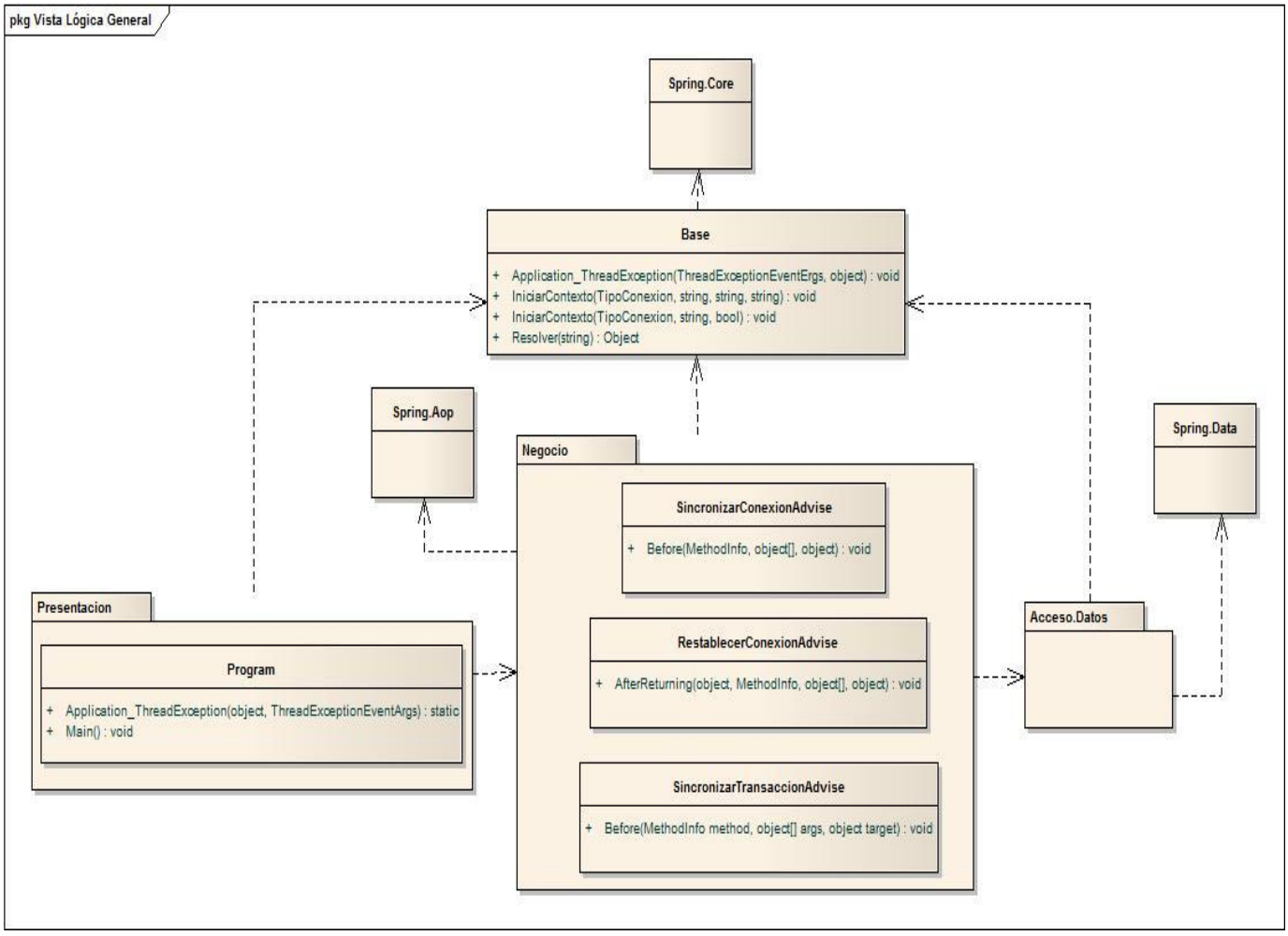

Figura 2.2 Vista Lógica General.

La clase Base en integración con Spring.Net, gestiona el manejo de excepciones Application_ThreadException, se encarga de cargar toda la configuración para que la arquitectura del sistema trabaje adecuadamente y la funcionalidad *Resolver* permite realizar la instanciación de los diversos objetos mediante el patrón Abstract Factory.

2.2 Patrones de diseño orientado a objetos utilizados

Entre los patrones de diseño orientados a objetos utilizados se encuentran:

Inyección de dependencias: Es cuando una clase principal hace uso de una clase secundaria sin hacer referencia directa a ella. Alguna entidad externa proveerá la clase secundaria a la clase principal en tiempo de ejecución, inyectará la dependencia. La clase secundaria deberá implementar una interfaz que la clase principal conocerá.

Este patrón en la aplicación es utilizado apoyándose en las facilidades de uso que brinda Spring.Net para la utilización del mismo. El empleo de este patrón se pone de manifiesto en la aplicación con la instanciación de los objetos de acceso a datos (DAOs) en RecursosNegocio.xml. A continuación, en la Figura 2.3 se muestra un ejemplo de cómo es utilizado este patrón.

```
<object id="IGtrImportar" type="GestionNegocio.GtrImportar, GestionNegocio">
  <property name="DaoImportar" ref="IDaoImportar" />
</object>
```

Figura 2.3 Ejemplo de utilización del patrón de diseño Inyección de Dependencias.

Fachada: Se utiliza para proporcionar una interfaz unificada de alto nivel para un conjunto de clases en un subsistema, haciéndolo más fácil de usar. Simplifica el acceso a dicho conjunto de clases, ya que el cliente sólo se comunica con ellas a través de una única interfaz.

Se trabaja con este patrón en todo momento, haciendo uso del *framework* Spring.Net a través de la Arquitectura Base. Es utilizado para abstraer al programador de la complejidad de las configuraciones requeridas por Spring.Net para aspectos como el manejo de excepciones, la instanciación de objetos y el sistema de mensajería. De esta manera la instanciación de un objeto por el programador quedaría de la forma que se muestra en la Figura 2.4.

```
IGtrImportar gtrImportar = Base.Resolver<IGtrImportar>();
```

Figura 2.4 Ejemplo de utilización del patrón de diseño Fachada.

Interceptor: Permite agregar fácilmente funcionalidad al sistema para cambiar su comportamiento dinámicamente, sin necesidad de recompilarlo, es decir, hace el cambio de comportamiento en tiempo de ejecución ya sea interponiendo una nueva capa de procesamiento o cambiando el destino dinámicamente,

ya que interpone objetos, los cuales pueden interceptar llamadas e insertar un procesamiento específico que puede estar basado en el análisis del contenido, además de redireccionar una llamada a un punto diferente.

Este patrón se utiliza en la administración de transacciones. En la aplicación se utilizan los interceptores del *framework* Spring.Net con el objetivo de realizar el análisis de métodos para determinar si necesitan algún tratamiento especial antes, durante y después de su ejecución. En la Figura 2.5 se muestra un ejemplo de su uso.

```
[Transaction]
public void SalvarMetadatos(UDocumental documento, DatosTomoPub tomo)
{
    DaoImportar.SalvarMetadatos(documento,tomo);
}
```

Figura 2.5 Ejemplo de utilización del patrón de diseño Interceptor.

2.3 Descripción del proceso de Importación de Metadatos

El sistema DigiPyrus es una aplicación de escritorio hecha a la medida y que está dedicada a la digitalización de los documentos del fondo documental histórico de los Registros y Notarías de la República Bolivariana de Venezuela. En su diseño cuenta con un conjunto de módulos para la gestión de la información, entre los que se encuentra el módulo desarrollado para el proceso de importación de metadatos. En el Anexo 2 se puede observar una vista del módulo.

2.3.1 Importar Metadatos

Se inicia con el objetivo de importar los objetos digitales multipáginas firmados digitalmente y los metadatos correspondientes, que se obtuvieron del proceso de digitalización. Una vez seleccionado en el menú la opción Importar Metadatos se muestra la interfaz donde se debe especificar el Camino para la Importación, donde se encuentra la carpeta con los objetos digitales y metadatos exportados del Centro de Digitalización.

La importación se realiza cumpliendo con el formato definido durante la exportación, y siguiendo el formato predefinido en documentos anteriores de Fase I de conjunto con el equipo de desarrollo del SAREN, el mismo sigue una estructura de directorios que permite el almacenamiento ordenado de la información y la identificación fácil de la misma para su importación en el Centro de Datos del SAREN. En la Figura 2.6 se muestra el formato antes mencionado.

Figura 2.6 Formato predefinido para el almacenamiento de la información.

Los objetos digitales multipáginas antes de ser importados tienen que cumplir de igual forma con dos requisitos, uno que estén firmados digitalmente con la firma electrónica suministrada a través de los PSC al SAREN, acreditados por la Superintendencia de Servicios de Certificación Electrónica (SUSCERTE), organismo creado a partir del DLMDFE y asignado al Centro de Digitalización, y que los metadatos presentes en el archivo DatosTomo.xml estén en el formato definido para su correcta administración por parte del sistema. En la Figura 2.7 se ilustra una imagen de un objeto digital firmado electrónicamente.

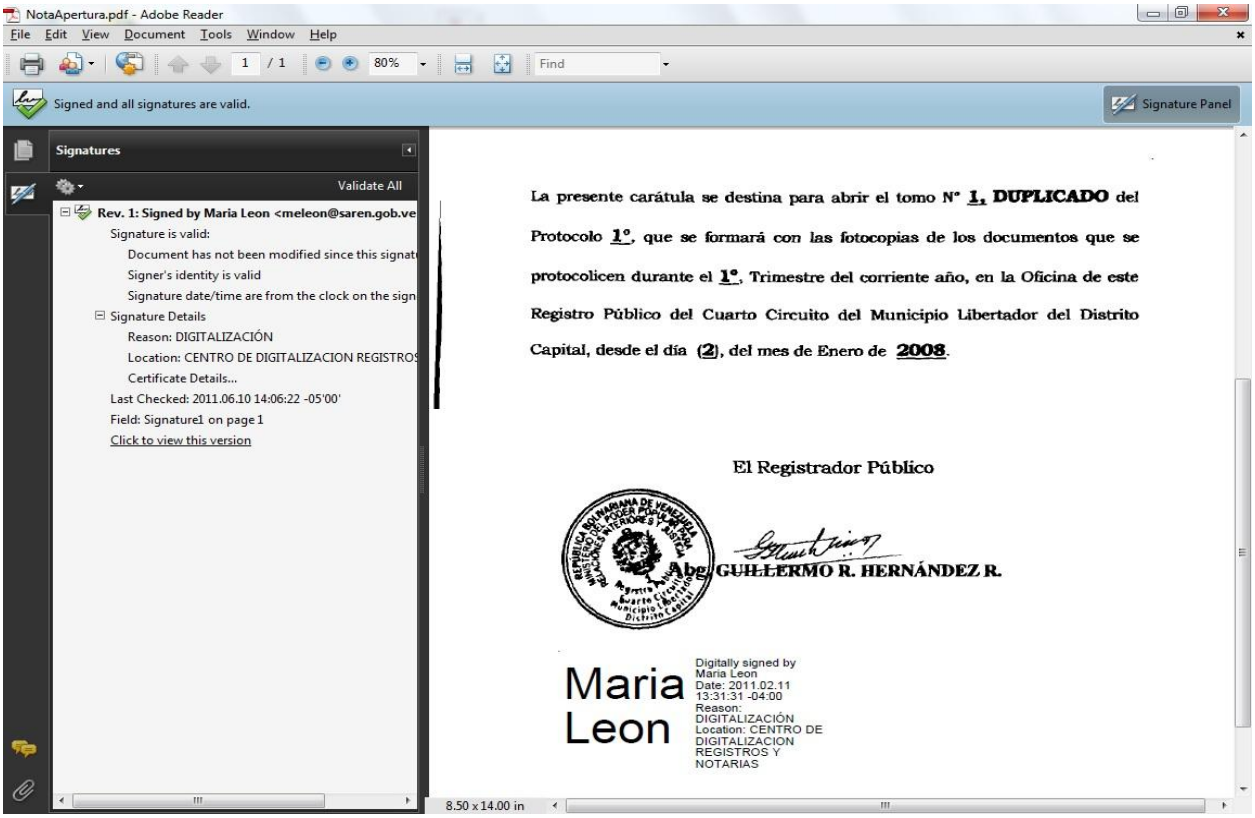

Figura 2.7 Imagen de un objeto digital firmado electrónicamente.

Una vez que el camino está seleccionado se comienza la importación de los objetos digitales multipáginas y sus metadatos correspondientes.

Puede darse el caso que uno o varios archivos seleccionados en la ruta especificada no cumplan con el formato predefinido en documentos exportados, por lo que el sistema mostrará un mensaje de error durante el proceso de importación de los mismos y este se irá visualizando en la lista de conclusiones del proceso conjuntamente con los datos de los archivos con información válida que se vaya importando, ya sean datos del Tomo, unidades documentales, actas de apertura o actas de cierre. Los errores que puedan ocurrir durante el proceso de importación son resumidos en ficheros logs que posteriormente son incorporados en el proceso de identificación de metadatos, el equipo de trabajo de esta área es el encargado de subsanar estos errores.

2.3.2 Buscar Tomos

Se inicia con el objetivo de buscar y mostrar los datos de los objetos digitales multipáginas y los metadatos correspondientes a los Tomos importados de la digitalización del fondo documental de los Registros y Notarías. Para pasar a buscar los datos del Tomo, se debe ingresar el Número del Tomo, el Trimestre, el Año y seleccionar la Oficina.

Una vez listados los Tomos que coincidan con el criterio de búsqueda especificado se pueden ver las unidades documentales pertenecientes a los mismos, para ello se selecciona la opción detalles y se muestra una interfaz con la lista de las unidades documentales que pertenecen al Tomo seleccionado y de cada una de ellas se muestra:

- Número de la Unidad.
- Folio de Inicio.
- Folio Fin.
- Fecha de Importación al SAREN.

Las búsquedas realizadas en este subproceso son lineales, basándose en consultas SQL las cuales son construidas a partir de la selección de los criterios de búsqueda definidos anteriormente.

2.3.3 Servicio de Importación en 2do plano

Se inicia con el objetivo de importar los objetos digitales multipáginas firmados digitalmente y los metadatos correspondientes, que se obtuvieron del proceso de digitalización. A diferencia de lo descrito en el Epígrafe 2.3.1 que interviene un usuario en la importación de los metadatos, este servicio una vez iniciado es capaz de realizar la importación de los objetos automáticamente con un mínimo de interacción humana.

Para iniciar el servicio se debe especificar primero el camino de importación, lugar donde van a ser copiados periódicamente los datos exportados en el Centro de Digitalización, además de establecer el intervalo de tiempo en que el servicio va a verificar si hay nuevos objetos para importar a la BD, ya sea en segundos, minutos, horas o días.

Al estar iniciado el servicio, lo único a tener en cuenta es que el usuario encargado de realizar la importación de metadatos copie los datos exportados del Centro de Digitalización en el directorio especificado para que el servicio busque las actualizaciones realizadas en el mismo y ejecute en el tiempo

antes establecido la importación de los objetos digitales y sus metadatos correspondientes hacia la BD del Centro de Datos del SAREN.

Este servicio no es independiente del sistema, funciona como parte del mismo módulo. Una vez iniciado, la aplicación pasa a trabajar en 2do plano, pero en caso de ocurrir una mala operación por parte de un usuario o una falla externa que conlleve al cierre de la aplicación, entonces el servicio también se detendría y sólo continuaría su ejecución si se abre nuevamente el sistema, volviéndose a iniciar el servicio como se explicó anteriormente.

2.3.4 Reporte de Fondo Documental

Se ejecuta con el fin de crear un reporte con el histórico de los Tomos y objetos digitales importados por cada uno de los estados de la República Bolivariana de Venezuela, en un período de tiempo determinado. Muestra un resumen de los Tomos y metadatos importados, los cuales se agrupan por el Estado de la nación, Tipo de oficina, Oficina a la pertenecen los documentos, y por cada Tomo se muestra el Número de Tomo, Año, Trimestre, Cantidad de unidades documentales y las unidades documentales no incluidas.

El histórico de los Tomos importados se muestra según el criterio de búsqueda especificado, para ello se debe seleccionar un Rango de fecha, el Estado y el Tipo de oficina, estos criterios pueden ser seleccionados todos a la vez o se puede seleccionar solamente los que estime conveniente a su informe.

2.4 Diseño e Implementación

En el diseño se modela el sistema, de forma que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se aplican. El modelo de implementación describe cómo los elementos obtenidos del modelo de diseño se implementan en términos de componentes, proporcionándole una ubicación a estos en los nodos físicos en los que funcionará la aplicación.

A continuación, se explican de manera general los modelos antes mencionados y se ilustran algunos diagramas generados durante el diseño del módulo.

2.4.1 Modelo de Diseño

El modelo de diseño es un modelo de objetos que describe la realización de los casos de usos y sirve como abstracción del modelo de implementación y su código fuente. Es usado tanto para concebir como para documentar el diseño de un sistema de *software*, 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. Incluye los diagramas de interacción, de clases y el modelo de datos.

2.4.1.1 Diagrama de Clases

Un diagrama de Clases representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas. Nos sirve para visualizar las relaciones entre las clases que involucran el sistema. Está compuesto por los siguientes elementos:

- Clase: atributos, métodos y visibilidad.
- Relaciones: herencia, composición, agregación, asociación y uso.

2.4.1.1.1 Diagrama de clases del proceso de Importación de Metadatos

Los diagramas de clases del módulo de Importación de Metadatos se dividen en la aplicación en cuatro procesos diferentes como se explicó anteriormente. Cada uno de estos procesos se divide en varios casos de usos y para cada caso de uso se realiza un diagrama de clases.

En el Anexo 3 se puede observar el diagrama de clases del dominio correspondiente al caso de uso Importar Metadatos. Se evidencian las entidades del dominio que permiten manejar los datos relacionados con el proceso de importación de metadatos y sus relaciones.

Las entidades que se muestran en el Anexo 3, van a permitir manejar los datos relacionados con la importación de los objetos digitales multipáginas y sus metadatos correspondientes hacia la BD del SAREN, organizando los mismos por oficina, protocolo, año, trimestre y número del Tomo.

En la Figura 2.8 se muestran las clases que implementan las operaciones de persistencia y obtención de datos y sus respectivas interfaces, las mismas están identificadas por <DaoNombreDao> y <IDaoNombreDao> respectivamente.

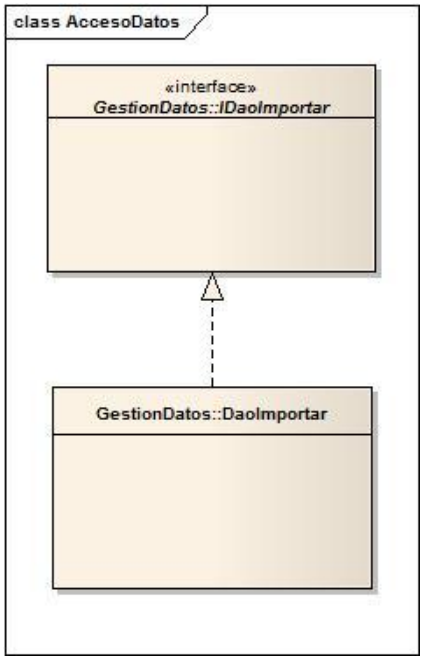

Figura 2.8 Diagrama de clases de la Capa de Acceso a Datos para el caso de uso Importar Metadatos.

Las operaciones de persistencia y obtención de datos pertenecientes al caso de uso Importar Metadatos están todas ubicadas en el DaoImportar. Estas clases se encuentran en la Capa de Acceso a Datos pues son las que controlan lo referente a la información que se encuentra o se persiste en la BD.

En la Figura 2.9 se muestran las clases Gestores que tienen como objetivo resolver determinadas funcionalidades dentro de los casos de uso y sus interfaces, estas están identificadas por <GtrNombreGestor> y <IGtrNombreGestor> respectivamente.

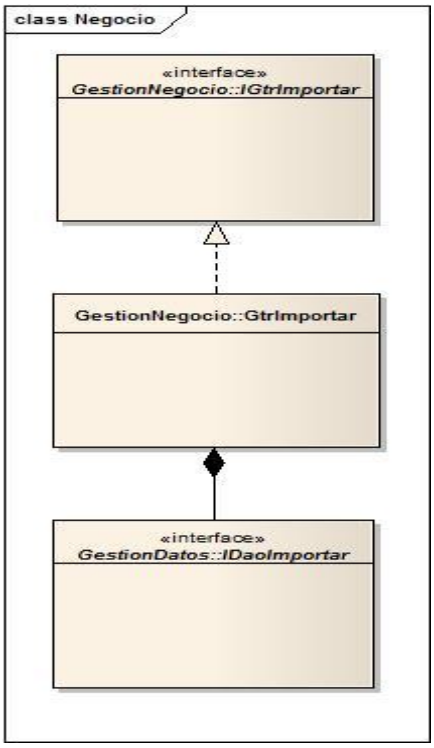

Figura 2.9 Diagrama de clases de la Capa de Negocio relacionadas con el caso de uso Importar Metadatos.

El gestor mostrado en el diagrama de clases de la Figura 2.9 está creado según la necesidad del diseño. Este realizará solamente las funciones de importación para el caso de uso Importar Metadatos, de esta forma se evidencia que la capa de Negocio puede realizar peticiones a la capa de Acceso a Datos, para responder a las solicitudes realizadas por la capa de presentación.

En el Anexo 4 se reflejan las relaciones entre clases existentes en el diagrama de clases del caso de uso Importar Metadatos. Este diagrama está compuesto por las entidades de dominio, las clases daos y sus interfaces, las clases gestoras de negocio y sus interfaces, y las clases de interfaz, estas últimas identificadas por <FrmNombreFormulario>.

2.4.1.2 Diagramas de Interacción

Los diagramas de interacción ilustran el flujo de interacciones entre las clases y subsistemas participantes. Estos diagramas de interacción son normalmente diagramas de secuencia y diagramas de colaboración, los primeros muestran las interacciones entre objetos, ordenadas en secuencia temporal durante un escenario concreto y son los que encontrará en el Modelo de Diseño, mientras que los segundos destacan la organización de los objetos que participan en una interacción. En la Figura 2.10 se muestra el diagrama

de secuencias perteneciente al escenario de Buscar Tomos, en el caso de uso Emitir Reporte de Fondo Documental.

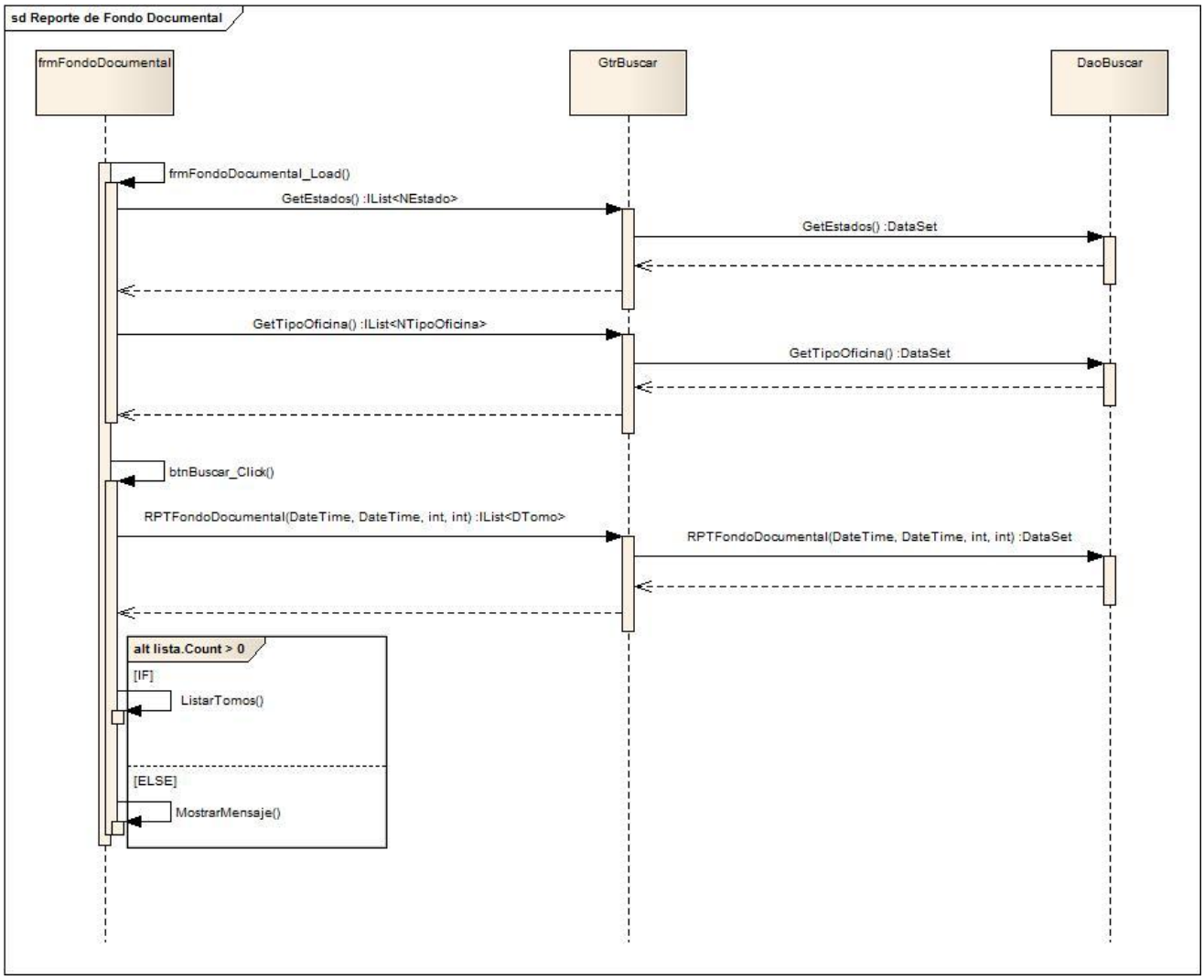

Figura 2.10 Diagrama de secuencia del escenario Buscar Tomos.

Este escenario tiene como objetivo realizar la búsqueda de los Tomos importados a la BD del Centro de Datos del SAREN, para posteriormente formular el reporte con los datos de los Tomos y objetos digitales encontrados después de haber aplicado el filtro de búsqueda por Fecha de importación, Oficina y Tipo de oficina.

2.4.2 Modelo de Datos

El modelo de datos es usado para describir las propiedades estáticas de una aplicación con un manejo intensivo de datos, es creado a partir de las clases persistentes y contienen tres elementos fundamentales los cuales son: los objetos, que son todas las entidades que manipulan los datos a persistir; los atributos, que son las características básicas de los objetos; y las relaciones, que son las que enlazan a dichos objetos entre sí.

En el Anexo 5 se muestra una porción del modelo de datos perteneciente al módulo de Importación de Metadatos. Este modelo de datos corresponde al proceso de importación de metadatos en los Registros Principales y Notarías, el modelo completo abarca un total de cuatro tipos de oficinas, Registros Principales, Registros Mercantiles, Registros Públicos y Notarías.

Las tablas presentes en el modelo que se puede observar en el Anexo 5 son las encargadas de almacenar la información referente a Tomos, Documentos (Unidades Documentales), Personas, Títulos y Sentencias, además de las imágenes digitales multipáginas importadas en conjunto con sus metadatos.

2.4.3 Modelo de Implementación

En este modelo se implementa el sistema diseñado en términos de componentes, ficheros de código fuente, ficheros de códigos binarios, ejecutables, entre otros.

Se podría decir también que la implementación sigue varios propósitos consigo como se muestra a continuación:

- Planificar las integraciones necesarias del sistema en cada iteración. Se sigue un enfoque incremental dando lugar a un sistema que se implementa en una sucesión de pasos pequeños y manejables.
- Distribuir el sistema asignando componentes ejecutables a nodos en el Diagrama de Despliegue.
- Implementar las clases y subsistemas encontrados durante el diseño. Las clases se implementan como componentes de ficheros que contienen código fuente.
- Probar los componentes individualmente y a continuación integrarlos y enlazándolos en uno o más ejecutables, antes de ser enviados para ser integrados, donde seguidamente se llevan a cabo las comprobaciones o prueba de unidad del sistema.

2.4.3.1 Diagrama de Componentes

Los diagramas de componentes muestran las dependencias del compilador y del *runtime* entre los componentes del *software*; por ejemplo, los archivos del código fuente y las DLL, estos diagramas son utilizados para representar la vista estática de un sistema. En el Anexo 6 se muestra el diagrama de componentes del sistema propuesto.

Como se puede observar en el Anexo 6 todas las capas de la aplicación hacen uso de los servicios generales de la arquitectura base, así como esta misma utiliza la librería Spring.Core. Además, la capa de Negocio utiliza directamente la librería Spring.AOP y la capa de Acceso a Datos utiliza Spring.Data para realizar las operaciones sobre la fuente de datos. El resto de las relaciones se obtienen de la representación de la arquitectura. Por último se muestran los componentes que componen cada capa los cuales se describen posteriormente.

En la capa de Presentación se encuentra la clase Program que constituye el punto de partida de la aplicación, esta establece las configuraciones necesarias para el correcto funcionamiento del sistema. A su vez encontramos también la librería que contiene las clases de interfaz visual que contiene el formulario principal del sistema y la plantilla del área de trabajo.

En la capa de Negocio se encuentra RecursosNegocio.xml que es donde se configura las instancias de los DAO´s a través de la inyección de dependencias y podemos encontrar la librería que contiene las clases Gestores.

En la capa de Acceso a Datos está ubicado dentro de la librería GestionDatos.dll el fichero ConfiguracionGestionDatos.xml, encargado de configurar las consultas de acceso a datos. Dentro de esta librería se puede encontrar el recurso Consultas, el cual incluye las consultas a utilizar durante todo el proceso de importación de metadatos, así como las clases DAO´s encargadas de realizar el acceso a datos hacia la BD del Centro de Datos del SAREN.

De igual manera se puede hallar la librería Entidades.dll que contiene todas las entidades con las que interactúa el sistema a la hora de realizar las operaciones de persistencia en la BD.

2.4.3.2 Diagrama de Despliegue

Los diagramas de despliegue muestran a los nodos procesadores, la distribución de los procesos y de los componentes. En estos se realiza la descripción de los dispositivos, sus distribuciones, así como el modo en el que se encuentran interconectados entre ellos en el despliegue del sistema.

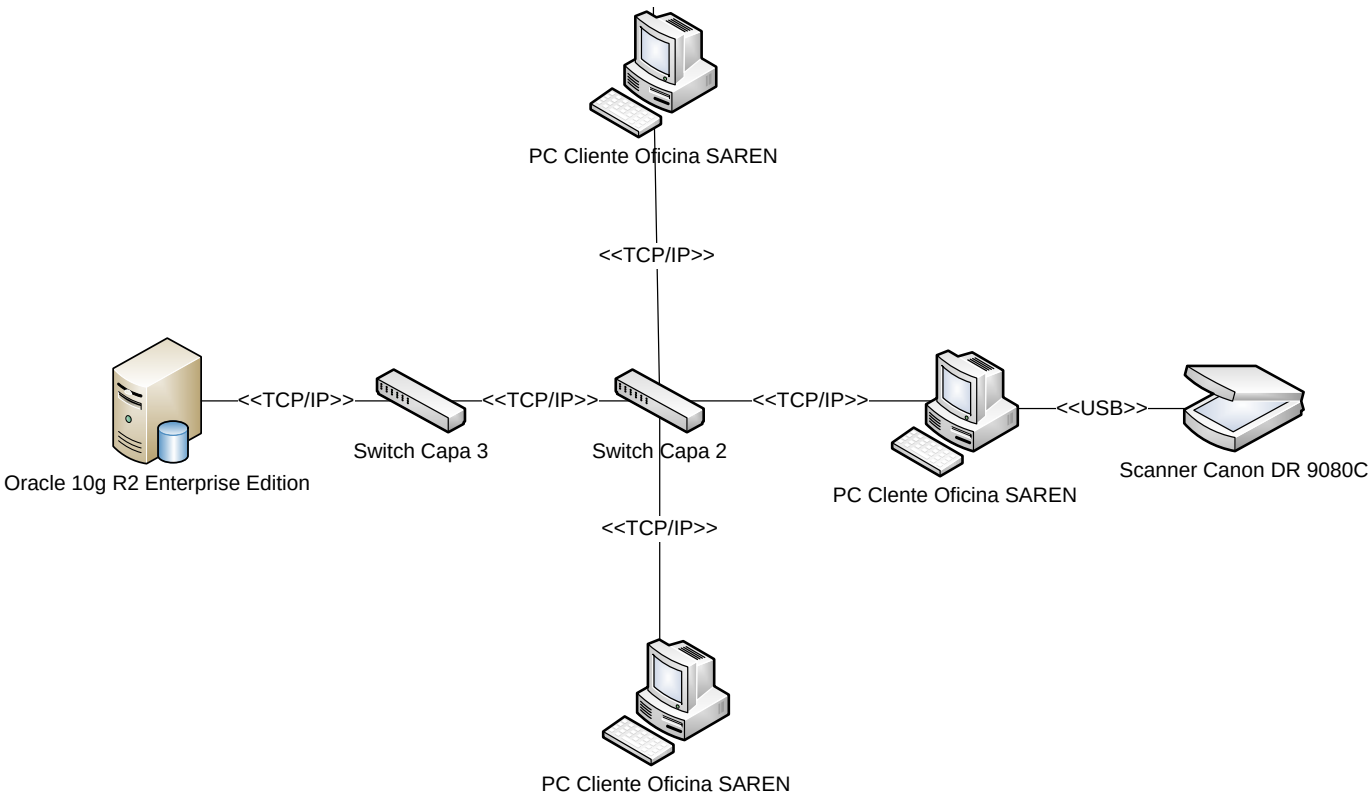

Figura 2.11 Diagrama de Despliegue del sistema DigiPyrus.

Seguidamente se muestra una breve descripción de los dispositivos que intervienen en el diagrama de despliegue ilustrado en la Figura 2.11.

Impresora HP Laser Jet 2430: Este dispositivo garantiza las funcionalidades de impresión de reportes y documentos que genera la aplicación dependiendo de las funcionalidades de la máquina cliente que la utilice a través de la red.

PC Cliente: Este nodo se corresponde con la máquina donde se encuentra instalado el módulo de Importación de Metadatos del sistema DigiPyrus, con las funcionalidades que se correspondan a cada puesto de trabajo.

Servidor Oracle 10g R2 Enterprise Edition: Este es el servidor de BD que se encuentra en el Centro de Datos. Está montado sobre el RedHat AS v4.0 y en este es donde se encontrará almacenada la totalidad de la información del sistema de todas las oficinas y hacia donde se realizará la importación de los objetos digitales y los metadatos.

Scanner Canon DR 9080C: Dispositivo que se utiliza para escanear los Tomos en el Centro de Digitalización del SAREN y la obtención de los objetos digitales multipáginas.

Switch Capa 2: Es el encargado de mantener conectados a la red todas las PC Clientes y las Impresoras dentro de la oficina, así como la comunicación con el switch del Centro de Datos.

Switch Capa 3: Está dedicado a la comunicación entre el servidor de BD y el resto de las oficinas.

2.5 Conclusiones del capítulo

En este capítulo se expusieron las principales características del diseño e implementación del módulo propuesto. Se explicaron los elementos que se tomaron en cuenta para obtención del modelo de diseño y el modelo de implementación. Se mostraron los patrones utilizados en la implementación, se analizó la arquitectura del módulo desarrollado y se explicaron los distintos subprocesos que forman parte del proceso de Importación de Metadatos.

Se obtuvo el diseño e implementación de un módulo para la importación de metadatos del sistema DigiPyrus del SAREN.

CAPÍTULO 3. VALIDACIÓN DE LOS RESULTADOS

Teniendo en consideración los objetivos planteados al inicio de la investigación y a modo de verificar el cumplimiento de los mismos, la validación de la solución informática propuesta es importante, pues da fe de la calidad del sistema desarrollado y del diseño realizado. En el presente capítulo se muestran los resultados obtenidos de la aplicación de pruebas y métricas de diseño, así como de la utilización de mecanismos que garantizan la integridad y confidencialidad de la información.

3.1 Integridad de los Datos

La integridad de datos se refiere a los valores reales que se almacenan y se utilizan en las estructuras de datos de la aplicación. La aplicación debe ejercer un control deliberado sobre todos los procesos que utilicen los datos para garantizar la corrección permanente de la información (2008).

Para obtener un modelo de datos que garantice la integridad en la BD existen una serie de reglas que se deben tener en cuenta a la hora de realizar el diseño, estas se ejemplifican a continuación.

3.1.1 Normalización de Datos

La normalización es la expresión formal del modo de realizar un buen diseño y provee los medios necesarios para describir la estructura lógica de los datos en un sistema informático (2011). Por ello se pretendió que de las 48 tablas que contiene el modelo de datos, sólo dejar en segunda forma normal una tabla que por necesidad de rendimiento en la BD lo requirió. Por lo que se concluye que el 97.9 % de las tablas están en tercera forma normal y sólo el 2.1% de ellas en segunda.

En la Figura 3.1 se muestra una porción del modelo de datos donde se observa un ejemplo de la normalización presente en las tablas de dicho modelo.

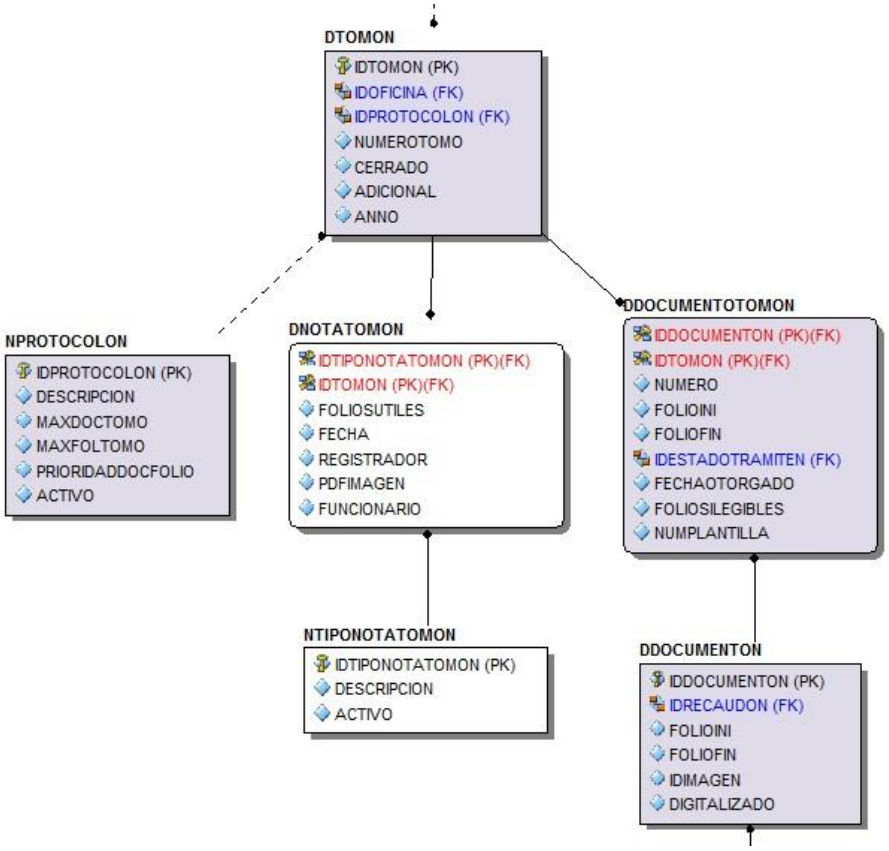

Figura 3.1 Ejemplo de modelo de datos en Tercera Forma Normal.

3.1.2 Integridad Referencial

La integridad referencial garantiza que las relaciones entre filas de tablas relacionadas son válidas y que no se eliminan o se cambian datos relacionados de forma accidental, esto significa que la clave externa de una tabla de referencia siempre debe aludir a una fila válida de la tabla a la que se haga referencia. La integridad referencial garantiza que la relación entre dos (o más) tablas permanezca sincronizada durante las operaciones de actualización y eliminación (2008).

En la Figura 3.1 mostrada anteriormente en el Epígrafe 3.1.1 Normalización de Datos también se puede observar cómo se comporta la Integridad Referencial en el modelo de datos perteneciente al módulo de Importación de Metadatos y como existen las siguientes restricciones:

- No se puede insertar una Nota en la tabla DNOTANTOMON, sin antes tener en la tabla NTIPONOTATOMON los datos que corresponden al Tipo de nota perteneciente a la Nota que se desea importar.
- No se puede insertar una Nota en la tabla DNOTATOMON, sin antes haber insertado en la tabla DTOMON los datos correspondientes al Tomo cuya Nota se está insertando.
- No se puede eliminar un Tomo de la tabla DTOMON, si este tiene asociados registros en las tablas DNOTATOMON o DDOCUMENTOTOMON.
- No se puede eliminar un Tipo de nota de la tabla NTIPONOTATOMON, si esta tiene algún registro en la tabla DNOTATOMON asociado a ella.
- No se puede modificar el identificador de un Tomo en la tabla DTOMON, si este tiene algún registro relacionado a él en las tablas DNOTATOMON o DDOCUMENTOTOMON.

Por lo expuesto anteriormente, se puede llegar a la siguiente conclusión sobre la integridad referencial presente en el modelo de datos del módulo de Importación de Metadatos:

- No se podrá introducir un valor en la tabla relacionada, si antes no ha sido introducida en la tabla principal.
- No se puede eliminar un registro de una tabla principal, si existen registros coincidentes en la tabla relacionada.
- No se puede cambiar un valor de la clave primaria en la tabla principal, si el registro tiene registros relacionados.

3.1.3 Manejo de Transacciones

Realizar el acceso a datos a través de transacciones, garantiza que si en un momento dado llegara a ocurrir algún error en la transferencia de datos, no se cree inconsistencia pues una transacción brinda la posibilidad de que todas las operaciones que se realicen dentro de ella se acepten o se reviertan de una vez. Si en algún momento determinado llegara a pasar una situación que pudiera provocar pérdida de información, esta no tendría lugar, pues al realizar las operaciones sobre la fuente de datos por medio de transacciones no terminaría la operación por lo que no quedarían datos incoherentes en la BD.

En la solución informática desarrollada todo acceso a datos está regido a través de estas transacciones, de manera que garantice que no haya incoherencia en los datos si llegara a ocurrir una anomalía en la transferencia de datos como se explicó anteriormente.

En la Figura 3.2 se puede observar un ejemplo de cómo a través de los interceptores de Spring.Net se realiza el manejo de transacciones. El método SalvarMetadatos del gestor GtrImportar se encarga de salvar los metadatos de un objeto digital perteneciente al Tomo activo en el momento de la importación, debido a que este posee datos que son salvados en varias tablas, el uso de una transacción permite que si en una de las operaciones se desata un error, entonces la operación completa falla por lo que no se salva hacia la BD ningún dato perteneciente al objeto en cuestión, de modo que no cree inconsistencia en la fuente de datos.

```
[Transaction]
public void SalvarMetadatos(UDocumental documento, DatosTomoPub tomo)
{
    DaoImportar.SalvarMetadatos(documento,tomo);
}
```

Figura 3.2 Ejemplo de utilización de transacciones en el método SalvarMetadatos.

3.2 Confidencialidad

Según ha sido certificado por la norma ISO-17799, la confidencialidad va dirigida a “garantizar que la información es accesible sólo para aquellos autorizados a tener acceso” (D’ANGELO, F.; GARCIA, D.; HERRERA, C., 2005). En el módulo de Importación de Metadatos del sistema DigiPyrus su aplicación práctica está basada fundamentalmente en proteger la aplicación informática y la BD contra accesos no autorizados.

Para realizar cualquier operación sobre la aplicación o la BD existen tres preguntas muy importantes que se tuvieron en cuenta en el desarrollo de la misma, las cuáles son:

¿Qué datos necesita?

¿Qué operaciones va a realizar sobre los datos?

¿Qué usuarios tienen privilegios sobre dichas operaciones?

3.2.1 Autenticación y Cuentas de Usuario

Es necesario que cada usuario de la aplicación posea un identificador único que le permitirá tener acceso a las funcionalidades del sistema y de esta forma acceder a las operaciones permitidas según su nivel en la BD.

La aplicación informática desarrollada cuenta con un control de acceso basado en usuarios y roles, por lo que inicialmente no podrá acceder a ninguna funcionalidad, si antes no se autentica satisfactoriamente con un usuario válido y con permisos en la aplicación. Un ejemplo de esto se muestra en la Tabla 3.1.

Tabla 3.1 Ejemplo de permisos en la aplicación para un usuario.

Propiedad	Valor
Nombre de Usuario	usuario1
Contraseña	**********
Roles del Sistema	Identificador, Importador
Nombre de Usuario	usuario2
Contraseña	**********
Roles del Sistema	Importador

Cada usuario para que tenga permisos de autenticación sobre la aplicación debe tener antes asignado al menos uno de los roles válidos del sistema, que le dan acceso sobre operaciones específicas dentro del mismo.

El usuario1 presente en el ejemplo mostrado en la Tabla 3.1, es válido para el sistema DigiPyrus pues posee permisos sobre áreas de procesos específicas dentro de la aplicación. Sin embargo si el usuario sólo tiene asignado el rol de Importador como usuario2, el sistema mostrará las acciones pertenecientes al módulo de Importación de Metadatos.

Identificador: Se refiere a la persona encargada de identificar los metadatos en los objetos digitales multipáginas obtenidos del proceso de Digitalización.

Importador: Está referido a la persona encargada de realizar la importación de los objetos digitales multipáginas al Centro de Datos del SAREN.

Medidas preventivas presentes en el sistema DigiPyrus

Como forma de prevenir autenticaciones indebidas de personas no autorizadas se tomaron algunas decisiones claves sobre la seguridad en la aplicación, a continuación se muestran algunas aplicadas en el desarrollo del sistema:

- Demorar la respuesta de la aplicación ante claves erróneas.
- Ir aumentando la demora de la respuesta por cada intento erróneo.
- Cerrar la aplicación una vez alcanzado el tercer intento de autenticación fallido y alertar sobre dicho evento.
- Hacer chequeos periódicos de procesos que llevan demasiado tiempo corriendo, permisos erróneos y actividades extrañas.

3.2.2 Permisos y Roles

Con el fin de otorgar o revocar privilegios de acceso a la BD se utilizaron los permisos y roles en la creación de los usuarios de la aplicación.

Permiso: Está definido como el privilegio de acceso sobre objetos de la Base de Datos.

Rol: Se define como el conjunto de privilegios que simplifica la gestión de los mismos.

El primer privilegio que debe tener un usuario de la aplicación es el de poder conectarse a la BD y esto se puede hacer de la forma mostrada en la Figura 3.3.

```
-- Grant object privileges
grant create session to REGISTRO;
grant connect to REGISTRO;
```

Figura 3.3 Asignar privilegio de inicio de sesión al usuario registro.

En el ejemplo de la Figura 3.3 se muestra la asignación de privilegios para crear sesión y conexión del usuario REGISTRO, esto garantiza de forma exclusiva que dicho usuario se pueda conectar, pero no le da permisos para realizar operaciones sobre objetos de la BD. Para asignar estos permisos se definieron Roles que se van asignando a cada usuario según su nivel de autoridad en la aplicación.

Los roles mostrados en el Epígrafe 3.2.1 carecen de validez o autoridad si antes no tienen un respaldo físico en la BD, por lo que para el Rol Identificador e Importador de la aplicación existen los roles físicos Identificador e Importador respectivamente en la BD.

Los usuarios de la BD no tienen asignados directamente privilegios sobre objetos, si no que se le asignan roles específicos y a estos roles se le dan los privilegios necesarios para el trabajo con los objetos.

Por ejemplo si el usuario usuario2 sólo tiene asignado el rol de Importador en la aplicación entonces en la BD le corresponden los roles físicos Importador e IniciarAplicacion.

IniciarAplicacion: Este es un rol común para todos los usuarios del sistema que da acceso a los paquetes y procedimientos necesarios para iniciar sesión y poder acceder a las operaciones de autenticación. En la Figura 3.4 se muestra el código de ejemplo que contiene este rol.

```
-- Create the role
create role INICIARAPLICACION;
-- Grant/Revoke object privileges
grant execute on DEVCONPC to INICIARAPLICACION;
grant execute on DEV_GRACE_TIME to INICIARAPLICACION;
grant execute on PKG_APLICACION to INICIARAPLICACION;
grant execute on PKG_DFRAMEVERSION to INICIARAPLICACION;
grant execute on PKG_OFICINA to INICIARAPLICACION;
grant execute on PKG_PUNTO to INICIARAPLICACION;
grant execute on PKG_ROL to INICIARAPLICACION;
grant execute on PKG_USUARIO to INICIARAPLICACION;
grant execute on PKG_USUARIOROL to INICIARAPLICACION;
grant execute on SPAPLICACIONINSERT to INICIARAPLICACION;
grant execute on SPAPLICACIONUPDATE to INICIARAPLICACION;
grant execute on SPCHANGEUSUARIO_PASSWORD to INICIARAPLICACION;
grant execute on SPEXCEPCIONINSERT to INICIARAPLICACION;
grant execute on SPUSUARIOACTUALIZARESTADOPASS to INICIARAPLICACION;
```

Figura 3.4 Creación del rol IniciarAplicacion.

Se puede observar que al usuario2 sólo se le han asignado permisos específicos agrupados en el rol IniciarAplicacion. Este no tiene privilegios de inserción, modificación o eliminación sobre ninguna tabla de la BD, en cambio los paquetes sobre los que se le asignaron permisos realizan estas operaciones una vez que el usuario se ha autenticado para cargar las configuraciones iniciales y registrar las trazas en la aplicación.

Importador: Este rol es el responsable de dar permisos al usuario del área de importación de metadatos, de manera que pueda ejecutar sin problemas cada una de las operaciones del módulo. En la Figura 3.5 se muestra el código de ejemplo de dicho rol.

```
-- Create the role
create role IMPORTADOR;
-- Grant/Revoke object privileges
grant execute on CERRARSESION to IMPORTADOR;
grant execute on INICIARSESION to IMPORTADOR;
grant execute on PKG_SESIONES to IMPORTADOR;
grant execute on PKG_UTILES to IMPORTADOR;
grant execute on PKG_UTILIDADES to IMPORTADOR;
grant execute on PKG_UNIDADESDOC to IMPORTADOR;
grant execute on RESETEARSESION to IMPORTADOR;
-- Grant/Revoke role privileges
grant iniciaraplicacion to IMPORTADOR;
```

Figura 3.5 Creación del rol Importador.

En esta Figura 3.5 se observa cómo además de asignar permisos sobre paquetes se asigna al rol Importador el rol IniciarAplicacion y de igual forma sólo se le asignan permisos de ejecución sobre los paquetes que son necesarios en el proceso de importación de metadatos, excluyendo al usuario de las tablas y objetos que son utilizadas en el desarrollo del mismo.

Cualquier usuario creado en la BD a partir de la solución informática desarrollada cumplirá con lo explicado anteriormente, ello demuestra que con una distribución efectiva de permisos a través de roles se puede garantizar que la información es accesible sólo para aquellos autorizados a tener acceso. En la Figura 3.6 se muestra un ejemplo de cómo debe quedar un usuario de la aplicación dentro de la BD.

```
-- Create the user
create user carlos
  identified by ImportacionRN@CentroDatos
  default tablespace TS_USUARIOS
  temporary tablespace TS_TEMP
  profile RN_PROFILE;
-- Grant/Revoke role privileges
grant iniciaraplicacion to carlos;
grant importador to carlos;
grant connect to carlos;
-- Grant/Revoke system privileges
grant create session to carlos;
```

Figura 3.6 Creación del usuario carlos en la BD.

3.3 Firma electrónica

La utilización de la firma electrónica presente en cada uno de los objetos digitales multipáginas exportados hacia la BD del SAREN, permitió brindar al cliente la seguridad de que los datos importados en la BD del Centro de Datos no serían alterados en el trayecto del Centro de Digitalización al MPPRIJ, ya que la firma electrónica permite al usuario detectar cualquier modificación o alteración de la información contenida en el mensaje de datos firmado, durante los procesos de transferencia, almacenamiento o manipulación de este. Por tanto, proporciona una protección en caso de modificaciones del contenido ya sean intencionales o accidentales como se abordó en el Capítulo 1.

El paquete empleado brinda la certificación de la firma electrónica en Venezuela pues fue suministrado a través de SUSCERTE. Una imagen de este paquete se puede observar en el Anexo 7.

3.4 Pruebas de Importación

Con el fin de comprobar la fidelidad de los objetos digitales (OD) importados y que el sistema informático desarrollado cumpliera con lo pactado con el cliente referente a la pérdida y duplicidad de información, así como a la optimización del proceso de importación, se implementó una prueba que consistió en importar de manera trimestral y de forma continua los metadatos y objetos digitales hacia una computadora personal. Los objetos digitales fueron importados por la operación de Importación de Metadatos del módulo desarrollado. En las pruebas realizadas durante las Fases I y II del proyecto, las cantidades importadas de objetos digitales se realizaron durante aproximadamente unas 10 y 6 horas de duración en Fase I y II respectivamente.

En la Tabla 3.2 se muestra un resumen de los valores obtenidos durante la importación de objetos digitales en el Centro de Datos del SAREN, y a su vez se establece una comparación entre las Fases I y II del proyecto.

Tabla 3.2 Resumen de los valores obtenidos de la importación durante las Fases I y II.

Fase	Período	Exportado (OD)	Importado (OD)
I	Ene-Mar	544750	545423
	Abr-Jun	654300	654001
	Jul-Sep	754900	753522
	Oct-Dic	665100	655200
II	Ene-Mar	3523	3523
	Abr-Jun	4565	4565
	Jul-Sep	5807	5807
	Oct-Dic	6211	6211

Como se muestra en la tabla anterior, en la Fase I del proyecto a pesar de que se digitalizó un mayor volumen de información, la importación de metadatos hacia el Centro de Datos estuvo sujeta a errores, pues en ocasiones al no llevar un buen control sobre cuándo y qué se exportaba se tendía a importar varias veces la información que estaba presente en dicha BD, o bien por no contar con un mecanismo fiable de validación de metadatos, habían objetos digitales que no eran insertados en el proceso de importación y se perdía esta información. Esto traía como consecuencia que después de terminado el

proceso de importación, el DBA tenía que estar ejecutando consultas SQL directamente en el gestor de BD para comparar datos e identificar posibles errores cometidos por la aplicación, una vez identificado proceder de forma manual a reparar el daño ocasionado, tarea que podía conllevar a varios días.

En la Fase II del proyecto, con la utilización del módulo de Importación de Metadatos desarrollado, se evidenció que con un control estricto sobre los datos exportados e importados y la implementación de un mecanismo de validación certero sobre los metadatos, se garantiza que una vez terminado el proceso de importación y mediante la obtención de los reportes se pueda verificar que los objetos digitales con metadatos exportados coinciden en su totalidad con los importados en dicho proceso.

Como se puede observar los resultados obtenidos fueron satisfactorios pues se logró eliminar durante la Fase II del proyecto, la pérdida y duplicidad de los objetos digitales importados.

En el Anexo 8 se muestra un ejemplo de un reporte obtenido una vez finalizado el proceso de importación de metadatos. En ese caso se importaron 280 Tomos para un total de 14000 objetos digitales importados, en el ejemplo mostrado cada Tomo está conformado por 50 objetos digitales.

3.5 Métricas aplicadas al Diseño del Software

3.5.1 Tamaño de Clase

La clase es la unidad fundamental del diseño orientado a objetos, por lo que es importante tener una forma de saber si se han diseñado con buena calidad. Para ello se miden las clases individuales, las jerarquías de clases y sus colaboraciones.

Su tamaño general está determinado por las siguientes medidas:

- El número total de operaciones que están encapsulados dentro de la clase.
- El número de atributos que están encapsulados por la clase.

Los valores grandes de Tamaño de Clase (TC) indican que tal vez tenga demasiada responsabilidad, haciendo difícil su implementación y prueba, mientras que valores pequeños permiten que sea altamente reutilizable.

En la Tabla 3.3 se muestran los intervalos de valores a través de los cuales se determina si una clase según su TC es de tamaño pequeño, medio o grande.

Tabla 3.3 Relación de intervalos de valores de TC.

Umbral	TC (Total de Atributos y Operaciones)
Menor o igual que 20 (TC <= 20)	Pequeño
Entre 20 y 30 (20 < TC <= 30)	Medio
Mayor que 30 (TC > 30)	Grande

En la Tabla 3.4 se resume el resultado de la aplicación de esta métrica.

Tabla 3.4 Resultados de la métrica Tamaño de Clase.

Total de Clases	TC Pequeño	TC Medio	TC Grande	Promedio de Atributos	Promedio de Operaciones
25	24	1	0	4.44	2.36

Con la aplicación de esta métrica al diseño del módulo desarrollado para el proceso de Importación de Metadatos, se obtuvo que de las 25 clases que contiene el diseño, 24 son de tamaño Pequeño y 1 de tamaño Medio, con un promedio de 4.44 atributos y 2.36 operaciones, por lo que se deduce que el resultado obtenido de la aplicación de esta métrica es positivo pues corrobora que las clases implementadas son reusables gracias al bajo nivel de responsabilidades y fáciles de comprobar.

3.5.2 Árbol de Profundidad de Herencia

Esta métrica se aplica a una jerarquía de clases, permitiendo conocer el nivel de complejidad en el momento de predecir el comportamiento de alguna de sus clases y la complejidad del diseño realizado. Se define como la longitud máxima desde el nodo padre hasta la raíz más lejana en el árbol que representa a la jerarquía y el valor obtenido se denomina Árbol de Profundidad de Herencia (APH) (MILANÉS, 2007).

A medida que crece el valor del APH, es más probable que las clases de niveles inferiores hereden muchos métodos. Esto da lugar a posibles dificultades cuando se intenta predecir el comportamiento de una clase y por lo tanto, mantenerla. Una jerarquía profunda lleva también a una mayor complejidad de diseño. Los valores grandes de APH implican que se pueden reutilizar muchos métodos, lo que debe ser considerado como un elemento a favor del fácil mantenimiento del sistema.

Al analizar la jerarquía de clases presentes en todas las capas se obtuvo que el número mayor de APH sea de 2 por lo que se puede concluir además que el diseño realizado presenta poca complejidad.

3.5.3 Número de Descendientes

Un descendiente es una subclase que se encuentra inmediatamente subordinada a otra en la jerarquía de clases. A medida que crece el número de descendientes (NDD) se incrementa la reutilización, pero podría diluirse la abstracción que presenta la clase predecesora si alguno de los descendientes no es un miembro apropiado de la clase padre.

En la solución propuesta las entidades tienen NDD máximo de 2, mientras que los formularios heredan todos de frmPlantilla, por lo que se obtiene un número elevado de NDD aumentando la reutilización del código, puesto que todos los formularios tienen una estructura y comportamiento común en forma de asistente.

3.6 Resultados obtenidos de las pruebas del NUnit

Con el fin de realizar las pruebas con NUnit se realizaron varias iteraciones en las cuales en etapas tempranas se obtuvieron algunos errores, pero luego se fueron corrigiendo hasta lograr que todos los métodos pasaran estas pruebas.

Para poder realizarlas es necesario primeramente desarrollar clases que contengan métodos de prueba, a continuación se muestran algunos de los métodos desarrollados.

En la Figura 3.7 se muestra el método Existe, encargado de buscar un objeto digital en la BD para saber si existe.

```
/// <summary>
/// Método Existe, se utiliza para saber si una Unidad Documental
/// ya existe en la Base de Datos.
///</summary>
[TestMethod()]
public void ExisteTest()
{
    var target = Base.Resolver<GtrImportar>();
    var documento = Base.Resolver <UDocumental>();
    var tomo = Base.Resolver <DatosTomoPub>();
    const bool expected = true;
    bool actual = target.Existe(documento, tomo);
    Assert.AreEqual(expected, actual);
}
```

Figura 3.7 Prueba al método Existe.

En la Figura 3.8 se muestra uno de los principales métodos del proceso de importación de metadatos, este se dedica a salvar hacia la BD un objeto digital junto a sus metadatos, en este caso este método es

utilizado para insertar lo metadatos de las oficinas de Registros Públicos, Registros Mercantiles y Notarías.

```
/// <summary>
/// Se utiliza para salvar los metadatos correspondientes
/// a las restantes oficinas (Reg. Publico, Reg. Principal, Notaria)
///</summary>
[TestMethod()]
public void SalvarMetadatosTest()
{
    var target = Base.Resolver<GtrImportar>();
    var documento = Base.Resolver<UDocumental>();
    var tomo = Base.Resolver<DatosTomoPub>();
    const bool expected = true;

    target.SalvarMetadatos(documento, tomo);

    bool actual = target.Existe(documento, tomo);
    Assert.AreEqual(expected, actual);
}
```

Figura 3.8 Prueba al método SalvarMetadatos.

En la Figura 3.9 se muestra el método SalvarMetadatosMercantil, este al igual que el mostrado en la Figura 3.8 se dedica a salvar un objeto digital junto a sus metadatos para la BD, pero en este caso se especializa en los objetos pertenecientes a las oficinas de los Registros Mercantiles.

```
/// <summary>
///Se utiliza para salvar los metadatos correspondientes
///a las oficinas Mercantiles.
///</summary>
[TestMethod()]
public void SalvarMetadatosMercantilTest()
{
    var target = Base.Resolver<GtrImportar>();
    var documento = Base.Resolver<UDocumental>();
    var tomo = Base.Resolver<DatosTomoPub>();
    const bool expected = true;

    target.SalvarMetadatosMercantil(documento, tomo);

    bool actual = target.Existe(documento, tomo);
    Assert.AreEqual(expected, actual);
}
```

Figura 3.9 Prueba al método SalvarMetadatosMercantil.

En la Figura 3.10 se muestra el método encargado de obtener los datos para generar el reporte de Fondo Documental Importado.

```
/// <summary>
/// Se utiliza para obtener el reporte de Fondo
/// Documental Importado
///</summary>
[TestMethod()]
public void RPTFondoDocumentalTest()
{
    var target = Base.Resolver<GtrImportar>();
    var fechaDesde = DateTime.Now.Date;
    var fechaHasta = DateTime.Now.Date;
    int idEstado = 4;
    int idTipoOficina = 1;
    IList<DTomo> expected = null;
    IList<DTomo> actual = target.RPTFondoDocumental(fechaDesde, fechaHasta, idEstado, idTipoOficina);
    Assert.AreEqual(expected, actual);
}
```

Figura 3.10 Prueba al método RPTFondoDocumental.

Se desarrollaron un total de 16 pruebas con el NUnit, distribuidas en tres iteraciones. Se obtuvieron en la primera un total de 3 errores que fueron subsanados, mientras que en la segunda iteración se obtuvo solamente un error condicionado por uno de los cambios realizados en los métodos rediseñados. Finalmente en la última se obtuvieron resultados satisfactorios para cada prueba realizada, en la Figura 3.11 se muestra el resumen brindado por el NUnit en esta iteración.

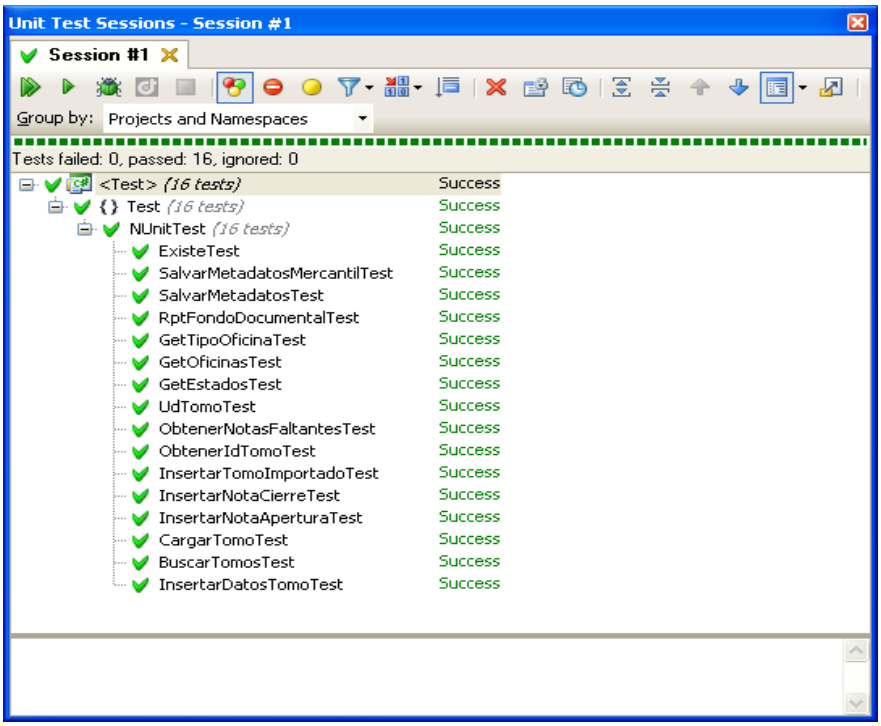

Figura 3.11 Vista resumen de pruebas con NUnit.

3.7 Conclusiones del capítulo

A partir de los resultados obtenidos, se puede afirmar que la solución propuesta cumple con los requisitos planteados al inicio de la investigación. El módulo desarrollado cumple con los requisitos de diseño e implementación como se pudo apreciar.

Se llevó a cabo la medición de algunos aspectos del *software* tales como el tamaño de clase y la profundidad de la herencia. Los valores numéricos obtenidos mediante la aplicación de estas métricas, proveen un valor cuantitativo que aseguran la calidad del diseño obtenido. Además, se mostró el resultado de las pruebas unitarias realizadas al módulo y después de varias iteraciones de pruebas, se puede concluir que se obtuvo un módulo de alta calidad.

El uso de la firma electrónica en la obtención de los objetos digitales importados en las pruebas de importación realizadas, en conjunto con la aplicación de los conceptos de integridad y confidencialidad de la información, demuestran que el módulo desarrollado satisface los requisitos de seguridad trazados en la concepción del sistema.

CONCLUSIONES

El presente trabajo de tesis cumplió satisfactoriamente sus objetivos. Se concluyó, de forma satisfactoria, el diseño e implementación del módulo de Importación de Metadatos del sistema DigiPyrus de los Registros y Notarías Públicas de la República Bolivariana de Venezuela.

El módulo desarrollado contribuye a la seguridad de los datos y permite importar los objetos digitales y metadatos sin pérdida y duplicidad de la información como quedó demostrado en las pruebas realizadas de acuerdo a la norma ISO- 17799.

Los algoritmos y clases implementadas en la importación de los metadatos y objetos digitales obtenidos en el sistema han dado resultados satisfactorios en los ensayos realizados, por lo que la solución propuesta contribuye a mitigar los problemas de gestión de la información en el SAREN. En la actualidad este módulo se encuentra en uso, y se han importado más de 20000 objetos digitales con resultados satisfactorios.

RECOMENDACIONES

Como recomendaciones se plantean:

- La implementación de un canal de comunicación segura entre las BD del Centro de Digitalización y el Centro de Datos del SAREN.
- Migrar la solución propuesta hacia una plataforma de software libre según lo expuesto en el Decreto-Ley 3390 del 23 de diciembre del 2004.

BIBLIOGRAFÍA

ABADI, M. y Cardelli, L. 1996. *A Theory of Objects.* s.l. : Springer-Verlag New York, Inc., 1996. ISBN 0387947752.

ADOLPH, S. P., BRAMBLE. 2001. *Patterns for Effective Use Cases.* s.l. : Prentice Hall , 2001. 85-325-399-8.

ALEXANDER, C., Ishikawa, Sara y Silve, Murray. 1977. A Pattern Language: Towns, Buildings, Construction. *Architect.* s.l. : Oxford University Press, 1977. 0195019199.

BECK, K. 1999. *Extreme Programming Explained: Embrace Change.* France : Prentice Hall, 1999.

BOOCH, G. 1996. *Análisis y diseño orientado a objetos 2.* España : Netalia S.L., 1996.

CÁNOS, J. H. y LETELIER, P. 2002. *Metodologías Ágiles en el Desarrollo de Software.* Caracas : Netalia S.L., 2002.

CAPLAN, P. 1995. You call it corn, we call it syntax-independent metadata for document-like objects. Belgium : The Public Access Computer Systems Review, 1995. Vol. 4, 6.

CHILVERS, A. y FEATHER, J. 1998. The management of digital data: a metadata approach. Belgium : Electronic Library, 1998. Vol. 16, 5.

CODD, E.F. 2000. A Relational Model of Data for Large Shared Data Banks. s.l. : Communications of the ACM, 2000. Vol. 13, 6. 10: 0321122267.

CrystalReports. 2010. MSDN. Microsoft. [En línea] February de 2010. http://msdn.microsoft.com/es-es/library/aa287920(v=VS.71).aspx.

D'ANGELO, F., GARCIA, D. y HERRERA, C. 2005. INTERNACIONAL ORGANIZATION FOR STANDARDIZATION. *Norma ISO 9000-3.* [En línea] April de 2005. http://www.iso.org/iso/iso_catalogue.htm.

2009. Desarrollo de una Aplicación en tres Capas con VS .NET. *www.microsoft.com.* [En línea] February de 2009. http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art140.asp.

HAMILTON. 2004. *NUnit Pocket Reference.* s.l. : Shroff / O'Reilly , 2004. 9788173667848.

HEERY, R. 1997. Biblink: LB4034 D1.1 metadata formats. Belgium : Biblink, 1997. Vol. 13, 5.

HERNÁNDEZ R.S., C.F. COLLADO. 2003. *Metodología de la investigación.* Colombia : Mc Graw Hill, 2003.

HERNÁNDEZ, L. R. A. 2000. *El paradigma cuantitativo de la investigación científica.* Sancti Spíritus, Cuba : Sede Universitaria, UCLV, 2000.

HUDGINS, J., AGNEW, G y BROWN, E. 1999. *Library and Information Technology Association: getting mileage out of metadata applications for the library.* Chicago : American Library Association, 1999.

HUNT, A. y THOMAS, D. 2004. *Pragmatic Unit Testing in C# with NUnit.* s.l. : The Pragmatic Programmers; 1st edition, 2004. 978-0974514024.

Hunt, Andrew y Thomas, David. 2004. *Pragmatic Unit Testing in C# with NUnit The Pragmatic Bookshelf.* Raleigh : s.n., 2004. ISBN 0-9745140-2-0.

2005. INTERNACIONAL ORGANIZATION FOR STANDARDIZATION. *ISO 17799:2005 Information technology - Security techniques - Code of practice for information security management.* [En línea] April de 2005. http://www.iso.org/iso/iso_catalogue.htm.

JACOBSON, I.G. 2002. Introducción a los Sistemas y Herramientas CASE. [aut. libro] ZAVALA. *Build Quality Applications Faster, Better and Cheaper.* España : OReilly, 2002.

JACOBSON, I.G. y BOOCH. 2000. *El Proceso Unificado de Desarrollo de Software.* España : Prentice Hall , 2000. 84-7829-36-2.

KATRIB, M. y DEL VALLE, M. 2008. *et al. (). Visual Studio 2008 Desafía todos los retos.* Ciudad de la Habana : Capitán San Luis, 2008. 978-959-211-329-9.

KERHERVÉ, B. y GERBÉ, O. 1999. Models for metadata or metamodels for data. s.l. : IEEE METADATA CONFERENCE, 1999. Vol. 12, 2.

KICILLOF, N. 2007. *Programación Orientada a Aspectos (AOP).* España : OReilly, 2007. http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art152.asp.

KICZALES, G., y otros. 1997. Aspect-Oriented Programming. *Proceedings of the European Conference on Object-Oriented Programming ECOOP´97.* Belgium : Springer-Verlag LNCS 1241, 1997. Vol. 1241.

KNOX, D. 2004. *Effective Oracle Database 10g Security by Design.* Osborne Media : McGraw-Hill, 2004. 978-0072231304.

KUMAR, Arun y Kanagaraj, John. 2005. *Oracle Database 10g Insider Solutions.* 1 edition. s.l. : Sams, 2005. 978-0672327919.

LARMAN, C. 1999. *UML y Patrones. Introducción al análisis y diseño orientado a objeto.* Mexico : OReilly, 1999.

2009. LEAD Technologies Inc. Corporate Overview. [En línea] LEAD Technologies Inc., April de 2009. http://www.leadtools.com/Home2/press/corporateovr.htm.

MACLLWAINE, J. y M., COMMENT J. 2002. *Directrices para Proyectos de Digitalización de colecciones y fondos de dominio público, en particular para aquellos custodiados en bibliotecas y archivos.* Lima : s.n., 2002.

MARTÍNEZ, N. 2002. Firma electrónica, certificados y entidades de certificación, RCE. s.l. : Revista de la Contratación Electrónica, 2002. Vol. 27, 4, págs. 59-75.

Metadata. **HUSBY, O. 1997.** Elag'97 : Prentice Hall , 1997.

MEYER, B. 2009. Touch of Class: Learning to Program Wel with Object and Contracts. s.l. : Springer, 2009. Vol. LXIV, pág. 876. 978-3-540-92144-8.

2010. MICROSOFT CORPORATION. *The .NET Standard Query Operators.* [En línea] April de 2010. http://msdn2.microsoft.com/en-us/library/bb931739. aspx.

2010. MICROSOFT CORPORATION. *ListView Class.* [En línea] March de 2010. http://msdn2.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.aspx.

MILANÉS, J.M. 2007. *Midiendo la Complejidad del Software.* España : Netalia S.L., 2007. http://www.gruposeti.com/complejidad.htm.

2008. MSDN Microsoft. Integridad de datos. [En línea] February de 2008. http://msdn.microsoft.com/es-es/library/aa291812(v=vs.71).aspx.

2008. MSDN Microsoft. Integridad Referencial. [En línea] February de 2008. http://msdn.microsoft.com/es-es/library/aa292166(v=vs.71).aspx.

2011. Normalización de Datos. *Microsoft.* [En línea] February de 2011. http://msdn.microsoft.com/es-es/library/aa291817(v=vs.71).aspx.

NUÑO, I. C. y G., EGIDO J. 2002. *Metodologías Orientadas a Objeto.* España : s.n., 2002.

OLAF, O., Krogdah, S. y Lyche, T. 2004. From Object-Orientation to Formal Methods: Essays in Memory of Ole-Johan Dahl. *Lecture Notes in Computer Science.* Belgium : Springer, 2004. Vol. 2635, 55. ISBN 3-540-21366-X.

POLLACK, Mark; EVANS, Rick; SEOVIC, Aleksandar, The Spring Java Team. 2010. The Spring.NET Framework Reference Documentation. [En línea] 1.3.1, April de 2010. http://www.springframework.net/doc-latest/reference/html/index.html.

PRADA, V. 2000. El documento informático y la seguridad jurídica. 1 [ed.] Ponencia española en el XX Congreso Internacional del Notariado Latino. [Ponencia española en el XX Congreso Internacional del Notariado Latino]. Sevilla, España : Revista jurídica del Notariado, 2000. pág. 113.

PRESSMAN, Roger S. 2005. *La Ingeniría del Software, un enfoque práctico.* Caracas : McGraw-Hill, 2005.

QUINTANA, A. 2006. *Metodología de Investigación Cualitativa en la Salud: Tópicos de Actualidad.* Lima : Grupo de Biomédica, Facultad de Ingeniería., 2006.

QUINTERO, A.M.R. 2000. *Visión General de la Programación Orientada a Aspectos.* Sevilla : Netalia S.L., 2000.

RAMIREZ, Elmasri y SHAMKANT, B.N. 2006. *Fundamentals of DATABASE SYSTEMS.* s.l. : Wesley-Addison, 2006. 978:0321369574.

RODRIGUEZ, A. 2004. *Firma electrónica y documento electrónico.* Sevilla : Ensayos de actualidad de escritura pública, 2004. 84-95176-42-4.

SANCHEZ, M. A. 2004. *Metodologías de Desarrollo de Software.* España : s.n., 2004. 45:233-289.

SCHACH, S. 2006. Object-Oriented and Classical Software Engineering. Seventh Edition s.l. : McGraw-Hill, 2006. ISBN 0-073-19126-4.

SCHWABER, K. y Beedle, M. 2001. *Agile Software Development with Scrum.* 2001.

SENSO, J.A. 2003. El concepto de metadato. Algo más que descripción de recursos electrónicos. Brasília : Ci. Inf., 2003. Vol. 32, 2.

SPARKS, G. 2005. *Una Introducción al UML. El Modelado de Proceso de Negocio.* Caracas : Wesley-Addison, 2005.

TENA, R. y DE LA NUEZ, E. 2001. La firma electrónica, ¿un poder al portador? Barcelona : Colegios Notariales de España, 2001. Vol. 10, 5. 155:1695-1700.

2003. THE SEARCH ENGINE REPORT. [En línea] The new meta tag are coming – or are they?, February de 2003. http://searchenginewatch.internet.com/sereport/97/12-metatags.html.

TORRES, J. L,. 2005. *Especificación de Requisitos en Ingeniería de Software.* 2da. 2005. págs. 345-398.

URDANETA, J.V. 2010. *Los mensajes de datos y la firma electrónica.* Caracas : Academia de Ciencias Sociales y Políticas, 2010. 978-980-6396-72-2.

2011. VISUAL STUDIO TOOLS. *MSDN Online Documentation.* [En línea] February de 2011. http://msdn2.microsoft.com/en-us/library/d2tx7z6d. aspx.

ZAVALA, J. 2002. *Ingeniería de Software.* Sevilla : s.n., 2002. 52:4426-078-3.