UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 15

Título: Diseño e implementación del proceso de inscripción de documentos que se realizan en las Notarías Públicas de la República Bolivariana de Venezuela.

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

Autor (es): Ivelisse Mercedes Díaz Rodríguez.

Nallelys Rodríguez Santos.

Tutor(es): Ing. Armando Esteban Pacheco Iglesias.

Ing. Rodolfo Pérez Ávila.

Ciudad de La Habana

Junio del 2010

DECLARACIÓN DE AUTORÍA.

Declaramos ser autores de la presente tesis y recomendamos 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 _____.

Ivelisse Mercedes Díaz Rodríguez

_____

Firma del Autor.

Ing. Armando Esteban Pacheco Iglesias

_____

Firma del Tutor.

Nayelis Rodríguez Santos

_____

Firma del Autor.

Ing. Rodolfo Pérez Ávila.

_____

Firma del Tutor.

AGRADECIMIENTOS

A nuestra Universidad de las Ciencias Informáticas por permitirnos crecer profesionalmente, a nuestro Comandante en Jefe por sus brillantes ideas que permitieron su creación, y por darnos la oportunidad de cursar nuestros estudios superiores en la Universidad del Futuro.

A Rodolfo y Armando, nuestros tutores, por su apoyo y guía en este trabajo.

A nuestros compañeros de trabajo del proyecto Registros y Notarías y a los de estudios a través de estos 5 largos años, por soportarnos, ayudarnos y estar ahí en todo momento.

A todos aquellos que de una forma u otra han marcado la diferencia, contribuyendo a nuestro desarrollo como profesionales y sobre todo como seres humanos en el transcurso de estos 5 años.

Agradecemos inmensamente a todas las personas, conocidos, amigos y profesores, que de alguna manera se preocuparon y aportaron su granito de arena en la confección de este trabajo.

A nuestra familia por su apoyo incondicional durante la carrera y la confección del mismo.

DEDICATORIA

Nallelys

Agradecerle principalmente a mi papá y a mi mamá por darme todo lo que necesité para crecer, por apoyarme en los momentos dificiles y quererme así como lo hacen. Todo eso hizo que este momento se hiciera realidad. Yo los amo... A mi hermana por ayudarme en todo lo que necesité aunque estuvieramos lejos.. Mi gordi yo te adoro...A mis dos Abuelas, a mis tíos, en especial mi Tío el niño por ayudarme tanto en estos 5 años, a mis tías y a mis primos, a ustedes gracias por todo...A mis super amigos...

Marlen(por su organización en el apto), Yuliet(por todas sus locuras), Flak(por sus consejos), Yaily(por su despistes), Odelkis(por ser como una hermana aquí), Chino(por ser tan mono), Misael(por pinareño), Danay(por sus inventos),Lisbey(por sus meriendas),a Yadira(por ayudarme a ser mejor en los temas esos de peinarse, jejeje) al Santi que aunque no esta aquí siempre lo recuerdo, Edel, Anel, Aliuska, Yoiner, Barli, Denia(La Negri) y a mi compañera de tesis por su ayuda en la realización de la tesis...Sin ustedes la vida aquí en la UCI no hubiese sido la misma... A mis tutores y al tribunal por toda su ayuda y consejos...

Ivelisse

A mi familia, mis padres, mi abuelita y mi hermano, por su amor y comprensión. Se que hoy están orgullosos de mi, de haber obtenido otra ingeniera en la familia y que no los defraude. A ellos por ser mi ejemplo y guías durante toda la carrera. A ellos, por su ternura y apoyo.

A mis amigos del alma, Denia, Dollys, Dania, Jonni. Se que sin su apoyo y ayuda no hubiera salido exitosa durante todos estos 5 años. Por estar ahí cuando siempre los necesité, en los buenos y en los malos momentos.

A Miguel, gracias por confiar en mí. Gracias por su apoyo diario y constante.

A todos los que de una forma u otra me apoyaron y me guiaron a lo largo de estos meses, a mis tutores, y a los compañeros del proyecto, sin ellos no hubiera sido posible la realización de este trabajo.

RESUMEN.

Las Notarías Públicas en Venezuela hoy no cuentan con la automatización de un sistema centralizado que permita determinar de manera rápida y oportuna la información de cada una de las actividades que se realizan en el proceso de Inscripción de Documentos que en ellas se realiza. El presente trabajo plantea una estrategia de cómo eliminar los problemas existentes en dicho proceso, a partir del diseño e implementación de una solución de software que satisfaga las necesidades existentes. Se presenta una detallada descripción de las principales tendencias y tecnologías usadas actualmente para el desarrollo de software, incluyendo las escogidas para el sistema que se propone como solución. Se exponen además los artefactos generados durante el diseño y la implementación del sistema sobre la base de la arquitectura planteada para la ejecución del mismo. Cada uno de estos artefactos se evalúan por medio de distintas métricas mundialmente usadas mostrándose los resultados obtenidos. Con esta solución la Dirección de Registros y Notarías contará con una herramienta que permita obtener la información de las actividades de las Notarías Públicas de forma oportuna y eficiente a lo largo del territorio.

ÍNDICE

Introducción

Debido al gran avance de la tecnología y los grandes cambios surgidos en el mundo, los países están llevando a cabo la informatización de cada una de las grandes empresas e instituciones presentes en los mismos. La República Bolivariana de Venezuela no ha estado exenta a este proceso. Uno de los sectores beneficiados, es el jurídico, donde desde hace algún tiempo se han venido estudiando diversas variantes en cuanto a la modernización de su Sistema Registral y Notarial.

El organismo rector en cuestión de todos los procesos jurídicos es el Servicio Autónomo de Registros y Notarías de la República Bolivariana de Venezuela (SAREN). Este es el órgano encargado de ordenar, supervisar y controlar todos los procesos que se llevan a cabo tanto en los registros, Mercantiles, Públicos, Principales, y las Notarías Públicas.

SAREN tiene como misión garantizar la seguridad jurídica de las actuaciones de los usuarios mediante la publicidad registral y fe pública, en el marco de la legalidad, de procesos expeditos y oportunos; ejerciendo el control de las operaciones a nivel nacional.

Todo comienza con la promulgación de la Ley de Registro Público en 1993 cuando el gobierno autoriza el inicio de un proceso de digitalización de documentos en todos los Registros y Notarías del país, con el objetivo fundamental de preservar el patrimonio registral de cada oficina. (1)

A partir de este momento, en cada una de las oficinas de registro, comenzaron a ejecutarse planes de modernización de sistemas a través de contratos con otras empresas de dentro o fuera del país. Estos fructificaron en ocasiones con soluciones informáticas muy buenas y completas, mientras que otras, con soluciones puntuales para algún caso determinado, sin embargo, siempre se adaptaron a las necesidades particulares de cada oficina y no una de manera uniforme para todas.

El último de estos convenios lo constituye el proyecto Registros y Notarías, a modo de colaboración entre Cuba y Venezuela, con el objetivo de controlar y estandarizar los procesos adjuntos que se llevan a cabo en los registros antes mencionados.

Se ha trabajado en los Registros Mercantiles y Públicos pero aún queda por automatizar los procesos de los Registros Principales y las Notarías Públicas de Venezuela. Lograr un correcto funcionamiento en las Notarías Públicas de Venezuela es uno de los objetivos fundamentales que tiene hoy el proyecto de Registros y Notarías.

Uno de los procesos a automatizar por dicho proyecto, es la inscripción de documentos en las Notarías Públicas. Para administrar dicho proceso se debe tener en cuenta diferentes fases o subprocesos que se deben tratar simultáneamente debido a la fuerte integración existente entre ellos. Estos se listan a continuación:

- Revisión: se revisa la correctitud de los documentos, además de registrar el acto al cual pertenecen y los timbres fiscales correspondientes, también se registran el presentante del documento, las prohibiciones y exenciones correspondientes al trámite.
- Cálculo: se identifican los aranceles correspondientes al trámite y se realizan los cálculos correspondientes para llevar a cabo el mismo.
- Presentación: se encarga de presentar el trámite a autenticar, para así darle entrada en la oficina de registro.
- Procesamiento: se encarga de procesar los documentos correspondientes a los trámites que se realizan, se registran los datos del documento además de generar las notas correspondientes a cada uno de los trámites.
- Otorgamiento: se realiza el otorgamiento del documento y se registra la fecha de realización del mismo.
- Digitalización: se encarga de la digitalización de cada trámite.
- Archivo: se archivan copias de cada uno de los documentos de los trámites que se llevan a cabo, además se generan las notas de apertura y cierre de los tomos del archivo.
- Firma Digital: se lleva a cabo la firma digital de los documentos garantizando su integridad.

Las Notarías Públicas hoy no cuentan con la automatización de un sistema centralizado que permita determinar oportuna y rápidamente la información de cada uno de estos importantes procesos. Actualmente la mayor parte del trabajo se realiza manualmente, y lo que se ha automatizado hasta hoy no es uniforme para todas las entidades. Todavía una parte de la recogida de datos es hecha en papel, aún cuando los datos pueden existir ya en el archivo; lo que trae como consecuencia: duplicado de información, un mayor volumen de papeles en el archivo innecesariamente y demora en la tramitación; por tanto la búsqueda y recuperación de la información se hace más lenta.

Uno de los problemas fundamentales hoy en día es la publicidad registral. Al no tener un sistema que organice toda la información generada por los procesos registrales y contar con grandes archivos

físicos de documentos, provoca que el acceso a los registros sea algo complicado y que sólo se pueda consultar en el lugar que se originó. Todo esto conlleva a la pérdida de la confiabilidad del ciudadano venezolano, incluyendo que la seguridad jurídica de estos documentos se encuentra severamente afectada ya que pueden inclusive hasta falsificarse, perderse o destruirse.

Según la problemática existente en las Notarías Públicas se llega al siguiente **problema científico**, ¿cómo lograr una gestión eficaz en los procesos de inscripción de documentos que se realizan en las Notarías Públicas de Venezuela a partir de los requerimientos identificados?

Por lo que el **objeto de estudio** de la investigación sería el Proceso de Desarrollo de Software.

Dado el problema planteado anteriormente se define como **objetivo general** de la investigación: desarrollar el diseño y la implementación como fases dentro de la solución del software en el proyecto Registros y Notarías, para los procesos de inscripción de documentos en las Notarías Públicas de la República Bolivariana de Venezuela.

Para dar cumplimiento al objetivo propuesto se han derivado un conjunto de **objetivos específicos** orientados fundamentalmente a proveer los elementos necesarios para el diseño e implementación de la solución, los cuales se listan a continuación:

- Realizar el marco teórico de la investigación.
- Analizar la arquitectura definida para la implementación de la solución informática.
- Definir una solución de diseño para los procesos de inscripción de documentos en las Notarías Públicas.
- Diseñar la solución, diagramas de clases y los diagramas de interacción, para dicha solución informática.
- Implementar la solución diseñada.
- Analizar y evaluar los resultados obtenidos, luego de aplicar diferentes métricas para el control y aseguramiento de la calidad en diseños de sistemas de software.

Visto el problema, el objeto de estudio y los objetivos específicos planteados se determina como **campo de acción** de esta investigación el diseño e implementación como fases específicas dentro del proceso de desarrollo de software posibilitando la realización de un sistema de inscripción de documentos en las Notarías Públicas de la República Bolivariana de Venezuela.

Como **hipótesis** de la investigación se tendría: si se desarrolla un sistema informático que gestione las actividades de inscripción de documentos, entonces se garantizaría una mejora en la gestión y

control de las actividades de inscripción de documentos en cada una de las Notarías Públicas de la República Bolivariana de Venezuela, donde también se garantizaría la seguridad jurídica del ciudadano venezolano.

Con este producto quedarán automatizadas las funcionalidades principales de las Notarías Públicas de la República Bolivariana de Venezuela. Entre los principales **resultados esperados** de la investigación se encuentran:

- Solución integral para la configuración y gestión de la inscripción de los documentos de las Notarías Públicas.
- Incremento sustancial de la seguridad jurídica, lo que conlleva al incremento de la confianza y seguridad del ciudadano venezolano a la hora de realizar cualquier tipo de trámite de inscripción de documentos en las Notarías Públicas de la República Bolivariana de Venezuela.
- Completamiento de la plataforma que interrelaciona todas las oficinas públicas que se subordinan al Servicio Autónomo de Registros y Notarías.

La presente investigación se encuentra estructurada por 3 capítulos los cuales se describen a continuación:

Capítulo 1. Fundamentación Teórica

En él se expone un estudio de la Ley de Registros Públicos y del Notariado promulgada en diciembre del 2006. También se explican aspectos sobre las Notarías Públicas en Venezuela, su funcionamiento, objetivos y servicios que brindan, haciendo énfasis en el proceso referente a la inscripción de documentos. Se aborda el estado actual de este tipo de sistemas en la República Bolivariana de Venezuela. Finalmente, se hace un estudio de las tendencias, metodologías y herramientas de desarrollo de software, concluyendo en las utilizadas en la solución expuesta.

Capítulo 2. Arquitectura, Diseño e Implementación

Se analiza la arquitectura del sistema a desarrollar, las diferentes capas que lo componen y sus componentes. Se exponen los patrones de diseño utilizados en la solución propuesta. Presenta la solución técnica aplicada al proceso de inscripción de documentos en las Notarías Públicas fundamentada con el modelo de diseño conformado por los diagramas de clases y los diagramas de secuencia y además el modelo de implementación con los diagramas de componentes y el diagrama de despliegue previamente diseñados.

Capítulo 3. Análisis de Resultados

Expone toda la validación hecha a la solución obtenida. Se presentan los resultados obtenidos de la aplicación de métricas al diseño y la implementación del sistema; y se analizan y evalúan dichos resultados.

Capítulo 1. Fundamentación Teórica.

Este capítulo enmarca la fundamentación teórica de la investigación. Se abordan aspectos sobre las Notarías Públicas en Venezuela, las leyes que las rigen, su funcionamiento, objetivos y servicios que brindan, haciendo énfasis en el proceso de inscripción de documentos. Posteriormente se hace un estudio del estado actual de este tipo de sistema en Venezuela. Se hace un recorrido por las diferentes tendencias, metodologías y herramientas de desarrollo de software, su clasificación y características.

1.1 Ley del Registro Público y del Notariado.

En la *Exposición de motivos del decreto con fuerza de Ley de Registro y del Notariado*, el presidente Hugo Rafael Chávez Frías dijo: "… la Constitución Bolivariana abre un nuevo camino hacia la modernización de las instituciones del sector público y esa apertura nos ofrece todas las posibilidades de adaptación del ordenamiento jurídico a los notables cambios que hoy tienen lugar en ese país, siendo más específico el vertiginoso avance de las tecnologías para la automatización de procesos. Esto significa darle prioridad a la seguridad jurídica en aquellos espacios institucionales que requieren con urgencia cambios profundos en el orden estructural, político, económico y social. Uno de esos ámbitos institucionales es el actual sistema registral y notarial venezolano, signado por la idea y la práctica tradicional de coleccionar manualmente en libros o protocolos los documentos que sirven para constituir, modificar o extinguir los derechos inscribibles de los ciudadanos." (2)

La función notarial en Venezuela había quedado marcadamente rezagada desde el punto de vista jurídico conceptual, limitando la actividad de los notarios a la autenticación de firmas en documentos privados. Es por ello que se propone la modernización de los servicios registrales y notariales, la cual implicaría alcanzar los siguientes objetivos, previstos en la normativa[1] propuesta:

- Automatizar los procedimientos y Sistemas Registrales y Notariales. Se ha previsto como medida prioritaria la implantación de un sistema automatizado, tanto para la gestión jurídica registral como para la gestión contable y administrativa, que requieren los procesos institucionales.

[1] Decreto con Fuerza de Ley de Registro Público y del Notariado, 2001.

- Ampliación del Sistema Notarial. El notariado es una función pública que el Estado puede delegar en los abogados que cumplen los requisitos establecidos en la Ley. Los notarios están autorizados para otorgar autenticidad a los hechos o actos jurídicos ocurridos en su presencia física o a través de medios electrónicos, indicando en este último caso los instrumentos mediante los cuales le otorga presunción de certeza al acto.

Todo el proceso de reforma notarial y registral se completa con la promulgación de la Ley del Registro Público y del Notariado del 2006. Esta ley que rige hoy los procesos jurídicos en cada una de las notarías públicas, constituye un cuerpo normativo que incorpora al ordenamiento jurídico venezolano nuevas leyes, cuyo cumplimiento son de vital importancia para instaurar la seguridad jurídica en estas entidades. Algunas se analizan a continuación.

Artículo 8. Sólo se inscribirán en el Registro los títulos que reúnan los requisitos de fondo y forma establecidos por la ley.

En cada una de las notarías públicas se inscribe cualquier tipo de documento siempre y cuando este tenga los requerimientos necesarios para llevar a cabo todo este proceso.

Artículo 23. Todos los soportes físicos del sistema registral y notarial actual se digitalizarán y se transferirán a las bases de datos correspondientes.

Todo lo que se genere de cada una de las inscripciones hechas por el personal que trabaja en las Notarías Públicas así como la documentación ya existente en las mismas, debe quedar registrado en una base datos, con una salva ante la ocurrencia de algún problema de consistencia de datos.

Artículo 24. La firma electrónica de los… notarios o notarias tendrá la misma validez y eficacia probatoria que la ley otorga.

Cada firma que otorgue el notario(a) es legítima para los trámites que se realizan en las notarías.

Artículo 78. La publicidad notarial reside en la base de datos del sistema automatizado de las notarías, en la documentación archivada que de ellas emanen y en las certificaciones que se expidan.

La existencia de un sistema centralizado que organice toda la información, proveerá con un acertado acceso y manejo de la información dentro de los registros.

Todos estos artículos en mayor o menor medida estarán reflejados en los procesos u operaciones implementados para la solución propuesta en vías de lograr una mejora en la inscripción de documentos. A continuación se procede a explicar el funcionamiento de las Notarías Públicas.

1.2 Notarías Públicas. Tareas Fundamentales.

Según la Real Academia de la Lengua Española el notario es el funcionario autorizado para dar fe pública de los contratos, testamentos y otros actos extrajudiciales, conforme a las leyes. Entiéndase por fe pública, la autoridad legítima atribuida a los notarios, para que los documentos que autorizan en debida forma sean considerados como auténticos y lo contenido en ellos sea tenido por verdadero mientras no se haga prueba de lo contrario.

El propósito fundamental de los Registros y Notarías en la República Bolivariana de Venezuela es garantizar, mediante la publicidad registral, la certeza y la seguridad jurídica de los bienes o derechos inscritos, otorgándoles la presunción de verdad legal, oponible a terceros. Los asientos registrales están bajo la salvaguarda de los Tribunales y producen todos sus efectos mientras no se declare su inexactitud. (2)

En el caso de las Notarías Públicas, el funcionario rector de todos los procesos que en ellas ocurren es el notario perteneciente al Servicio Autónomo de Registros y Notarías que tienen la potestad de dar fe pública de los hechos o actos jurídicos ocurridos en su presencia física o a través de medios electrónicos, indicando en este último caso los instrumentos mediante los cuales le otorga presunción de certeza al acto.

Según el artículo 75 de la Ley de Registro Público y del Notariado promulgada el 22 de Diciembre de 2006 los notarios o notarias son competentes, en el ámbito de su jurisdicción, para dar fe pública de todos los actos, hechos y declaraciones que autoricen con tal carácter, particularmente de los siguientes:

1. Documentos, contratos y demás negocios jurídicos, unilaterales, bilaterales y plurilaterales.
2. Poderes, sustituciones, renuncias y revocatorias, con excepción de las sustituciones, renuncias y revocatorias que se efectúen en los expedientes judiciales.
3. Los contratos de opción para adquirir derechos sobre bienes inmuebles.
4. Justificaciones para perpetua memoria, con excepción de lo señalado en el artículo 937 del Código de Procedimiento Civil.
5. Protestos de los títulos de crédito, de conformidad con lo previsto en el Código de Comercio.
6. Otorgamiento de testamentos abiertos, de conformidad con los artículos 852 al 856 del Código Civil.

7. Presentación y entrega de testamentos cerrados, con expresión de las formalidades requeridas en los numerales 1, 2 y 3 del artículo 857 del Código Civil.
8. Apertura de testamentos cerrados, de conformidad con lo previsto en los artículos 986 al 989 del Código Civil y 913 al 920 del Código de Procedimiento Civil. El Notario o Notaria tendrá potestades para realizar los actos que se atribuyen al Registrador o Registradora Subalterno en el Código Civil.
9. Autorizaciones de administración separada de comunidad conyugal.
10. Autorizaciones de administración de bienes de niños, niñas o adolescentes e incapacitados.
11. Otorgamiento de cualquier caución o garantía civil o mercantil.
12. Constancias de cualquier hecho o acto a través de inspección extrajudicial.
13. Transcripciones en acta o por cualquier medio de reproducción o de grabación del contenido de archivos públicos o de documentos privados, siempre y cuando no esté expresamente prohibido en el primer caso, o lo autorice el dueño o depositario del documento, en el segundo caso.
14. Celebración de asambleas, reuniones o manifestaciones, dejando las constancias personales, gráficas y sonoras del caso.
15. Transacciones que ocurran en medios electrónicos.
16. Apertura de libros de asambleas de propietarios, actas de juntas de condominios, sociedades y juntas directivas.
17. Autenticar firmas autógrafas, electrónicas y huellas digitales.
18. Las demás que le atribuyan las leyes.

Al notario también le competen las siguientes atribuciones:

- Archivar, en los casos en que fuere procedente, los instrumentos privados a que se contrae el artículo 1369 del Código Civil; archivar los documentos relativos a los contratos de venta con reserva de dominio, a los efectos de la fecha cierta de los mismos; extender y autorizar actas notariales, a instancia de parte, que constituyan, modifiquen o extingan un acto o negocio jurídico. Estas actas deben incorporarse cronológicamente en el archivo físico o electrónico notarial. (Artículo 75)
- Copias. Los Notarios expedirán copias certificadas o simples de los documentos y demás asientos que reposen en su oficina, siempre que las copias se soliciten con indicación de la clase de actos o de sus otorgantes, circunstancias éstas que se harán

constar en la correspondiente nota de certificación. También podrán expedir copias de documentos originales por procedimientos electrónicos, fotostáticos u otros semejantes de reproducción. (Artículo 76)

1.2.1 Proceso de Inscripción de documentos.

En las Notarías Públicas se pueden encontrar dos procesos fundamentales:

- Inscripción de Documentos: es el proceso encargado de inscribir los documentos presentados en la Notaría para su protocolización y archivado.
- Proceso de Solicitudes: estas pueden ser trámites de sellado de libros, expedición de actas notariales, justificativos o copias de documentos archivados en el Registro.

Para el desarrollo de este trabajo de investigación y el cumplimiento de los objetivos propuestos se hace necesario abordar el Proceso Inscripción de Documentos.

El proceso de inscripción es una formalidad contenida en el Código de Comercio[2] cuya principal finalidad es la oponibilidad a terceros del contenido de los actos y contratos inscritos. Este efecto sólo se obtiene con la publicidad legal, resultado de este proceso. La omisión de la inscripción trae como consecuencia la inoponibilidad de estos actos y contratos, por lo tanto, las personas jurídicas se sienten en el interés de inscribir sus actos para hacerlos constar ante terceros en caso de negocios por interés de ambas partes. (3)

La persona jurídica según la Real academia de la Lengua Española es la organización de personas o de personas y de bienes a la que el derecho reconoce capacidad unitaria para ser sujeto de derechos y obligaciones, como las corporaciones, asociaciones, sociedades y fundaciones.

El proceso de inscripción de documentos es el proceso más importante que se lleva a cabo dentro de un Notaría Pública, debido a que por medio de este se le otorga la fe pública y validez legal a un documento. Este proceso se encuentra organizado en una secuencia de pasos o fases por las cuales debe transitar el documento obligatoriamente, hasta convertirse en la constancia legal que permanece almacenada y registrada, formando parte del archivo notarial. A continuación se describen las fases inmersas en el proceso.

[2] Congreso de la República Bolivariana de Venezuela, 1955.

- Revisar documento: en este proceso se realiza la revisión legal del documento, verificando fondo, forma, prohibiciones del trámite y correspondencia de los datos del documento con los recaudos presentados.
- Realizar cálculo: en este proceso se calculan los aranceles que debe pagar el usuario para llevar a cabo el trámite, se verifica si el mismo está exento de pago o no, si el trámite no está exento de pago se genera la planilla única bancaria (PUB) con los cálculos de los derechos arancelarios a cancelar por la realización del trámite, en caso de que esté exento de pago se genera la PUB con monto cero.
- Presentar documento: en este proceso es donde se le da entrada al documento en la notaría. El usuario se presenta con la PUB pagada y sus respectivas copias, además de los timbres fiscales si han sido requeridos anteriormente. En el se genera la planilla de entrada del documento quedando el documento asentado en el Libro de Control de Entrada de Documentos de la notaría.
- Procesar notas: en este proceso se elabora la nota de autenticación del documento para autenticar el acto. Se obtienen las copias de los documentos presentados por el usuario y de la nota de autenticación elaborada y se engrapan junto a la PUB. Después de ser revisados por el jefe revisor son archivados temporalmente en espera de la fecha del otorgamiento.
- Otorgar documento: en este proceso se realiza el otorgamiento del documento presentado. En él, los otorgantes plasman su firma y sus huellas dactilares en los documentos que han sido autenticados, estos son revisados y firmados por el notario y posteriormente se entrega al usuario el documento original y las copias son enviadas al archivo.
- Archivar documento: en este proceso se deja una copia en el archivo de la notaría de cada uno de los actos realizados en la misma. Si el trámite es susceptible a autenticación, se archiva una copia de los documentos autenticados en los tomos Principal y Duplicado.

1.3 Soluciones informáticas existentes.

Actualmente en Venezuela existe el Sistema Automatizado para Oficinas de Registro y Notarías Públicas (SARENOT). Este es el resultado de un arduo trabajo de investigación que le ha permitido a sus autores conocer ampliamente todas las características de los procesos requeridos para el cumplimiento de la actividad de cada uno de los funcionarios que laboran en dichas oficinas y diseñar

las aplicaciones informáticas necesarias para esos fines, utilizando modernas herramientas que garantizan el manejo integro, consistente y seguro de la información.

Según la página de los autores es un sistema integral e interactivo, esto es, cada uno de los módulos que componen su estructura forma parte de un todo integrado, perfectamente relacionado con el resto de los módulos, lo cual permite que los procesos sean complementarios. Vincula de manera inseparable el proceso de facturación con el de procesamiento de la información relacionada con los documentos otorgados u otras actuaciones realizadas, lo cual garantiza la integridad de la información contenida en la base de datos. El sistema contiene herramientas de control que permiten garantizar la integridad de los datos introducida en el mismo, por lo cual es recomendable que todas las funciones del personal sean procesadas de forma automatizada. El sistema contiene además las herramientas necesarias para la digitalización de los documentos, la cual contendrá las vinculaciones y relaciones requeridas para que sea objeto de un trato coordinado y organizado.

Esta aplicación es un sistema integrado con respecto a la interrelación de los módulos de procesos subyacentes a la inscripción de documentos, no de integración a nivel nacional. Así como tampoco la digitalización de documentos llega a este nivel. SARENOT es un software propietario y no todas las notarías cuentan con el dinero para adquirirlo con todas sus funcionalidades, trayendo como consecuencia que en algunas sólo se encuentra habilitado el módulo de revisión y en otras sólo la parte de cálculo, demostrando que aún cuando las oficinas lo compren, no está garantizado el alto grado de automatización requerido. Se le añade además que es un sistema distribuido y se dificulta la adquisición de los datos por parte del SAREN.

1.4 Desarrollo de Software. Tendencias y tecnologías.

1.4.1 Metodologías de Desarrollo de Software.

Durante el ciclo de vida del software se deben completar una serie de tareas para obtener un producto. A menudo, se dice que los distintos componentes de software deben pasar por distintas fases o etapas durante el ciclo de vida.

Pues bien, cada una de esas tareas pueden ser abordadas y resueltas de múltiples maneras, con distintas herramientas y utilizando distintas técnicas. Es necesario saber cuándo se puede dar por

concluida una tarea, quién debe realizarla, qué tareas preceden o anteceden a una dada, y qué documentación se utiliza para llevarla a cabo.

Se habla de detalles organizativos, de un "estilo" de hacer las cosas. Pero yendo un poco más allá de un simple estilo, formalizando ese "estilo" añadiendo algo de rigurosidad y normas se obtiene una metodología.

Existen dos grupos en los cuales se dividen estas metodologías: los métodos tradicionales y los procesos ágiles, con marcadas diferencias.

A continuación se listan algunas de las más usadas en la actualidad:

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 del convencional. En palabras de Beck: "XP es una metodología ligera, eficiente, con bajo riesgo, flexible, predecible y divertida para desarrollar software"[3].

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 del ciclo de la programación. 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 costes. XP propone valores que deben poseer los miembros del equipo de desarrollo para desarrollar el trabajo y conseguir los planteamientos iniciales. Estos cuatro valores son: comunicación, sencillez, retroalimentación y valentía. Además, define cuatro variables para proyectos de software: coste, tiempo, calidad y ámbito; y un conjunto de prácticas básicas tales como: 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.

Scrum: tiene su primera aparición física en 1995, pero no es hasta el 2001 que sus creadores describen completamente la metodología[4]. Scrum es un proceso en el que se aplican de manera regular un conjunto de mejores prácticas para trabajar en equipo y obtener el mejor resultado posible de un proyecto. Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita

[3] Extreme Programming Explained: Embrace Change. Kent Beck, (1999).

[4] Agile Software Development with Scrum. Ken Schwaber y Mike Beedle. (2001)

obtener resultados pronto, los requisitos son cambiantes o poco definidos y la innovación, la competitividad y la productividad son fundamentales.

Scrum también se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, específicamente cuando: (5)

- Las entregas se alargan demasiado, los costes se disparan o la calidad no es aceptable.
- Se necesita capacidad de reacción ante la competencia.
- La moral de los equipos es baja y la rotación alta.
- Es necesario identificar y solucionar ineficiencias sistemáticamente.
- Se quiere trabajar utilizando un proceso especializado en el desarrollo de producto.

En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos (iteraciones de un mes natural y hasta de dos semanas, si así se necesita). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite. Las actividades fundamentales del proceso son la planificación y la revisión. Para ello Scrum plantea la ejecución de una serie de reuniones, estas son:

- Scrum Diario (Daily Scrum)
- Scrum de Scrum
- Reunión de Planificación del Sprint (Sprint Planning Meeting)
- Reunión de Revisión del Sprint (Sprint Review Meeting)
- Retrospectiva del Sprint (Sprint Retrospective)

El Proceso Unificado Racional (Rational Unified Process - RUP) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML[5], constituye una metodología robusta, la más estándar y utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. 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.

[5] Unified Modelling Language por sus siglas en inglés. Versión original en 1997. Se ha convertido en el estándar para notaciones de modelado por idea de tres expertos en modelado: Booch, Rumbaugh y Jacobson.

Como RUP es un proceso, en su modelación define como sus principales elementos:

- Trabajadores (“quién”): define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.
- Actividades (“cómo”): es una tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos.
- Artefactos (”qué”): productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.
- Flujo de actividades (“Cuándo”): secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.

En RUP se han agrupado las actividades en grupos lógicos definiéndose 9 flujos de trabajo principales. Los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como de apoyo. (*Ver figura 1*)

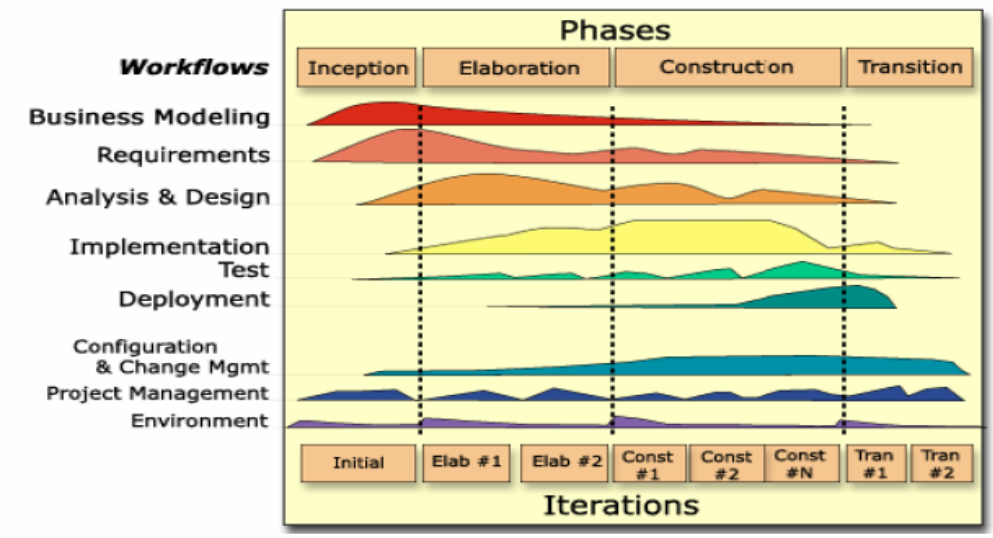

Fig. 1 RUP en dos dimensiones

Características principales de RUP:

- 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 (que se acercan al producto terminado) del producto en desarrollo.

Registros y Notarías se caracteriza por ser un proyecto grande en lo que se refiere a recursos humanos y a tiempo de desarrollo, con gran número de iteraciones, y por consiguiente de artefactos generados en cada una de sus fases. Es por ello, que se decide como metodología a utilizar para el desarrollo, RUP, teniendo en cuenta las dimensiones que posee el proyecto, y a causa de su característica de ser a distancia, añadiendo a esto el gran volumen de documentación que se precisa. Se considera además, que resulta más adaptable para los proyectos a largo plazo que necesiten un desarrollo iterativo capaz de mantener un buen control sobre los cambios y que facilite sobre todo el trabajo a distancia con los clientes, aplicando los criterios de flexibilidad que ofrece, robustez en la gestión del proyecto y los cambios, la cantidad de artefactos que se generan en forma de entregables al cliente y la adaptabilidad ante un sistema de gran tamaño. En base a esta decisión, se define como lenguaje de modelado UML.

1.4.2 Herramientas CASE

De acuerdo con Kendall y Kendall la ingeniería de sistemas asistida por ordenador es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo. Su objetivo es acelerar el proceso para el que han sido diseñadas, en el caso de CASE para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas.

Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas ayudan 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 costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Se clasifican en disímiles tipos, según las funciones específicas que brindan, fundamentalmente las referidas a la fase que dan apoyo dentro del desarrollo de software, ya sea la captura de requisitos y el análisis y diseño o todo el ciclo de vida hasta la implementación y pruebas. A continuación se muestran algunas de ellas:

Rational Rose Enterprise: brinda soporte para la generación de código a partir de modelos en Ada, ANSI C++, C++, CORBA, Java™/J2EE™, Visual C++® y Visual Basic®. Como todos los demás productos Rational Rose, proporciona un lenguaje común de modelado para el equipo que facilita la creación de software de calidad rápidamente. (6)

Características generales:

- Crea modelos independientemente del lenguaje de arquitectura de software, necesidades empresariales, activos reutilizables y comunicación a nivel de gestión.
- Posee un único entorno de diseño.
- Presenta un desarrollo basado en modelos con soporte para UML.
- Brinda soporte para múltiples modelos en la Arquitectura basada en Modelos (MDA).
- Se ejecuta de forma independiente o integrada con Microsoft Visual Studio .NET.
- Crea arquitecturas independientes de la plataforma que se pueden implementar en plataformas Java y .NET.
- Uso de patrones definibles por el usuario para crear, personalizar y aprovechar patrones de diseño arquitectónico.
- Usa referencias cruzadas entre modelos y el control de versiones a nivel de clase y diagrama para la estructuración ajustable a cualquier proyecto.
- Conserva la capacidad de rastreo entre los modelos de análisis, diseño e implementación.
- Modela de formas libres.
- Soporta publicación web y generación de informes.

Visual Paradigm: es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software, análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar

código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. A continuación se listan sus características:

- Soporte de UML versión 2.1.
- Diagramas de Procesos de Negocio.
- Modelado colaborativo con CVS y Subversion (nueva característica).
- Interoperabilidad con modelos UML2 (metamodelos UML 2.x para plataforma Eclipse) a través de XMI (nueva característica).
- Ingeniería de ida y vuelta. Ingeniería inversa - Código a modelo, código a diagrama. Ingeniería inversa Java, C++, Esquemas XML, XML, .NET exe/dll, CORBA IDL.
- Generación de código - Modelo a código, diagrama a código.
- Editor de Detalles de Casos de Uso - Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Diagramas de flujo de datos.
- Soporte ORM - Generación de objetos Java desde la base de datos.
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación.
- Generador de informes para generación de documentación.
- Distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML.

Enterprise Architect: es una herramienta de modelamiento basada en UML 2.1 flexible, completa y poderosa para plataformas Windows. Es orientada a objetos y provee el limite competitivo para el desarrollo de sistemas, administración de proyectos y análisis de negocios. Ayuda al desarrollador de software a trazar especificaciones de alto nivel a modelos de análisis, diseño, implementación, pruebas y mantenimiento, usando UML, SysML, BPMN y otros estándares abiertos para modelado.
Entre sus principales características se pueden destacar: (7)

- Con un repositorio de alto desempeño, facilita que los grandes equipos compartan la misma visión del desarrollo del producto.
- Provee trazabilidad completa desde los modelos de requisitos, análisis y diseño, hasta la implementación y despliegue.
- Generación potente de documentación.
- Generación e ingeniería inversa de código fuente. Soporta la generación e ingeniería inversa de código fuente para muchos lenguajes populares. Las integraciones para Eclipse y Visual Studio .NET, les proveen a los desarrolladores un acceso directo a los diseños y capacidades de modelado justo dentro del IDE. Las plantillas de generación de código le permiten personalizar el código fuente generado de acuerdo con las especificaciones de su compañía.
- Permite que se construya, pruebe, depure, active y ejecuten scripts de despliegue, todo dentro del entorno de desarrollo de Enterprise Architect. Con la capacidad de generar clases de prueba NUnit y JUnit desde clases origen usando transformaciones MDA y la integración del proceso de pruebas directamente en el IDE de Enterprise Architect, ahora puede integrar el modelado con UML y el proceso de compilación/prueba/ejecución/despliegue.

Enterprise Architect es utilizado por varios tipos de desarrolladores de programas de sistemas para una amplia gama de industrias. Enterprise Architect tiene una fuerte base de usuarios en más de 60 países del mundo. 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.

ER/Studio Enterprise: cuando se hace la planificación de la base de datos, la primera etapa del ciclo de vida de las aplicaciones de bases de datos, también se puede escoger una herramienta CASE. Su uso puede mejorar considerablemente la productividad en el desarrollo de una aplicación de bases de datos. ER / Studio Enterprise dentro de la industria de software es una herramienta de modelado para el análisis, visualización y comunicación de base de datos, aplicaciones de diseños de datos y arquitectura de la información. La combinación de procesos, datos, modelado UML, y presentación de reportes en un potente entorno de diseño en multi-niveles son algunas de sus principales características. (8)

El uso del ER / Studio Enterprise trae como beneficios:

• Promoción de la reutilización de datos y la colaboración en tiempo real a través de una gestión eficaz del modelo de la empresa y la capacidad de publicación de metadatos.
• Mejora de la visibilidad y la calidad de información con la ingeniería inversa y el soporte al ciclo de vida completo de base de datos.
• Comunicación efectiva de los modelos de toda la empresa con informes de metadatos en tiempo real y herramientas de publicación.

En el estudio de herramientas CASE debe resaltarse **Enterprise Arquitect** como una herramienta de análisis evaluada muy potente y orientada a proyectos grandes donde el trabajo en equipo es esencial. Tal es el caso del proyecto Registros y Notarías cuya dirección ha seleccionado esta herramienta por sus amplias potencialidades ya señaladas con anterioridad. Esta herramienta CASE describe la metodología seleccionada, RUP en este caso, en muy buena medida. Enterprise Arquitect se integra bien con UML en sus últimas versiones y proporciona generación de código C# entre otros lenguajes. Además ya ha sido utilizada por los roles del proyecto implicados y resulta fácil de usar, rápida de entender y muy manejable. Además, se selecciona para el modelado de los datos el Embarcadero ERStudio, una herramienta fácil de usar y multinivel, para el diseño y construcción de bases de datos tanto a nivel físico como lógico, direccionando las necesidades diarias de los desarrolladores que construyen y mantienen aplicaciones de bases de datos grandes y de gran complejidad.

1.4.3 Paradigmas de Programación.

La historia del desarrollo de los lenguajes de programación muestra una creciente evolución en la que se van incorporando elementos que permiten ir creando programas cada vez más sólidos y eficientes a la vez que facilitan la tarea del programador para su desarrollo, mantenimiento y adaptación.
Un paradigma de programación es un modelo básico de diseño e implementación de programas, un modelo que permite desarrollar programas conforme a ciertos principios o fundamentos específicos que se aceptan como válidos. (9)
Los principales paradigmas se clasifican en dos grupos: procedimentales y declarativos. Esta clasificación está dada partiendo de los principios fundamentales de cada paradigma en cuanto a las orientaciones sobre la forma para construir las soluciones. A continuación se abordan aspectos relacionados con los paradigmas que serán útiles en la construcción de la solución propuesta.

Dentro de los paradigmas procedimentales se encuentra el imperativo y el de objetos, este último surgido a partir del imperativo pero renueva sus principios en un modelo más amplio. El paradigma de objetos, o como se lo conoce generalmente, la Programación Orientada a Objetos, se fundamenta en concebir a un sistema como un conjunto de entidades que representan al mundo real, los "objetos", que tienen distribuida la funcionalidad e información necesaria y que cooperan entre sí para el logro de un objetivo común. Introduce nuevos conceptos de programación provocando un giro importante en sus principios y consideraciones básicas, en el que más que dejarlos de lado, los reorganiza y capitaliza dentro de un modelo más amplio, con otros conceptos característicos. Ejemplos de ellos son el Encapsulamiento, Polimorfismo y Herencia.

De forma particular vale mencionar la Programación Orientada a Aspectos (POA), la cual es una nueva metodología de programación que aspira a soportar la separación de competencias para aspectos tales como la sincronización, la distribución, el manejo de errores, la optimización de la memoria, la gestión de seguridad, etc. Es decir, que intenta separar los componentes y los aspectos unos de otros, proporcionando mecanismos que hagan posible abstraerlos y componerlos para formar todo el sistema. En definitiva, lo que se persigue es implementar una aplicación de forma eficiente y fácil de entender. POA es un desarrollo que sigue al paradigma de la orientación a objetos, y como tal, soporta la descomposición orientada a objetos, además de la procedimental y la descomposición funcional.

1.4.4 Patrones de Diseño de Software.

Los patrones son una disciplina de resolución de problemas reciente en la ingeniería del software que ha emergido en mayor medida de la comunidad de orientación a objetos, aunque pueden ser aplicados en cualquier ámbito de la informática y las ciencias en general.

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 diseño de programas orientados a objetos, haciendo más fácil reutilizar con éxito los diseños y arquitecturas, estos 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.

Los patrones de diseño se dividen en tres grupos principales:

- **Patrones de creación:** muestran la guía de cómo crear objetos cuando sus creaciones requieren tomar decisiones. Estas decisiones normalmente serán resueltas dinámicamente decidiendo que clases instanciar o sobre que objetos un objeto delegará responsabilidades. Dentro de ellos clasifican Patrón de Fábrica Abstracta, Patrón Constructor, Patrón del Método de Fabricación, Patrón Prototipo, y Patrón de Instancia Única (Singleton).
- **Patrones estructurales:** describen la forma en que diferentes tipos de objetos pueden ser organizados para trabajar unos con otros. Dentro de ellos clasifican Patrón Adaptador, Patrón Puente, Patrón Compuesto, Patrón Decorador, Patrón de Fachada, Patrón de Peso Mosca y Patrón Apoderado.
- **Patrones funcionales o de comportamiento:** se utilizan para organizar, manejar y combinar comportamientos. Dentro de ellos clasifican Patrón de Cadena de Responsabilidad, Patrón de Comando, Patrón Intérprete, Patrón Iterador, Patrón Mediador, Patrón Memento, Patrón Observador, Patrón de Estado, Patrón de Estrategia, Patrón del Método Plantilla y Patrón Visitante.

Paralelamente asignar correctamente las responsabilidades es muy importante en el diseño orientado a objetos. Los **patrones GRASP**[6] 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.

1.4.5 Plataformas de Desarrollo.

Para lograr el buen funcionamiento de un sistema que se desee implementar, es necesario tener un conocimiento previo de cada una de las plataformas existentes, para así escoger la que reúna las características acordes al sistema y las personas e instituciones que se beneficiarán del mismo.
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

[6] GRASP es un acrónimo que significa General Responsibility Asignment Software Patterns (patrones generales de software para asignar responsabilidades. El nombre se eligió para indicar la importancia de captar (grasping) estos principios, si se quiere diseñar eficazmente el software orientado a objetos.

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 por sus siglas en inglés).

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 los son, la Web y las aplicaciones de escritorio. A continuación se exponen algunas de las diferentes plataformas existentes en el mundo para el desarrollo de aplicaciones de escritorio.

Plataforma Java: la plataforma Java es el nombre de un entorno o plataforma de computación originaria de Sun Microsystems, capaz de ejecutar aplicaciones desarrolladas usando el Lenguaje de programación Java u otros lenguajes que compilen a bytecode y un conjunto de herramientas de desarrollo. En este caso, la plataforma no es un hardware específico o un sistema operativo, sino más bien una máquina virtual encargada de la ejecución de aplicaciones, y un conjunto de librerías estándar que ofrecen funcionalidad común. Desde 2006, la versión actual de la Plataforma Java Standard Edition se le conoce como Java SE 6 como versión externa, y 1.6 como versión interna.

Java no fue la primera plataforma basada en el concepto de una máquina virtual, aunque es la que de más amplia difusión ha gozado. El empleo de máquinas virtuales se había centrado principalmente en el uso de emuladores para ayudar al desarrollo de hardware en construcción o sistemas operativos, pero la JVM (Java Virtual Machine) fue diseñada para ser implementada completamente en software, y al mismo tiempo hacer que fuera portable a todo tipo de hardware. Esta plataforma se compone de un amplio abanico de tecnologías, cada una de las cuales ofrece una parte del complejo de desarrollo o del entorno de ejecución en tiempo real. Por ejemplo, los usuarios finales suelen interactuar con la maquina virtual de Java y el conjunto estándar de bibliotecas. Además, las aplicaciones Java pueden usarse de forma variada, como por ejemplo ser incrustadas en una página Web. Para el desarrollo de aplicaciones, se utiliza un conjunto de herramientas conocidas como JDK (Java Development Kit, o herramientas de desarrollo para Java).

Esta plataforma está compuesta por bibliotecas que tienen como objetivo ofrecer al programador un conjunto bien definido de funciones para realizar tareas comunes, como manejar listas de elementos u operar de forma sofisticada sobre cadenas de caracteres. Además, estas proporcionan una interfaz abstracta para tareas que son altamente dependientes del hardware de la plataforma destino y de su

sistema operativo. Tareas tales como manejo de las funciones de red o acceso a ficheros, suelen depender fuertemente de la funcionalidad nativa de la plataforma destino.

Plataforma Mono: su nombre proviene de un proyecto de código abierto iniciado por Ximian (empresa proveedora de software libre para Linux y Unix) y actualmente impulsado por Novell (tras la adquisición de Ximian) para crear un grupo de herramientas libres, basadas en GNU/Linux y compatibles con .Net según lo especificado por el ECMA (European Computer Manufacturers Association).

Mono posee importantes componentes útiles para desarrollar software:

- Una máquina virtual de infraestructura de lenguaje común (CLI) que contiene un cargador de clases, un compilador en tiempo de ejecución (JIT), y unas rutinas de recolección de memoria.
- Una biblioteca de clases que puede funcionar en cualquier lenguaje que funcione en el CLR (Common Language Runtime).
- Un compilador para el lenguaje C#, MonoBasic (la versión para mono de Visual Basic), Java y Python.
- El CLR (Common Language Runtime) o Lenguaje Común de Infraestructura y el Sistema de tipos común (CTS) permite que la aplicación y las bibliotecas sean escritas en una amplia variedad de lenguajes diferentes que compilen para byte code.
- Esto significa que si, por ejemplo, se define una clase que realice una manipulación algebraica en C#, ésta pueda ser reutilizada en cualquier lenguaje compatible con CLI. Puede crear una clase en C#, una subclase en C++ e instanciar esa clase en un programa en Eiffel (Lenguaje de programación orientado a objeto).
- Un sistema de objetos único: sistema de hilos, bibliotecas de clases y sistema recolector de memoria pueden ser compartidos por todos estos lenguajes.
- Es un proyecto independiente de la plataforma. Actualmente Mono funciona en GNU/Linux, FreeBSD, UNIX, Mac OS X, Solaris y plataformas Windows.

Esta plataforma presenta algunas ventajas: es ideal para desarrollo de plataformas cruzadas y se ha posicionado como un entorno que permite ejecutar en Linux aplicaciones diseñadas para Microsoft .Net en entorno Windows, facilitando la migración de aplicaciones a Linux y aumentando su base de desarrolladores y usuarios. A diferencia de los programas tradicionales que se ejecutan sobre el sistema

directamente, los programas en la plataforma Mono se ejecutan sobre un entorno controlado de ejecución conocido como la máquina virtual. Este entorno proporciona numerosas ventajas sobre la ejecución tradicional directa: gestión de memoria automática (el sistema se encarga de recuperar automáticamente la memoria no usada por las aplicaciones simplificando la gestión a las aplicaciones), un entorno seguro de ejecución (donde se puede definir los recursos físicos y lógicos a los que la aplicación tiene acceso) y un sistema de control de errores y ejecución que permite una gestión de errores avanzada.

Plataforma .Net: .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. 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.

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.

Es un componente de software que puede ser añadido al sistema operativo Windows. Provee un extenso conjunto de soluciones predefinidas para necesidades generales de la programación de aplicaciones, y administra la ejecución de los programas escritos específicamente con la plataforma. Esta solución es el producto principal en la oferta de Microsoft, y pretende ser utilizada por la mayoría de las aplicaciones creadas para la plataforma Windows.

Basado en ella, la empresa intenta desarrollar una estrategia horizontal que integre todos sus productos, desde el sistema operativo hasta las herramientas de mercado. Su propuesta es ofrecer una manera rápida y económica, a la vez que segura y robusta, de desarrollar aplicaciones –o como la misma plataforma las denomina, soluciones– permitiendo una integración más rápida y ágil entre empresas y un acceso más simple y universal a todo tipo de información desde cualquier tipo de dispositivo.

Dentro de las desventajas que presenta esta plataforma se puede mencionar 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

sólo está disponible para la familia de Windows. El tema de las licencias es también otro inconveniente, es una herramienta de código cerrado y no hay licencias libres.

Se escoge la plataforma .Net para el desarrollo del software primeramente por ser la conveniada con los clientes por la dirección de Registros y Notarías. Se incluye además la valoración de las enormes ventajas que aporta su uso dentro del desarrollo de software expuestas anteriormente. Por otra parte, el número de personas que tienen dominio sobre la plataforma .NET en el proyecto es significativamente superior a los de las demás plataformas de desarrollo existentes. Además, el software que se va a implementar debe integrarse con los anteriores módulos de la aplicación hechos en Fase I, por lo que deben ser implementados en la misma plataforma.

Se utilizará como entorno de desarrollo integrado el Visual Studio .NET 2008, el cual provee al desarrollador con mejoras de desempeño, escalabilidad y seguridad con respecto a la versión anterior; y como lenguaje de programación se selecciona el C#, un lenguaje orientado a objetos y que además puede ser usado como lenguaje base para la programación orientada a aspectos.

1.4.6 Frameworks de Desarrollo.

En general, un framework es una estructura real o conceptual que pretende servir como soporte o guía para la construcción de algo que expande su estructura en algo usable. En sistemas informáticos, un framework es con frecuencia una estructura que indica cuales tipos de programas pueden o deben ser construidos y cómo estos se interrelacionarán.

Algunos pueden además incluir programas existentes, interfaces de programación específicas u ofrecer herramientas de programación para usar el framework. Un framework puede ser para un conjunto de funciones dentro de un sistema y como ellos se interrelacionan, las capas de un sistema operativo, de un subsistema de aplicación, cómo la comunicación debe ser estandarizada a un cierto nivel de la red de trabajo, y así sucesivamente. Es generalmente más comprensivo que un protocolo y más prescriptivo que una estructura. (10)

Básicamente un framework evita construir una aplicación desde cero, los frameworks orientados a objetos están estructurados en librerías de clases y ahorran de cierta forma el trabajo largo y a veces tedioso de la programación, se debe tener en cuenta que para comenzar a desarrollar en un framework se deben conocer sus especificaciones. A continuación se exponen algunos que serán de gran utilidad a la hora de implementar la solución que se propone en la investigación.

.NET Framework: es la infraestructura básica subyacente de .NET. De esta manera, 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.

El .NET Framework está formado por:

- 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.

Algunas de las ventajas de utilizar el .NET Framework para desarrollar aplicaciones incluyen:

- 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 lenguaje 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. Se puede acceder y extender las clases .NET (a menos que estén protegidas) mediante la herencia. También es posible 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 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.

Framework Común: inicialmente fue el desarrollado por el Proyecto Identidad de la Universidad. Contiene elementos del patrón Modelo-Vista-Controlador. Estas ideas se encapsulan en una capa de presentación que queda compuesta por los elementos del patrón. Su uso no cambia el estilo de arquitectura pues la arquitectura es basada en capas habiéndose empleado o no este framework. Es utilizado la en estandarización y gestión de todas las interfaces del sistema.

La intención de este patrón es separar estos tres componentes de modo tal que puedan ser variados y desarrollados en paralelo, en el sistema la vista sería el conjunto de formularios de Windows la cual muestra la información necesaria por el usuario, el controlador es el encargado de responder a todos los eventos de usuario, las entradas del teclado y del ratón e informa al modelo de los cambios en la vista y viceversa. El modelo es el conjunto de estructuras, clases e interacciones entre estos que son modelados sobre la arquitectura y soportan las reglas del negocio.

Inicialmente este framework no respondía a todas las características que se buscaban en los objetivos del Proyecto de Registros y Notarías, por lo que el equipo de desarrollo se vio forzado a incluir o mejorar funcionalidades que cubrieran sus expectativas:

- Se mejoró el chequeo y tratamiento de excepciones, con lo que se organizó de una forma más eficiente el tratamiento de estas.
- Se crearon opciones para la gestión de la sesión vigente en el sistema y la configuración local de las oficinas.
- Se mejoró la seguridad para una mayor protección de los datos creando canales más seguros y confiables.
- Se creó una funcionalidad para el chequeo de la versión del framework, el cual ha sido muy útil a la hora de desplegar nuevas versiones en los diferentes escenarios.

Ahora se analizarán las ventajas y las desventajas de este framework, algunas son propias del MVC.

Entre las ventajas del framework escogido están las siguientes:

- **Múltiples vistas usando el mismo modelo:** dado que la vista se encuentra separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente**.** En el framework el controlador y el modelo se pueden unir, pero si los desarrolladores tienen esto en cuenta, separando los dos componentes esta ventaja se puede aprovechar.
- **Fácil para soportar nuevos tipos de clientes:** para soportar nuevos tipos de clientes sólo se debe escribir la vista y el controlador para este y se reutiliza el modelo ya existente. Importante si alguna vez se decide migrar el sistema a un entorno de software libre.
- **Claridad de diseño:** mirando la lista de métodos públicos, debe ser fácil entender cómo controlar el funcionamiento del modelo. Esto depende además en lo complejo de la solución, a medida que el software crece es necesario ir desarrollando otros diagramas como apoyo.
- **Fácil de crecer:** los controladores y las vistas pueden crecer según crece el modelo, viejas versiones de las vistas y los controladores pueden usarse mientras la interfaz común es mantenida.
- **Distribuible:** con conjunto de proxies se puede distribuir una aplicación MVC, solamente alterando el método de inicio de la aplicación.
- **Seguridad a nivel de aplicación**: el framework sienta las bases para brindar seguridad a nivel de aplicación, mediante un fichero de configuración donde se definen roles y en base a estos se gestiona el acceso a las acciones (los controladores) y la visualización de las opciones de la interfaz que le pertenecen a cada rol.
- **Colores de Interfaz Configurable:** se puede cambiar el color de toda la aplicación con unos pocos cambios en la configuración.
- **Configuración del Menú de Opciones**: mediante un fichero de configuración se puede cambiar la estructura del menú de opciones, donde cada opción puede contener opciones y así sucesivamente por varios niveles.
- **Múltiples controladores usando la misma vista**: debido a que la vista está completamente separada del controlador y no lo conoce, sino que es al revés, la misma vista puede ser utilizada por distintos controladores, además permite que la vista sea modificada en su estructura y diseño sin afectar a los controladores.

Entre las desventajas del framework escogido están las siguientes:

- **Complejidad:** introduce nuevos niveles de indirección y por tanto aumenta la complejidad de la solución. Incrementa además la naturaleza basada a eventos del código de la interfaz de usuario, lo cual puede ser más difícil de depurar.
- **Acoplamiento de la Vista y el Controlador con el Modelo**: cambios en la interfaz del modelo requieren cambios paralelos en la vista y cambios adicionales en el controlador, algunos cambios del código pueden ser difíciles.
- **Interfaz:** el framework posee una interfaz al estilo Web donde en la parte derecha se muestra un menú con todas las opciones, si se desea cambiar esta apariencia puede conllevar un gran esfuerzo por el equipo de desarrollo, partiendo desde el punto que se cuente con el código del mismo, de lo contrario no es posible.

Spring. Net : 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 reales beneficios y se utiliza en miles de aplicaciones empresariales de todo el mundo. Spring.NET es una metaframework basado en los patrones de arquitectura y diseño que han sido probados y que no se vinculan a una plataforma determinada. Puede utilizar la funcionalidad de sus módulos de forma independiente.

Sus principales potencialidades tributan significativamente a la arquitectura, entre ellas se pueden destacar la Inyección de Dependencia, la Programación Orientada a Aspectos (POA por sus siglas en inglés), interoperabilidad e integración con otros frameworks de factible utilización, así como el hecho de ser un framework de software libre dando la posibilidad de tener acceso a su código fuente.

Al utilizar Spring.NET en aplicaciones se aseguran una serie de puntos importantes en toda la aplicación, puntos que comprende el framework y no requieren prácticamente ningún código por parte del equipo de desarrollo. A continuación se exponen las fundamentales ventajas:

- Implementación de Patrones de Diseño: la utilización de patrones en una arquitectura resulta una decisión muy acertada y eficiente puesto que constituyen buenas prácticas y soluciones a problemas muy comunes en el desarrollo de cualquier aplicación. Spring.NET implementa una

gran variedad de estos patrones de diseño de una forma bastante sencilla y práctica, por ejemplo: *Fábrica, Fábrica Abstracta, Constructor, Decorador, y Localizador de Servicio*, entre otros.

- Inversión del Control (IoC): delega la responsabilidad de instanciar los objetos en un archivo de configuración XML, esto resulta muy útil puesto que evita cualquier dependencia que pueda existir entre los objetos, más si son de capas o aplicaciones externas. El nombre viene precisamente de que los objetos generalmente son instanciados antes de ser utilizados en el momento en que se carga el XML.

NHibernate: framework de Object-Relational-Mapping (mapeo de objetos relacionales) de código abierto, que resuelve en forma automática la persistencia de objetos de dominio .NET. NHibernate está basado en el popular Framework Hibernate surgido en la comunidad Java en el año 2002. (11)
Maneja objetos persistentes simples de una base de datos relacional subyacente. Dada la descripción XML de las entidades y relaciones, NHibernate genera automáticamente el SQL para cargar y guardar los objetos. Opcionalmente, se puede describir el mapeo de metadatos con atributos en el código fuente. NHibernate soporta persistencia transparente, las clases de objetos no tienen que seguir un modelo de programación restrictivo. Las clases persistentes no necesitan implementar ninguna interfaz o heredar de una clase base especial. Esto posibilita diseñar la lógica del negocio usando objetos simples .NET (CLR) y el idioma de los objetos orientados.
Entre sus ventajas se destacan:

- Código abierto.
- Soporta DataBinding[7].
- Puede aceptar consultas SQL directas.
- Se puede integrar con frameworks de MVC.
- Soporta las relaciones entre objetos.
- Soporta agrupamiento (GROUP BY).
- Soporta agregación (COUNT, AVG, ETC.).

[7] También llamado enlace de datos, es el proceso que establece una conexión entre la la interfaz de usuario de una aplicación y la lógica de negocio.

- Soporta llaves primarias compuestas.
- Soporta asociaciones muchos a muchos y uno a muchos.
- Soporta persistencia de propiedades a través de los campos de propiedades.
- Soporta persistencia de propiedades a través de métodos get/set o propiedades.
- Soporta trabajo offline y luego aplicar los cambios a la base de datos.
- Soporta WebServices (la tecnología más moderna para la Integración de aplicaciones web, y el paradigma de programación moderno de programación orientada a servicios, publicación de servicios).
- Soporta tipos nulos.
- No se requiere generar código pre compilado.

Presenta también algunas desventajas como son:

- No soporta carga no transaccional lazy de relaciones.
- No soporta Aggregate Mappings – Single (un campo a muchos campos en la base de datos).
- No sorpota consultas transparentes a múltiples recursos de datos.

NUnit: es un framework de código abierto para pruebas unitarias de sistemas realizados con la plataforma Microsoft .NET. Se compone por todo un conjunto de meta-atributos y aserciones que permiten probar los métodos de una clase especificada. El NUnit se puede ejecutar desde la consola o a través de una interfaz gráfica y además se integra con el Visual Studio en cualquiera de sus versiones. Actualmente soporta los frameworks 1.1/2.0. También soporta la plataforma Mono.

Posee soporte de archivos de configuración y ejecución de múltiples ensamblados. Es extensible y autodetecta cambios de los ensamblados.

NUnit basa su funcionamiento en dos aspectos, el primero es la utilización de atributos personalizados. Estos atributos le indican al framework de NUnit que debe hacer con determinado método o clase, es decir, estos atributos le indican a NUnit como interpretar y ejecutar las pruebas implementadas en el método o clase. El segundo son las aserciones, que no son más que métodos del framework de NUnit utilizados para comprobar y comparar valores.

Entre sus ventajas se pueden mencionar:

- Pruebas automatizadas, por lo cual se hacen repetibles.

- Fomento del cambio: ya que se permite probar cambios en el código y asegurar que en éstos no se hayan introducido errores funcionales; habilitando la refactorización del código.
- Se simplifica la integración: se llega a la fase de integración con un grado alto de seguridad sobre el código.
- Documentación del código.
- Separación de la interfaz y la implementación.
- Los defectos están acotados y fáciles de localizar.
- Se le permite al desarrollador pensar como el consumidor del código y no como el productor.

Sin embargo presenta algunas desventajas o limitaciones tales como:

- No se llegan a descubrir todos los defectos del código.
- No se pueden determinar problemas de integración o desempeño.
- No es trivial anticipar todos los casos especiales de entradas.
- Las pruebas unitarias determinan la presencia de defectos, no la ausencia de éstos. Son efectivas al combinarse con otras actividades de pruebas.

1.4.7 Sistemas Gestores de Base de Datos

En este tipo de sistemas los datos se centralizan en una base de datos común a todas las aplicaciones. Para que los usuarios puedan procesar, describir, administrar y recuperar los datos almacenados en dicha base de datos se utiliza un sistema gestor de bases de datos o SGBD.

En estos sistemas se proporciona un conjunto coordinado de programas, procedimientos y lenguajes que permiten a los distintos usuarios realizar sus tareas habituales con los datos, garantizando además la seguridad de los mismos. El éxito del SGBD reside en mantener la seguridad e integridad de los datos. Lógicamente tiene que proporcionar herramientas a los distintos usuarios. Dentro de este concepto hoy en día son más populares debido a sus funcionalidades los Sistemas de Gestión de Bases de Datos Relacionales (SGBDR). Bajo este concepto se oculta toda la complejidad informática necesaria para gestionar:

- El acceso controlado de los procesos de los usuarios a los datos
- La gestión del almacenamiento de los datos
- La gestión de las comunicaciones entre procesos clientes y servidores

- Interconexión con distintos protocolos y sistemas operativos
- Acceso a los recursos conectados a la red

A continuación se mencionan dos de los principales candidatos para utilizar en la solución debido a sus características.

PostgreSQL: es un poderoso Sistema de Base de Datos Relacional de código abierto. Cuenta con más de 15 años de desarrollo activo y una arquitectura probada que se ha ganado una sólida reputación de confiabilidad, integridad de datos y corrección. Funciona en todos los principales sistemas operativos, incluyendo Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), y Windows. Es totalmente compatible con ACID, tiene soporte completo para claves foráneas, uniones, vistas, disparadores y procedimientos almacenados (en varios idiomas). También es compatible con el almacenamiento de objetos binarios, incluyendo imágenes, sonidos o vídeo. Tiene interfaces de programación nativa de C / C + +, Java, NET, Perl, Python, Ruby, Tcl, ODBC, entre otros, y la documentación en varios idiomas.

PostgreSQL cuenta con sofisticadas funciones como el Control de Concurrencia Multi-Versión (MVCC), punto en el tiempo de recuperación, tablespaces, replicación asincrónica, transacciones anidadas (puntos de retorno), backups en línea, un planificador / optimizador de consultas sofisticadas, y escribir por delante de registro para la tolerancia a fallos. Es compatible con conjuntos de caracteres internacionales, codificación de caracteres multibyte, Unicode, y es consciente de la configuración regional para la clasificación, caso-sensibilidad, y el formato. Es altamente escalable, tanto en la enorme cantidad de datos que puede manejar y en el número de usuarios concurrentes que puede acomodar. No hay sistemas activos de PostgreSQL en entornos de producción que manejan en exceso de 4 terabytes de datos.

Una falencia importante que gira entorno a PostgreSQL es la replicación[8]. Es un problema que algunas herramientas externas abordan sólo en parte, las formas más complicadas no están hechas todavía. Además, podría tornarse en desventaja el hecho de lo lento de importar datos desde archivos de texto o que hay un sobrecosto por tupla que en algunos casos puede ser muy grande. (12)

Oracle: es un Sistema de Gestión de Bases de Datos Relacionales (SGBDR), desarrollado por la empresa Corporación Oracle surgida a finales de la década de 1970. Permite el almacenamiento de

[8] Copiar datos de una base de datos a otra.

datos en tablas formadas por filas y columnas, y su posterior consulta y mantenimiento mediante un sencillo y potente lenguaje de consulta estructurado (SQL).

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 multitud de usuarios (entornos multi-usuarios) hacia los mismos datos.

Las principales funcionalidades aportadas por todo el SGBD Oracle son:

- Soporte y tratamiento de una gran cantidad de datos (Gbytes).
- 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.

El alto precio de las patentes y licencias del equipamiento que requiere Oracle, con respecto a otros sistemas gestores de bases de datos constituye uno de sus principales problemas. Sin embargo, estas licencias son más costosas al comienzo de la utilización del producto pues luego solamente cabe reactivarlas, proceso que es mucho más económico.

Oracle se encuentra disponible en una variedad de ediciones: Express Edition, Standard Edition One, Standard Edition, y Enterprise Edition. Todas las ediciones fueron creadas utilizando la misma base de código, lo cual significa que sus aplicaciones de base de datos pueden fácilmente escalar de servidores de procesadores sin cambiar ninguna línea de código. (12)

Las versiones de Oracle propuestas implementan un buen proceso de réplica, en este caso, los servidores de ubicados en los Registros replican diariamente la información almacenada hacia el

servidor central ubicado en SAREN, lo que permite la persistencia y seguridad de la información. Además, Oracle desde la concepción inicial de Registros y Notarías ha tenido la aceptación del personal de base de datos del proyecto por la experiencia en el trabajo con el gestor, por sus potencialidades y porque se cuenta con las licencias para su uso. Es así como es el seleccionado por la dirección del proyecto una vez más para su uso en el subsistema de Notarías Públicas.

Para ello se hará uso de **Oracle 10*g* Release 2**. Esta versión provee una técnica de comprensión única que resulta muy factible para los almacenes de datos grandes. Su reducción de espacio en disco puede ser significativamente mayor que los algoritmos de compresión estándar, ya que está optimizado para datos relacionales. Prácticamente no tiene impacto negativo en la realización de consultas en los datos comprimidos, de hecho, puede tener un impacto positivo y significativo en el acceso a las consultas de grandes cantidades de datos, así como sobre las operaciones de gestión de datos, como backup y recuperación. Además, asegura que datos comprimidos nunca son mayores que los datos sin comprimir.

En cada una de las oficinas se hará uso de **Oracle 10*g* R2 Standard Edition One** es una solución de middleware[9] probada, segura, completa, basada en estándares y accesible que permite compartir información en forma instantánea y segura con los clientes, empleados y colegas. Con la familia "SE One", Oracle proporciona una solución flexible, modular y de fácil implementación. Es muy fácil de instalar y configurar; y permite una gestión automatizada y automantenida. Además, soporta Oracle Application Express.

Está confeccionado para permitir un crecimiento rápido de los negocios para avanzar de manera rápida en las soluciones implementadas, a través de las funcionalidades preconstruidas, reduce el tiempo de desarrollo y simplifica el mantenimiento y las operaciones. Provee una plataforma amigable con todo lo necesario para implementar un portal para compartir información, un sitio web para promoción, o aplicaciones basadas en Web/Java, proporcionando funcionalidades comunes. Todo lo anterior está previamente pre-ensamblado y probado.

Para el servidor de datos central se utilizará **Oracle 10*g* R2 Enterprise Edition.** Esta edición provee una única de manejar el costo comprimiendo los datos almacenados en bases de datos relacionales, con

[9] Es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas.

muy pocos impactos negativos en el tiempo de las consultas a datos y de esta forma, permite sustanciales ahorros en el costo.

Es ideal para empresas que necesitan soporte para transacciones en línea de grandes volúmenes de datos, procesando y consultando aplicaciones de almacenamiento intensivo de datos. Provee escalabilidad probada en todas las configuraciones de hardware y puede ser usada para manejar grandes cantidades de información con los niveles de seguridad más altos dentro de la industria de software. Provee beneficios de disponibilidad únicos que resguardan los datos de errores humanos costosos. Además, reduce el periodo de inactividad asociado con el mantenimiento de la rutina, e incluye las capacidades de automanejo para ayudar a disminuir los costos operacionales.

Como la primera base datos diseñada para Grid Computing[10] se puede usar para reducir significativamente los costos de infraestructura, a través del uso eficiente de recursos comunes de bajo costo, componentes estandarizados de hardware. Con cada lanzamiento de producto, esta edición de Oracle, potencia su compromiso con Grid Computing, automatización de base de datos y capacidad de autogestión.

1.4.8 Arquitectura de Software.

La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de estos componentes según se les percibe del resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones. (13)

La definición oficial de Arquitectura del Software es la IEEE Std 1471-2000 que reza así: “La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución”.

Para lograr un fácil manejo durante el desarrollo, se hace uso de los estilos arquitectónicos. Un estilo afecta a toda la arquitectura de software y puede combinarse de acuerdo con la propuesta de solución.

[10] Sistema de computación distribuido que permite compartir recursos no centrados geográficamente para resolver problemas de gran escala.

La imposición de ciertos estilos arquitectónicos mejora o disminuye las posibilidades de satisfacción de ciertos atributos de calidad del sistema. Cada estilo propicia atributos de calidad, y la decisión de implementar alguno de los existentes depende de los requerimientos de calidad del sistema.

No es objetivo de este epígrafe hacer un análisis de los distintos estilos arquitectónicos en su totalidad, sino sólo aquellos que por su estructura y solución podrían considerarse para la construcción del sistema, analizando las ventajas y desventajas que estos poseen para determinar si debe ser o no aceptado para el desarrollo del software. Algunos de estilos son:

Estilos de Flujo de Datos

La familia de los estilos de Flujos de Datos, enfatiza la reutilización y la modificabilidad, es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos, ejemplos de estas se pueden encontrar las arquitecturas de proceso secuencial por lotes, red de flujos de datos, y tuberías y filtros.

De las arquitecturas que ofrece esta familia de estilos no se recomienda ninguna de ellas para el desarrollo de un sistema con las características que se necesita. A pesar de que es simple de entender, implementar, al igual que enfatiza la reutilización y la modificabilidad, en los procesos que complementan al proceso de inscripción el trámite pasa por determinados pasos, no se escoge pues no se hacen modificaciones sino que se le agregan metadatos a lo largo de todo el flujo y en otros casos lo único que se le hace es transmitir esos datos a diferentes departamentos pero sin modificaciones a este documento.

Estilos Centrados en Datos

Otro de los estilos arquitectónicos es el Centrado en Datos, esta familia de estilos enfatiza la integridad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Entre los subestilos característicos están los Repositorios, Base de Datos y Arquitectura de Pizarra.

No se considera dicha arquitectura apropiada para la realización del sistema, pues esta más bien enfatiza los datos y no el modelo, además no son objetivos de este tipo de estilo la reutilización y la flexibilidad del código.

Estilos Peer-to-Peer

Los estilos Peer-to-Peer o también llamado de componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo

general en procesos independientes o entidades que se comunican a través de mensajes. Algunos de los miembros de esta familia de estilos son las Arquitecturas Basadas en Eventos, Arquitectura Basadas en Servicios y la Arquitectura Basada en Recursos.

El estilo basado en eventos más bien se utiliza en ambientes de integración de herramientas, interfaces de usuario o cuando no se quiere que exista un acoplamiento entre el emisor y el receptor. La Arquitectura Basada en Servicios no se acopla al contexto de las oficinas de las Notarías Públicas, por ser un ente que tiene cierta forma independiente y que debe funcionar pese a problemas de conectividad u otros problemas que surjan.

No se selecciona los estilos Peer-to-Peer, porque el negocio de los procesos que se desean automatizar, tienen sus propias reglas, donde es muy importante el rendimiento del sistema, destacando la flexibilidad y la reusabilidad, además de que existen ciertas condiciones en las que el sistema no podría funcionar, rompiendo así con los esquemas propuestos por dichos estilos de arquitectura.

Estilos de Llamada y Retorno

Los estilos de Llamada y Retorno enfatizan la modificabilidad y la escalabilidad, son los estilos más utilizados o más generalizados por los sistemas a gran escala. Dentro de esta familia se encuentran el Modelo-Vista-Controlador (MVC), la Arquitectura Orientada a Objetos y la Arquitectura en Capas.

- **Modelo-Vista-Controlador (MVC):** en ocasiones se le define más bien como un patrón de diseño o como práctica recurrente, y en estos términos es referido en el marco de la estrategia arquitectónica de Microsoft. Lo que se persigue con este estilo de arquitectura es separar sus tres componentes, el modelo, la vista y el controlador de manera que puedan ser reutilizados y desarrollados por diferentes personas de forma paralela.

  Entre las ventajas del estilo señaladas en la documentación de Patterns & Practices de Microsoft vale mencionar el soporte de vistas múltiples y la adaptación al cambio. Sin embargo, presenta algunas desventajas a considerar tales como el nivel de complejidad y el costo de actualizaciones frecuentes.
- **Arquitecturas Orientadas a Objetos**: nombres alternativos para este estilo han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. Los objetos representan una clase de componentes que en algunas ocasiones son

denominados managers, debido a que son responsables de preservar la integridad de su propia representación. El principal problema de esta arquitectura es que es necesario tener un nivel de abstracción superior que agrupe estos objetos de modo tal que sean más fáciles de entender y desarrollar en paralelos por diferentes grupos de desarrolladores; en sistemas de gran tamaño, las interacciones entre los objetos y su volumen se pueden volver inmanejables.

- **La Arquitectura en Capas:** podría decirse que su finalidad es abstraer las funcionalidades de una capa, de manera tal que esta pueda ser totalmente reemplazada. La Arquitectura de Capas más común es la que está compuesta por tres: Presentación, Modelo o Reglas del Negocio de la Empresa y Acceso a Datos. De esta forma, se podrá cambiar cualquiera de estas sin afectar a las restantes. Aunque tres capas es lo más común, a medida que aumenta la complejidad de los sistemas, las capas crecen. A su vez cada capa puede estar compuesta por subcapas y una capa o subcapa puede estar compuesta por una o más clases del diseño.

Para la solución se elige por la dirección del Proyecto la Arquitectura en Capas. Primero, el estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. En segundo lugar, el estilo admite muy naturalmente optimizaciones y refinamientos. En tercer lugar, proporciona amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas. Para la aplicación se hará uso de una arquitectura en capas que proveerá alto nivel de modificabilidad y escalabilidad, permitiéndole al sistema cambios ya sea de requisitos funcionales, agregaciones o mejoras en las funcionalidades.

1.4.9 Métricas para la medición del diseño de software.

Cuando se construye un sistema de software es de gran importancia tener en cuenta todos los aspectos necesarios para obtener un producto de software con la mayor calidad posible, es por ello que para este tipo de empresas en la actualidad este es un objetivo primordial a alcanzar.

Según la terminología de la IEEE, la calidad de un sistema, componente o proceso de desarrollo de software, se obtiene en función del cumplimiento de los requerimientos iniciales especificados por el cliente o el usuario final. (14)

Es importante entonces destacar que una métrica se puede definir como la aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo del software y sus productos para suministrar información relevante a tiempo, así el administrador junto con el empleo de estas técnicas mejorará el proceso y sus productos. (15)

Métricas para Sistemas Orientados a Objetos

Los sistemas de software orientados a objetos, como el propuesto en este trabajo, poseen características especiales que lo diferencian de otros sistemas de software construidos utilizando métodos convencionales, es por ello que las métricas aplicadas a sistemas orientados a objetos deben ajustarse a estas características.

Métricas orientadas a clases.

Se incluyen dentro de las métricas de diseño arquitectónico. Teniendo en cuenta que la clase es la unidad fundamental de un sistema orientados a objetos, es que las métricas aplicadas a las clases, sus jerarquías y colaboraciones entre ellas son un recurso de suma importancia para la estimación de la calidad del sistema. Entre algunas de las métricas que se han definido a nivel de clases se pueden mencionar:

- **Métricas de Chidamber y Kemener (CK).**

Se sitúan entre las métricas más difundidas en cuanto a las basadas en clases para este tipo de sistemas, fueron propuestas por Shyam R. Chidamber y Chris F. Kemener, las mismas se dividen en: Métodos Ponderados por Clases (MPC), Árbol de Profundidad de Herencia (APH), Número de Descendientes (NDD), Acoplamiento entre Clases Objetos (ACO), Respuesta Para una Clase (RPC) y Carencia de Cohesión en los Métodos (CCM).

- **Métricas de Lorenz y Kidd**

Estas métricas se dividen en cuatro categorías: tamaño, herencia, valores internos, valores externos. Las métricas orientadas a tamaño se centran en el cálculo de atributos y operaciones de una clase de manera individual; las basadas en herencia se enfocan en la reutilización de las operaciones en una jerarquía de clases; las basadas en valores internos examinan la cohesión y otros aspectos relacionados

con el código; y las métricas orientadas a valores externos se centran en el acoplamiento y la reutilización. (16)

En este grupo de métricas se encuentran: Tamaño de Clase (TC), Número de Operaciones Redefinidas por una clase (NOR) y Número de Operaciones Añadidas por una clase (NOA).

Existen además otras familias de métricas que se pueden aplicar para determinar la calidad de un diseño en un sistema, entre estos grupos se encuentran: Métricas de Li y Henry que son una modificación de las métricas de CK más cuatro nuevas métricas; y las Métricas de Hendersson-Sellers que están relacionadas fundamentalmente con el acoplamiento y la cohesión del diseño. (16)

1.4.10 Métricas de Prueba

Las métricas de prueba que existen se concentran en el proceso de prueba y no en las características técnicas de la prueba. Los responsables de la prueba se guían por las métricas de análisis, diseño y código. Entre ellas vale destacar la métrica de Puntos de Función, Bang, Halstead y la Complejidad Ciclomática. (12)

Otras a destacar dentro de la mencionada clasificación son las **Métricas de Pruebas de Unidad**. Estas se incluyen dentro de las Métricas de Prueba. El término prueba de unidad se refiere a la prueba individual de unidades separadas de un sistema de software. En sistemas orientados a objetos, estas unidades son, típicamente, clases y métodos. La prueba es un proceso de ejecución de un programa con la intención de descubrir un error. Se dice que tiene éxito si descubre un error no detectado hasta entonces. Para llevarlas a cabo se utilizan los casos de prueba, los cuales se consideran más efectivos cuando tienen una alta probabilidad de mostrar un error no descubierto hasta entonces.

Las **pruebas de caja blanca** permiten examinar la estructura interna del programa realizando un seguimiento del código fuente según va ejecutando los casos de prueba, de manera que se determinan concretamente las instrucciones, bloques en los que existen errores. Para la realización de las pruebas de unidad se diseñan casos de prueba que se encargan de examinar la lógica del sistema. Los casos de prueba garantizan que se ejerciten todos los caminos independientes de cada módulo o clase y todas las decisiones lógicas así como que se ejecuten todos los bucles y las estructuras de datos internas.

Las **pruebas de caja negra** comprueban que cada función es operativa. En la prueba de caja negra, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta. Permiten detectar funcionamiento

incorrecto o incompleto, errores de interfaz, errores de acceso a estructuras de datos externas, problemas de rendimiento y/o errores de inicio y terminación.

1.5 Conclusiones Parciales.

En este capítulo se realizó un estudio de las leyes que rigen el funcionamiento de las Notarías Públicas. Se analizaron los procesos presentes en ellas, haciendo énfasis en la inscripción de documentos como objeto dentro de la investigación. Se ejemplificó la existencia de soluciones informáticas que automatizan algunos de estos procesos en Venezuela, culminado con el análisis de las tendencias, metodologías y las herramientas útiles para el desarrollo de software, viendo de ellas las ventajas y desventajas existentes, concluyendo con las seleccionadas para la solución propuesta.

Capítulo 2. Arquitectura, Diseño e Implementación.

Propone la solución técnica que se propone en la investigación. Se analiza la arquitectura del sistema a desarrollar, las diferentes capas que los componen y sus componentes. Se exponen los patrones de diseño utilizados en la solución propuesta. Presenta el modelo de diseño conformado por los diagramas de clases y los diagramas de secuencia (Diagramas de Interacción) y además el modelo de implementación con los diagramas de componentes y el diagrama de despliegue previamente diseñados.

2.1 Descripción de la arquitectura del sistema.

Teniendo en cuenta las características del software que se está desarrollando así como las cualidades o atributos de calidad que se desean alcanzar, la Arquitectura sustenta su base en los componentes, siendo estos bloques de construcción que conformarán las partes del mismo. La Arquitectura empleada en la solución informática del Proyecto Registros y Notarías de la República Bolivariana de Venezuela está organizada desde dos enfoques, uno horizontal y otro vertical los cuales se describen a continuación.

2.1.1 Enfoque Horizontal

El sistema está dividido en 2 subsistemas para los diferentes tipos de oficinas, cada uno de los cuales responde a un conjunto de funcionalidades y casos de uso específicos del cliente. Estos subsistemas interactúan y comparten datos de interés en dependencia de la funcionalidad de cada uno y siguiendo un estricto régimen de seguridad de la información. Los subsistemas son Notarías Públicas y Registros Principales(*ver figura 2*).

Estos subsistemas comparten un Framework Común que contiene las funcionalidades básicas comunes para todos los subsistemas como la Gestión de Sesión, Gestión de Roles y todo lo referente a la autenticación de los usuarios. Además, establece una guía para el pautado del diseño visual de las pantallas del sistema dándole uniformidad a los 2 subsistemas mencionados anteriormente.

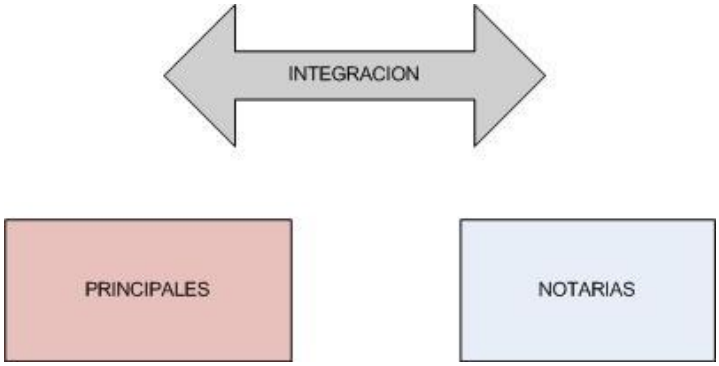

Fig. 2 Perspectiva Horizontal

2.1.2 Enfoque Vertical

El enfoque vertical abarca la distribución de los componentes que dan vida a la arquitectura para cada subsistema de manera individual. En este sentido, se presenta un modelo multicapa(*ver Figura 3*), donde las capas tienen un orden lógico de procedencia, es decir, las funcionalidades y recursos están encapsulados en cada nivel de acuerdo con la responsabilidad que se establece para cada uno de ellos.

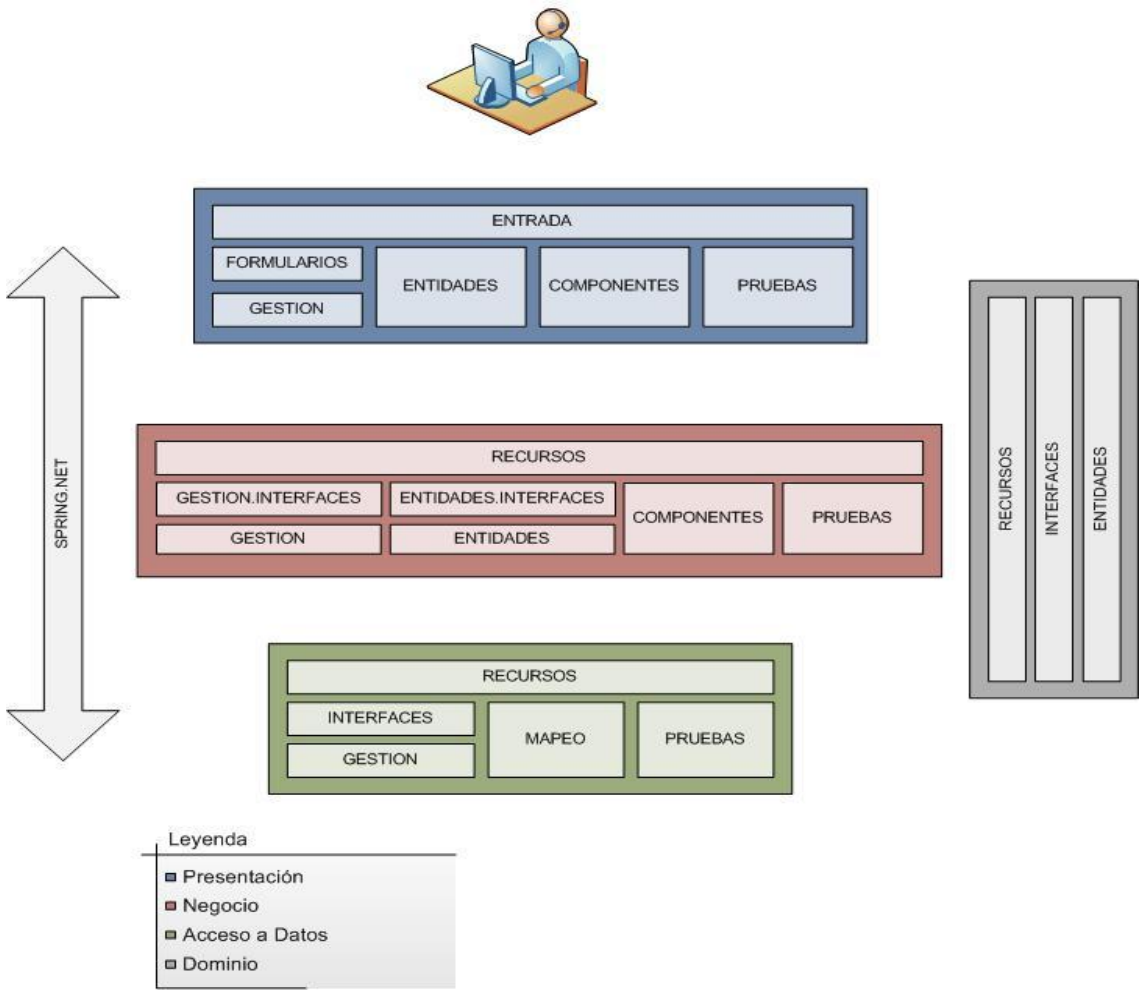

Fig.3 Perspectiva Vertical

Nivel Presentación

Es la capa superior, contiene los componentes con las que va a interactuar el usuario. Desde este nivel se pueden alcanzar las funcionalidades que brinda el negocio y mostrar o capturar la información a través de los diferentes elementos que comprende.

Inicio: constituye el punto de partida del sistema, esta estructura contiene los mecanismos de inicialización y configuración de algunos componentes radicados en la Arquitectura Base que dan vida a los principales servicios que brinda. Por tanto, el trabajo posterior del módulo depende de dichas configuraciones que van desde la autenticación a través del Framework Común hasta los eventos y métodos que se adopten durante y después de la misma. Teniendo en cuenta el nivel de visibilidad y

responsabilidad que tiene este espacio sobre el resto y que justamente el usuario interactúa implícitamente con él, el Inicio se ubica en la capa superior, de esta manera, se puede hacer referencia a cada uno de los Recursos que se brindan en las diferentes capas del subsistema.

Formularios: responden a un comportamiento y diseño estándar, los módulos tienen la misma forma con vistas a facilitar el trabajo con ellos y la estandarización del sistema completo. Su uso se basa en la explotación de los Formularios de Interfaz de Usuario que realmente provee el Framework Común simplificando el desarrollo del sistema. Resulta importante puntualizar que en las interfaces de usuario no se programará el flujo normal de eventos que en ellas ocurre, dicha responsabilidad está delegada en la Gestión a través de las acciones, las cuales se asociarán a estos formularios. En este sentido, cada uno de ellos tienen que heredar de las clases que se brindan en el framework en dependencia del tipo de interfaz que se desee obtener (frmAreaTrabajo, frmMensaje).

Gestión: contiene las acciones que se asocian a los formularios a través del Framework Común.

- Cada acción define un bloque de código reusable por varias operaciones sobre el sistema.
- Una acción es una clase que representa precisamente la ejecución de alguna tarea concreta. Estas tareas no deben ser excesivamente complejas y la clase debe heredar de la clase “Accion” o “AccionSegura” (Framework Común), según el nivel de visibilidad que se requiera.
- Se le debe reescribir el método CrearForma con vistas a controlar los eventos del formulario asociado.
- Se deben programar dichos eventos teniendo en cuenta su utilización a partir de la interacción con la forma visual.

El hecho de tener separada la lógica de presentación de los formularios de interfaz de usuario en acciones asociadas a los mismos, brinda mayor flexibilidad, modificabilidad y mantenibilidad del software.

Componentes de Interfaz de Usuarios: permiten encapsular una función determinada que será utilizada por más de un proceso en el subsistema. El hecho de que estén en la Capa de Presentación se debe a que trabajan con los formularios directamente, ya sea de forma visual o con los datos que se encuentran relacionados en el mismo a través de otros componentes o controles. Cada componente puede incorporar clases de negocio e inclusive elementos de acceso a datos encapsulados y distribuidos según la estructura que tiene la Arquitectura Vertical.

Entidades: contiene las clases que son útiles para la presentación, es decir, representan datos que

son necesarios para facilitar el trabajo en esta capa teniendo en cuenta que los existentes, por el hecho de ser generales, pueden ser demasiado grandes o insuficientes según sea el caso, para una funcionalidad determinada.

Pruebas: contiene las clases diseñadas para realizar pruebas de unidad u otras a los diferentes bloques de código de este nivel. Esta estructura estará implementada utilizando las facilidades que brinda Spring.NET a través de su integración con el Framework NUnit.

Nivel de Negocio

Su diseño depende directamente del negocio específico al que se refiera cada subsistema. Esta capa recibe una petición del nivel superior, gestiona o procesa la misma, de ser necesario solicitándola a la capa de Acceso a Datos y finalmente envía una respuesta continuando el proceso en el punto donde se inició dicha petición.

Recursos: contiene las configuraciones y estructuras que presenta la capa para el trabajo correcto de determinadas funcionalidades (ej. Administrador de Transacciones). Asimismo establece una fachada que facilita la comunicación con los objetos que se deseen lógicamente ubicados en el nivel.

Entidades: contiene las clases que son útiles para el negocio, es decir, representan datos que son necesarios para facilitar el trabajo en esta capa teniendo en cuenta que los existentes, por el hecho de ser generales, pueden ser demasiado grandes o insuficientes, según sea el caso, para una funcionalidad determinada. Cada entidad es un elemento auto sustentado, es decir, pueden procesar sus propios datos o valores sin interactuar con los demás componentes del negocio, esto quiere decir que consiguen persistir la información, en caso de ser requerido, según la competencia que tengan sobre la misma.

Gestión: contiene las clases que tienen como objetivo resolver determinadas funcionalidades correspondientes a los Caso de Uso del subsistema. En dichas funcionalidades puede ser útil emplear los datos resultantes de las entidades que se obtengan en este nivel. Estas clases correspondientes a las funcionalidades se denominan Gestores, los cuales pueden agruparse según la necesidad del diseño.

Componentes: son recursos utilizados para encapsular una función determinada que será utilizada por más de un proceso de negocio en el subsistema. Lógicamente estas funcionalidades son únicamente de este nivel, es decir, no tienen presentación pero si pueden utilizar recursos de acceso a datos.

Interfaces Gestión/Entidades: se utilizan para ofrecer funcionalidades que van a representar un

nivel de abstracción. Esta distribución precisamente asegura el patrón Bajo Acoplamiento y tributa en la mantenibilidad, modificabilidad, flexibilidad entre otros.

Pruebas: contiene las clases diseñadas para realizar pruebas de unidad u otras a los diferentes bloques de código de este nivel. Esta estructura estará implementada utilizando las facilidades que brinda Spring.NET a través de su integración con el Framework NUnit.

Nivel de Acceso a Datos

Definir la estrategia de persistencia de un sistema es una de las decisiones de Arquitectura más importantes. En una aplicación estándar más del 50% del código generado está relacionado con lógica de persistencia. Por tanto, esta es la capa más crítica y sensible a cambios, pues controla todo lo concerniente a la información que se encuentra en la fuente de almacenamiento (Capa de Datos).

Al ser la capa inferior los componentes que contiene desconocen los niveles superiores, únicamente se limitan al manejo de la información, ya sea para persistirla u obtenerla para su procesamiento y propagación por la aplicación. Todo este manejo es responsabilidad de los Objetos de Acceso a Datos (DAOs por sus siglas en inglés).

Gestión: todas las funcionalidades del Acceso a Datos radican en los DAOs, es decir estas clases implementan la totalidad de las operaciones de persistencia y obtención de datos explotando los recursos que brinda NHibernate Framework que cumple perfectamente con el objetivo de este nivel, dígase trabajo con procedimientos almacenados, métodos de persistencia y consultas.

Recursos: contiene las configuraciones y estructuras que presenta la capa para el trabajo correcto de determinadas funcionalidades para la persistencia (ej. Fábrica de Sesiones). Asimismo se establece una fachada que facilita la comunicación con los objetos que se deseen contenidos en este nivel.

Mapeo: este componente es de uso exclusivo del **Object-Relational Mapping** (**ORM)**, en este caso el framework NHibernate. Se decide encapsularlo en un ensamblado ya que son ficheros XML de configuración independiente de cualquier implementación; por tanto resulta útil tenerlos separados del código garantizando una alta cohesión.

Interfaces: representan las funcionalidades que brinda esta capa, es decir, las referidas a los DAOs. Su uso e importancia es la misma que las de la capa Lógica de Negocio.

Nivel de Dominio

Se encuentra distribuido verticalmente teniendo en cuenta que contiene todas las entidades y recursos que representan los valores correspondientes a los datos o servicios que se manejan en el

dominio completo del subsistema.

Recursos: contiene las configuraciones y estructuras que presenta la capa para el trabajo correcto de determinadas funcionalidades que pueden ser utilizadas en la totalidad del subsistema (ej. Almacén de Mensajes, Registro de Excepciones).

Entidades: son los datos concretos que se manejan en el dominio total del subsistema. Las clases radicadas en este ensamblado no realizan persistencia ni otro tipo de operación, simplemente constituyen la materia prima para los diferentes procesos.

Interfaces: representan una fachada para los datos que se encuentran en este nivel, es decir, las referidas a las Entidades de Dominio.

De manera general cada uno de los niveles que se relacionaron anteriormente tiene su responsabilidad, siendo la forma en que uno u otro se relacionan única y restrictivamente.

2.2 Estándar de Codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas. Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento y mantenimiento del software.

La adopción de un estándar de codificación sólo es viable si se sigue desde el principio hasta el final del proyecto de software. No es práctico, ni prudente, imponer un estándar de codificación una vez iniciado el trabajo. En el caso de Registros y Notarías, el estándar de codificación fue definido por la dirección del proyecto en sus inicios, se define el formato para los siguientes puntos:

- Organización de los ficheros.
- Indentación.
- Comentarios.
- Declaraciones.

- Espacios en blanco.
- Convenciones de declaración.
- Prácticas de programación.

Una vez visto y explicado las bases sobre las cuales se construyó el sistema se presentarán algunos de los patrones de diseño que más significación tuvieron en la construcción de la aplicación, para una mayor flexibilidad en todos los procesos.

2.3 Patrones Utilizados en la Solución Propuesta

Los patrones más usados en el desarrollo de este sistema son los siguientes:

- **Patrón Abtract Factory (Fábrica Abstracta):** proporciona una interfaz para la creación de familias de objetos interdependientes o interrelacionados, sin especificar sus clases concretas. El patrón se usó para la creación de las clases del dominio, usándose la implementación que brinda Spring, configurando un XML para que usara una fábrica propia diseñada por el equipo de desarrollo.
- **Patrón Solitario (Singleton):** la intención del patrón es mantener un punto de acceso global a una instancia de una clase garantizando que esta sea única. Este patrón fue implementado en las clases Gestoras del Negocio, y los DAOs, además en las fábricas implementadas para la creación de objetos de las clases del Dominio.
- **Patrón Proxy (Intermediario):** su funcionamiento se base en proporcionar un delegado para que se encargue de controlar el acceso a una instancia de una clase mayormente por motivo de eficiencia. Este objeto intermediario actúa en lugar del verdadero objeto, ofreciendo la misma interfaz, solicitándola en el objeto cuando sea necesario. Este patrón es muy usado en cada momento que se necesita el acceso a los datos de los trámites que el intermediario se encarga sólo de cargar los datos necesarios para ese paso o flujo de manera que no se ponga en peligro el rendimiento de la solución.
- **Patrón State (Estado):** permite que un objeto modifique su comportamiento cada vez que cambie su estado interno. Está motivado por aquellas clases en que, según su estado actual, varía su comportamiento ante los diferentes mensajes. Es utilizado para manejar los diferentes estados por los que pasa el trámite, ya sea por cada uno de los flujos dentro de la inscripción de documentos, así como dentro del mismo subproceso.

2.4 Componentes Visuales Usados

Para el desarrollo del sistema informático que se propone fueron utilizados algunos componentes visuales (*Véase Figura 4*). Dichos componentes fueron heredados de anteriores soluciones implementadas en el proyecto Registros y Notarías Fase I con el objetivo de para facilitar la implementación del diseño que se propone. Estos componentes son orientados a objetos, lo que dotó al equipo de desarrollo de productividad en el trabajo, eficiencia y claridad del código. A continuación se mencionan los componentes y sus características fundamentales.

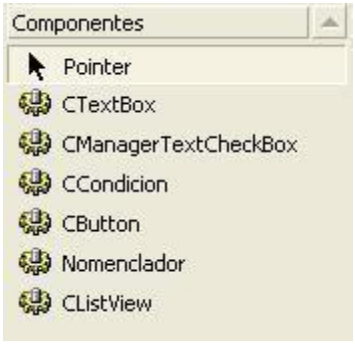

Fig. 4 Componentes Visuales

- **Componente CTextBox**

Principales propiedades

Propiedad: nombre de la propiedad que va a ser entrada y/o visualizada por el CTextBox.

Filtro: caracteres que acepta el CTextBox que poseen el siguiente formato:

Por ejemplo: “0&9|a&z|., “

De esta forma, el componente acepta sólo números del 0 al 9, letras de la *a* a la *z* minúsculas, además de caracteres como el punto (.), la coma (,), y el espacio ( ).

ColorBien y **ColorMal**: permiten seleccionar los colores que debe mostrar el CTextBox para resaltar datos válidos e inválidos.

MaxLength: define la longitud máxima de caracteres usados.

Moneda: define el símbolo monetario a mostrar.

DigitosDecimales: cantidad de dígitos decimales admitidos.

UsarSeparadorGrupos: permite separar o no las cifras numéricas por grupos.
Ejemplo: 12345678 ➔ 12’345’678.
Separador: indica el caracter que hará función de separador decimal.
MultiEspacio: permite validar si se desea entrar o no varios espacios consecutivos.
Tipo: permite seleccionar el tipo de dato a manejar.

- **Componente CManagerTextCheckBox**

Componente no visual de importancia para el control de todos los componentes **CTextBox** y de las operaciones con ellos, provee eficiencia y economicidad de código. La asociación del componente **CManagerTextCheckBox** con los componentes **CTextBox** tiene lugar en el evento Load o en el constructor del formulario. Por ejemplo:

```
forma.CManagerTextCheckBox.Add(forma.CTextBox1); ……
forma.CManagerTextCheckBox.Add(forma.CTextBox2);
```

El componente define dos métodos fundamentales: GetValor (Object) y SetValor (Object). Por ejemplo:

```
forma.CManagerTextCheckBox.GetValor(persona);
```

De esta forma, el objeto persona se llena con todos los valores de los componentes CTextBox relacionados con el CManagerTextCheckBox correspondiente.

```
forma.CManagerTextCheckBox.SetValor(persona);
```

De esta forma, el objeto *persona* muestra sus datos en los componentes **CTextBox** relacionados con el **CManagerTextCheckBox** correspondiente.

- **Componente CCondicion**

Componente no visual asociado a un CTextBox que permite definir reglas a ser aplicadas en este último. Las reglas definidas incluyen operadores contenidos en el componente como son: “*Igual*”, “*No Igual*”, “*Mayor*”, “*Menor*”, “*MayorIgual*”, “*MenorIgual*”. Su función principal es determinar si los valores de los CTextBox cumplen las reglas definidas.
Ejemplo: 50 < cantidad < 100 y cantidad != 75.

- **Componente CButton**

Principales propiedades CListViewAsociado: propiedad para asociar un CListView (ver posteriormente) a un **CButton.** Tiene el objetivo de permitir que el CButton esté activo o no en caso

de existir elemento (s) seleccionado (s) en el CListView, usado para validar acciones del tipo Eliminar o Modificar principalmente.

Las propiedades que se muestran a continuación son editadas en caso de que el **CButton** haga la función de trasladar elementos desde un CListView (**CListViewFuente**) hacia otro (**CListViewDestino**) y/o viceversa. **CListViewFuente** y **CListViewDestino**: Estas propiedades permiten seleccionar un CListViewFuente y CListViewDestino respectivamente. **Accion**: Permite seleccionar una de las siguientes acciones:

- MoveDtoF: mover un elemento desde el **CListViewDestino** hasta el **CListViewFuente.**
- MoveFtoD: mover un elemento desde el **CListViewFuente** hasta el **CListViewDestino.**
- MoveFtoDAll: mover todos los elementos desde el **CListViewFuente** hasta el **CListViewDestino.**
- MoveDtoFAll: mover todos los elementos desde el **CListViewDestino** hasta el **CListViewFuente.**

CManagerTextCheckBox: esta propiedad permite seleccionar el **CManagerTextCheckBox** asociado al CButton, para acciones del tipo Aceptar, en las cuales se confirma la captura de valores, validando de forma automática que todos los datos de entrada estén correctos.

- **Componente Nomenclador**

Facilita el trabajo con las tablas nomencladoras de la BD asegurando un conjunto de funcionalidades que resultan útiles para el trabajo con este tipo de datos durante la implementación. Su representación visual es similar a la del componente “ComboBox” definido en Visual Studio.NET.

Principales propiedades

Servidor: especifica el servidor para autenticar el Nomenclador en tiempo de diseño.

Usuario: especifica nombre de usuario para autenticar el Nomenclador en tiempo de diseño.

Contraseña: especifica la contraseña para autenticar el Nomenclador en tiempo de diseño.

Activo: permite activar el nomenclador para su uso.

TipoOrigenDatos: especifica el tipo de origen de datos que va a cargar el nomenclador, puede ser “Vista” o “Procedimiento Almacenado”.

NombreOrigenDatos: especifica el nombre de la tabla de donde cargará los datos el nomenclador.

Dependencia: define dependencias entre nomencladores.

ColumnaMostrar: especifica el nombre de la columna que se quiere visualizar.

- **Componente CListView**

Principales propiedades

Editable: permite seleccionar que el **CListView** sea editable o no.

NoMostrarCero: permite seleccionar si se pretende mostrar o no el valor cero.

PermitirOrdenar: permite seleccionar si se quiere ordenar los objetos del **CListView** al hacer Clic en el encabezado de la columna.

Si el **CListView** es editable pueden definirse para cada columna mediante la propiedad **ColumnsC** los componentes asociados, por ejemplo: un CTextBox, un Nomencaldor, un Combobox, un CButton. En este caso además se definen o redefinen eventos:

antesAdicionarEventHandler: permite validar si se puede adicionar o no.

caminandoEventHandler: permite validar si es permitido caminar con la tecla *Enter*.

CClic: se dispara al seleccionar un nuevo Item (Válido también para cuando el componente no es editable).

subItemBeginEditing: se dispara antes de comenzar a editar un item.

subItemEndEditing: se dispara al terminar de editar un item.

2.4 Diseño e Implementación

En este epígrafe se muestra lo concerniente al diseño e implementación del sistema informático integrado que se propone. Para obtener de manera más específica la solución técnica aplicada a la inscripción de documentos ver los artefactos Modelo de Diseño y Modelo de Implementación, adjuntos al documento de tesis.

2.4.1 Modelo de Diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar.

Como parte del modelo de diseño del proceso de Inscripción de documentos, se realizaron los casos de usos esenciales según su alto valor para la arquitectura, representados a través de los diagramas de interacción y diagramas de clases. En casos particulares estos diagramas fueron agrupados en

pequeños grupos equivalentes a los casos de usos significativos arquitectónicamente identificados, pues constituyen las funciones fundamentales del sistema. A continuación se exponen los artefactos generados dentro del flujo de trabajo diseño definido por RUP.

Los diagramas de clases: describen gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Normalmente, contienen clases, asociaciones y atributos, interfaces, con sus operaciones y constantes, métodos, información sobre los tipos de los atributos, navegabilidad y dependencias. Un diagrama de este tipo contiene las definiciones de las entidades del software. (17)

Como artefactos generados dentro del flujo de trabajo diseño existen también los diagramas de interacción que explican gráficamente las interacciones existentes entre las instancias (y las clases) del modelo de estas. El UML define dos tipos de estos diagramas: diagramas de colaboración y diagramas de secuencia. Ambos sirven para expresar interacciones semejantes o idénticas de mensaje.

Para la realización de la solución se diseñaron diagramas de secuencia. En ellos se muestran las interacciones entre objetos mediante transferencias de mensajes entre objetos o subsistemas. Cuando se dice que un subsistema “recibe” un mensaje, se quiere decir en realidad, que es un objeto de una clase del subsistema el que envía el mensaje. El nombre del mensaje debería indicar una operación del objeto que recibe la invocación o de una interfaz que el objeto proporciona. (18)

Cuando el diagrama de secuencias sólo se centra en un escenario (una instancia) dentro del caso de uso se conoce como diagrama de secuencias de instancias, si por el contrario tuviera en cuenta todos los escenarios de un caso de uso, se trata de un diagrama de secuencias genérico.

Por otra parte, un modelo de datos no es más que la representación de un fenómeno de la realidad objetiva a través de los objetos, sus propiedades y las relaciones que se establecen entre ellos. (19) El modelo de datos es usado para describir la lógica y física de la información persistente manejada por el sistema. Puede ser inicialmente creado a través de ingeniería inversa de un almacenamiento de datos persistentes que ya exista (base de datos) o puede ser inicialmente creado a partir de un conjunto de clases del diseño persistentes en el modelo de diseño. El creado para este sistema es una traza directa del diseño de clases del dominio, este describe de una forma abstracta como están representadas físicamente las entidades de dominio en una base de datos relacional.

2.4.2 Modelo de Implementación.

El modelo de implementación está comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Entre los componentes se pueden encontrar datos, archivos, ejecutables, código fuente y los directorios. Este artefacto describe cómo se implementan los componentes, congregándolos en subsistemas organizados en capas y jerarquías, y señala las dependencias entre éstos. (12)

El modelo de implementación muestra las dependencias entre las partes de código del sistema (diagrama de componentes) o la estructura del sistema en ejecución (diagrama de despliegue), siendo estos sus fundamentales artefactos. 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; mientras que los diagramas de despliegue muestran a los nodos procesadores, la distribución de los procesos y de los componentes.

2.4.3 Estrategia trazada para el diseño

Para la realización de la solución propuesta en el proyecto implementó una estrategia para el diseño, basada en los patrones arquitectónicos y de diseño propuestos para el desarrollo. A continuación se exponen los pasos a seguir en dicha estrategia.

1. Confección de los modelos conceptuales para cada uno de las fases dentro de la inscripción de documentos.
2. Confección del modelo de datos para la solución informática para el subsistema de Notarías Públicas.
3. Realización de los diagramas de clases correspondientes a cada caso de uso que se implementa, unido a los diagramas de secuencia correspondientes para cada uno de los escenarios que se necesiten.
4. Generación de código de las clases del dominio, y los gestores del negocio y de acceso a datos.
5. Capacitación al programador de los diagramas hechos para cada caso de uso, en específico aquel que le toque implementar.

2.5 Conclusiones Parciales

En este capítulo se muestra la solución propuesta para el sistema, mostrándose cada uno de los elementos que componen la arquitectura definida, algunos de los patrones de diseño y componentes usados para la implementación, culminando con los artefactos propios de los flujos de trabajo bases a defender en este trabajo, el modelo de diseño y el de implementación.

Capítulo 3. Análisis de Resultados

En este capítulo se evalúa el grado de calidad y fiabilidad de los resultados obtenidos con el desarrollo de este trabajo, teniendo por objeto la aplicación de algunas de las métricas empleadas internacionalmente para tal fin, específicamente las asociadas a las de la medición de la calidad del diseño y la implementación de software.

3.1 Aplicación de métricas para la evaluación de la calidad del sistema

El presente epígrafe hace un análisis de los resultados al aplicar las métricas de diseño e implementación del software. Se eligieron algunas de las métricas orientadas a objetos, ya que estas se han introducido para ayudar al ingeniero de software a usar el análisis cuantitativo y de esta manera evaluar la calidad en el diseño antes de que el sistema sea construido.

Para la validación del diseño del software se escogieron dos de las series de métricas más conocidas y divulgadas a nivel mundial, la primera fue la serie de métricas CK, la cual de sus seis (6) métricas plantadas, se escogieron dos (2) en particular, por su gran importancia, Árbol de Profundidad de Herencia (APH) y Número de Descendientes (NDD). La segunda serie que se escogió fue las métricas propuestas por Lorenz y Kidd, donde se eligió dentro de sus cuatro (4) propuestas para la medición del diseño, la de Tamaño de Clases (TC). También se aplicó la Complejidad Ciclomática como una de las métricas de prueba. A continuación se exponen los resultados.

3.1.1 Métrica Tamaño de Clase (TC)

Esta métrica consiste en medir el tamaño de una clase a partir de las siguientes medidas:

- Total de operaciones (operaciones tanto heredadas como privadas de la instancia), que se encapsulan dentro de la clase.
- Número de atributos (atributos tanto heredados como privados de la instancia), encapsulados por la clase.
- Promedio general de las dos métricas anteriores para el sistema en general.

La siguiente tabla muestra la relación de los intervalos de valores para determinar si el valor de TC se considera Grande, Medio o Pequeño en dependencia de los resultados obtenidos.

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

Para aplicar la métrica en el sistema se seleccionaron las clases dentro del módulo de gestión dentro de la capa de negocio que intervienen en el flujo de inscripción.

Resultado:

Después de haber aplicado la métrica TC, se llegó a la conclusión de que para un total de 25 clases, existe un promedio de 1,64 de atributos de y 4,44 de operaciones, lo que representa un 100% de clases de tamaño pequeño, 0% de clases de tamaño mediano y 0% de tamaño grande.

Umbral	Tamaño	Cantidad de Clases
<= 20	Pequeño	25
>20 y <= 30	Medio	ninguna
> 30	Grande	ninguna

Es válido señalar además, que a pesar de que el negocio a implementar posee un alto grado de complejidad, su implementación en el sistema ofrece grandes posibilidades de reusabilidad, no es difícil de comprobar y la mayor parte de las clases poseen un bajo nivel de responsabilidades.

3.1.2 Métrica Árbol de Profundidad de Herencia

Esta métrica describe la longitud máxima desde el nodo hasta la raíz del árbol, permitiendo indicar que muchos métodos se han heredado. Además de que 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.

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. Por otro lado, los valores grandes de APH implican que se pueden reutilizar muchos métodos, lo que debe ser considerado como un elemento a favor de la mantenibilidad.
Después de haber aplicado esta métrica dio como resultado que el nivel más alto de herencia en las clases del diseño es 2, lo cual se encuentra dentro del umbral definido para determinar que el diseño no es complejo, existe un bajo acoplamiento entre las clases y no es difícil su mantenimiento.

3.1.3 Métrica Número de Descendientes

Las subclases que son inmediatamente subordinadas a una clase de la jerarquía de clases se denominan sus descendientes. A medida que crece el número de descendientes (NDD), se incrementa la reutilización, pero también implica que la abstracción representada por la clase predecesora se ve diluida. Esto dificulta el mantenimiento, ya que existe la posibilidad de que algunos de los descendientes no sean realmente miembros propios de la clase. El valor aceptable al aplicar esta métrica es de 1 a 4.
En el caso del diseño presentado en la solución del software, al aplicar la métrica se obtuvo un valor de 4 NDD lo cual indica que se ofrece un cierto grado de reutilización y no se ve afectada el nivel de abstracción representado por la clase predecesora.

3.1.4 Complejidad Ciclomática

Se incluye dentro de las métricas del Código Fuente, específicamente dentro de las de análisis del código y constituye una medida del software que aporta una valoración cuantitativa de la complejidad lógica del sistema informático que se propone con la investigación. Como se puede apreciar en los métodos correspondientes al Gestor de Persona Natural y Persona Jurídica (*ver Figura 5*), los valores de complejidad ciclomática de los métodos en su totalidad de los casos es relativamente baja, con valores entre 1 y 2. Se arriba a la conclusión de que los códigos de la aplicación no son complejos, más bien legibles y de no muchas líneas.

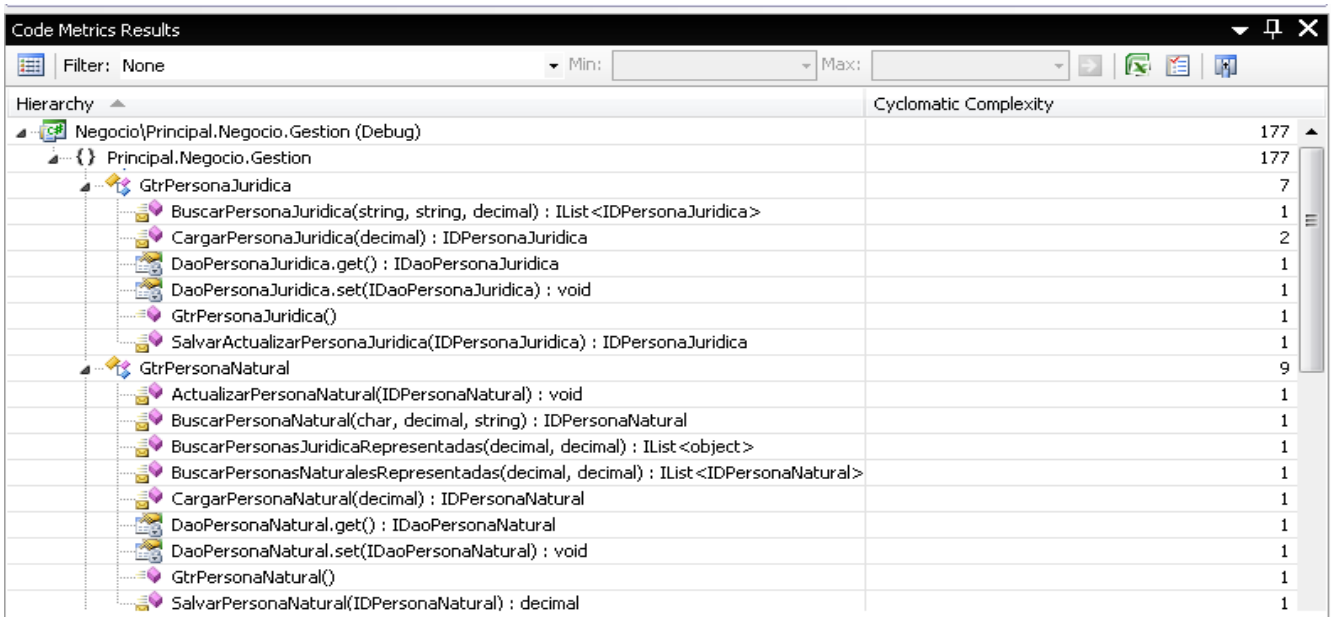

Fig. 6 Complejidad Ciclomática obtenida de los gestores GtrPersonaNatural y GtrPersonaJuridica

3.2 Resultados obtenidos de las pruebas del NUnit

Como se explicó en el capítulo 1 se utiliza el Framework NUnit en integración con Spring para la realización de pruebas de unidad. Se validaron funcionalidades implementadas en las capas de Presentación, en la del Negocio y en Acceso a Datos. Se muestra a continuación pruebas que se le hicieron a métodos relacionados de los gestores GtrPersonaNatural, GtrPersonaJuridica y GtrNomenclador, específicamente en la capa de Negocio.

Este es el método donde se declara el punto inicial de las pruebas de unidad, en él se declaran las inicializaciones de la Base de Datos de Notarías Públicas para las futuras pruebas.

```
[SetUp]
public void IniciarContexto()
{
    string conxString =
    "Data Source=rp213p.saren.mij.gov.ve;
     User Id=usuario_213;
     Password=Data Source=rp213p.saren.mij.gov.ve;
     User Id=usuario_213;
     Password=CslNMWU73oQb0Ny1dFos3361tW0=;;";

    Base.IniciarContexto(conxString,Base.TipoConexion.Oracle,true);
}
```

Figura 6. Método de inicio de las pruebas.

Este método se encarga de validar que la búsqueda de la persona sea correcta, para ello se pasa por parámetro un pasaporte real "57584" y comprueba si la persona "BARBARA" es igual al valor esperado de la persona.

```
[Test]
public void ProbandoBuscarPersonaNatural1()
{
    persona = Base.Resolver<IGtrPersonaNatural>();
    var letra = char.MinValue;
    IDPersonaNatural pers= persona.BuscarPersonaNatural(letra,-1, "57584");

    Assert.AreEqual("BARBARA", pers.PrimerNombre);

}
```

Figura 7. Prueba al método de Buscar Persona Natural.

Este método al igual que el anterior se encarga de validar en caso de que el valor esperado sea distinto de la persona pasada por parámetro.

```
[Test]
public void ProbandoBuscarPersonaNatural()
{
    persona = Base.Resolver<IGtrPersonaNatural>();
    var letra = char.MinValue;
    IDPersonaNatural pers = persona.BuscarPersonaNatural(letra, -1, "57584");

    Assert.AreNotEqual("BARBAR", pers.PrimerNombre);

}
```

Fig. 8 Prueba al método de Buscar Persona Natural con parámetros erróneos.

Este método se encarga de validar que la búsqueda de la persona jurídica sea correcta, para ello se pasa por parámetro tres atributos y comprueba si la persona jurídica es igual al valor esperado.

```
[Test]
public void ProbandoBuscarPersonaJuridica()
{
    persona = Base.Resolver<IGtrPersonaJuridica>();

    IList<IDPersonaJuridica> aux = persona.BuscarPersonaJuridica("BNV", "V1234", 10500);
    Assert.AreEqual(10500, aux[0].EstadoDomicilio);
}
```

Figura 9. Prueba al método de Buscar Persona Jurídica.

Este método se encarga de validar que cuando se ejecuta el método del gestor de Negocio CargarNomencladorPais el nomenclador no este vacío.

```
[Test]
public void ProbandoCargarNomencladorPais()
{
    //especificar datos con la BD
    obj = Base.Resolver<IGtrNomenclador>();


    IList<IDPais> aux = obj.CargarNomencladorPais();
    Assert.AreNotEqual(0, aux.Count);


}
```

Figura 10. Prueba al método de CargarNomancladorPais.

Este método al igual que el anterior se encarga de validar que cuando se ejecuta el método del gestor de Negocio CargarNomencladorEstadoCivil el nomenclador no este vacío.

```
[Test]
public void ProbandoCargarNomencladorEstadoCivil()
{
    //especificar datos con la BD
    obj = Base.Resolver<IGtrNomenclador>();


    IList<IDEstadoCivil> aux = obj.CargarNomencladorEstadoCivil();
    Assert.AreNotEqual(0, aux.Count);


}
```

Figura 11. Prueba al método de CargarNomancladorEstadoCivil.

Los resultados obtenidos con las pruebas realizadas fueron satisfactorios. Esto quedó corroborado con la interfaz mostrada por la herramienta NUnit (*ver Figura 12*), en la cual se muestra que no hubo alguna salida fallida. Para interpretar estos resultados vale destacar que el color verde representa el resultado satisfactorio, en caso contrario sería rojo.

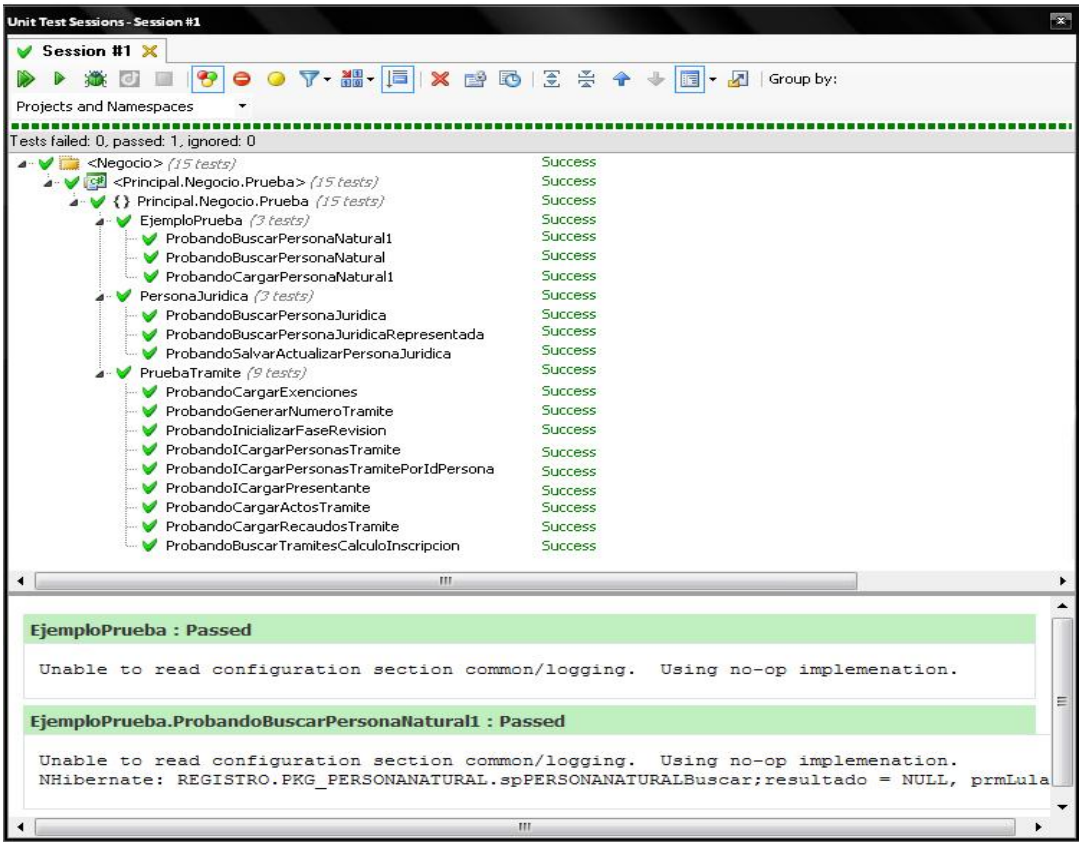

Figura 12. Pantalla de resumen del NUnit.

Unido a todas las anteriores validaciones vale mencionar que los artefactos generados en cada una de las etapas de desarrollo de software realizadas, fueron entregados al equipo de calidad de la facultad para su revisión y aprobación. Luego de haber procedido a rectificar las no conformidades (NC) resultantes, se generó una nueva entrega, obteniéndose como resultado esta vez, el acta de liberación de los artefactos del trabajo de investigación.

3.3 Conclusiones Parciales

En este capítulo se logró aplicar una serie de métricas para la validación del diseño propuesto, con lo cual se determinó que no presenta una alta complejidad estructural, de datos, ni del sistema en general, permitiendo que la implementación y las pruebas realizadas no fuesen complejas y afectando en un mínimo de tiempo algún cambio a realizar. La métrica de Tamaño de Clases

evidencia que la mayoría de las clases son reutilizables y la implementación no es complicada. Los niveles de profundidad de la herencia están acordes con los umbrales definidos por autores consultados permitiendo un bajo acoplamiento entre las clases. La métrica de la complejidad ciclomática arrojó resultados satisfactorios, concluyendo que los códigos de la aplicación no son complejos. Por último, se logró aplicar las pruebas de unidad con el NUnit Framework en las cuales se demostró haber implementado los requisitos primarios, donde los resultados fueron satisfactorios, llevando esto a que el producto responda a todas las necesidades y propósitos de los clientes.

Conclusiones Generales

Como conclusiones generales de la investigación se tiene que:

- Quedó definida la metodología, tecnología y las herramientas, para la construcción del software a partir de un estudio previo de las funcionalidades de los procesos notariales, mencionados en la fundamentación teórica.
- Se analizó la arquitectura planteada por la dirección del proyecto, fundamentando la elección hecha para una adecuada implementación del sistema que se propone.
- Se definió una estrategia para la construcción de un diseño que fuese flexible y escalable, a partir de la implantación de patrones y técnicas de diseño.
- Se lograron obtener los artefactos necesarios para las etapas de diseño e implementación, según la metodología de desarrollo de software definida (RUP).
- Se aplicaron distintos tipos de métricas que permitieron evaluar el nivel de calidad que presenta dicha solución.
- El sistema se apoya totalmente en las Leyes Registrales y Notariales, facilitando la organización y control de todos los procesos notariales planteados.
- Con el nuevo sistema, todas las operaciones en un trámite son accesibles y conocidas desde cada oficina, brindando una gran publicidad registral en todo momento.

Recomendaciones

Luego de haber concluido el trabajo de investigación se recomienda:

- Hacer el estudio y análisis de alguna plataforma o lenguaje en software libre que soporte la migración del software, sin que tenga que emplearse un gran tiempo de ejecución para esta migración.
- Desarrollar los diagramas de diseño e implementación de los otros procesos que complementan al proceso de Inscripción que se ejecutan en las Notarías Públicas.
- Documentar el Framework Común para que sea posible su utilización por otros desarrolladores y proyectos.

Bibliografía

1. **ALBET, Ingeniería en Sistemas.** *Estándares de codificación .* Ciudad Habana, Cuba : s.n., Octubre 10, 2006.

2. **Frías, Hugo Chávez.** *Exposición de motivos del decreto con fuerza de Ley de Registro y del Notariado.* Caracas, Venezuela : s.n., 2001.

3. **Kiosmy Almenares Herrera, Rubén de León Becerra.** *Diseño e Implementación del proceso de inscripción del Módulo Mercantil en las Oficinas Registrales de la República Bolivariana de Venezuela.* La Habana : s.n., 2007.

4. **Robles, Gregorio y Comunicaciones, Grupo de Sistemas y.** *Programación Extrema (y software libre).* Madrid : s.n., 2003.

5. **Baufest, proyecto Exactas.** *Resumen del seminario corto acerca de la metodología SCRUM.* Buenos Aires, Argentina : s.n., 2006.

6. **Rational Software Corporation.** *Product.* 2002.

7. Enterprise Architect-Herramienta CASE para diseño UML y desarrollo de software. [En línea] Sparx Systems. [Citado el: 26 de 3 de 2010.] http://www.sparxsystems.com.ar/products/ea/index.html.

8. Data Modeling Enterprise Software - Embarcadero ER/Studio. [En línea] [Citado el: 26 de 3 de 2010.] http://www.embarcadero.com/products/er-studio-enterprise.

9. **Cátedra de Paradigmas de Programación, Faculdad Regional Buenos Aires, Universidad Tecnológica Nacional.** *Fundamentos teóricos de los Paradigmas de Programación.* Buenos Aires : s.n., abril del 2005.

10. TechTarget. Whatis.com. The leading IT Encyclopedia and learning center. [En línea] TechTarget. [Citado el: 23 de 3 de 2010.] http://whatis.techtarget.com/definition/0,,sid9_gci1103696,00.html.

11. **Alejandro Abiague Napoles, Maikel Yonelis García Batista.** *Diseño e Implementación de la Solución Informática del proceso de Recaudación en la Administración Financiera de los Registros y Notarías de la República Bolivariana de Venezuela.* La Habana, Cuba : s.n., 2008.

12. **Maribel Silva Muñoz, Sándor Rodríguez Prieto.** *Diseño e Implementación de un sistema informático integrado para la Gestión de Compras de Bienes y Contratación de Servicios en los Registros y las Notarías de la República Bolivariana de Venezuela.* La Habana : s.n., 2008.

13. **Clements, Paul.** *A Survey of Architecture Description Languages.* Alemania : s.n., 1996.

14. **Gillies., A.C.** *Software Quality, Theory and Management.* s.l. : Thomson Computer Press, 1999.

15. **Kan, S. H.** *Metrics and Models in Software Quality Engineering. .* s.l. : Addison - Wesley, 2003.

16. **Arregui, Juan José Olmedilla.** *Revisión Sistemática de Métricas de Diseño Orientado a Objetos. .* Madrid : Universidad Politécnica de Madrid, Facultad de Informática, 2005.

17. **Larman, C.** *UML y Patrones Introducción al Análisis y Diseño orientados a objetos.* 1999.

18. **Jacobson, Ivar, Rumbaugh, James y Booch, Grady.** *El proceso unificado de desarrollo de software. .* s.l. : Addison Wesley, 2000.

19. **Mato Garcia, R. M.** *Diseño de Bases de Datos .* 1999.

20. **ALBET, Ingeniería y Sistemas.** *Proyecto Técnico-Económico para la Modernización de los Registros y Notarías.* Caracas, Venezuela : s.n., 2005.

21. **Venezuela, Asamblea Nacional de la República Bolivariana de.** *Ley de Registro Público y Notariado.* Caracas, Venezuela : s.n., 2006.

22. **Gamma, e., Helm, R., Johnson, R., Vlissides, J.** *Design Patterns. Elements of Reusable Object-Oriented.* s.l. : Addison Wesley, 1994.

23. **Corporation, Microsoft.** *Microsoft Solutions Framework version 3.0 Overview. White paper.* June, 2003.

24. **Hensgen, Paul y Authors, Umbrello UML Modeller.** *Umbrello UML Modeller Handbook.* October 15, 2003.

25. **Barzanallana, Rafael.** Universidad de Murcia.Página Profesor Rafael Barzanalla. [En línea] [Citado el: 2010 de 3 de 26.] http://www.um.es/docencia/barzana/IAGP/Enlaces/CASE_principales.html.

26. **Proyecto Registros y Notarías.** *Documento Línea Base de la Arquitectura.* La Habana, Cuba : s.n., 2010.

27. **Alejandro Casanova, Armando Pacheco Iglesisas.** *Diseño e implementación de funcionalidades que se llevan a cabo en los registros mercantiles: solicitudes de expedientes, copias de documentos y sellado de libros.* La Habana : s.n., 2008.

28. Optima, your product development partner. [En línea] [Citado el: 15 de 6 de 2010.] http://www.optima-design.co.uk/products/oracle/oracle_10g_standard_one.html.

29. eBizinet.com. [En línea] [Citado el: 15 de 6 de 2010.] http://www.ebizinet.com/Home/Oracle/Oracle_Database10gR2.pdf.

30. Oracle. [En línea] [Citado el: 15 de 6 de 2010.] http://www.oracle.com/technology/products/ias/pdf/as_se1_datasheet.pdf.

Glosario de Términos

Documento: es el documento redactado por un abogado, donde se refleja cuál es la operación que se va a realizar en el trámite, en el se especifican los datos de los bienes y las personas involucradas en la operación, y luego es llevado a la notaría para ser registrado. El usuario lo presenta para ingresarlo al proceso de incripción, de él emana toda la información que comprobarán los funcionarios pertinentes en cada fase.

Encapsulamiento: forma de abstracción que separa las interfaces de las implementaciones de la funcionalidad del sistema (métodos) y oculta la información (variables).

Funcionario: personal que labora en la notaría, los mismos pueden cumplir diferentes roles, tales como: Funcionario de Cálculo, Funcionario de Presentación, Funcionario de Revisión, Funcionario de Archivo, Funcionario de Otorgamiento.

Herencia: concepto que permite que una clase sea definida como una extensión o modificación de otra.

Modelo Conceptual: un modelo conceptual explica los conceptos más significativos en un dominio del problema, identificando los atributos y las asociaciones, y es la herramienta más importante del análisis orientado a objetos.

Nota de Autenticación: es la nota que se genera al final del paso de procesamiento de los trámites de inscripción de documentos susceptibles a autenticación.

Prohibición: impedimento judicial, medida cautelar, que impide realizar una trámite en específico, ya sea porque pese sobre la persona o sobre el bien involucrado en la operación.

Planilla Única Bancaria: planilla emitida en el departamento de cálculo donde se asientan los conceptos por los cuales se debe pagar al servicio autónomo y el monto a pagar por cada uno, según el cálculo hecho al documento.

Polimorfismo: concepto usado en programación de forma que las entidades del programa pueda referenciar en tiempo de ejecución a objetos de diferentes clases.

Recaudos: documentos, comprobantes, avales, certificaciones y constancias. Que deben acompañar a los documentos a la hora de presentarlos, para conferirle valor legal al proceso y respaldar las operaciones contenidas en el mismo.

Tomo: libro físico, compuesto por varios folios, donde están inscritos y asentados los documentos, se numeran consecutivamente y también, al igual que los folios, sirven para referenciar dónde está inscrito el documento físicamente.