Desarrollo de los Subsistemas de Empresas Cubanas y Sucursales Extranjeras para el Sistema de Informatización Registral de la Cámara de Comercio de la República de Cuba

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

Autores

Ariel Jesús Batista Guerra
Antonio Diaz Soutuyo

Tutor
Ing. Alain Osorio Rodríguez
Co-tutor
Ing. Reinier Silverio Figueroa

Facultad 3 - Centro de Gobierno Electrónico

"Lo fundamental es que seamos capaces de hacer cada día algo que perfeccione lo que hicimos el día anterior"

Che

Declaración de Autoría

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

Para que así conste firmamos la presente a los _____días del mes de _____ del año 2012.

_____ _____

Ariel Jesús Batista Guerra Antonio Diaz Soutuyo

_____ _____

Ing. Alain Osorio Rodríguez Ing. Reinier Silverio Figueroa

Agradecimientos

Agradezco a mi mamá por apoyarme en todos los momentos de mi vida tanto en los buenos como en los malos y convertirme en la persona que soy, por luchar tanto por mí, además de quererme mucho y aguantarme, a mi hermanito linda por tener tanta confianza en mí, a mis abuelos Matilde y Antonio que a pesar de ya no estar presentes siempre los llevo conmigo.

A mis amigos Yadelis, Yoyi, Amarilis y Abel sin ustedes no soy nada, gracias por estar siempre listos para todo conmigo, dándome consejos, apoyándome especialmente en los momentos difíciles y levantándome los ánimos para seguir adelante,. A mi cotutor Reinier por haberme brindado toda la ayuda posible y a Danaysa que sin ella la defensa de nuestra tesis no hubiera sido posible. A los profesores que de una forma u otra han contribuido a mi formación profesional.

A todos mis amigos de la Universidad por compartir estos años juntos, muchas gracias.

TONY

A mi madre por ser la persona que más quiero en esta vida, por su apoyo incondicional, por luchar cada día para que pudiera lograr lo que soy hoy.

A mi papa Eduardo que me crió, me educó y me brindo su cariño a pesar que de no ser de su misma sangre, te quiero mucho.

A mi hermana por todo su cariño y preocupación.

A mi padre Audelio, a mis hermanos Audelín y William por apoyarme y brindarme su amistad.

A mis abuelos por ser tan buenos conmigo y quererme tanto.

A mis 4 sobrinitas lindas.

A todos mis tíos por el ejemplo y cariño que me han brindado.

A toda la familia por la confianza y el amor brindado.

A mi compañero de tesis por trabajar duro a mi lado.

A mi tutor y co-tutor por su apoyo en la realización de este trabajo.

A el presente tribunal, en especial a la profesora Danaysa.

A las personas del laboratorio 315 en especial los de 4to año de mi proyecto, al profe Orlando y Julio por su ayuda.

A todos los educadores que contribuyeron a mi formación en especial a las profesoras Dariela, Mónica y Dina.

A todos mis socios a lo largo de estos 5 años en especial a Luis Ernesto, Randy, Josue, Ricardo, Carlito.

Ariel

RESUMEN

La Cámara de Comercio de la República de Cuba es una institución vinculada al comercio, la industria y los servicios. La institución guía las mejores alternativas para el desarrollo de la actividad empresarial. Actualmente, este organismo no posee un sistema informático que le brinde seguridad y usabilidad a los procesos de registros que realiza, por lo que surgen varios problemas relacionados con el seguimiento y actualización de los datos. Para mejorar el proceso de registro se decide desarrollar el Sistema de Informatización Registral de la Cámara de Comercio de la República de Cuba. Este sistema informatizará varios procesos entre los que se encuentran la Afiliación de Empresas Cubanas y Sucursales Extranjeras.

El desarrollo de los subsistemas estará guiado por el Proceso Unificado de Desarrollo de Software y se utilizará el paradigma de desarrollo basado en componentes. Se hará uso de GEFORT[1] como marco de trabajo, el cual se apoya en el lenguaje de programación Java, y está compuesto por EJB[2] que es un componente que encapsula la lógica de negocio en el servidor. En la investigación se describen las principales características y funcionalidades del sistema, para realizar su posterior implementación. Durante la etapa de diseño e implementación se emplearon un conjunto de estándares que permitieron la correcta comprensión del código y contribuyeron a la obtención de los artefactos necesarios para dar solución al problema planteado. Estos artefactos fueron evaluados mediante un conjunto de pruebas y métricas para comprobar su calidad y correcto funcionamiento.

[1] Gobierno Electrónico Fortalecido.
[2] Empresa de Java Bean.

ÍNDICE

Índice de Tablas

Índice de Figuras

INTRODUCCIÓN

El comercio es una forma de intercambio antigua que constituye la base para el desarrollo económico y social de la humanidad. Entre la segunda mitad del siglo XVIII, e inicios de XIX, con la Revolución Industrial la humanidad sufre las mayores trasformaciones socioeconómicas, tecnológicas y culturales debido a que el trabajo manual fue remplazado por la industria y la manufactura. Esto propició un aumento sin precedentes de las actividades comerciales por lo que se comenzó a necesitar una estructura capaz de organizar las mismas. Las cámaras de comercio surgen con el objetivo de controlar y proteger intereses personales, así como mejorar el intercambio comercial, las cuales agrupan a un conjunto de comerciantes. Estas son organismos extendidos por todo el mundo, regulados por diferentes leyes según el propósito para el cual fueron creadas y las disposiciones de comercio de cada país.

El desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) ha contribuido al perfeccionamiento empresarial, con tal avance se ha logrado una mayor eficiencia y facilidad para realizar procesos de la vida diaria. En Cuba a pesar del bloqueo económico se ha logrado llevar paralelamente estos avances apoyado de la Universidad de las Ciencias Informáticas (UCI). El Centro de Gobierno Electrónico (CEGEL) de la Facultad 3 de la UCI en convenio con la Cámara de Comercio de la República de Cuba, aúnan esfuerzos en la elaboración de un sistema registral que le permita a esta última una informatización en los diferentes procesos que realizan. Por este motivo se creó el proyecto Sistema de Informatización Registral de la Cámara de Comercio (SIRECC) que tiene el objetivo de llevar a cabo esta tarea.

La Cámara de Comercio gestiona gran parte de los datos de las empresas asociadas de manera tradicional, es decir, de forma manual, lo que no siempre brinda la confiabilidad necesaria, ni asegura que la información sea la correcta. Frecuentemente esta información enfrenta problemas como el difícil acceso, desactualización y pérdida en el peor de los casos. Al concluir el proceso registral de las empresas se insertan los datos de las mismas en hojas de cálculo creadas en Microsoft Excel, herramienta de software de corto alcance para el cúmulo de información que presenta dicha entidad, y en Microsoft Access que es una base de datos para pequeños escenarios no apropiada para la magnitud de los datos necesitada. Debido a esta situación el registro de los datos se vuelve lento, inseguro y engorroso pues estas herramientas no son las más adecuadas para apoyar y agilizar el proceso de búsqueda.

Partiendo de la situación expuesta, en una primera etapa se realizó un análisis del proceso, donde fueron identificados 43 requisitos para el subsistema Afiliación de Empresas Cubanas y 52 para Afiliación de Sucursales Extranjeras. En esta nueva etapa se realizará el desarrollo de los procesos, partiendo de nuevas consideraciones arquitectónicas.

Tomando en consideración lo antes mencionado, surge como **problema de la investigación**: ¿Cómo elevar la usabilidad y seguridad de los datos, durante el proceso de registro de Afiliación de Empresas Cubanas y Sucursales Extranjeras a partir del levantamiento de requisitos?

En consecuencia el **objeto de estudio** es: el proceso de desarrollo de software del Sistema de Informatización Registral de la Cámara de Comercio.

De esta manera el **objetivo general** del presente trabajo es: desarrollar los requisitos acordados para que contribuyan a la usabilidad y seguridad de los procesos de registro de Afiliación de Empresas Cubanas y Sucursales extranjeras del proyecto SIRECC.

Identificándose como **Campo de Acción:** desarrollo de los requisitos acordados de los procesos de Afiliación de Empresas Cubanas y Sucursales Extranjeras del proyecto SIRECC.

En la presente investigación se sustenta la siguiente **hipótesis:** si se satisfacen los requisitos acordados, entonces se contribuirá a la usabilidad y seguridad de los procesos de registro de Afiliación de Empresas Cubanas y Sucursales extranjeras del proyecto SIRECC.

Del objetivo general se derivan los siguientes **objetivos específicos:**

- Elaborar el marco teórico, para fundamentar la investigación de una síntesis de los resultados alcanzados en la revisión bibliográfica e indagaciones.
- Realizar el diseño de los subsistemas de Afiliación de Empresas Cubanas, Sucursales Extranjeras, acorde a las nuevas consideraciones arquitectónicas.
- Implementar la solución.
- Validar la solución propuesta aplicando pruebas de unidad, caja negra y validación de las variables dependientes.

Para satisfacer estos objetivos específicos quedan definidas las siguientes **tareas de investigación**:

- Estudio y análisis de los procesos Afiliación de Empresas Cubanas de la Cámara de Comercio de la República de Cuba (Ariel Jesús Batista Guerra).
- Estudio y análisis de los procesos Sucursales Extranjeras de la Cámara de Comercio de la República de Cuba (Antonio Díaz Soutuyo).
- Estudio y aplicación de las metodologías, tecnología y lenguajes más utilizada a nivel mundial para el desarrollo de este tipo de aplicaciones (Ariel Jesús Batista Guerra – Antonio Díaz Soutuyo).
- Diseño de clases del subsistema Afiliación de Empresas Cubanas (Ariel Jesús Batista Guerra).
- Diseño de clases del subsistema Afiliación Sucursales Extranjeras (Antonio Díaz Soutuyo).
- Implementación del subsistema Afiliación de Empresas Cubanas (Ariel Jesús Batista Guerra).
- Implementación del subsistema Afiliación Sucursales Extranjeras (Antonio Díaz Soutuyo).
- Pruebas de calidad de software a los subsistemas Afiliación de Empresas Cubanas y Sucursales Extranjeras (Ariel Jesús Batista Guerra – Antonio Díaz Soutuyo).

Métodos Teóricos:

- **Analítico-Sintético:** se realizó un análisis de los referentes teóricos y la bibliografía en cuestión donde se hizo una extracción y síntesis de los elementos más significativos relacionados con el desarrollo que han tenido los procesos de Afiliación de Empresas Cubanas y Sucursales Extranjeras en la Cámara de Comercio. Esto sirvió como base teórica para el desarrollo de la investigación.
- **Modelación:** para crear abstracciones de la solución en aras de comprender la realidad; es usado en el proceso de desarrollo de software para hacer diagramas y crear otros artefactos.

Métodos Empíricos:

- **Medición:** se pone de manifiesto en las métricas de calidad, las diferentes pruebas de calidad de software para garantizar el correcto funcionamiento de los requisitos y artefactos generados en el desarrollo. La información así obtenida puede ser de carácter cualitativo o cuantitativo.

Estructura de la Tesis:

El presente trabajo está compuesto por 3 capítulos, el contenido de los mismos se describe a continuación:

Capítulo 1. Fundamentación teórica: en este capítulo se aborda los aspectos teóricos relacionados con la investigación. Se escoge mediante decisiones arquitectónicas las diferentes herramientas y tecnologías que se utilizaran para el desarrollo de la solución.

Capítulo 2. Diseño e Implementación de los subsistemas: se enfatiza en la parte técnica, se presentan los diferentes artefactos que se obtienen del diseño realizado, también se detallará la implementación desarrollada a partir del diagrama de clases, el modelo de datos, diagrama de componentes y en general toda una descripción del diseño e implementación.

Capítulo 3. Validación de la solución: se realizarán las pruebas unitarias y funcionales al producto donde se dará a conocer los resultados de la solución.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

Este capítulo constituye la base sobre la cual se construye el posible resultado del presente trabajo. Con los aspectos teóricos y técnicos adquiridos por la investigación se realizará la selección de la metodología, lenguajes de programación, marcos de trabajo y patrones a utilizar en el Desarrollo de los Subsistemas de Empresas Cubanas y Sucursales Extranjeras para el proyecto.

1.2.1 Seguridad

La seguridad informática es el conjunto de métodos y herramientas destinados a proteger los bienes informáticos de una institución. La información es un activo muy preciado y que se mantenga segura es una necesidad vital. La seguridad de la información tiene como objetivo de garantizar:

- **Confidencialidad:** la información o los activos informáticos son accedidos solo por las personas autorizadas para hacerlo.
- **Integridad:** la información solo puede ser modificada por las personas autorizadas y de la forma autorizada.
- **Disponibilidad:** los activos informáticos son accedidos por las personas autorizadas en el momento requerido.

Existe hoy día un gran número de amenazas que pueden afectar la seguridad de un sistema. A medida que se desarrollan las tecnologías informáticas y las herramientas de comunicaciones, los piratas disponen de posibilidades mayores para la propagación de amenazas. La seguridad informática cuenta con un conjunto de principios básico que debe presentar un sistema, como son:

Mínimo privilegio: se deben otorgar los permisos estrictamente necesarios para efectuar las acciones que se requieran, ni más ni menos de lo solicitado.

Eslabón más débil: la seguridad de un sistema es tan fuerte como su parte más débil. Un atacante primero analiza cuál es el punto más débil del sistema y concentra sus esfuerzos en ese lugar.

Proporcionalidad: las medidas de seguridad deben estar en correspondencia con lo que se protege y con el nivel de riesgo existente. No sería lógico proteger con múltiples recursos un activo informático que no posee valor o que la probabilidad de ocurrencia de un ataque sobre el mismo es muy baja.

Dinamismo: la seguridad informática no es un producto, es un proceso. No se termina con la implementación de los medios tecnológicos, se requiere permanentemente monitoreo y mantenimiento.

Participación universal: la gestión de la seguridad informática necesita de la participación de todo el personal de una institución. La seguridad que puede ser alcanzada mediante medios técnicos es limitada y debiera ser apoyada por una gestión y procedimientos adecuados, que involucren a todos los individuos. (Pfleeger, 2006)

Los procesos de registro de Afiliación de Empresas Cubana y Sucursales Extranjeras se realizan con un sistema que presenta una frágil seguridad, por lo que se hace necesario la creación de un producto seguro, que se base de los elementos anteriormente expuesto.

1.2.2 Usabilidad

El término usabilidad, no forma parte del Diccionario de la Real Academia Española (RAE); pero en la informática es bastante habitual. La expresión se refiera al grado de facilidad con que un usuario puede utilizar una herramienta fabricada por otras personas con el fin de alcanzar un cierto objetivo. La usabilidad sostiene que si el sistema en cuestión no puede ser utilizado por el usuario en todo su potencial, no es útil y, por lo tanto, es deficiente. El grado de usabilidad se mide a partir de pruebas empíricas (con trabajo de campo entre múltiples usuarios) y relativas.

El estándar ISO/IEC 9126 propone como usabilidad a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso. La usabilidad se compone de las siguientes características que serán comprobadas para validar el objetivo.

- Entendibilidad.
- Capacidad de Aprendizaje.
- Operabilidad.

Un sistema que presente alto grado de usabilidad implica una reducción y optimización general de los costos de producción, así como un aumento en la productividad. La usabilidad permite mayor rapidez en la realización de tareas y reduce las pérdidas de tiempo. (Nielsen, 2003)

1.3 Descripción de los procesos

La Cámara de Comercio de Cuba es una asociación de empresas vinculadas al comercio, la industria y los servicios, con reconocimiento ante los organismos del Estado. La institución permite orientar las mejores alternativas para el desarrollo de la actividad empresarial de las entidades que a esa asociación pertenecen, y constituye una herramienta para la reinserción de la economía cubana en el mundo de las relaciones económicas internacionales. Es una entidad que promueve las ofertas exportables de productos y servicios, así como las oportunidades de negocios e inversión de las empresas cubanas y la sustitución de importaciones, en beneficio de la economía nacional. Presta servicios a sus asociados, a las sucursales de compañías extranjeras acreditadas en el país, así como a empresarios extranjeros interesados en negociar con Cuba. (Cámara Comercio, 2009)

A continuación se describen los subsistemas a desarrollar en la investigación, donde se precisa un conjunto de acciones a tener en cuenta por los funcionarios de la Cámara de Comercio.

1.3.1 Proceso de Afiliación de Empresas Cubanas

El proceso de Afiliación de las Empresas Nacionales a la Cámara de Comercio de la República de Cuba se realiza de la manera siguiente: primeramente hay que explicarle a la empresa los detalles de los servicios que brinda la Cámara de Comercio a sus asociados, qué documentos debe presentar la empresa interesada y en qué consiste el proceso de afiliación, encargándose de esta tarea el *Especialista en Desarrollo Empresarial*. Las empresas interesadas en asociarse presentan ante la Cámara todos los documentos necesarios para la solicitud de afiliación, ellos son: *Carta de solicitud de afiliación*, *Ficha de la entidad*, *Copia de la Resolución de constitución de la Empresa* y *Copia de la Resolución de nombramiento del Director*, los cuales son revisados rigurosamente (Director de Formación y Membrecia, 2009). Con estos documentos se prepara un expediente por parte de los funcionarios de la Cámara, donde se resumen los datos principales de la empresa que solicita la afiliación; documentación que también es revisada y firmada por el *Director de Formación y Membrecía* de la Cámara de Comercio.

Posteriormente se envía el expediente a la Junta Directiva para que se realice la Admisión de Asociación. Una vez revisado el expediente, la Junta Ejecutiva emite el documento *Criterio de la Junta Ejecutiva* sobre *Admisión de Asociación* (Director de Formación y Membrecía, 2009). En caso de no ser aprobado el expediente se le comunica a la empresa la denegación de su solicitud. Si la solicitud es aprobada se disponen a elaborar los documentos asociados al proceso de afiliación con las firmas pertinentes. Al concluir el proceso de aprobación y asentamiento jurídico, el *Técnico en Información Comercial* registra en la base de datos toda la información de la empresa y se realiza el *Aviso de Cobro*. Concluido este proceso, se comunica a la empresa solicitante la aprobación de la solicitud y se le entregan los documentos asociados al proceso de afiliación, publicaciones y actividades programadas para su participación (Director de Formación y Membrecia, 2009).

1.3.2 Proceso de Afiliación de Sucursales Extranjeras

Para llevar a cabo el proceso de Registro de Sucursales Extranjeras, la empresa interesada presenta la solicitud de inscripción y la documentación correspondiente. El *Encargado del Registro* hará una revisión de la documentación en cuanto a completitud y legalidad, y verificará que se ajuste a los requisitos y principios establecidos. Luego de ser verificada la completitud y la correctitud de la documentación presentada, se elaborará un expediente que será revisado por el *Director Jurídico* y una vez aprobado por este, se elevará para la aprobación de la *Junta Directiva*. Después de ser revisado el expediente y aprobado el registro de la sucursal por la *Junta Directiva*, se elevará la propuesta al Ministerio del Comercio Exterior (MINCEX) para la aprobación final del registro. El MINCEX realizará la aprobación del registro de la sucursal extranjera, durante los días hábiles establecidos y emitirá una resolución la cual será enviada a la Cámara de Comercio para la continuación del proceso.

Posteriormente de recibida la resolución si es aprobada la sucursal, el Encargado del Registro informará de la decisión al representante de dicha sucursal y lo citará para el registro, en el cual se archivarán los datos de la misma y se le entregará la licencia para comercializar, así como toda la información relativa al funcionamiento de las Sucursales en el país y otras cuestiones de interés. En la *Licencia* que se expide hará constar la vigencia de la inscripción, así como el término establecido para presentar la correspondiente solicitud de renovación y se envía copia de la documentación entregada al MINCEX.

Una vez que se registre la sucursal se procede a realizar las correspondientes actualizaciones de la misma en la medida que se presenten. Estas actualizaciones pueden ser debido a un cambio de resolución social, un cambio de representante, la apertura de una Oficina Secundaria, o la ampliación o modificación de la licencia y por último la cancelación de la sucursal. En caso que no se apruebe el registro de la sucursal por el MINCEX se notifica al interesado y se archiva el expediente (Directora Jurídica, 2009).

Para el desarrollo de los anteriores procesos es necesario una metodología que guíe los diferentes pasos del ciclo de vida del software, así se puede sistematizar o formalizar la complicada tarea de desarrollar software.

1.4 Metodologías de desarrollo de software

Las metodologías de desarrollo de son un conjunto de procedimientos, técnicas, herramientas y soporte documental que estructura, planifica y controla el desarrollo de un software. Estas metodologías se dividen en dos grandes grupos, las orientadas al control de procesos o metodologías pesadas, y las metodologías orientadas a las interacciones con el cliente y el desarrollo incremental del software, conocidas como metodologías ligeras o ágiles. Las metodologías pesadas velan por un riguroso cumplimiento de las actividades a realizar, requieren de mucha documentación para prever todo de ante mano. Las metodologías ágiles se centran en crear un producto que funcione antes de estar llenando mucha documentación y el cliente está interactuando en todo momento en el proyecto. A continuación se conocerá de algunos ejemplos de metodologías y se decidirá por alguna para realizar el proyecto. (Metodologías Pesadas, 2011)

Se comenzará abordando acerca de **Rational Unified Process** (RUP) que es una de las metodologías pesadas más conocidas y usadas, permite el desarrollo de software a gran escala mediante un proceso continuo de pruebas y retroalimentación. Las principales características de esta metodología son:

- Iterativo e incremental.
- Centrado en la Arquitectura.
- Dirigido por Casos de Uso.

RUP divide el desarrollo de software en 4 fases que definen su ciclo de vida:

- **Inicio:** propone una visión general de la arquitectura de software, se define el alcance y se identifican los posibles riesgos asociados al proyecto.
- **Elaboración:** el objetivo de esta fase es desarrollar el plan del proyecto, establecer los cimientos de la arquitectura, elaborar un plan para la siguiente fase y al finalizar se debe contar con un dominio del problema.
- **Construcción:** esta fase se propone obtener un producto funcional del proyecto resultado de las iteraciones realizadas.
- **Transición:** en la presente fase se le entrega el producto al cliente brindándole las respectivas y sucesivas actualizaciones.

RUP es un proceso que define “quién” está haciendo “qué”, “cuándo” y “cómo” para alcanzar un determinado objetivo. También define un conjunto de actividades que son necesarias para convertir los requisitos del sistema en artefactos funcionales. (Jacobson, 1999)

También se investigó acerca de **Extreme Programming** (XP), otra de las metodologías de desarrollo de software más difundidas. XP es una de las metodologías de software que se clasifica entre las ágiles, se caracteriza por la rapidez del desarrollo, reduce en gran medida el costo del proyecto, minimiza el tiempo en cambio de etapas del proyecto, aprovecha al máximo las mejores prácticas de programación, se van realizando en todo momento pruebas funcionales para que los clientes vayan comprobando que el proyecto está satisfaciendo sus expectativas. XP desarrolla cuatro actividades que guían el desarrollo las cuales son:

- Codificar.
- Testear.
- Atender.
- Diseñar.

No es recomendable usar XP para aplicaciones complejas o de gran escala. XP intenta minimizar el riesgo de fallo del proceso por medio de la disposición permanente de un representante competente del cliente a disposición del equipo de desarrollo.

La simplicidad y la comunicación son esenciales para el desarrollo sobre esta metodología que sin dudas constituye una herramienta poderosa para la construcción de proyectos simples y de desarrollo rápido (Sánchez, 2004).

Metodología seleccionada

Sobre la base de las consideraciones anteriores la metodología que se escoge para el Desarrollo de los Subsistemas de Empresas Cubanas y Sucursales Extranjeras es RUP, y la presente investigación se enmarcará en los siguientes flujos de trabajos: diseño, implementación y pruebas. Dicha selección está sustentada en el hecho de que es una metodología que brinda un seguimiento detallado de cada una de las fases del desarrollo, se cuenta con una alta familiarización de la misma y su objetivo principal es crear software de alta calidad.

1.5 Herramientas CASE

Las herramientas CASE (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Ordenador) tienen diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo función 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 costos, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras (Kendall, 2010).

Rational Rose es una herramienta de producción y comercialización establecida por *Rational Software Corporation*. Rational Rose que utiliza el Lenguaje Unificado de Modelado (UML) como medio para facilitar la captura de dominio de la semántica, la arquitectura y el diseño. Sus características principales son (Zhao, 2005):

- No es gratuito, se debe hacer un previo pago para poder adquirir el producto.
- La ingeniería de código (directa e inversa) es posible
- Rational Rose habilita asistentes para crear clases y provee plantillas de código que pueden aumentar significativamente la cantidad de código fuente generada.

- Adicionalmente, se pueden aplicar los patrones de diseño, Racional Rose ha provisto 20 de los patrones de diseño GOF (*Gang of Four*, La Banda de los cuatro) para Java.
- Admite la integración con otras herramientas de desarrollo.

Visual Paradigm (VP) es una herramienta para el diseño en UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. También proporciona abundantes tutoriales de UML, demostraciones interactivas y proyectos. Presenta licencia gratuita y comercial. Es fácil de instalar, actualizar y compatible entre ediciones. Sus características principales son: (López, 2012)

- Soporte de UML.
- Ingeniería inversa: código a modelo, código a diagrama.
- Generación de código: modelo a código, diagrama a código.
- Diagramas EJB (Enterprise Java Beans): visualización de sistemas EJB.
- Generación de código y despliegue de EJB: generación de Beans para el desarrollo y despliegue de aplicaciones.
- Soporte de Mapeo Relacional de Objetos (Object Relational Mapping, por sus siglas en inglés): 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.

Selección de la herramienta CASE para el desarrollo de software

Visual Paradigm (VP), es la herramienta CASE seleccionada para modelar todos los diagramas del sistema. Entre las principales funcionalidades del Visual Paradigm 8.0 se encuentran capturar, diseñar, gestionar y documentar los artefactos del software en un único entorno compatible con varias metodologías de desarrollo, como RUP y XP, y generar diagramas de proceso de negocios. Tiene la ventaja de ser multiplataforma, soporta el ciclo de vida completo del desarrollo de software. Es una herramienta de gran

utilidad para el proyecto, por su nivel de integración con el entorno de desarrollo y por la posibilidad que brinda de trabajar sobre la plataforma libre GNU/Linux, sistema operativo sobre el cual se desarrolla el proyecto.

1.6 Lenguaje de modelado

En consecuencia de la elección de Visual Paradigm y que esta herramienta CASE utiliza como lenguaje de modelado UML. Este lenguaje permite modelar, construir y documentar los artefactos que forman un sistema de software orientado a objeto. Es considerado un estándar en la industria del software creado por Grady Booch, James Rumbaugh e Ivar Jacobson. UML es un conjunto de notaciones estándares que permite crear diagramas como: *diagrama de clases*, *diagramas de secuencias* y *diagrama de paquetes*. Las principales características que presenta dicho lenguaje son (Jacobson, 1999):

- Corrección de errores viables en todas las etapas.
- El cliente participa en todas las etapas del proyecto.
- Permite documentar todos los artefactos de un proceso de desarrollo.

Con la utilización de UML se mejora el tiempo total de desarrollo, el cual brinda un soporte a la planeación y control del proyecto.

1.7 Patrones de diseño

Los patrones de diseño son la base para la búsqueda de la solución de un problema en un contexto particular. Cada uno describe que ocurre una y otra vez en el entorno, y describe también el núcleo del resultado al problema. En general, un patrón está compuesto por cuatro elementos esenciales:

- **El nombre:** identificador que se utiliza para describir un problema de diseño, sus soluciones y sus consecuencias en una o dos palabras.
- **El problema:** describe cuando se utiliza el patrón.
- **La solución:** se describen los elementos que componen el diseño, sus relaciones, responsabilidades y colaboraciones.
- **Las consecuencias:** son los resultados de la aplicación del patrón.

En el diseño de los subsistemas a desarrollar es muy necesario asignar las responsabilidades adecuadamente, para realizar esta tarea se hace uso de los patrones de diseño GRASP (*General Responsibility Assignment Software Patterns* por sus siglas en inglés). A continuación se mencionan y se explican cada uno de los patrones usados:

Experto: este patrón plantea que la responsabilidad debe ser asignada al experto en la información, es decir, a la clase que cuenta con la información necesaria para cumplir esta responsabilidad.

Alta cohesión y bajo acoplamiento: constituyen dos patrones pero se recomienda tener un mayor grado de cohesión con un menor grado de acoplamiento. De esta forma, se tiene menor dependencia y se especifican los propósitos de cada objeto en el sistema. La alta cohesión se traduce en que la información pertenezca a la clase que le corresponda y el bajo acoplamiento es tener la menor dependencia entre clases posibles.

Otros patrones son los GOF (*Gang of Four*, la Pandilla de los Cuatro):

Fachada: provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.

Instancia Única: garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

Otro patrón muy utilizado por la tecnología EJB es la **Inyección de Dependencias** el cual se especializa en inyectar objetos en una clase, en lugar de ser la propia clase quien cree el objeto (Larman, 2011).

Los patrones brindan soluciones a problemas que se presentan en el proceso de desarrollo del software. Una buena estructura organizacional es esencial para un sistema, para ello existen diferentes patrones vinculados a la arquitectura del sistema, pero requieren de un mayor nivel de abstracción respecto a los patrones de diseño.

1.8 Patrones arquitectónicos

El estilo arquitectónico elegido para la realización del sistema es **Cliente-Servidor** sobre la base de una aplicación de escritorio. Este se basa en la existencia de un componente servidor, que ofrece ciertos servicios, escucha que algún otro componente requiera uno; un componente cliente solicita ese servicio al servidor a través de un conector. El servidor ejecuta el requerimiento (o lo rechaza) y devuelve una respuesta.

Además, cada una de las capas está regida por el patrón **N-Capas** para relacionar cada uno de los componentes contenidos dentro de estas. La ventaja principal de este patrón arquitectónico es que en caso de que sobrevenga algún cambio, sólo se modifica el nivel requerido evitando cambios relevantes en los niveles adyacentes. Estas capas deben estar débilmente acopladas entre ellas y con alta cohesión internamente, lo cual posibilita variar de una forma sencilla diferentes combinaciones de capas (Reynoso, 2004).

1.9 Entorno de desarrollo

Se decidió el desarrollo orientado a componentes el cual tiene su base en la reutilización de código elaborado con anterioridad. Este código fue probado y su funcionalidad fue comprobada. Para lograr comprender este paradigma se debe conocer que un componente es una porción de software reutilizable, con dependencias explícitas en un único contexto que suple una necesidad evidente de la arquitectura definida. El uso del desarrollo basado en componentes posee algunas ventajas (Terreros, 2013):

- **Reutilización del software:** se alcanzará un mayor nivel de reutilización de software.
- **Simplifica las pruebas:** permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- **Simplifica el mantenimiento del sistema:** cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
- **Mayor calidad:** dado que un componente puede ser construido y luego mejorado continuamente, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

El Desarrollo de los Subsistemas de Empresas Cubanas y Sucursales Extranjeras para el Sistema de Informatización Registral de la Cámara de Comercio de la República de Cuba está basado en componentes. Esto permitió la reutilización de componentes que eran comunes en diferentes procesos, incluyendo varios de los provistos por el marco de trabajo utilizado.

1.10 El marco de trabajo GEFORT

El marco de trabajo GEFORT desarrollado por el Centro de Gobierno Electrónico (CEGEL) perteneciente a la UCI, proporciona a las aplicaciones que lo utilicen funcionalidades y componentes que simplifican su desarrollo. Está diseñado para la creación de aplicaciones con arquitectura Cliente-Servidor utilizando la plataforma Ediciones Empresariales de Java (*Java Enterprise Edition* JEE, por sus siglas en inglés). Por esta razón GEFORT consta de dos partes, una que se encuentra en el lado del cliente facilitando la creación de la capa de presentación y la comunicación con los objetos remotos desplegados en el servidor, y una segunda que se encuentra en el lado del servidor donde se encuentra la lógica de negocio.

Algunas de las funcionalidades y componentes que provee son:

- Una interfaz de usuario básica.
- Un mecanismo para la internacionalización de las interfaces de usuario.
- Un mecanismo para el acceso a los procedimientos remotos publicados en un servidor de aplicaciones.
- Un mecanismo de mensajería.
- Un mecanismo para gestionar las excepciones lanzadas por la aplicación que lo utilice.

GEFORT provee un alto nivel de configuración y adaptabilidad, lo cual lo convierte en un software altamente reusable. Actualmente este marco de trabajo tiene grandes cualidades para la administración de las aplicaciones informáticas.

1.10.1 Java

La plataforma que utiliza el marco de trabajo GEFORT, es Java y dentro de esta específicamente, la Edición Empresarial de Java, la cual cuenta con varias tecnologías que facilitan en gran medida el trabajo como son: la administración de transacciones, un mecanismo para la invocación de procedimientos remotos y una

Interfaz de Programación de Aplicaciones (*Application Programming Interface* API, por sus siglas en inglés) para la persistencia de datos.

Java es un lenguaje de programación pensado de forma tal que sea sencillo, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutral, portátil, de gran rendimiento, multitarea y dinámico que permite a los desarrolladores realizar cualquier tipo de programación (Zukowski, 2003).

Orientado a Objetos: tiene una colección de componentes reutilizables, extensibles y sostenibles llamados bibliotecas de clases que van a permitir desarrollar software describiendo problemas mediante el uso de elementos u objetos desde el espacio del problema y no mediante un conjunto de pasos secuenciales que se ejecutarán en el ordenador.

Distribuido: Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir *sockets*[3], establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

Interpretado y Compilado: el código Java en lugar de ser nada más compilado en ejecutables nativos, es también traducido en códigos de bytes que se pueden transferir a cualquier plataforma que tenga JRE (*Java Runtime Environment*), que consiste en una Máquina Virtual de Java (JVM, por sus siglas en inglés) y de este modo pueden ejecutarse sin volver a compilarlos.

Robusto: Java verifica su código al mismo tiempo que lo escribe, y una vez más antes de ejecutarse, de manera que se consigue un alto margen de codificación sin errores. Este lenguaje posee una gestión avanzada de memoria llamada gestión de basura, y un manejo de excepciones orientado a objetos integrados.

Seguro: presenta varias funciones de seguridad para asegurar la correcta construcción y ejecución de programas distribuidos, entre ellas se pueden mencionar: el verificador de código de bit, el cargador de clases y el gestor de seguridad, garantizando así la certeza del código que se está ejecutando y evitando

[3] Los sockets: son básicamente formas en las que podemos interconectar 2 (o más) programas mediante el uso de la internet.

que el código no seguro realice operaciones seguras, además de las relacionadas con la autenticación, autorización y encriptación para proteger la privacidad y asegurar la integridad de los datos (Eckel, 2013)

1.10.2 Enterprise Java Beans (EJB)

Los *Enterprise Java Beans* (EJB) son componentes del lado del servidor para la plataforma *Java Enterprise Edition* (Java EE), que apuntan a crear un desarrollo rápido y simple para aplicaciones distribuidas, transaccionales, seguras y portables. EJB es un componente del lado del servidor que encapsula la lógica de negocio de una aplicación.

EJB de Sesión: gestionan el flujo de la información en el servidor. Generalmente sirven a los clientes como una fachada de los servicios proporcionados por otros componentes disponibles en el servidor. Puede haber dos tipos:

- **Con estado (`stateful`)**. En un Bean de sesión con estado, las variables de instancia del Bean almacenan datos específicos obtenidos durante la conexión con el cliente. Cada Bean de sesión con estado, por tanto, almacena el estado conversacional de un cliente que interactúa con el Bean. Este estado conversacional se modifica conforme el cliente va realizando llamadas a los métodos de negocio del Bean. El estado conversacional no se guarda cuando el cliente termina la sesión.
- **Sin estado (`stateless`).** Los Beans de sesión sin estado son objetos distribuidos que carecen de estado asociado permitiendo por tanto que se los acceda concurrentemente. No se garantiza que los contenidos de las variables de instancia se conserven entre llamadas al método.

Interfaz "*Home*"

La interfaz "*Home*" permite al código cliente manipular métodos de clase del EJB que no están asociados a ninguna instancia particular. La Interface "*Home*" permite crear las instancias de EJB de entidad o sesión a través del método `create` que puede ser sobrecargado.

Interfaz remota

La interfaz remota especifica los métodos de instancia públicos encargados de realizar las operaciones. Una sesión Bean puede implementar una sola interfaz, con la interfaz apuntada por un tipo de cliente diferente. La interfaz local es para aquellos clientes que corren en la misma máquina virtual que el

contenedor EJB. La interfaz remota es para clientes fuera del EJB. Frente a una consulta del cliente, el contenedor retorna un `stub` serializado del objeto que implementa la interfaz remota. El `stub` conoce cómo pasar llamadas a procedimientos remotos al servidor. (Monson-Haefel, 2004)

1.10.3 Tecnología Java Persistence API (JPA)

Para hacer el mapeo objeto-relacional se seleccionó JPA (*Java Persistence API*) por ser un estándar que permite acceder a los datos de una base datos relacional con un estilo orientado a objeto.

JPA fue desarrollada para la plataforma JEE por un grupo de expertos de EJB 3.0 como parte de JSR 220 (*Java Specification Request*) y su uso no se limita para los componentes de software EJB. JPA es un modelo de persistencia basado en POJO's (*Plain Old Java Object*) para mapear bases de datos relacionales en Java (Schincariol, 2009). *Java Persistence API* consta de tres áreas:

- El *Java Persistence API*.
- El leguaje de consultas.
- El mapeo de los metadatos objeto/relacional.

El *Java Persistence API* simplifica el modelo de programación para la persistencia de la entidad y agrega capacidades como (Schincariol, 2009):

- Requiere menos clases e interfaces.
- Prácticamente elimina los descriptores de despliegue a través de largas anotaciones.
- Proporciona una manera fácil y estandarizada para el mapeo objeto-relacional.
- Elimina la necesidad de código de búsqueda.
- Agrega soporte para herencia, polimorfismo y consultas polimórficas.
- Proporciona un lenguaje de consulta de *Java Persistence*, EJB QL.

1.10.4 NetBeans

Se eligió a NetBeans 7.2 pues permite el desarrollo de todo tipo de aplicación en Java. Posee buena integración con las herramientas que se utilizan en el marco de trabajo, como son el servidor de aplicaciones

Glassfish *Open Source Edition* y el Gestor de Base de Datos PostgreSQL. Además mejora el soporte de versiones anteriores para el desarrollo de aplicaciones empresariales en Java.

La plataforma Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés, *Integrated Development Environment*) NetBeans es una base modular y extensible usada como una estructura de integración para crear aplicaciones de escritorio grandes. Empresas independientes asociadas, especializadas en desarrollo de software, proporcionan extensiones adicionales que se integran fácilmente en la plataforma y que pueden también utilizarse para desarrollar sus propias herramientas y soluciones. La plataforma ofrece servicios comunes a las aplicaciones de escritorio, permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación.

El IDE NetBeans una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Es un producto libre y gratuito sin restricciones de uso. El IDE NetBeans es un entorno integrado desarrollo de código abierto escrito completamente en Java. (NeatBeans.org, 2013)

1.10.5 Servidor de aplicaciones

Los servidores de aplicaciones manejan generalmente la mayor parte de las funciones lógica de negocio y el acceso a datos de la aplicación. Un servidor de aplicaciones permite facilitar el desarrollo la alta disponibilidad, la escalabilidad y el mantenimiento al sistema.

Oracle Glassfish Open Source Server: es un servidor de aplicaciones de software, posee licencia de libre y de código abierto e implementa las tecnologías en la plataforma Java EE. También posee una excelente interfaz para facilitar las configuraciones, es un servidor rápido, presenta una alta escalabilidad y consume poca memoria. Glassfish tiene una consola de administración web que posibilita al administrador manejar el servidor de manera remota. Posee una implementación robusta para el control de concurrencias, tolerancias a fallos, seguridad y alta disponibilidad. (Oracle, 2003)

1.11 Sistema Gestor de Base de Datos (SGBD)

Un SGBD (En inglés DBMS: *Data Base Management System*) es un conjunto de programas para acceder a los datos de una base de datos. La base de datos contiene información relevante para una empresa. El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una

base de datos, de manera que sea tanto práctica como eficiente. Los sistemas de bases de datos se diseñan para gestionar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos para la manipulación de la información. Los Sistemas de Bases de Datos deben proporcionar la fiabilidad de la información almacenada, a pesar de las caídas del sistema o los intentos de acceso sin autorización (Silberschatz, 2002).

MySQL: es de los SGBD de software libre populares en la actualidad, licenciado bajo GNU GPL (Licencia Pública General de GNU, por sus siglas en inglés *General Public License*), actualmente subsidiada por la corporación Oracle. A diferencia de otros proyectos como Apache, donde el encargado de desarrollar software es la comunidad, MySQL está patrocinada por una empresa privada, la que brinda soporte y servicios.

GPL es una licencia creada por la Fundación de Software Libre, orientada a la protección de la libre distribución, modificación y uso de software. MySQL proporciona un servidor de base datos rápido, multiusuario, multiplataforma y robusto. El servidor MySQL se diseñó para entornos de producción críticos, con abundante carga de trabajo y para integrarse en software distribuido. Dentro de sus características están además:

- Replicación.
- Conexión segura.
- Transacciones y llaves foráneas.
- Consume pocos recursos.

Según las propias palabras del fabricante: “MySQL se ejecuta en más de 20 plataformas, incluyendo Linux, Windows, Mac OS, Solaris, IBM AIX, brindando el tipo de flexibilidad que te pone en control. Sin importar si eres nuevo en tecnología de bases de datos o un desarrollador experimentado o administrador de bases de datos, MySQL ofrece una variedad de herramientas para la base de datos, soporte, entrenamiento y servicios de consultas para su éxito.” (Oracle Corporation, 2012).

PostgreSQL: PostgreSQL se puede considerar como uno de los SGBD objeto-relacional de código abierto más avanzado. Publicado bajo la licencia BSD10 y desarrollado en la Universidad de California por el

Departamento de Ciencias de la Computación Berkeley, lleva más de 15 años de desarrollo y consta de una probada arquitectura basada en la fiabilidad e integridad de los datos.

PostgreSQL puede ser comparado favorablemente con otros SGBD. Contiene casi todas las características que se pueden encontrar en otras bases de datos comerciales u *Open Source*, y algunas extras que no se pueden encontrar en más ningún otro (Matthew, 2005).

En el sitio web del fabricante se plantea: “Lo mejor de todo es que el código fuente PostgreSQL está disponible bajo la licencia *Open Source*. Esta licencia permite la libertad para usar, modificar y distribuir PostgreSQL de cualquier manera que se desee, código abierto o propietario. Cualquier modificación, mejora o cambios que realices son suyos para hacer como desees. Por consiguiente, PostgreSQL no es solamente una bases de datos poderosa capaz de ejecutarse por empresas, es una plataforma de desarrollo en la que se puede desarrollar productos comerciales, web y domésticos que requieran de un SGBD objeto-relacional capaz.” (PostgreSQL Global Development Group, 2012).

Sistema de Gestor de Base Datos empleado

Después de lo anterior expuesto se escogió como gestor de la base de datos a PostgreSQL en su versión 9.1. La razón principal de porque se escogió este gestor es que su licencia es de software libre por lo que se puede hacer uso de él y comercializar el producto sin necesidad de pagar licencias, como sí hay que hacerlo con Oracle y SQL Server. No es solo eso, sino que es un potente SGBD en la actualidad, presenta características atractivas como la orientación a objetos y una excelente seguridad. Es multiplataforma y consume pocos recursos.

1.12 JUnit

JUnit es un marco de trabajo que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente. JUnit es también un medio de controlar las pruebas de regresión,

necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación.

Para realizar las pruebas de unidad a la solución de software desarrollada se utilizará el marco de trabajo JUnit, el cual es de código libre y gratuito. Además se encuentra integrado al IDE NetBeans 7.2 que permite la generación de las clases de pruebas y visualizar los resultados de las pruebas en forma de árbol o de reporte con formato html.

1.13 Pruebas de software

Al concluirse la implementación del sistema se deben realizar un conjunto de pruebas que permitan determinar si se cumplieron los objetivos propuestos en la investigación. Las pruebas de software consisten en la dinámica de la verificación del comportamiento en un conjunto finito de casos de pruebas. Son unas series de actividades que se realizan con la finalidad de encontrar fallos de implementación, calidad o usabilidad de un programa. Para comprobar el comportamiento del Sistema de Informatización registral de la Cámara de Comercio de la República de Cuba se realizarán las siguientes pruebas de software:

Pruebas de caja blanca: son las que se realizan sobre las funcionalidades internas del software. Para probar el correcto funcionamiento de los principales algoritmos se realizarán las pruebas unitarias que permiten controlar los caminos lógicos del software proponiendo casos de prueba que se ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o mencionado.

La herramienta para realizar las pruebas de unidad será JUnit, el cual es un entorno de trabajo que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.

Pruebas de caja negra: se llevan a cabo sobre la interfaz del sistema, obviando el comportamiento interno. El objetivo es demostrar que las funcionalidades del sistema son operativas. Estas permiten encontrar:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento. (Edu, 2011)

1.14 Conclusiones parciales

Con la realización de la fundamentación teórica de la investigación en curso se llegó a las siguientes conclusiones:

- Se realizó la selección de las metodologías, tecnologías y herramientas garantizando la mantenibilidad y la independencia tecnológica necesaria.
- Se analizaron diferentes metodologías de desarrollo de software donde se decidió el uso de RUP en sus flujos de trabajo Diseño, Implementación y Pruebas.
- Se acordó utilizar patrones de diseño y arquitectónicos para resolver problemas concurrentes.

CAPÍTULO 2. PROPUESTA DE SOLUCIÓN

2.1 Introducción

En el presente capítulo se tratan los elementos y características relacionados con el diseño e implementación de los subsistemas de Afiliación de Empresas Cubanas y Sucursales Extranjeras. Se exponen también los patrones de diseño empleados durante el desarrollo con ejemplos prácticos de su utilización. Se muestra el modelado de diagramas de paquetes, componentes y de clases del diseño que componen el modelo de diseño de los subsistemas a desarrollar. Comprenderá la definición de los estándares de codificación y la representación de la estrategia de integración entre los componentes que conforman el sistema.

2.2 Objeto de informatización

El objeto de informatización de la solución que se propone, está enmarcado esencialmente en el proceso de aprobación de afiliación, el cual está divido en cuatro etapas fundamentales que son: Procesar Documentación que incluye dentro de sí el registro de los datos de la empresa ya sea extranjera o nacional, la creación del expediente con todos los documentos necesarios y las observaciones; la segunda etapa es Aceptar Documentación donde se revisan los documentos del expediente y la siguiente etapa es Aprobar Afiliación que se encarga de gestionar la carta de aprobación. Siendo entonces la última etapa el Registro de Datos de Afiliación en la cual se añaden datos como: el tomo, el folio y la fecha de inscripción que constituyen los datos finales para el registro. A continuación se puede observar un diagrama de procesos donde se muestra las diferentes etapas que lo componen.

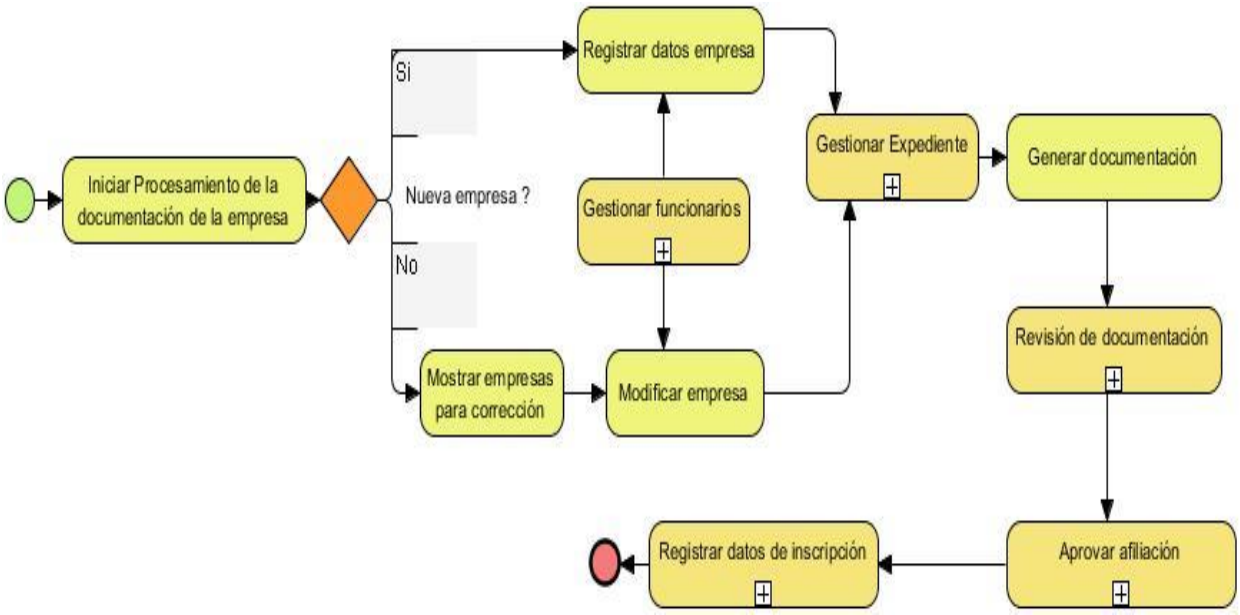

Figura 1: Diagrama de procesos.

La propuesta de solución de este trabajo consiste en el diseño, implementación y prueba de los subsistemas Empresas Cubanas y Sucursales Extranjeras. Dicha solución estará provista de las funcionalidades necesarias para que exista un correcto funcionamiento de las tareas relacionadas con la afiliación de dichas empresas y que contribuya a la gestión de la información.

Las funciones realizadas por los subsistemas están evidenciadas en los requisitos funcionales. En un análisis realizado se detectaron un conjunto de requisitos que deben estar presentes en la solución. A continuación se detallan un conjunto de requisitos pertenecientes al subsistema Afiliación de Empresas Cubanas, los restantes requisitos pueden ser consultados en el Anexo 3.

- RF_1: Registrar datos de la empresa: El sistema registra los datos que intervienen en el momento de guardar los datos de la empresa: entidad, siglas, directivo, representante, cargo, dirección, datos de la actividad económica, entre otros.
- RF_2: Modificar datos de la empresa: El sistema permitirá actualizar los datos de la empresa, teniendo como punto de partida los datos ya insertados los cuales también se pueden modificar.
- RF_3: Cancelar empresa: El sistema permitirá modificar el estado de una empresa registrada a pasiva cuando deje de ser miembro de la cámara.

- RF_4: Visualizar datos de la empresa: El sistema permitirá mostrar los siguientes datos que se tienen registrados de la empresa seleccionada.
- RF_5: Buscar empresa: El sistema permitirá buscar una empresa según los criterios de búsquedas especificados.
- RF_6: Listar empresas: El sistema permitirá listar todas las empresas registradas según la búsqueda realizada.
- RF_7: Registrar datos del funcionario: El sistema permitirá registrar los nuevos datos a la persona y asociárselo a la empresa.
- RF_13: Crear expediente: El sistema permitirá introducir los datos del expediente, así como insertarles los documentos.
- RF_29: Aceptar o denegar documentación: El sistema permitirá realizar una revisión de los documentos presentados por la empresa y cambiarle el estado a la institución.
- RF_30: Aprobar o denegar Afiliación de Empresa: El sistema permite aprobar o denegar la Afiliación de la Empresa.
- RF_31: Registrar datos de inscripción: El sistema listará las empresas con estada Aceptada y mostrará la interfaz para insertar los datos de inscripción.

A partir de las funcionalidades que deben realizar los subsistemas, es necesario un conjunto de patrones y abstracciones coherentes que proporcionen el marco de desarrollo.

2.3 Patrones arquitectónicos utilizados

Debido a que el diseño arquitectónico del marco de trabajo GEFORT está enfocado en un estilo Cliente-Servidor, la solución de software de la presente investigación se basa en el mismo estilo, el cual permitirá contar con una solución distribuida donde se podrá establecer una comunicación contractual entre funcionalidades que se ejecutan en las estaciones clientes y otras en el servidor para dar solución a los requerimientos de los usuarios, abstrayéndolos de la distribución física de la solución.

A pesar de que el estilo arquitectónico Cliente-Servidor proporciona un alto nivel organizativo, no es suficiente ya que el diseño de estas aplicaciones continúa siendo complejo. Para organizar mejor el diseño de estas partes se utilizará el patrón n-capas teniendo en cuenta que este permite establecer una organización jerárquica entre cada una de las capas, las cuales agrupan funcionalidades similares,

garantizando que cada capa proporcione sus servicios a la capa inmediata superior y que esta a su vez se sirva de las prestaciones que le brinda la capa inmediata inferior.

Dentro de la parte del Cliente se encuentran la capa `Presentación`, que maneja los formularios a través de las acciones. La capa `NegocioCliente` se encarga de la gestión de los datos en el cliente y se comunica con el servidor apoyándose de la interfaz `GestorFachadasRemoto` que permite llamar a las funcionalidades provistas por el componente EJB. Existen otras capas propias del marco de trabajo que se encargan de la internacionalización y de gestionar los mensajes de la aplicación, esto evidencia la aplicación del patrón n-capas. Igualmente pasa en la parte del Servidor donde las capas `NegocioServidor` y `AccesoDatos` dividen las funciones para responder a las peticionas realizadas.

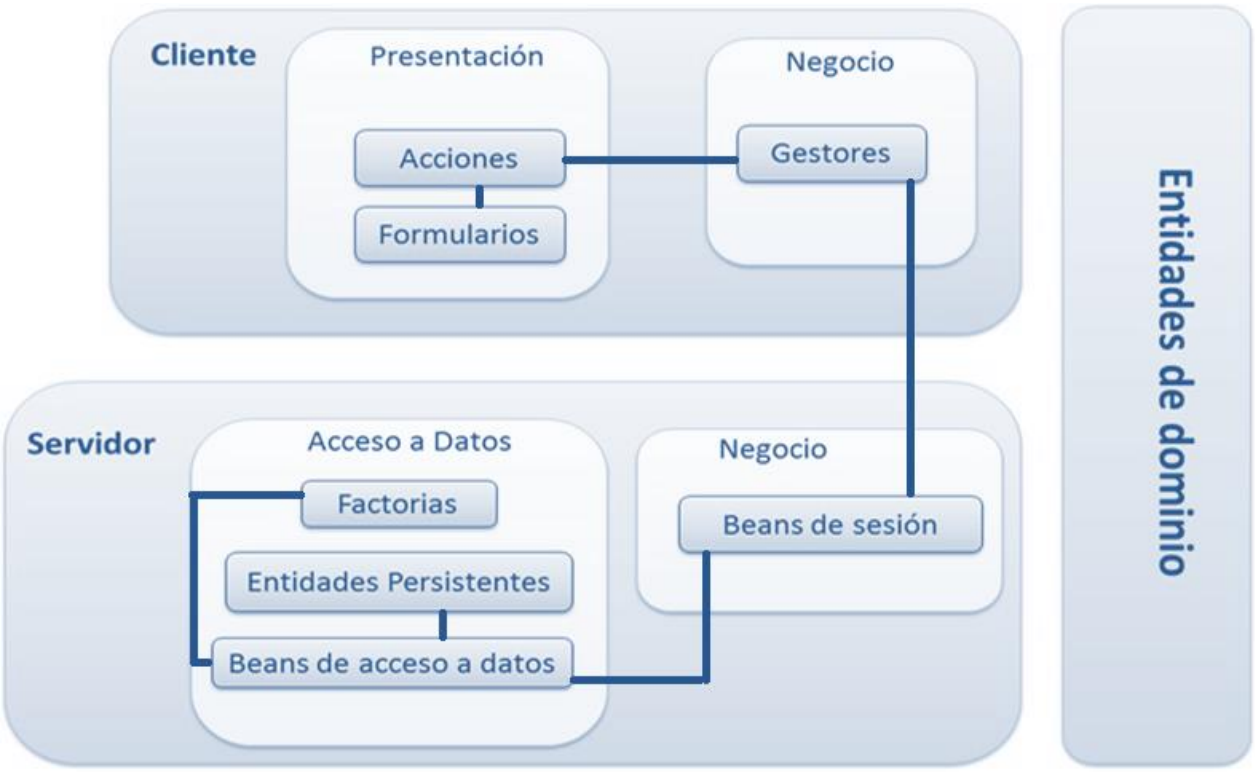

Figura 2: Arquitectura de los subsistemas.

2.4 Patrones de diseño utilizados

Patrón experto: Este patrón se utiliza en los Gestores de Acceso a Datos que son las clases que tienen la responsabilidad de devolver o insertar datos para realizar una funcionalidad determinada, o sea se comunica con la clase que cumple con una responsabilidad que otra clase no podría.

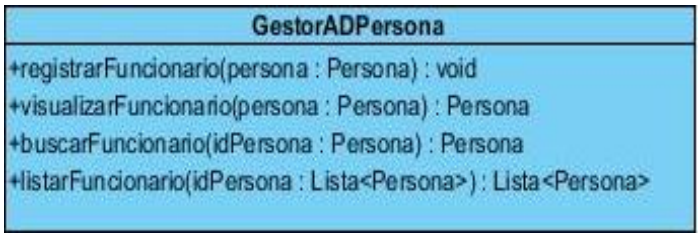

Figura 3: Aplicación del patrón Experto.

Alta cohesión y bajo acoplamiento: estos patrones se evidencian con el desarrollo basado en componentes debido a que un componente puede ser modificado y no implica grandes cambios a los demás que lo utilicen. Además para realizar una funcionalidad, cada capa se relaciona con su inferior o superior, apoyándose de las interfaces que permiten una menor dependencia entre clases, y así la objetividad de los patrones.

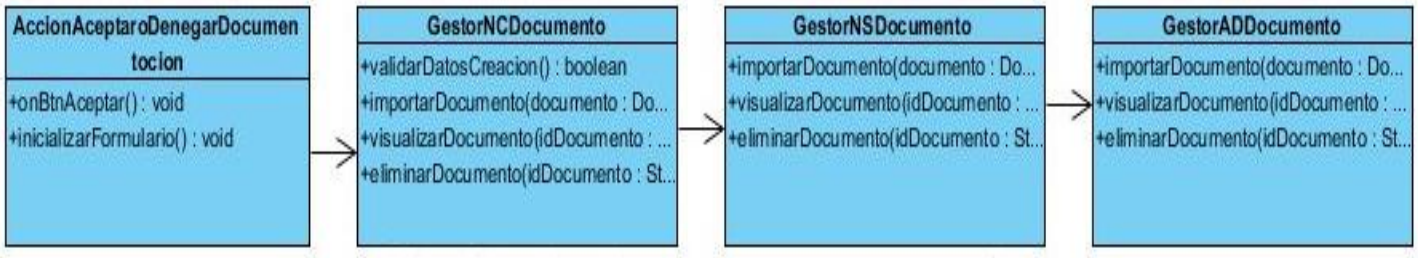

Figura 4: Aplicación de los patrones Alta Cohesión y Bajo Acoplamiento.

Fachada: el principal uso de este patrón se ve reflejado en la creación de la clase `FachadaGestoresRemoto` que permite comunicar el cliente con el servidor

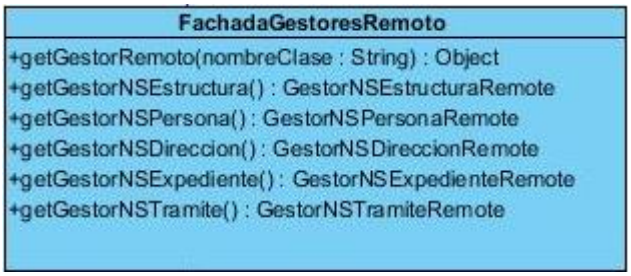

Figura 5: aplicación del patrón Fachada.

Instancia Única: este patrón es utilizado en la clase `GestorMarcoTrabajo` que permite abstraerse de la creación de instancias.

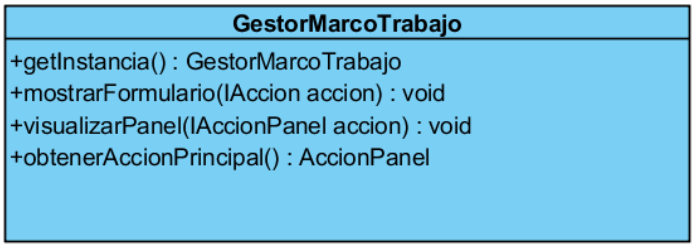

Figura 6: Aplicación del patrón Instancia Única.

Como se aprecia en la figura 6, el gestor devuelve su instancia y se declara el constructor de la clase como privado para que no sea instanciable directamente.

Inyección de Dependencias: este patrón es utilizado por EJB a la hora de crear objetos de otras clases como puede apresiar en la figura 7:

```
@EJB
private GestorADNombreGestorLocal gestorADNombreGestor
```

Figura 7: Aplicación del patrón Inyección de Dependencias.

La anotación @EJB permite abstraerte de inicializar el objeto, es decir, no será necesario el uso de new `NombreClase()`.

2.5 Modelo de diseño

El modelo de diseño es un prototipo de objetos que permite describir el posible resultado físico de los componentes concentrándose en los requisitos funcionales y las principales restricciones. Este modelo se construye a partir del resultado del análisis y constituye el preludio de las actividades de implementación. Está compuesto por varios artefactos como son: el diagrama de paquetes y diagramas de clases del diseño.

Diagrama de Paquetes: los diagramas de paquetes muestran como un sistema está dividido en agrupaciones lógicas, mostrando las dependencias entre ellas. Además brindan una descomposición de la jerarquía lógica y física del sistema. A continuación se muestra la representación del diagrama de paquetes de los subsistemas a desarrollar.

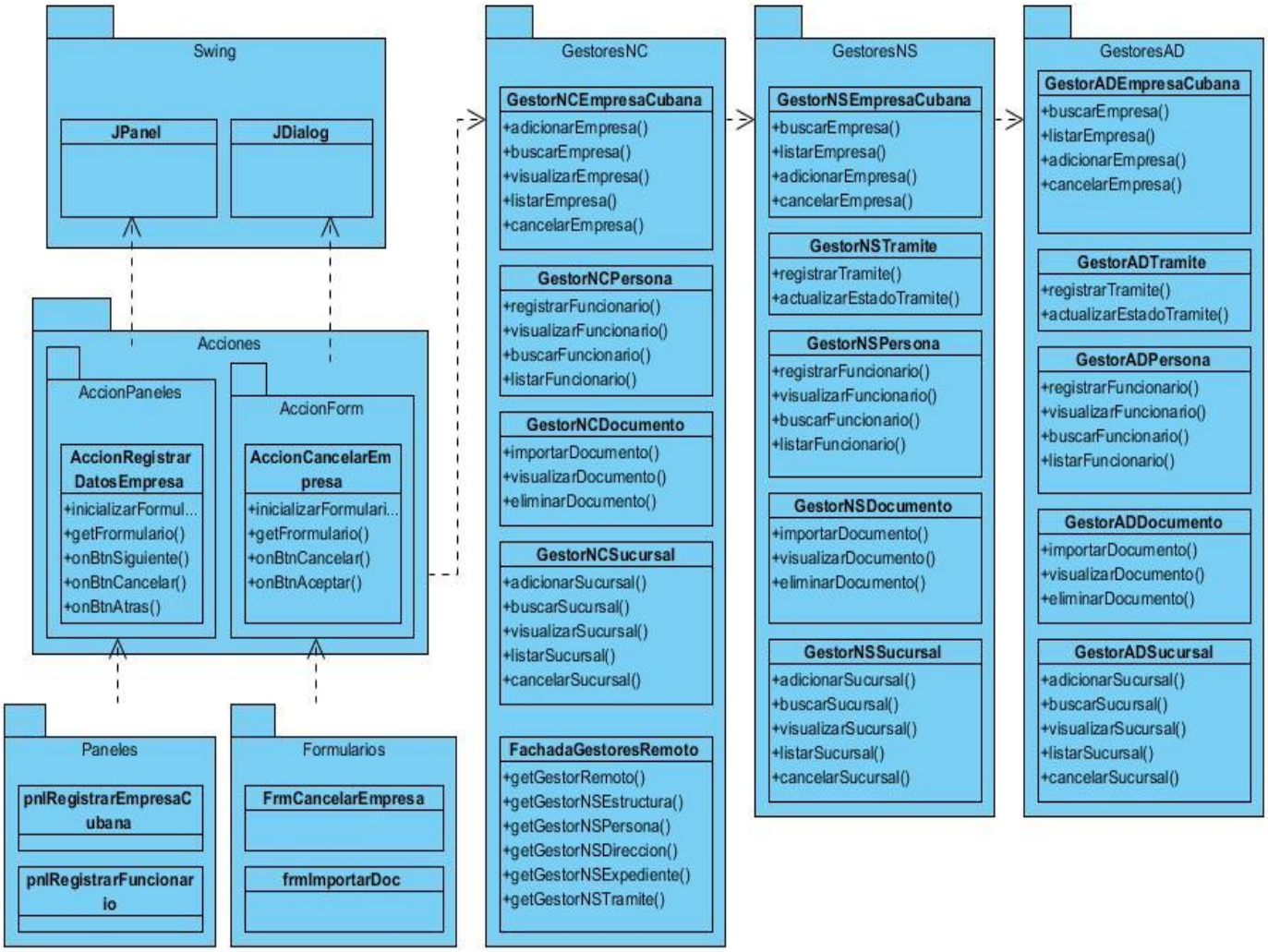

Figura 8: Diagrama de paquetes de los subsistemas.

Diagrama de Clases del Diseño: los diagramas de clases del diseño constituyen una representación estática de los elementos de la solución del sistema, mostrando sus clases, atributos y las relaciones entre ellos. Las clases se representan mediante una caja subdividida en tres partes, en la parte superior se muestra el nombre, en el medio los atributos y en la parte inferior las operaciones.

En primer lugar las acciones validan y recogen los datos los cuales son enviados al gestor de negocio del cliente el cual gestiona la información y le pide a la interfaz fachada de gestores remota comunicación con

el servidor. El gestor de negocio del servidor envía los datos a la capa de acceso a datos y estos son almacenados en las entidades persistentes, no sin antes utilizar las factorías para convertir el objeto de dominio en un objeto de entidad persistente. A continuación se muestra un fragmento del diagrama de clases del diseño del requisito Registrar Datos Empresa Cubana perteneciente al subsistema Afiliación de Empresas Cubanas. Ver anexos 1 y 2.

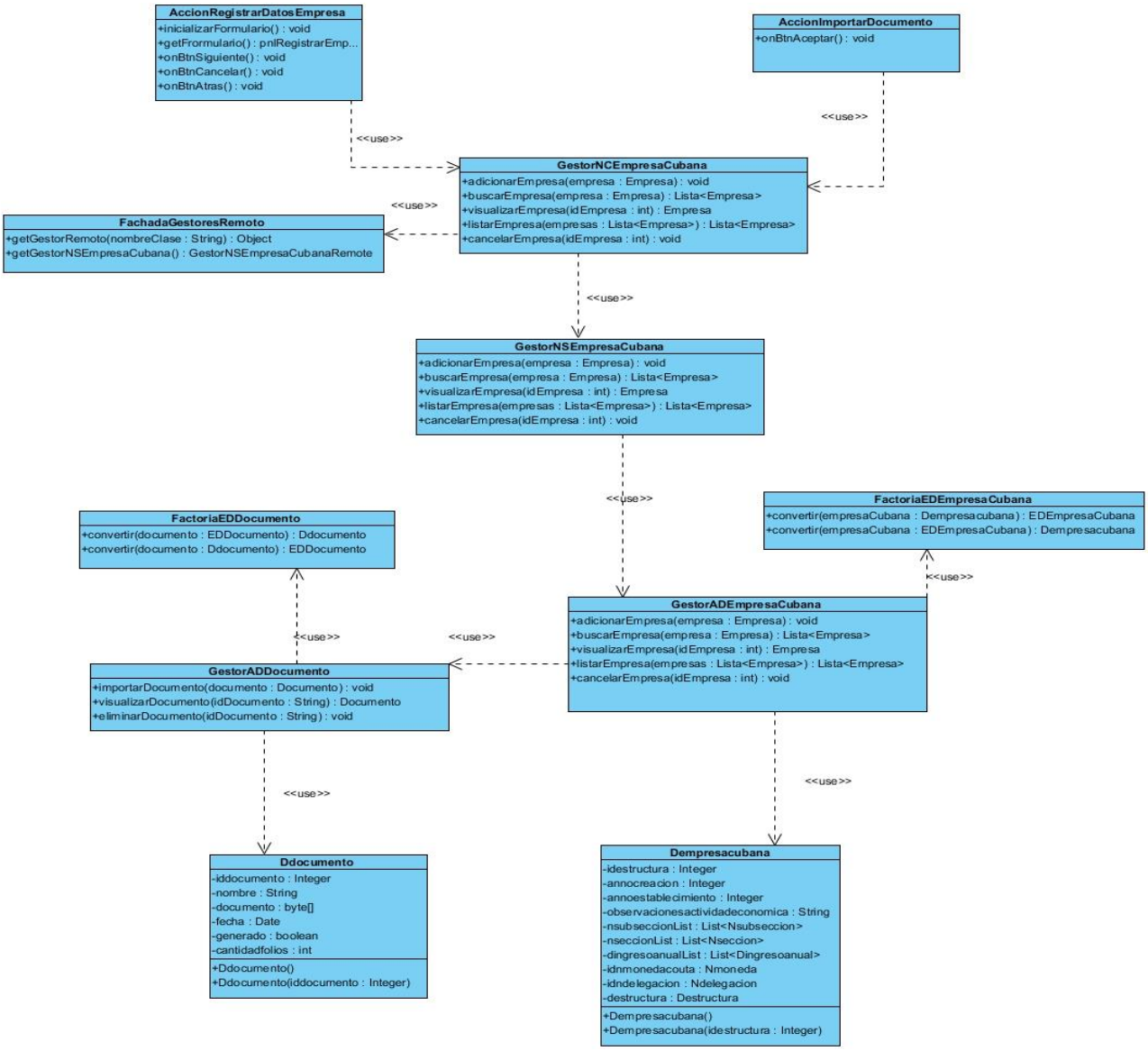

Figura 9: Diagrama de clases del diseño del requisito Registrar Datos de la Empresa.

Diagrama de Secuencia: un Diagrama de Secuencia muestra una interacción ordenada según la secuencia temporal de eventos. En particular, muestra los objetos participantes en la interacción y los mensajes que intercambian ordenados según su secuencia en el tiempo. En el eje vertical la representación del tiempo y en el horizontal es donde se colocan los objetos y actores participantes en la interacción, estos últimos no tienen que tener un orden prefijado pues cada uno tiene una línea vertical hacia donde se establecen los mensajes que se muestran mediante flechas, estableciendo el tiempo de arriba hacia abajo. En la Figura 10 se muestra el diagrama de secuencia correspondiente al requisito Registrar Datos de Empresa Cubana.

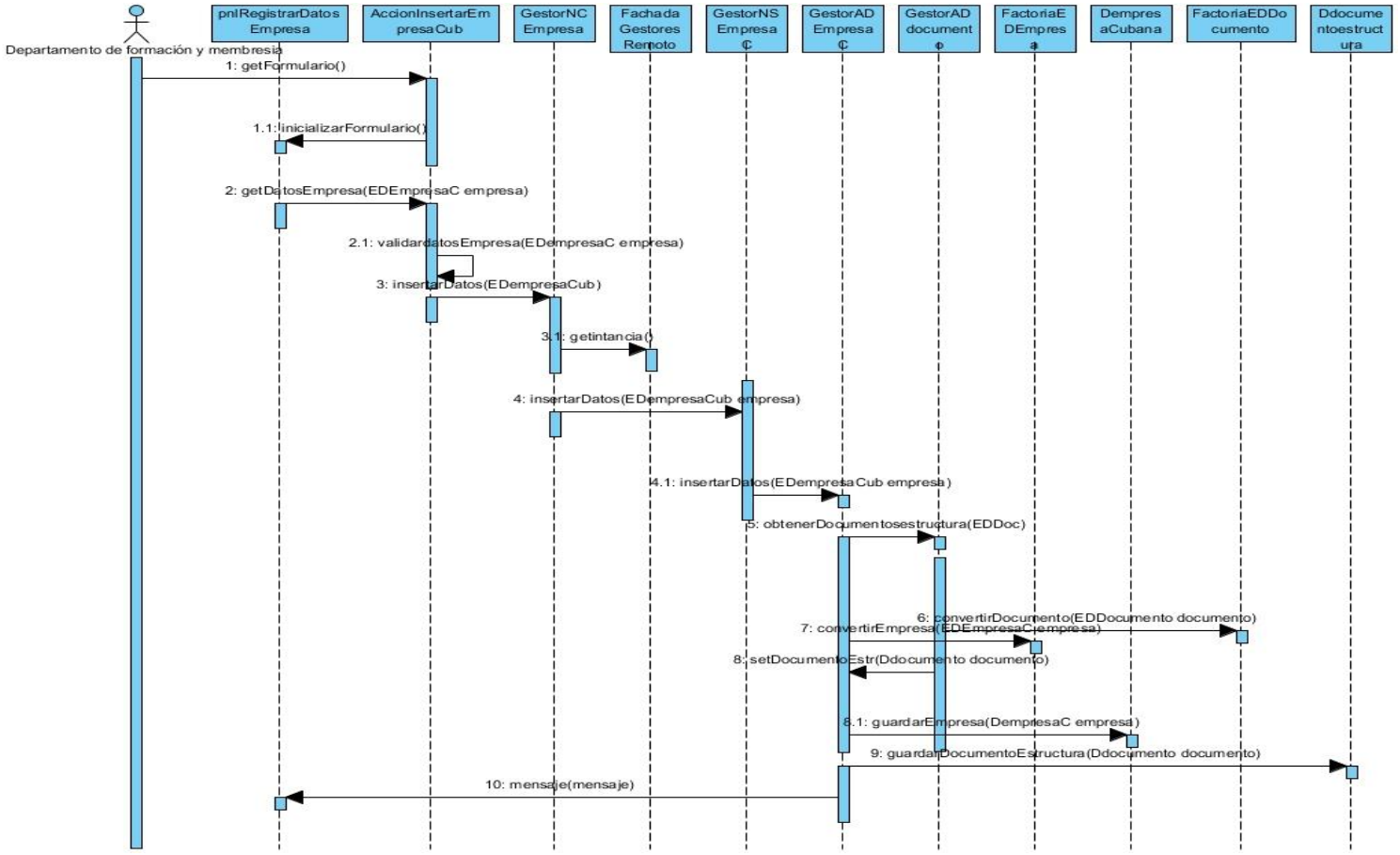

Figura 10: Diagrama de secuencia del requisito Registrar Datos de la Empresa.

Descripción de clases: Para comprender la aplicación es necesario realizar una descripción de las clases. La necesidad de realizarla se contempla en su debida magnitud cuando hay errores que reparar o hay que

expandir la aplicación con nuevas necesidades o adaptarla a nuevas tecnologías. A continuación se describen las clases del subsistema Afiliación de Empresas Cubanas.

Tabla 1: Descripción de las acciones del subsistema.

Nombre: Procesar Información	
Nombre de acción	**Propósito**
`AccionRegistrarDatosEmpresa`	La acción inicializa el formulario y envía los datos de la Empresa Cubana para ser registrados.
`AccionRegistrarDatosFuncionario`	La acción inicializa los componentes necesarios para registrar los datos de una persona y los envía para ser persistidos.
`AccionBuscarFuncionario`	La acción carga en los componentes para insertar los criterios de búsqueda del funcionario.
`AccionImportarDocumento`	La acción carga los componentes donde permite introducir el documento y sus datos.
`AccionModificarDocumento`	La acción carga el formulario donde permite insertar los nuevos valores del documento y así ser insertados.

Tabla 2: Descripción de los Gestores de Negocio del Cliente del subsistema.

Nombre: Procesar Información	
Nombre del gestor	**Propósito**
`GestorNCEmpresaCubana`	Gestiona la información del negocio en el cliente de las Empresas Cubanas registrados en el sistema.
GestorNCPersona	Gestiona la información del negocio en el cliente de las personas registrados en el sistema.

GestorNCDocumento	Gestiona la información del negocio en el cliente de los documentos registrados en el sistema.
FachadaGestoresRemoto	Comunica los gestores del cliente con los del servidor.

Tabla 3: Descripción de los Gestores de Negocio del Servidor subsistema.

Nombre: Procesar Información	
Nombre del gestor	**Propósito**
GestorNSEmpresaCubana	Gestiona la información del servidor de las Empresas Cubanas registradas en el sistema.
GestorNSPersona	Gestiona la información del negocio en el servidor de las personas registradas en el sistema
GestorNSDocumento	Gestiona la información del negocio en el servidor de los documentos que se registran en el sistema

Tabla 4: Descripción de los Gestores de Acceso a Datos del subsistema.

Nombre: Procesar Información	
Nombre del gestor	**Propósito**
GestorADEmpresaCubana	Gestiona la información del registro en la base datos de las Empresas Cubanas registradas en el sistema.
GestorADPersona	Gestiona la información del registro en la base datos de las personas registradas en el sistema.

GestorADDocumento	Gestiona la información del registro en la base datos los documentos que se registran en el sistema.
GestorADDireccion	Gestiona la información del registro en la base datos las direcciones que se registran en el sistema.

2.6 Modelo de datos

El modelo de datos constituye la representación abstracta de los datos de los objetos y las relaciones entre ellos. El objetivo con la creación del modelo de datos es escenificar la base de datos, para lograr comprender su estructura. Los modelos de datos se expresan en tres términos: las entidades, que manejan la información a persistir; los atributos, que son las características básicas de los objetos y relaciones, que son las que enlazan a dichos objetos entre sí.

El modelo de datos correspondiente a los subsistemas Afiliación de Empresas Cubanas y Sucursales Extranjeras está compuesto por 58 tablas, de ellas 32 nomencladores y las restantes para guardar la información correspondiente a los subsistemas a desarrollar. La creación y nombramiento de las tablas se realizará de la siguiente manera:

- El nombre de las tablas nomencladoras comenzará con la letra n. Ejemplo: `nEmpresaCubana`.
- El nombre de las tablas de datos comenzará con la letra d. Ejemplo: `dDocumento`. Las especializaciones se nombrarán: nombre de la tabla padre + nombre de la tabla hija.
- Para la notación de las llaves primarias se utilizará el siguiente patrón: `Id` + nombre de la tabla.

A continuación se muestra un fragmento del modelo de datos de los subsistemas a desarrollar por la presente investigación.

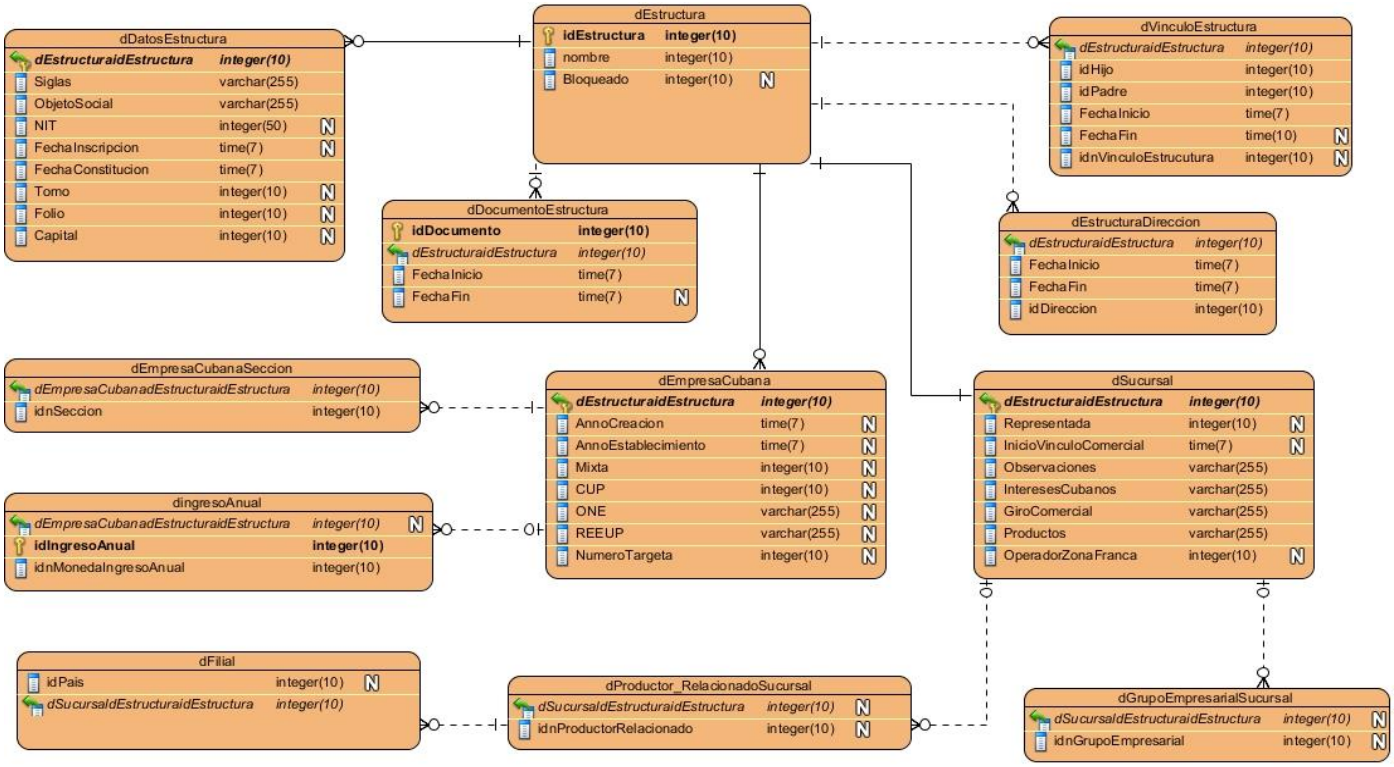

Figura 11: Fragmento del modelo de datos de los subsistemas.

2.7 Modelo de implementación

El modelo de implementación toma el resultado del modelo del diseño y lo transforma en el código final. Detalla la organización de los componentes de acuerdo a los mecanismos de estructuración, el lenguaje de programación utilizados y cómo dependen los componentes unos de otros.

Diagrama de componentes

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes de software, sean estos componentes de código fuente, binarios o ejecutables, por este motivo en muchos aspectos se puede considerar que un diagrama de componentes es un diagrama de clases a gran escala. Estos se representan como un grafo de componentes de software unidos por medio de relaciones de dependencia.

La figura 12 representa el diagrama de componentes, organizado de acuerdo a la arquitectura seleccionada.

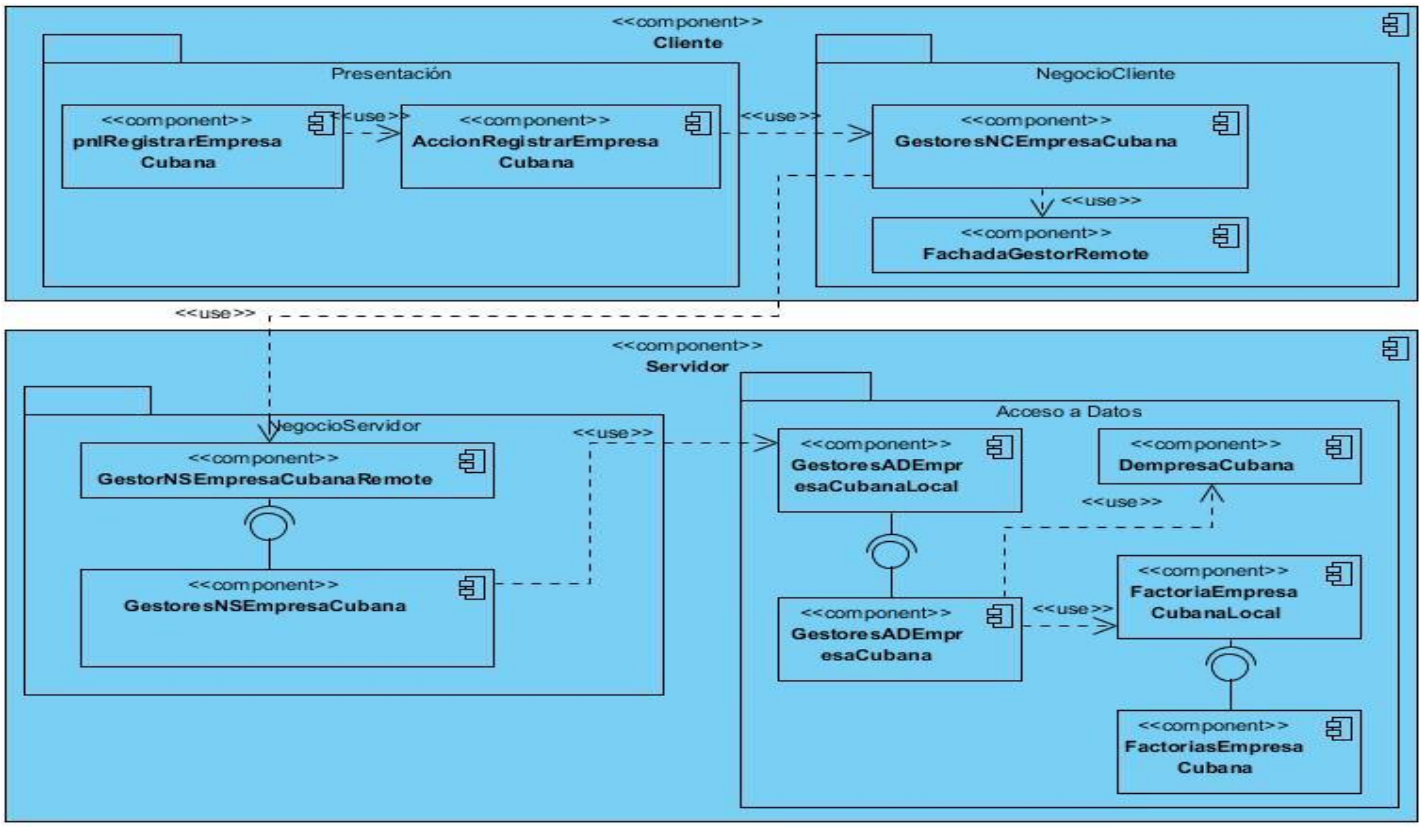

Figura 12: Diagrama de componentes del requisito Insertar Datos de Empresa Cubana.

Diagrama de despliegue

El modelo de despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. El propósito del modelo de despliegue es capturar la configuración de los elementos de procesamiento y las conexiones entre estos en el sistema. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria, los que se precisan a través de estereotipos que definen la naturaleza del equipo. En la figura 13 se muestra el diagrama de despliegue que se propone para los subsistemas Afiliación de Empresas Cubanas y Sucursales Extranjeras.

La distribución física del sistema estará compuesta por una PC-cliente en la cual estará instalada la máquina virtual de Java donde correrá la aplicación en su lado cliente. A la PC-cliente se le conectará por USB[4] una impresora o escáner para imprimir o escanear documentos respectivamente. Se contará con un servidor donde estará el Sistema de Gestor de Base Datos y el Servidor de Aplicaciones donde estará desplegada la aplicación empresarial que se encargará de gestionar las respuestas a las peticiones del cliente usando el protocolo TCP-IP[5]. El asistente se apoyará en el Servidor de Almacenamiento para realizar salvas como puede observar.

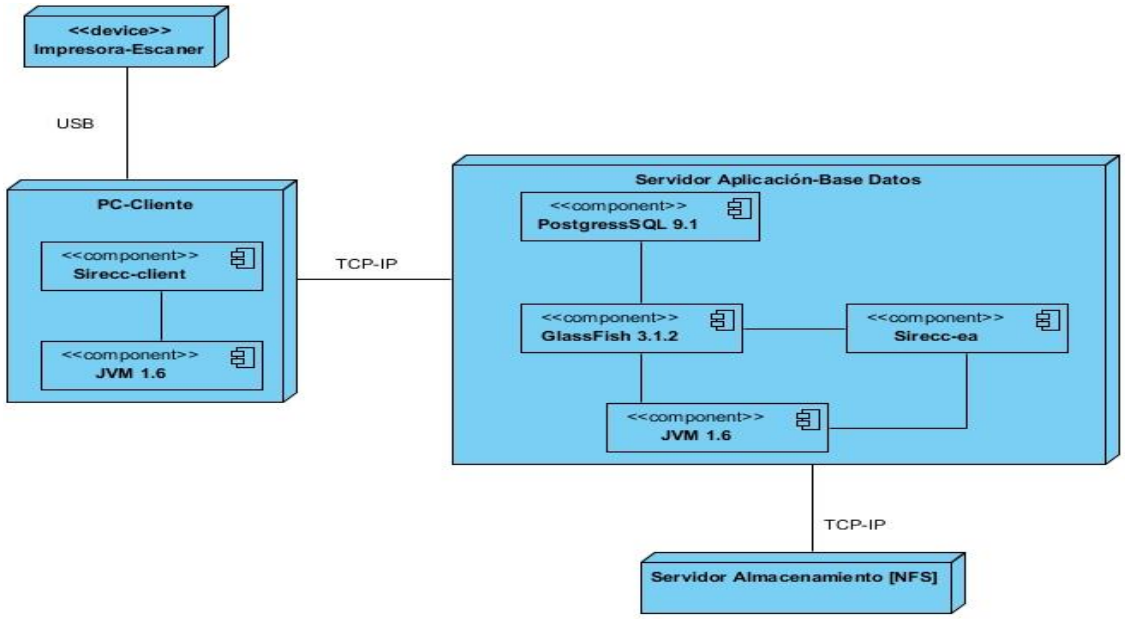

Figura 13: Diagrama de despliegue de los subsistemas Afiliación de Empresas Cubanas y Sucursales Extranjeras.

2.8 Estándares de codificación

Con el objetivo de lograr un enfoque uniforme en la implementación de la solución y que el código nuevo sea entendible rápidamente se utilizaron estándares de codificación, a continuación se evidenciarán sus usos:

[4] El Universal Serial Bus (USB).
[5] TCP/IP es un modelo de descripción de protocolos de red.

Nomenclatura General:

- Se exceptúan el uso de las tildes y la letra ñ, la que será sustituida por nn.
- En todo momento se utilizarán nombres que sean claros, concretos y libres de ambigüedades. Ejemplo: "`fechaCreacion`" y no solamente "`fecha`".
- El nombre de todas las variables y métodos comenzarán con letra minúscula y si este está compuesto por varias palabras se utilizará el estilo de escritura *lowerCamelCase*, que rige que para un nombre compuesto por varias palabras comenzará con minúscula pero todas las palabras internas que lo componen comienzan con mayúscula.

Nombres de ficheros: los nombres de fichero describen el uso o propósito de la clase y utilizan el estilo de escritura *UpperCamelCase* ejemplo: `GestorADEmpresaCubana` (Gestor de Acceso a Datos de la Empresa Cubana). Los ficheros tienen el siguiente orden:

- Comentarios de comienzo.
- Sentencias `package` e `import`.
- Declaraciones de clases e interfaces.

La siguiente lista describe las partes de la declaración de una clase o interface, en el orden en que deberían aparecer.

- Comentario de documentación de la clase o interface (/**...*/).
- Sentencia clases o interfaces.
- Comentario de implementación de la clase o interface si fuera necesario (/*...*/): este comentario debe contener cualquier información aplicable a toda la clase o interface que no era apropiada para estar en los comentarios de documentación de la clase o interface. Variables de clase (`static`): Primero las variables de clase `public`, después las `protected`, posteriormente las de nivel de paquete (sin modificador de acceso), y finalmente las `private`.
- Variables de instancia: Primero las `public`, después las `protected`, posteriormente las de nivel de paquete (sin modificador de acceso), y por último las `private`.
- Constructores

Indentación: se deben emplear cuatro espacios como unidad de indentación. La construcción exacta de la indentación (espacios en blanco contra tabuladores) no se especifica. Los tabuladores deben ser exactamente cada 8 espacios.

Longitud de la línea: evitar las líneas de más de 70 caracteres, cuando una expresión no entre en una línea, romperla de acuerdo con estos principios: romper después de una coma o antes de un operador.

Declaraciones: se realiza una declaración por línea debido a que facilita a la hora de comentar. Se inicializa la variable locales donde se declaran y se colocan al principio de los bloques.

Métodos: los métodos deben ser verbos, cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula.

Variables: excepto las constantes que todos sus caracteres son en mayúscula, todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas (Hommel, 1999).

2.9 Conclusiones parciales

Luego del desarrollo del presente capitulo se obtuvo como resultado:

- Los modelos que fueron la base utilizada para dar cumplimiento de manera satisfactoria a la implementación del sistema.
- Se obtuvieron los elementos esenciales del modelo de diseño a partir de las especificaciones de los requisitos funcionales, dentro de los que se encuentran diagrama de clases del diseño, diagramas de paquetes, diagramas de secuencia y diagrama de despliegue, en los cuales se evidencia el uso de patrones que permiten obtener un producto funcional.
- Se definieron un conjunto de estándares de codificación, que brindaron la posibilidad de una implementación clara y libre de ambigüedades.

CAPÍTULO 3. VALIDACIÓN DE LA SOLUCIÓN

3.1 Introducción.

Durante el desarrollo del software es común la aparición de errores debido a que en muchas ocasiones se programan rutinas o funciones y se obvian algunas posibilidades que pueden variar el resultado esperado durante la ejecución del código, por lo que hay que tratar de asegurar que no suceda, aplicando pruebas y validaciones de software. Los errores pueden suceder a causa del mal uso de estructuras de datos, errores de integración de componentes y errores lógicos. En el presente capítulo se evidencia el resultado de las pruebas y validaciones realizadas a los diferentes subsistemas, que permiten garantizar el correcto funcionamiento de la totalidad de sus requerimientos.

3.2 Medición de la calidad del diseño utilizando métricas.

Para el aseguramiento y control de calidad de los artefactos generados en el diseño se utilizan las métricas de software. En el siguiente epígrafe se validará el diseño de los subsistemas mediante las métricas con las que se podrá medir el estado de algunos atributos de calidad que permiten brindar criterios valorativos sobre la realización del diseño.

Los atributos de calidad que se engloban con la realización de estas métricas son:

Cohesión o Responsabilidad: consiste en el grado de responsabilidad asignada a una clase diseñada en un modelo de dominio específico.

Complejidad de implementación: un aumento del TOC (Tamaño Operacional de las Clases) implica un aumento de la complejidad de implementación de la clase.

Complejidad de mantenimiento: consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costos y la planificación del proyecto.

Acoplamiento: consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización.

Reutilización: consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.

Cantidad de Pruebas: Consiste en el grado de fuerza para realizar las pruebas de calidad de los subsistemas diseñados (Camacho, 2004).

3.2.1 Tamaño Operacional de las Clases.

Consiste en medir el tamaño general de una clase tomando el valor de la cantidad de operaciones que están encapsuladas dentro de dicha clase. Si el resultado obtenido indica valores grandes, significa que la clase posee un alto grado de responsabilidad, debido a esto se reducirá la reutilización, se hará mucho más difícil la implementación y la realización de pruebas de dicha clase.

Por tanto mientras menor sea el valor para el TOC se hará mucho más fácil la reutilización de dicha clase dentro del sistema. Los desarrolladores de esta métrica, dejan el esquema de peso, como una decisión de implementación. En la presente investigación se tomó como muestra el subsistema Afiliación de Empresas Cubanas, bajo los criterios de bajo, medio y alto; calculado de la siguiente forma:

A: alto M: medio B: bajo

P: promedio de operaciones que poseen las clases de la muestra

O: cantidad de operaciones de una clase.

Tabla 5 Umbrales para los atributos de calidad.

Atributos		Categoría	Umbral
Responsabilidad		B	O<=P
		M	P < O < 2P
		A	O > 2P

Complejidad de implementación		B	O<=P
		M	P < O < 2P
		A	O > 2P
Reutilización		B	O > 2P
		M	P < O < 2P
		A	O<=P

A partir de la aplicación de dicha métrica a las clases de la solución, se arrojaron los siguientes resultados.

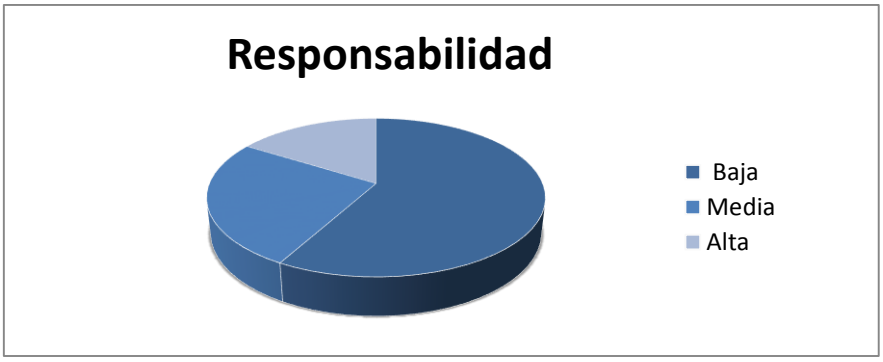

Figura 14: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad.

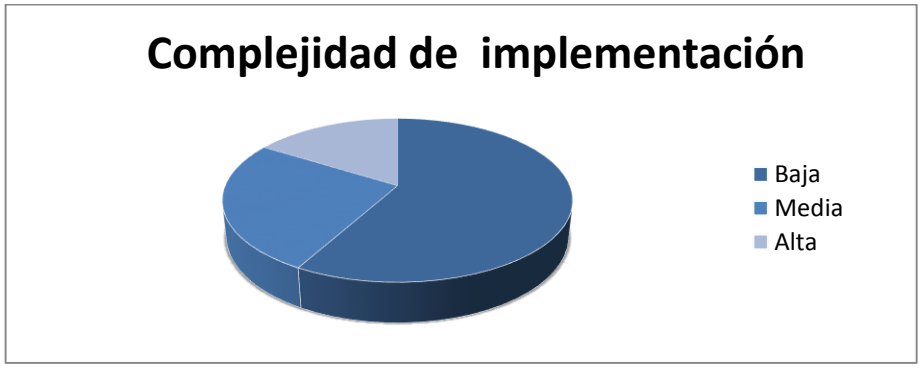

Figura 15: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de implementación.

Figura 16: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización.

Con la métrica anteriormente aplicada se puede concluir que el diseño de los subsistemas de Afiliación de Empresas Cubanas y Sucursales Extranjeras presentan una buena calidad, pues tuvo resultados satisfactorios en los diferentes atributos de calidad medidos, por lo que se puede afirmar que el diseño realizado fue correcto, con el cual se obtuvieron altos niveles de reutilización (64%), bajos niveles de responsabilidad (64%) y complejidad de implementación (64%), pues se le asignaron correctamente las responsabilidades a las clases involucradas en la solución.

3.2.2 Relaciones entre Clases

El resultado de la aplicación de la métrica Relaciones entre Clases (RC) está dado por el número de relaciones de uso de una clase con otras. La aplicación de dicha métrica permite evaluar atributos como el acoplamiento, la complejidad de mantenimiento y la cantidad de pruebas. Después de aplicar dicha métrica se obtuvieron los siguientes resultados:

A: alto M: medio B: bajo N: ninguno

P: promedio de relaciones que poseen las clases de la muestra.

O: cantidad de relaciones de una clase.

Tabla 6: Umbrales de atributos de calidad.

Atributos	Categoría	Umbral

Acoplamiento	N	O=0
	B	O=1
	M	O = 2
	A	O>=3
Complejidad de mantenimiento	B	O<=P
	M	P < O < 2P
	A	O > 2P
Cantidad de pruebas	B	O > 2P
	M	P < O < 2P
	A	O<=P

Para el atributo Acoplamiento se tienen en cuenta la cantidad de dependencias de una clase con otras, mientras que Complejidad de mantenimiento y Cantidad de pruebas se mide por la fórmula planteada anteriormente.

Figura 17: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento.

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

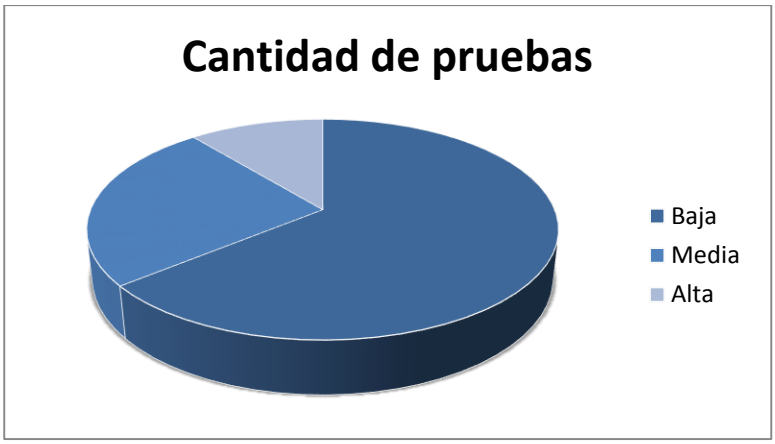

Figura 19: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.

La aplicación de esta métrica de software puntualizó que existe una baja dependencia entre las diferentes clases de la solución propuesta, además se demostró que dichas clases poseen un bajo acoplamiento lo que posibilita reutilizar los componentes. También arrojó resultados satisfactorios en el atributo complejidad de mantenimiento lo que facilita las tareas de corrección, modificación y mantenimiento de los componentes.

3.3 Pruebas de caja blanca

En la validación de la solución de los subsistemas, se realizó la prueba del camino básico a las funcionalidades de mayor complejidad, esta permite definir los diferentes caminos independientes de la

función y probar su funcionamiento al menos una vez. A continuación se muestra el proceso desarrollado al método "`insertarDirecciones`" de la clase "`GestorADDireccion`".

```
@Override
public List<Object> insertarDirecciones(List<EDDireccion> list, Object objetoAsociarDireccion) {
    List<Ddireccion> dList = new LinkedList<Ddireccion>();
    List<Object> result = new LinkedList<Object>();

    for (EDDireccion eDDireccion : list) {
        dList.add(insertarDireccion(eDDireccion));
    }
    if (objetoAsociarDireccion instanceof Dpersona) {
        Dpersonadireccion pd;
        for (Ddireccion d : dList) {
            pd = new Dpersonadireccion();
            pd.setIddireccion(d);
            pd.setFechainicio(Calendar.getInstance().getTime());
            pd.setIdpersona(((Dpersona) objetoAsociarDireccion));
            manager.persist(pd);
            result.add(pd);
        }
    } else if (objetoAsociarDireccion instanceof Destructura) {
        Destructuradireccion ed;
        for (Ddireccion d : dList) {
            ed = new Destructuradireccion();
            ed.setIddireccion(d);
            ed.setFechainicio(Calendar.getInstance().getTime());
            ed.setIdestructura((Destructura)objetoAsociarDireccion);
            manager.persist(ed);
            result.add(ed);
        }
    }
    return result;
}
```

Figura 20: Método `insertarDirecciones` de la clases `GestorADDireccion`.

Seguidamente es necesario representar el grafo de flujo (o grafo del programa), que constituye el flujo de control lógico del código anterior, a través de nodos, aristas y regiones. Como se muestra en la Figura 21.

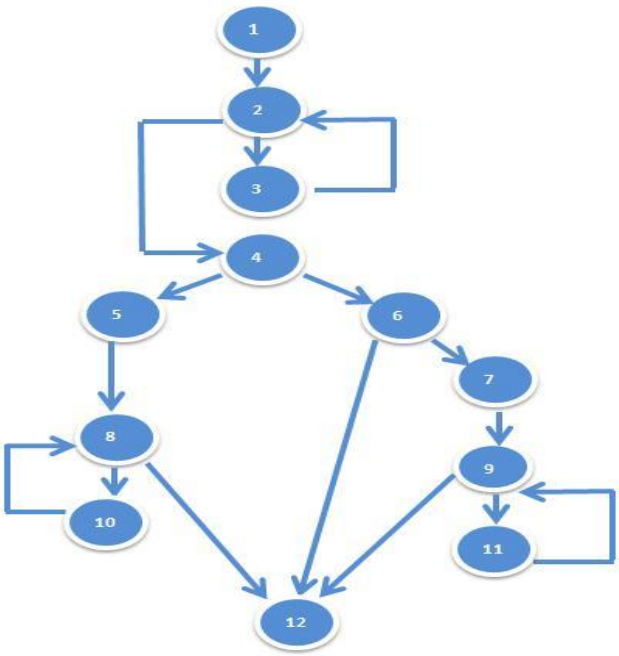

Figura 21: Grafo del flujo del código de la función de la figura anterior.

Luego de realizado el grafo es necesario conocer la cantidad de caminos independientes que se deben buscar para probar, y para esto se calcula la complejidad ciclomática, que se puede calcular de tres formas:

- El número de regiones del grafo de flujo coincide con la complejidad ciclomática.
  Por lo que V (G) = **6**
- V (G) = (A – N) + 2 donde "**A"** es el número de aristas del grafo de flujo y "**N**" es el número de nodos del mismo. V (G) = 16 – 12 +2 = **6**
- V (G) =P+ 1 donde "**P**" es el número de nodos predicado contenidos en el grafo. Los nodos 2 y 3 son nodos predicados. V (G) = 5+1 = **6**

Como se muestra en cualquiera de las tres formas anteriores la complejidad ciclomática es **6**, por lo que la cantidad de caminos básicos que puede tomar el algoritmo durante su ejecución es **6** y quedan definidos de la siguiente forma:

Camino básico #1: 1 -- 2 -- 4 -- 5 -- 8 -- 12

Camino básico #2: 1 -- 2 -- 3 -- 2 -- 4 -- 5 -- 8 -- 10 -- 8 -- 12

Camino básico #3: 1 -- 2 -- 4 -- 6 -- 7 -- 8 -- 12

Camino básico #4: 1 -- 2 -- 3 -- 2 -- 4 -- 6 -- 7 -- 8 -- 11 -- 8 -- 12

Camino básico #5: 1 -- 2 -- 4 -- 6 -- 12

Camino básico #6: 1 -- 2 -- 3 -- 2 -- 4 -- 6 -- 12

Seguidamente se preparan los casos de pruebas que forzarán la ejecución de cada camino del conjunto básico, quedando de la siguiente forma:

3.3.1 Casos de pruebas.

Descripción: este método se encarga de persistir en la base de datos la relación entre un objeto, ya sea de `Dpersona` o `Destructura` con sus respectivas direcciones, y retorna una lista de objetos de la relación persistida.

Caso de prueba para el camino básico # 1. Condición de ejecución: para esta prueba es necesario que la variable `List<EDDireccion> list` esté vacía y `Object objetoAsociarDireccion` sea de tipo `Dpersona`.

Resultados esperados: teniendo en cuenta la condición de ejecución se espera que recorra el camino #1 y devuelva una lista vacía.

El resultado obtenido fue correcto.

Caso de prueba para el camino básico # 2. Condición de ejecución: para esta prueba es necesario que la variable `List<EDDireccion> list` no esté vacía y `Object objetoAsociarDireccion` sea de tipo `Dpersona`.

Resultados esperados: teniendo en cuenta la condición de ejecución se espera que recorra el camino #2 y devuelva una lista de objetos de la relación entre persona y dirección.

El resultado obtenido fue correcto.

Caso de prueba para el camino básico # 3. Condición de ejecución: para esta prueba es necesario que la variable `List<EDDireccion> list` esté vacía y `Object objetoAsociarDireccion` sea de tipo `Destructura`.

Resultados esperados: teniendo en cuenta la condición de ejecución se espera que recorra el camino #3 y devuelva una lista vacía.

El resultado obtenido fue correcto.

Caso de prueba para el camino básico # 4. Condición de ejecución: para esta prueba es necesario que la variable `List<EDDireccion> list` no esté vacía y `Object objetoAsociarDireccion` sea de tipo `Destructura`.

Resultados esperados: teniendo en cuenta la condición de ejecución se espera que recorra el camino #4 y devuelva una lista de objetos de la relación entre estructura y dirección.

El resultado obtenido fue correcto.

Caso de prueba para el camino básico # 5. Condición de ejecución: para esta prueba es necesario que la variable `List<EDDireccion> list` esté vacía y `Object objetoAsociarDireccion` no sea ni `Destructura`, ni `Dpersona`.

Resultados esperados: teniendo en cuenta la condición de ejecución se espera que recorra el camino #5 y devuelva una lista vacía.

El resultado obtenido fue correcto.

Caso de prueba para el camino básico # 6. Condición de ejecución: para esta prueba es necesario que la variable `List<EDDireccion> list` no esté vacía y `Object objetoAsociarDireccion` no sea ni `Destructura`, ni `Dpersona`.

Resultados esperados: teniendo en cuenta la condición de ejecución se espera que recorra el camino #6 y devuelva una lista vacía.

El resultado obtenido fue correcto.

Al aplicar las pruebas a los algoritmos con mayor complejidad dentro de cada uno de los subsistemas se detectaron un total de 5 errores. Los principales errores consistían en variables no inicializadas, los cuales fueron corregidos y probados nuevamente asegurando su correcto funcionamiento, lo que demuestra que el código cumple con los caminos básicos de su ejecución dando la posibilidad de responder a las situaciones a las que se va a enfrentar.

3.4 Pruebas de caja negra

Las pruebas de caja negra se centran en las interfaces del software donde se especializan en la detección de errores mediante casos de pruebas. Los errores a detectar se agrupan en las siguientes categorías:

- Funciones incorrectas o ausentes
- Errores de interfaz

Para seleccionar el conjunto de entradas y salidas sobre las que trabajar, se tiene en cuenta que en todo programa existe un conjunto de entradas que causan un comportamiento erróneo en el sistema, y como consecuencia producen una serie de salidas que revelan la presencia de defectos .Con la aplicación de los casos de pruebas (ver en los artefactos Casos de pruebas Empresa Cubana y Casos de pruebas Sucursales Extranjeras entregables) basados en requisitos se realizaron tres iteraciones donde en la última se mitigaron los errores detectados. A continuación se muestran los resultados de cada una de las iteraciones y en el Anexo 4 se encuentra un ejemplo de los casos de prueba.

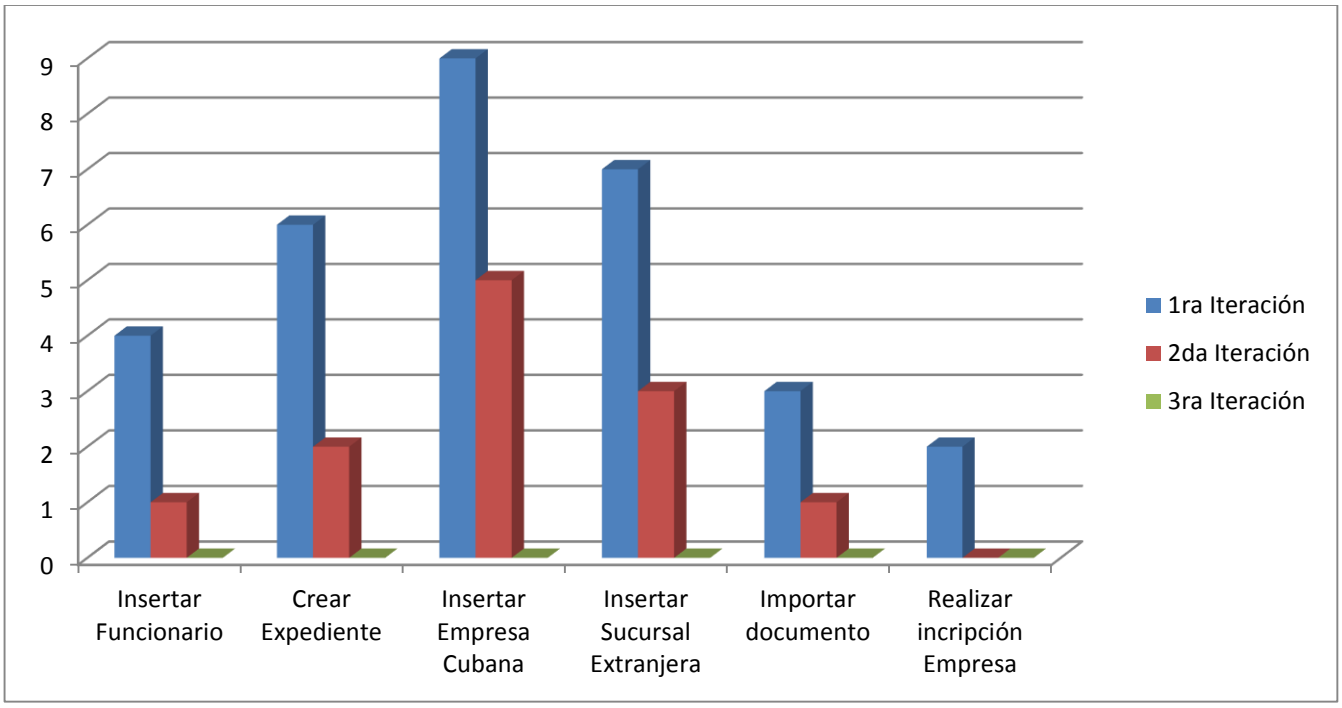

Figura 22: Cantidad de no conformidades detectadas a los requisitos por iteración.

3.5 Pruebas de unidad

Con el objetivo de corregir errores en la menor unidad del software, se realizaron las pruebas de unidad. Las pruebas de unidad son una manera de manejar los elementos de prueba combinados, puesto que se centra la atención inicialmente en unidades más pequeñas del programa.

Durante la realización del proceso de pruebas se decidió aplicar pruebas unitarias a los algoritmos que presentan una complejidad considerable dentro de cada uno de los subsistemas. Para la realización de estas pruebas se usó JUnit 4.0, el cual es una herramienta que se asocia al NetBeans para permitir la ejecución de este proceso. A continuación se muestran las pruebas realizadas al algoritmo `buscarEmpresasCubanas` perteneciente al subsistema Afiliación de Empresas Cubanas:

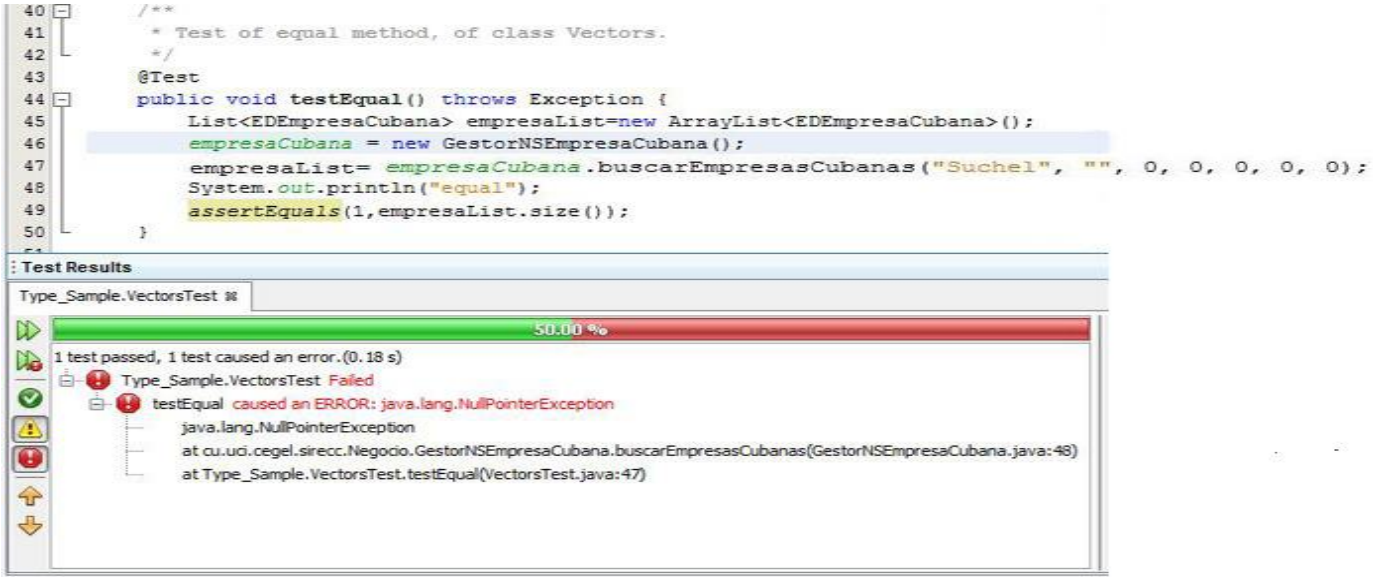

Figura 23: Resultado de la primera iteración de la prueba de unidad para el algoritmo buscarEmpresasCubanas.

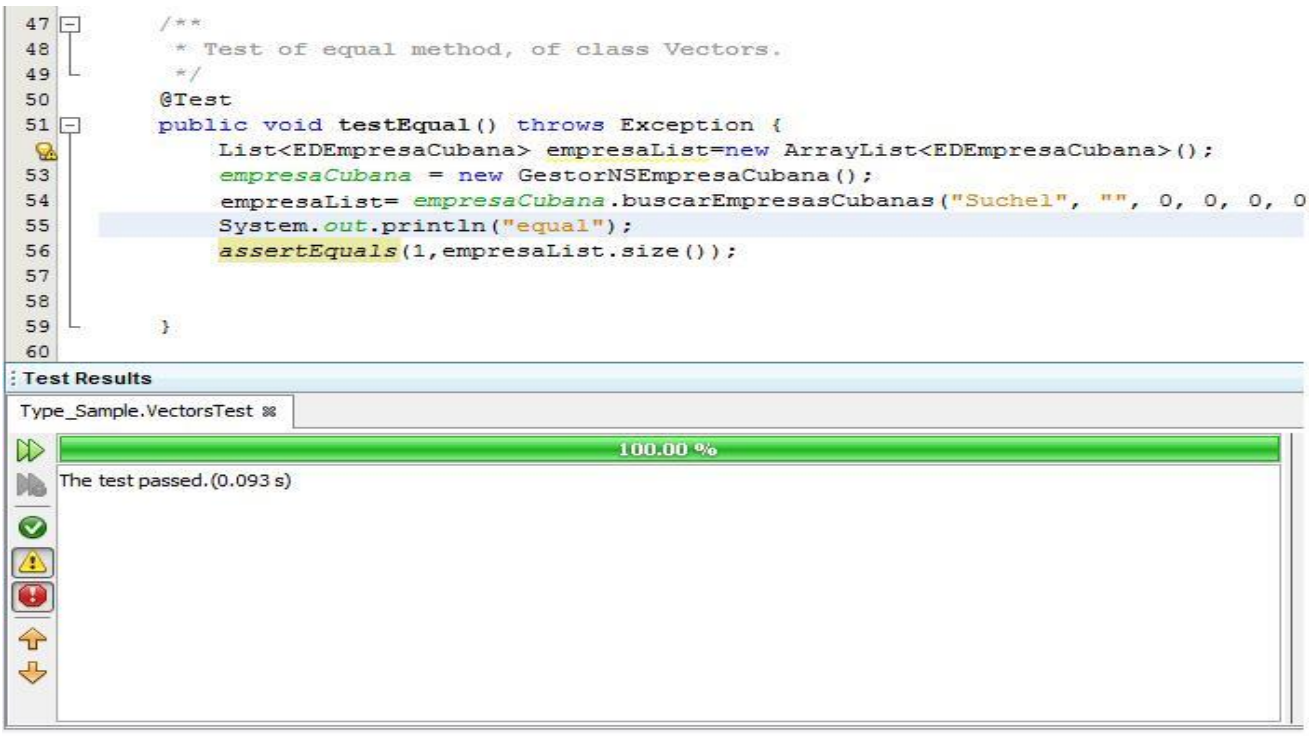

Figura 24: Resultado de la segunda iteración de la prueba de unidad para el algoritmo buscarEmpresasCubanas.

Con la aplicación de las pruebas de unidad se detectaron un total de cuatro errores, por lo que fue necesario corregirlos y realizar una segunda iteración de pruebas, donde se mitigaron todos los problemas existentes, con lo cual se comprobó el buen funcionamiento de los algoritmos implicados en dichos errores como se puede apreciar en la figura 25.

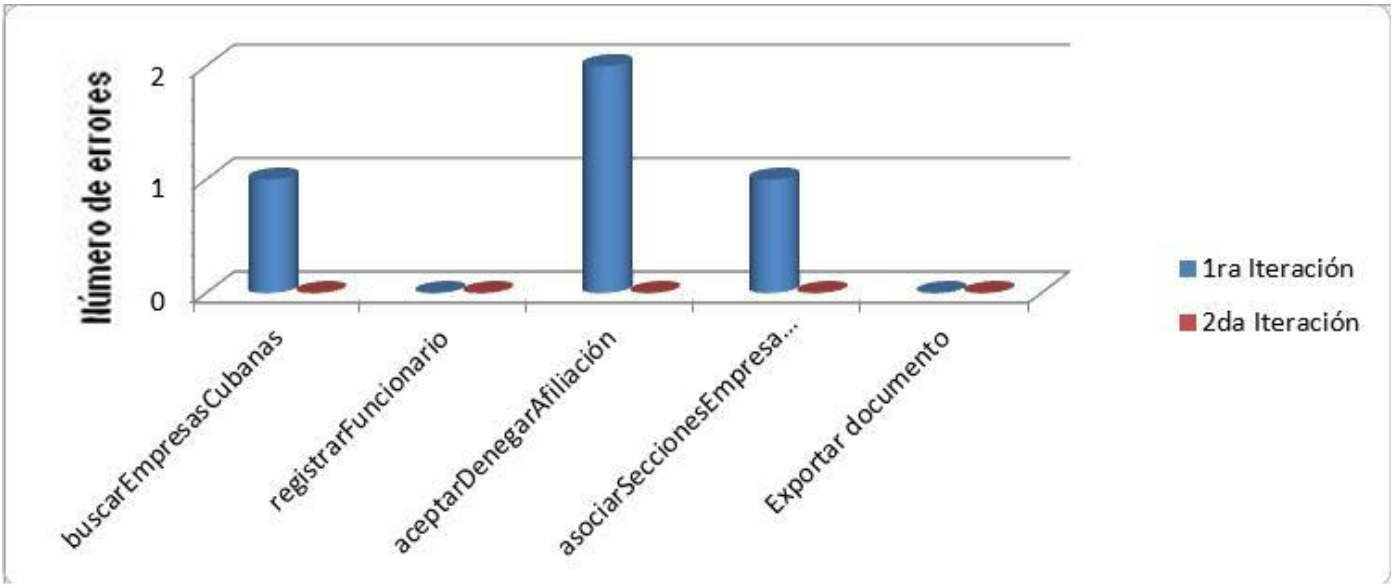

Figura 25: Resultados de la aplicación de las pruebas unitarias.

3.6 Validación de seguridad y usabilidad

Para validar el objetivo general planteado es necesario que los subsistemas presenten un alto grado de usabilidad para que el usuario se sienta a gusto con el uso de los subsistemas, además de que se cuente con un conjunto de aspectos que permitan garantizar la seguridad de los mismos.

3.6.1 Usabilidad

Para validar la seguridad y usabilidad en los subsistemas se utilizará el método de Juicios de Expertos el cual consiste en la selección de un grupo de personas calificadas en el tema desarrollado y formular un cuestionario, que es aplicado para obtener un consenso a partir de las valoraciones subjetivas que realicen cada uno de ellos de manera individual. El conjunto de opiniones que se obtiene de la consulta es sometido

a un procesamiento estadístico que genera una cuantificación sobre las que se pueden dar criterios acerca de las variables definidas.

Para aplicar el proceso planteado anteriormente se seleccionaron un conjunto de expertos pertenecientes a CEGEL, los cuales cuenten con experiencias en desarrollos de sistemas empresariales principalmente aplicaciones de escritorio.

El siguiente paso es la elaboración del cuestionario el cual se dirige a indagar el comportamiento previsible de la usabilidad y la seguridad del registro de los procesos en el cual se elaboraron 9 preguntas. Ver Anexo 5. Las respuestas son valoradas de la siguiente manera: Muy adecuado (P1), Bastante adecuado (P2), Adecuado (P3), Poco adecuado (P4), Nada adecuado (P5).

Tabla 7: Evaluación por preguntas de los expertos.

	1	2	3	4	5	6	7	8	9
P1	1					1	1		1
P2	2	1	3	2	2	3	2	1	1
P3	2	3	2	3	1	1	2	3	3
P4		1			2			1	
P5									

Como se puede observar los resultados obtenidos tuvieron su principal peso en la valoración Adecuado y Bastante adecuado. Con el objetivo de lograr una mayor validez de los resultados se verifica el grado de concordancia entre los expertos donde se utiliza el coeficiente de Kendall (W).

En la prueba estadística el Coeficiente de Concordancia de Kendall (W), ofrece el valor que posibilita conocer el nivel de concordancia entre los expertos. El valor de W oscila entre 0 y 1. El valor de 1 significa una concordancia de acuerdos total y el valor de 0 un desacuerdo total. La tendencia a 1 es lo deseado pudiéndose realizar nuevas rondas si en la primera no es alcanzada una significación en la concordancia.

$$(\mathbf{A}).\ W = \frac{12*S}{K^2\,(N^3+N)}$$

Dónde: W: es el coeficiente de concordancia. S: es la suma de los cuadrados de las desviaciones observadas de la media de Sj (rangos) y se calcula mediante la expresión:

$$(\mathbf{B}).\ S = \sum_{j=1}^{n} (Sj - \overline{S})^{2}$$

Dónde: $\overline{S}$ es la suma de los rangos dividido entre la cantidad de preguntas. N: cantidad de preguntas en este caso 9. K: Cantidad de expertos, por tanto K=5.

$$(\mathbf{C}).\ \overline{S} = \frac{\sum_{j=1}^{n} Sj}{N}$$

Tabla 8: Datos obtenidos de las encuestas.

	1	2	3	4	5	6	7	8	9
P1	3	4	4	3	3	3	4	4	3
P2	4	3	3	3	4	4	3	3	4
P3	3	2	4	4	4	4	3	2	5
P4	5	3	4	4	2	5	4	3	3

P5	4	3	3	3	2	4	5	3	3
Sj	19	15	18	17	15	20	19	15	18

Sustituyendo los valores en la ecuación (C), se calcula el valor de $\overline{S}$ y se obtiene como resultado:

$\overline{S} = \frac{156}{9} = 17$

Para calcular la desviación media (S) se sustituye los valores en la ecuación (B) y se obtiene como resultado:

$S = 31$

Finalmente se sustituyen los valores en la ecuación (A), donde se obtiene el valor de Kendall: $W = (9 * 31)/5^2(9^3 + 9)$

W = 0,01512

Con el coeficiente de Kendall (W) se hace necesario conocer significancia, para lo cual se debe trasformar el valor en Chi cuadrado de Pearson mediante la siguiente fórmula:

$X^2_{cal} = K(N-1)w$

Al sustituir los valores se obtiene:

$X^2 = 5(9 - 1)0{,}01512$
$X^2 = 0{,}6048$

Para buscar el Chi cuadrado en la tabla de distribución reflejada en el Anexo 6, se procede a calcular el grado de libertad (gl) donde: gl = N-1

Sustituyendo los valores en la ecuación se obtiene: gl = 9-1= 8

$X^2_{tabla} = X^2_{\alpha,gl}$ donde α es el nivel de significación utilizado para calcular el nivel de confianza. El nivel de confianza es igual a 100(1- α), un α=0,05 indica un nivel de confianza de 95 %. Una vez obtenidos los

valores se compara $X^2_{tabla} = X^2_{real}$, si $X^2_{tabla} > X^2_{real}$ entonces existe concordancia de criterios entre los expertos.

$X^2_{tabla} > X^2_{real}$

$15{,}51 > 0{,}6048$

Por lo tanto se puede afirmar que existe concordancia entre los expertos.

3.6.3 Seguridad

Para la informatización de los subsistemas de Afiliación de Empresas Cubanas y Sucursales Extrajeras, se utilizaron mecanismos que permiten una buena seguridad. Los subsistemas cuentan con distintos mecanismos que permiten la seguridad como son:

- **Autenticación desde el cliente y autorización en el servidor:** los mismos son gestionados por el servidor de aplicaciones y está basado en el uso de roles.
- **Validación de la fortaleza de la contraseña:** viene dada por la medición de parámetros como la cantidad total de caracteres, la cantidad de signos especiales, la cantidad total de dígitos y la cantidad de letras en mayúscula.
- **Cifrado de contraseña:** se lleva a cabo utilizando las funciones de resumen md5 y sha-1, las mismas son sólo de ida o irreversibles.
- **Mecanismos de confidencialidad e integridad para transmitir la información por la red:** los mismos se basan en algoritmos de cifrado de clave simétrica. El marco de trabajo cuenta con un componente para la generación de dicha llave de forma local, tanto en el cliente como en el servidor.

Establecer políticas que regulen el uso de aplicaciones y el acceso a las mismas constituye una tarea de vital importancia. Las empresas deben controlar el uso que se realiza por parte de los usuarios de estas nuevas herramientas colaborativas que pueden mejorar la productividad, pero evitando que se conviertan en un riesgo para la seguridad tanto de los sistemas como de los datos.

3.7 Conclusiones parciales

La realización de las pruebas a los subsistemas desarrollados permitió demostrar el cumplimiento de los objetivos de la investigación:

- El código implementado fue probado y corregido mediante la aplicación de las pruebas de caja blanca y las pruebas unitarias.
- Las pruebas de caja negra realizadas evidencian que las funcionalidades de los subsistemas tienen correspondencia con las que se definieron inicialmente.
- Se utilizaron métricas de diseño que permitieron validar algunos atributos de calidad, arrojando resultados satisfactorios en los atributos medidos.
- Se aplicó un análisis estadístico a partir de criterios realizados por varios expertos donde se comprobó el alto grado de usabilidad y seguridad de los subsistemas.
- Se realizó un aval por parte de la dirección del proyecto SIRECC donde se reconoce la calidad de los requisitos implementados. Ver anexo 7.

CONCLUSIONES

Con la culminación del presente trabajo de diploma se desarrollaron los subsistemas de Afiliación de Empresas Cubanas y Sucursales Extranjeras del proyecto Sistema de Informatización Registral de la Cámara de Comercio (SIRECC), contribuyendo a la seguridad y usabilidad de los procesos de registro. Los resultados alcanzados permiten concluir:

- Se efectuó un análisis del marco teórico de la investigación, demostrando las bases científicas del trabajo.
- La adecuada selección de la metodología, las tecnologías y la arquitectura permitió satisfacer las necesidades existentes.
- El uso de patrones permitió aplicar buenas prácticas de desarrollo, donde se obtuvo los modelos de diseño y de implementación transformando los requisitos identificados en una representación de la solución desarrollada.
- A partir de la representación física de los componentes se logró el efectivo desarrollo de los requisitos que componen los subsistemas, y así proveer una solución a los problemas existentes en la Cámara de Comercio de la República de Cuba.
- Las pruebas realizadas evidencian que las funcionalidades de los subsistemas tienen correspondencia con las que se definieron inicialmente.

RECOMENDACIONES

Luego de dar cumplimiento al objetivo de este trabajo de diploma y teniendo en cuenta las experiencias adquiridas durante el desarrollo, se recomienda:

- Reutilizar componentes de los subsistemas en aplicaciones registrales.
- Implementar los requisitos relacionados con los reportes de los subsistemas.
- Continuar con el mejoramiento de los subsistemas a partir de nuevos cambios que puedan surgir de acuerdo a nuevas necesidades de los clientes.

BIBLIOGRAFÍA

Apache. 2012. apache. [En línea] 2012. http://www.apache.org.

Bizagi. 2009. Patrones de Modelado. [En línea] 2009. http://wikibizagi.com/es/index.php?title=Patrones.

Bonnefoy, Juan Cristobal. 2006. *Indicadores de Desempeño.* s.l. : United Nations Publications, 2006.

Camacho, Erika. 2004. *Arquitectura de Software.* 2004.

Cámara Comercio. 2009. *Que es la Cámara.* La Habana : s.n., 2009.

Definicion.de. 2011. [En línea] 10 de 11 de 2011. http://definicion.de/comercio/.

Director de Formación y Membrecia. 2009. *Procedimiento de Afiliación de Empresas Cubanas.* Habana : s.n., 2009.

Directora Jurídica. 2009. *Manual de la Dirección Jurídica.* 2009.

Eckel, Bruce. 2013. *Thinking in Java.* 2013.

Edu, Mauri. 2011. *Testing funcional o pruebas de caja negra.* 2011.

Hommel, Scott. 1999. *Convenciones de código para el lenguaje de programación.* s.l. : JAVA, 1999.

http://www.camaras.org. 2010. *Organización y funciones.* Chile : http://www.camaras.org, 2010. 1.2.

Jacobson, Ivar. 1999. [En línea] 1999. http://www.rational.com.ar.

Juristo, Natalia. 2008. *Verification and validation: current and best practice.* 2008.

Kendall, Kendall &. 2010. *Analisis y Diseño de Sistemas .* 2010.

Larman, Craig. 2011. *UML y Patrones. Introducción al análisis y diseño orientado a objeto.* 2011.

Lenguaje C#. 2012. Universidad Tecnica de Cotopaxi. [En línea] 2012.

López, Patricia. 2012. *Trabajando con Visual Paradigm .* 2012.

Matthew, Neil y Stones, Richard. 2005. *Beginning Databases with PostgreSQL.* 2005.

McGraw-Hill Companies . 2002. *Oracle Designer.* s.l. : S.I, 2002.

Metodologías Pesadas. 2011. es.sribd. *informatizate.* [En línea] 2011. http://www.es.scribd.com/doc/62686746/10/Metodologias-Pesadas.

Microsoft. 2012. Net Framework de Microsoft. [En línea] 2012. http://www.support.microsoft.com.

Monson-Haefel, Richard. 2004. *Enterprise JavaBeans.* 2004.

Neatbeans.org. 2013. Informacion del lanzamineto del IDE Netbeans 7.2. [En línea] 2013. http://neatbeans.org/comunity.

Nielsen, Jakob. 2003. *Usability Engineering.* 2003.

Oracle. 2003. [En línea] 2003. http://www.oracle.com/goto/glassfish.

OriginalJavaWhitepaper. 2011. Sitio Oficial Java. *java.sun.com.* [En línea] 2011. http://www.java.sun.com.

Pfleeger, Charles P. 2006. *Security in computing.* 2006.

PostgreSQL Global Development Group. 2012. postgresql. [En línea] 2012. http://www.postgresql.org/about.

Reynoso, Carlos. 2004. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires : s.n., 2004.

Ruiz, Marcelo Hernan. 2006. *Programacion Web Avanzada.* Habana : Felix Varela, 2006.

Sánchez, María A. Mendoza. 2004. *Metodologías de desarrollo de software.* 2004.

Schincariol, Mike Keith and Merrick. 2009. *Pro JPA 2: Mastering the Java Persistence API.* s.l. : Apress, 2009. Edición: 1.

Silberschatz, Abraham, Korth, Henry F. y Sudarshan. 2002. *Fundamentos de Bases de Datos.* Madrid : McGraw-Hill, 2002.

Terreros, Julio Casal. 2013. msdn.microsoft.com. *Desarrollo de Software basado en Componentes.* [En línea] 2013. http://msdn.microsoft.com/es-es/library/bb972268.aspx.

Varios. 2011. *Guia de Arquitectura N-Capas.* s.l. : Krassis Press, 2011.

Zhao, J. y D. Thomas. 2005. *Comparación de Herramientas de modelado UML: Enterprise Architect y Rational Rose.* 2005.

Zukowski, John. 2003. *Java 2 J2SE 1.4.* 2003.

GLOSARIO DE TÉRMINOS

Internacionalización: es el proceso de diseñar software de manera tal que pueda adaptarse a diferentes idiomas y regiones sin la necesidad de realizar cambios de ingeniería ni en el código.

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

Paradigma: En términos generales se puede definir al término paradigma como la forma de visualizar e interpretar los múltiples conceptos, esquemas o modelos del comportamiento en todas las etapas de la humanidad en lo psicológico y filosófico, que influyen en el desarrollo de las diferentes sociedades así como de las empresas, integradas e influenciadas por lo económico, intelectual, tecnológico, científico, cultural, artístico, y religioso que al ser aplicados pueden sufrir modificaciones o evoluciones según las situaciones para el beneficio de todos.