Universidad de las Ciencias Informáticas

Facultad 3

Título: Diseño e implementación de los subsistemas Créditos y Depósitos del sistema Quarxo.

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

Autor: Manuel Alejandro Borroto Santana

Tutor: Ing. Yoan Antonio López Rodríguez

La Habana, 2011. "Año 53 de la Revolución"

Pensamiento

Las producciones intelectuales serán el sustento fundamental de Cuba. La idea es convertir la informática en una de las ramas más productivas y aportadoras de recursos para la nación…

Fidel Castro

Declaración de autoría

Declaro que soy el único autor del trabajo “Diseño e implementación de los subsistemas Créditos y Depósitos del sistema Quarxo” y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio. Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____	_____
Manuel Alejandro Borroto Santana	Ing. Yoan Antonio López Rodríguez
Autor	Tutor

Resumen

La informatización del sistema bancario cubano como parte del desarrollo tecnológico que precisa el país, demanda una elevada capacidad tecnológica y operativa, lo que trae consigo la utilización de herramientas computacionales para el procesamiento de la información.

La Universidad de las Ciencias Informáticas ha desempeñado un papel protagónico en ese sentido, de manera que está inmersa en el desarrollo del sistema Quarxo, que responde de forma rápida y certera a la actividad contable y financiera que se lleva a cabo en el Banco Nacional de Cuba, a partir de que en este se utiliza el Sistema Automatizado para la Banca Internacional de Comercio (SABIC), el cual está desarrollado sobre tecnología obsoleta, lo que hoy en día dificulta el manejo de los grandes volúmenes de datos, así como la gestión de los préstamos y los depósitos; procesos claves dentro de las operaciones bancarias. Con el propósito de llevar a cabo dichas funcionalidades en el BNC, el presente trabajo comprende el Diseño y la Implementación de los subsistemas Créditos y Depósitos que serán incorporados al sistema Quarxo.

La solución abarca el estudio de tecnologías, herramientas y lenguajes a utilizar, así como la elaboración de artefactos propios de los flujos Diseño e Implementación como parte del proceso de desarrollo de software.

Palabras clave: Préstamos, Créditos, Depósitos, Quarxo, Diseño, Implementación.

Indice de contenidos

Índice de figuras

Índice de tablas

Introducción

En los últimos años, las Tecnologías de la Información y las Comunicaciones (TIC) han evolucionado para dejar de ser un componente básico del negocio y convertirse en un elemento crítico primordial para la ejecución de las estrategias en el ámbito empresarial. Con el inapelable paso del tiempo las organizaciones, elemento fundamental dentro del ambiente competitivo que se vive actualmente, han tenido la necesidad de contar con información confiable, íntegra y oportuna para el cumplimiento de sus principales objetivos. La informatización de los procesos empresariales es uno de los mecanismos indispensables para el desarrollo económico de una empresa en busca del aumento en los niveles de eficiencia, organización de las actividades, control del flujo de información y optimización de los procesos.

Las empresas bancarias no son la excepción en este contexto en que urge incluir las tecnologías de la información como elemento clave para el logro eficiente de los objetivos organizacionales. El entorno en que se desarrolla la actividad bancaria, exige un constante esfuerzo de mejora en muchos frentes tales como: el rediseño de procesos, la mejora de la productividad, la reducción de costos y el alcance de una buena calidad para mejorar la satisfacción de los clientes, dejando clara la necesidad de perfeccionamiento y evolución de las soluciones informáticas.

Los bancos actualmente poseen el control económico del mundo y su actuación se ha visto favorecida en gran medida por la utilización de sistemas informáticos, como pilar fundamental para el incremento de la eficiencia en la toma de decisiones y en el resto de sus operaciones.

El Sistema Bancario Cubano y en específico el Banco Nacional de Cuba en función de lograr el manejo automático de la información y la interconexión entre los diferentes bancos y sucursales ha venido empleando los sistemas: SABIC (Sistema Automatizado para la Banca Internacional de Comercio) y SISCOM (Sistema de Comunicaciones) respectivamente. Independientemente de que el SABIC ha sido versionado en diferentes ocasiones e instalado en diversas entidades bancarias del país, el Banco Nacional de Cuba continúa afectado por utilizar la versión inicial, la que presenta una serie de limitaciones al utilizar como sistema operativo el MS-DOS, como plataforma tecnológica el FOX PRO y un servidor de ficheros que conlleva a un tráfico excesivo dentro de la red. Debido a las deficiencias del propio sistema el BNC se ha visto dificultado en la toma de decisiones, el control eficiente de los cambios, la obtención de los reportes y la ejecución del resto de las operaciones.

A raíz de eso, se determina la necesidad de realizar una nueva solución informática que resuelva esas dificultades. Para ello se le encargó la tarea de desarrollar dicha aplicación a un proyecto productivo de la Universidad de las Ciencias Informáticas. La solución denominada Quarxo se implementa a través de un conjunto de subsistemas, entre los cuales se encuentran: Contabilidad, Cuentas de Clientes, Cartas de Créditos, entre otros, existiendo una adecuada comunicación entre ellos. Actualmente no se cuenta con subsistemas que gestionen las operaciones de préstamos y depósitos bancarios, siendo dichos procesos de primordial importancia para el Banco Nacional de Cuba.

Atendiendo a la situación problemática descrita con anterioridad, se identifica el siguiente **problema a resolver**: ¿Cómo llevar a cabo los procesos de préstamos y depósitos del Banco Nacional de Cuba mediante los subsistemas del sistema Quarxo?

A partir del problema definido el **objeto de estudio** estará enfocado hacia los procesos de préstamos y depósitos en sistemas informáticos bancarios y el **campo de acción** hacia los procesos de préstamos y depósitos del BNC.

El **objetivo general** definido para el siguiente trabajo de diploma es: Realizar el diseño y la implementación de los subsistemas Créditos y Depósitos del sistema Quarxo.

Idea a defender: Si se realiza el diseño y la implementación de los subsistemas Créditos y Depósitos del sistema Quarxo, se logrará llevar a cabo los procesos de préstamos y depósitos del Banco Nacional de Cuba en el sistema Quarxo.

Objetivos específicos:

1. Fundamentar la investigación, mediante la elaboración del Marco Teórico.
2. Diseñar e implementar los subsistemas Crédito y Depósitos.
3. Validar los subsistemas implementados.

Tareas para cumplir los objetivos:

1. Revisión de sistemas informáticos bancarios existentes.
2. Estudio de las metodologías, lenguajes y herramientas establecidas en el proyecto SAGEB.
3. Revisión de las Especificaciones de Requisitos de los subsistemas Créditos y Depósitos.

4. Estudio de las tecnologías y patrones de diseño establecidas en el proyecto SAGEB.
5. Diseño de los subsistemas Créditos y Depósitos.
6. Estudio del comportamiento de los componentes de software a utilizar.
7. Implementación de los subsistemas Créditos y Depósitos.
8. Validación de los subsistemas Créditos y Depósitos.

Posibles resultados: Obtener mediante el diseño y la implementación los Subsistemas Créditos y Depósitos del sistema Quarxo.

Estructura del documento:

Capítulo 1: Se presenta la fundamentación teórica del tema mediante; la descripción de algunos sistemas bancarios existentes relacionados con el objeto de estudio, la valoración del estado del arte, el estudio de la Arquitectura Base definida para el sistema y la justificación las herramientas y tecnologías a utilizar para el diseño e implementación de los subsistemas.

Capítulo 2: En este capítulo se elabora el diseño a partir de la especificación de requerimientos de los subsistemas Créditos y Depósitos, como base para la implementación de los mismos. Está enfocado a la construcción de diagramas de clases de diseño, diagramas de paquetes y diagramas de secuencias para conformar el modelo de diseño, según la Arquitectura Base y los requerimientos del sistema, así como la realización del modelo de datos.

Capítulo 3: Estuvo centrado principalmente en la implementación de los subsistemas Créditos y Depósitos, mostrando y explicando en cada caso los estándares de codificación, la interacción entre componentes del sistema a través de diagramas, descripción de un subconjunto de clases y funcionalidades y una breve descripción acerca de la utilización de Spring WebFlow.

Por otra parte se realiza la validación a partir de la realización de las pruebas de unidad, brindando una explicación detallada de las pruebas de caja blanca que fueron realizadas al código del software y las pruebas de caja negra que se realizaron a la interfaz del mismo

Capítulo 1: Fundamentación teórica

1.1 Introducción

En el presente capítulo se abordan diferentes temas que constituyen la base para efectuar el diseño y la implementación de los subsistemas Créditos y Depósitos del sistema Quarxo. Primeramente se exponen las tendencias y tecnologías actuales en el mundo informático así como la metodología, la arquitectura base y las herramientas definidas para el desarrollo. En un segundo momento se presenta un estado del arte sobre el objeto de estudio y el campo de acción definidos en la investigación, así como una conceptualización de los principales términos relacionados con el dominio del problema.

1.2 Tendencias y tecnologías actuales

La compleja actividad del desarrollo de software ha logrado un gran auge en los últimos años incorporándose en casi todas las actividades de la sociedad. En la actualidad se desarrolla en un mundo que se encuentra en constante cambio, donde los usuarios y clientes son cada día más exigentes y para los cuales se hace necesario que el producto tenga la calidad requerida, donde las funcionalidades respondan lo mejor posible a sus necesidades y algo muy importante para ambas partes que es la reducción del tiempo de desarrollo y el coste del producto final. Para ello el mundo se mueve sobre una seria de tendencias y tecnologías que ayudan en gran medida a solventar estos problemas, a continuación se ofrecerá una valoración de algunas de ellas:

1.2.1 Arquitectura cliente-servidor

Básicamente es una arquitectura distribuida, que permite a los usuarios finales obtener acceso a la información en forma transparente independientemente de la plataforma. En esta el cliente envía un mensaje al servidor solicitando un determinado servicio y este responde a la solicitud enviando los datos de la respuesta. Esta arquitectura se ha convertido en una de las más utilizadas hoy día y entre sus características están:

✓ El ambiente es heterogéneo. La plataforma de hardware y el sistema operativo del cliente y del servidor no son siempre la misma. Precisamente una de las principales ventajas de esta arquitectura es la posibilidad de conectar clientes y servidores independientemente de sus plataformas.

✓ Las tareas del cliente y del servidor tienen diferentes requerimientos en cuanto a recursos de cómputo como velocidad del procesador, memoria, velocidad y capacidades del disco duro.

✓ Combinación de un cliente que interactúa con el usuario y un servidor que interactúa con los recursos compartidos. El proceso del cliente proporciona la interfaz entre el usuario y el resto del sistema. El proceso del servidor actúa como un motor de software que maneja recursos compartidos tales como bases de datos, impresoras, módems, entre otras. (1)

✓ El hecho de que esta arquitectura se haya propagado tanto es que posee una serie de ventajas debido a que puede usar componentes de software y hardware de diferentes fabricantes, ayudando a reducir los costos y la flexibilidad en la implantación, es más rápido el mantenimiento y desarrollo de las aplicaciones, la estructura inherentemente modular facilita además la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional, favoreciendo así la escalabilidad de las soluciones.

1.2.2 Aplicaciones Web

En la actualidad con el auge y el desarrollo desmedido de la Internet y el surgimiento de la WWW (en inglés: World Wide Web) se ha hecho muy común el desarrollo de aplicaciones web, que son un buen ejemplo de aplicaciones que emplean la arquitectura cliente-servidor y a su vez presenta todas sus características, con sus ventajas y desventajas.

La principal característica de estas aplicaciones y a su vez una de sus ventajas más relevantes, es el acceso inmediato desde cualquier lugar sin necesidad de descargar, instalar ni configurar nada, solo se necesita un navegador web instalado en el computador para poder acceder a la misma y de conexión con el servidor. No se necesitan grandes requerimientos de hardware, debido a que consumen poco o ningún espacio en disco duro y el consumo de memoria RAM es pobre, además no se necesitan procesadores potentes ya que la carga de trabajo se desplaza al servidor.

1.3 Metodología definida para el desarrollo

Una metodología es una colección de procedimientos, técnicas, herramientas y documentos auxiliares que ayudan a los desarrolladores de software en sus esfuerzos por implementar nuevos sistemas de información. Una metodología está formada por fases, cada una de las cuales se puede dividir en sub-fases, que guiarán a los desarrolladores de sistemas a elegir las técnicas más apropiadas en cada momento del proyecto y también a planificarlo, gestionarlo, controlarlo y evaluarlo. (2)

En la actualidad y a partir de la concepción de diferentes ideologías sobre cómo desarrollar un software han surgido un gran cúmulo de metodologías de desarrollo que atienden principalmente a aspectos importantes tales como, el tamaño del proyecto, la criticidad que presenta, la calificación del personal con el que se cuenta, así como la cultura que presenta el mismo. Comúnmente se dividen en dos grupos: las metodologías ágiles y las robustas. Las ágiles buscan minimizar los riesgos desarrollando software en cortos lapsos de tiempo y resaltan las comunicaciones cara a cara en vez de la documentación, además acentúan que la primera medida de progreso es el software funcional, entre ellas están: eXtreme Programming (XP), Scrum, Crystal, Open Unified Process (OpenUP), Agile Unified Proccess (AUP). Por su parte las metodologías robustas destacan por ser apropiadas para productos y proyectos grandes en las que existe un rigor de requisitos y diseño adecuado para los procesos de prueba, como ejemplo de estas están: RUP, Microsoft Solution Framework (MSF) y Iconix.

Para el desarrollo de esta investigación se ha seleccionado la metodología RUP (en inglés: Rational Unified Process) debido al alcance del proyecto y a que la misma aplica muchas de las mejores prácticas del desarrollo de software moderno, enfocadas a la producción de software con calidad.

1.3.1 Rational Unified Process

El Proceso Unificado es un proceso de desarrollo de software. Un proceso de desarrollo de software es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. El Proceso Unificado es más que un simple proceso, es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes niveles de aptitud y diferentes tamaños de proyecto.

El Proceso Unificado utiliza el UML (en inglés: Unified Modeling Language), para realizar todos los esquemas de un sistema de software, pero existen tres aspectos que realmente lo caracterizan y ellos son:

✓ **Dirigido por casos de uso:** Los casos de uso describen lo que los usuarios futuros necesitan y desean, lo que es captado en el modelado del negocio y representado a través de los requerimientos. A partir de este momento todos los artefactos que se obtienen en el desarrollo representan la realización de los casos de uso.

✓ **Centrado en la arquitectura:** La arquitectura provee al equipo de desarrollo y a los usuarios de una visión general del sistema en la que ambas partes deben de estar de acuerdo y describe los elementos del modelo que son más importantes para su construcción. La relación entre casos de usos y arquitectura es estrecha, es decir los casos de usos deben encajar en la arquitectura cuando se llevan a cabo mientras que la arquitectura debe permitir el desarrollo de todos los casos de usos requeridos.

✓ **Iterativo e incremental:** El desarrollo de un proyecto supone un gran esfuerzo que puede durar entre varios meses hasta posiblemente un año o más. Es práctico dividir el trabajo en partes más pequeñas o mini proyectos. Cada proyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en el flujo y los incrementos al crecimiento del producto. Para una efectividad máxima las iteraciones deben estar controladas.

El Proceso Unificado se repite a lo largo de una serie de ciclos que constituyen la vida del proyecto. Cada ciclo está dividido en cuatro fases: Inicio, Elaboración, Construcción y Transición y durante cada fase tienen lugar un grupo de flujos de trabajos que son: Modelado del negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas, Instalación entre otras.

Debido a la naturaleza del presente trabajo se hace necesario centrarse en los flujos de trabajo de Diseño e Implementación.

✓ **Diseño:** En el flujo se describe como el sistema será realizado a partir de las exigencias previstas en el flujo de Requisitos, su función principal es tratar de interpretar y traducir los requisitos a una detallada

descripción que indique a los implementadores como desarrollar el sistema, obteniendo como principal resultado el modelo de diseño.

✓ **Implementación:** En la implementación se comienza con los resultados del diseño e implementa el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares. El propósito general de este flujo es desarrollar la arquitectura y el sistema como un todo, obteniendo como resultado principal el modelo de implementación. (3)

1.3.2 Lenguaje de modelado

Se denomina lenguaje de modelado de objetos al conjunto estandarizado de símbolos y las distintas combinaciones de ellos para modelar un diseño de software.

UML

UML (en inglés: Unified Modeling Language) es un lenguaje de modelado visual que permite visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software, soportando además el paradigma orientado a objetos. Se caracteriza por dividir a los sistemas en una estructura estática y un comportamiento dinámico**.** (4)

1.4 Arquitectura definida para el proyecto

Uno de los elementos clave en todo proceso de desarrollo de software es la definición de la Arquitectura. Esta representación eleva el nivel de abstracción, facilitando así la comprensión de sistemas de software complejos, asimismo hace que aumenten las posibilidades de reutilizar tanto la arquitectura como los componentes que aparecen en ella. En la medida que sea concebida la arquitectura basada en los principios de cohesión, utilidad y flexibilidad de los componentes se obtendrá un mejor acabado del producto.

“La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán y los principios que orientan su diseño y evolución”. (5)

1.4.1 Características de la arquitectura del sistema

A continuación se presentan los principales componentes que conforman la arquitectura del sistema informático Quarxo, que tiene como primer elemento el desarrollo bajo la tecnología Java Enterprise Edition. La arquitectura base está compuesta por tres capas: Presentación, Negocio y Acceso a Datos y una capa transversal a las otras con las clases del dominio, como se muestra a continuación:

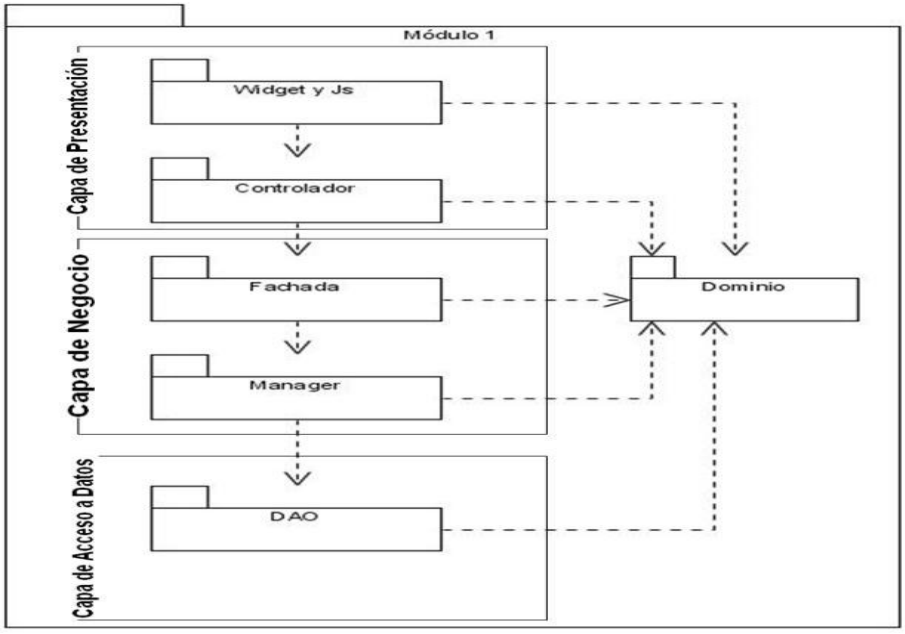

Figura 1: Representación de la Arquitectura del sistema.

- ✓ **Capa de Presentación**

En esta capa se desarrolla la lógica de presentación. En el lado del cliente se utiliza la librería Dojo Toolkit para generar las interfaces que interactúan con el usuario. En el lado del servidor se emplea Spring MVC para recibir, controlar y enviar respuestas a las peticiones realizadas desde el cliente y también se utilizará Spring WebFlow para representar y controlar los flujos complejos y reutilizables de la aplicación. La capa de Presentación está relacionada con la capa de Negocio y la capa de Dominio.

- ✓ **Capa de Negocio**

La capa de Negocio está divida en dos sub capas. Estas sub capas son Facade y Manager. La Facade es el punto de intercambio entre la capa de Presentación y la capa de Negocio. Esta capa no tiene

lógica de negocio, sino que agrupa las funcionalidades según su naturaleza para que pueda ser invocada desde la capa de Presentación. La subcapa Facade delega a la sub capa Manager la realización de la lógica del negocio.

Por otro lado, la subcapa Manager tiene la jerarquía de clases suficiente para implementar el negocio de la aplicación. Esta subcapa utiliza la capa de Acceso a Datos para obtener los datos persistidos y la capa de Dominio para generar las entidades del negocio.

Desde la capa de Negocio se envuelven transaccionalmente las funcionalidades de la aplicación para evitar inconsistencia en los datos. Se utiliza para esto, las políticas de transacciones que propone Spring Framework. Se utiliza el contenedor de Spring Framework para declarar y representar las relaciones de dependencia de cada una de las clases, Spring Security para asegurar las invocaciones a los métodos, Spring AOP para ejecutar las transacciones y la auditoría de cada uno de los métodos del negocio.

- ✓ **Capa de Acceso a Datos**

En esta capa se encuentran las operaciones que permiten la interacción con el gestor de base de datos desde la aplicación. Desde aquí se ejerce la conexión con el gestor de base de datos, permitiendo así la persistencia y el acceso a la información. La interacción con la capa de negocio se realiza a través de interfaces. Se utiliza el patrón DAO para el desarrollo de la capa, el framework de persistencia Hibernate y los módulos Spring ORM y Spring DAO. Hibernate como tecnología ORM para el manejo objetual de acciones de persistencia con la base de datos. Spring ORM y Spring DAO para soportar la integración con Hibernate y la utilización del patrón DAO.

La arquitectura definida presenta como otra de sus características la utilización del patrón MVC, patrón que propone dividir la aplicación en tres capas distintas, el Modelo, la Vista y el Controlador, potenciando la flexibilidad y la adaptabilidad a futuros cambios. Específicamente el Modelo es la representación de la información que maneja la aplicación, la Vista constituye la representación del modelo en forma gráfica disponible para la interacción con el usuario y el Controlador se encarga de responder a las solicitudes del usuario desde la Interfaz, manejando los diferentes eventos a través de las funcionalidades necesarias y la información perteneciente al Modelo.

Estructura del sistema

El sistema está estructurado a través de subsistemas, módulos y componentes:
Los Subsistemas, Módulos y Componentes son definidos según las funcionalidades identificadas en la Captura de Requisitos. Los Subsistemas agrupan un conjunto de Módulos relacionados con los procesos que ejecutan. Los Módulos agrupan un conjunto de Casos de Uso que representan uno o más procesos bancarios estrechamente relacionados y por último los Componentes son un conjunto de funcionalidades comunes que son reutilizados por otros módulos del sistema. (6)

1.5 Patrones de diseño

En el complejo mundo del desarrollo del software se ha hecho muy común el uso de los patrones, esto no se debe solamente a un gusto, sino que su empleo brinda a los equipos de desarrollo un arma que agiliza el diseño del sistema, debido a que establecen soluciones a problemas particulares del diseño, facilitan la reutilización del código y permiten una fácil comprensión debido a la documentación estándar que presentan. Una definición formal para el término patrón sería:

Un patrón es una pareja problema-solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas.

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

1.5.1 Patrón DAO

Plantea utilizar un DAO (en inglés: Data Access Object) para abstraer y encapsular todos los accesos a la fuente de datos. Maneja la conexión con la fuente de datos para obtener y almacenar datos, implementa el mecanismo de acceso requerido para trabajar con la fuente de datos y oculta completamente los detalles de implementación de la fuente de datos a sus clientes. Como la interfaz expuesta por el DAO no cambia cuando cambia la implementación de la fuente de datos subyacente, este patrón permite al DAO adaptarse a diferentes esquemas de almacenamiento sin que esto afecte a sus clientes o componentes de negocio. Esencialmente, actúa como un adaptador entre el componente y la fuente de datos. (7)

1.5.2 Patrones GRASP

Los Patrones Generales para Asignar Responsabilidades: GRASP (en inglés: General Responsibility Asignment Software Patterns) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones, su nombre se debe a la importancia de captar estos principios, si se quiere diseñar eficazmente el software orientado a objetos.

GRASP destaca 5 patrones principales: Experto, Creador, Bajo acoplamiento, Alta cohesión, Controlador.

Patrón Experto: Este patrón consiste en asignar una responsabilidad al experto en información, es decir a la clase que cuenta con la información necesaria para cumplir con la responsabilidad. Se refuerza el encapsulamiento y se favorece el bajo acoplamiento.

Patrón Creador: El patrón guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. Su propósito principal es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Brinda soporte al bajo acoplamiento.

Patrón Bajo Acoplamiento: Pretende asignar una responsabilidad para mantener el bajo acoplamiento, es decir el diseño de clases más independientes, que no se relacionen con muchas otras, que reducen el impacto de los cambios, que son más reutilizables y acrecienten la oportunidad de una mayor productividad.

Patrón Alta Cohesión: Su objetivo es asignar una responsabilidad de modo que la cohesión siga siendo alta. Las clases con alta cohesión se caracterizan por tener responsabilidades estrechamente relacionadas y no realizar un trabajo enorme. Una clase con alta cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizarla. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño.

Patrón Controlador: Consiste en asignar la responsabilidad a una clase de manejar los mensajes correspondientes a eventos de un sistema. El controlador es un intermediario entre la interfaz de usuario y el núcleo de las clases donde reside la lógica de la aplicación. Este patrón sugiere que la

lógica de negocios debe estar separada de la capa de presentación para aumentar la reutilización de código y a la vez tener un mayor control. (4)

1.5.3 Patrones GoF

Patrón Fachada: Patrón estructural que trata de simplificar la interfaz entre dos sistemas o componentes de software ocultando un sistema complejo detrás de una clase que hace las veces de pantalla o fachada. La idea principal es la de ocultar todo lo posible la complejidad de un sistema, el conjunto de clases o componentes que lo forman, de forma que solo se ofrezca un punto de entrada al sistema tapado por la fachada. Su uso aísla los posibles cambios que se puedan producir en alguna de las partes.

Patrón Mediador: Patrón de comportamiento que coordina las relaciones entre sus asociados. Permite la interacción de varios objetos, sin generar acoples fuertes en esas relaciones.

Patrón Cadena de Responsabilidad: La cadena de responsabilidad se encarga de evitar el acoplamiento del remitente de una petición a su receptor, dando a más de un objeto la posibilidad de manejar la petición.

Patrón Singleton: Patrón creacional diseñado para restringir la creación de objetos pertenecientes a una clase. Su objetivo es garantizar que una determinada clase solo tenga una instancia y proporcionar un punto de acceso global a esta. Es un patrón muy sencillo de diseñar y a menudo es implementado por otros patrones. (8)

1.6 Ambiente de Desarrollo

1.6.2 Lenguaje de programación en el lado del servidor

Hoy en día existe una fuerte tendencia al desarrollo de aplicaciones Web favorecido por el desarrollo de Internet y las facilidades que este brinda como se explicaba al principio de este capítulo. Un aspecto importante de este tipo de aplicaciones lo constituye el lenguaje que se emplee en el servidor, que es el encargado de ejecutarse en el mismo y del cual los usuarios obtienen el beneficio del procesamiento de la información. En el presente coexisten gran número de lenguajes por los que un equipo de desarrollo

puede optar para realizar esta función, pero hay tres particularmente que marcan la delantera en este sentido, atendiendo a la popularidad que presentan dentro de la comunidad de desarrollo de software, el cúmulo de sistemas que han sido desarrollados sobre ellos, además de los resultados que se han obtenido mediante la explotación de estas aplicaciones, evidenciando su robustez y dominio sobre los demás, ellos son: C# desarrollado y estandarizado por Microsoft que soporta el paradigma orientado a objetos, PHP diseñado originalmente para la creación de páginas web dinámicas siendo además multiparadigma y finalmente Java que fue el seleccionado por el equipo de arquitectura del proyecto para su empleo en el desarrollo atendiendo a una serie de características.

Java

Java es un lenguaje de programación orientado a objetos, que permite a los programadores realizar aplicaciones de múltiples tipos, ya sean de escritorio o web. Se caracteriza por ser un lenguaje simple, robusto y poderoso que se torna fácil de aprender, debido a que elimina sentencias de bajo nivel además del Garbage Collector (en español: Recolector de Basura) haciendo transparente para los programadores el manejo de la memoria. Se destaca por ser un lenguaje de código abierto, multiplataforma por lo cual ha logrado una gran expansión por todo el mundo. En la actualidad incluye un gran número de librerías para múltiples trabajos como: el trabajo con la red, tratamiento de excepciones, hilos para el procesamiento concurrente entre otras. (9)

Plataforma JEE

JEE (en inglés: Java Enterprise Edition) es una plataforma de programación, que define estándares para desarrollar y ejecutar aplicaciones en el lenguaje de programación Java, empleando arquitecturas que definen un modelo multicapa y que se apoyan en componentes de software modulares. JEE incluye tecnologías, tales como Servlets, JSP (en inglés: Java Server Pages) y varias tecnologías de servicios web. Las aplicaciones desarrolladas en esta plataforma tienden a ser portables, escalables, robustas y seguras a la vez que son integrables con tecnologías anteriores. (10)

1.6.3 Lenguajes en el lado del cliente

En el caso de los lenguajes que sustentan la aplicación en el cliente, se definieron por parte del proyecto el uso de:

XHTML:

XHTML (en inglés: Extensible Hypertext Markup Language) es el lenguaje de marcado pensado para sustituir a HTML (en inglés: Hypertext Markup Language), es una reformulación del mismo que es compatible con XML (en inglés: Extensible Markup Language) Se utiliza para generar documentos y contenidos de hipertexto generalmente publicados en la WEB. El documento se escribe en forma de etiquetas, que hasta cierto punto pueden establecer la apariencia del documento, aunque en la actualidad se suele mezclar con lenguajes script como JavaScript, para lograr mayor interacción con los usuarios. (11)

JavaScript

Es un lenguaje interpretado, es decir que no requiere compilación y se utiliza comúnmente para la construcción de páginas web en combinación con el XHTML. JavaScript no es orientado a objetos debido a que no soporta la herencia de objetos, sino que está basado en objetos que incorpora para su funcionalidad, aunque permite la creación de objetos propios. Se caracteriza por ser un lenguaje manejado por eventos por el hecho de responder a eventos generados ya sea por el usuario o por el navegador, es independiente de la plataforma debido a que solo se necesita un navegador para ejecutar el código, permite un desarrollo rápido y es relativamente fácil de aprender. Es soportado por la mayoría de los navegadores como Internet Explorer, Netscape y Mozilla Firefox.

1.6.4 Frameworks

Un framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. (12)

En el contexto actual de la industria de software se cuenta con grupo considerable de frameworks que facilitan y agilizan el desarrollo de sistemas. Específicamente para los lenguajes seleccionados con

anterioridad existen un conjunto de ellos que se emplean con mucha frecuencia. Para lo referente al trabajo de acceso a datos se pueden mencionar a algunos frameworks de elevado prestigio como Athena Framework, Ebean ORM e Hibernate Framework. Por otro lado están un conjunto de frameworks que hacen posible la aplicación del patrón MVC; propuesta de la arquitectura base, que poseen una elevada aceptación a nivel mundial, entre ellos: Struts, JBoss Seam y Spring Framework, vale destacar que los dos últimos presentan un marcado uso en la universidad. De igual manera para la lógica de presentación se utilizan frameworks, debido a la importancia que esta posee en los sistemas informáticos, teniendo en cuenta su interacción con los usuarios. Diversas son las librerías desarrolladas para JavaScript, que enriquecen las interfaz de usuario y brindan un ambiente de trabajo más limpio en el uso de Ajax, entre ellas destacan Qooxdoo, Ext JS y Dojo Toolkit.

Ateniendo a las características del proyecto y las bondades que brindan cada uno de estos frameworks el equipo de arquitectura decidió usar los que se caracterizan a continuación:

1.6.4.1 Spring Framework

Es un framework bajo licencia de código abierto concebido para el desarrollo de aplicaciones basadas en la plataforma Java/JEE. Spring ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas que permiten desarrollar un software seguro y robusto haciendo uso de las prácticas comunes en la industria de software. Su funcionamiento se basa en inversión de control e inyección de dependencia, apoyándose en el API de Java Reflection. El framework está dividido en módulos para su correcto funcionamiento, son estos:

✓ **Spring Core:** Este representa el núcleo de Spring, es donde se encuentran funcionalidades fundamentales que provee el framework.

✓ **Spring AOP:** Este módulo ofrece un extenso soporte para Programación Orientada a Aspectos, permitiendo definir entre otras cosas la política transaccional y de seguridad de una aplicación.

✓ **Spring ORM:** En este módulo se brinda el soporte necesario para la integración con los frameworks Hibernate e iBates.

✓ **Spring DAO:** Provee un trabajo con JDBC en aras de hacer el código de acceso a datos más limpio y entendible, ofrece además una capa de excepciones para el manejo de los errores emitidos por los servidores de base de datos.

✓ **Spring Web:** Es el encargado de crear el contexto para aplicaciones web, incluye soporte para una variedad de tareas como la subida de archivos, la vinculación de parámetros de las peticiones a objetos del negocio, etc.

✓ **Spring Web MVC:** Ofrece gran soporte para el desarrollo de aplicaciones web utilizando la filosofía Modelo-Vista-Controlador, emplea la inversión de control para brindar una separación entre la lógica de los controladores y los objetos del negocio.

✓ **Spring Context:** Este módulo es el que consagra a Spring como un framework, ofrece soporte para la internacionalización, la aplicación de eventos de ciclo de vida. Brinda soporte a servicios como correo electrónico, acceso JNDI, integración con EJB, etc.
Se hace uso de Spring Framework en su versión 2.5.

1.6.4.2 Spring WebFlow

Spring WebFlow es un framework y a su vez un subproyecto de Spring que permite controlar la navegación de la aplicación Web, guiando al usuario a través de una serie de pasos para completar una transacción de aplicación. Los flujos Web son diseñados para ser auto-controlados, dando la posibilidad de definir reglas de navegación múltiples y complejas. En Spring WebFlow un flujo se define mediante un archivo XML en el que se definen las reglas. Permite la creación de flujos reutilizables en toda la aplicación. (13)
Se hace uso de Spring WebFlow en su versión 2.0.8.

1.6.4.3 Hibernate Framework

Hibernate es una solución ORM (en inglés: Object Relacional Mapping) para el lenguaje de programación Java, concebido bajo la filosofía del código abierto. Busca solucionar el problema de la diferencia entre el modelo de objetos y el modelo relacional, realizando un mapeo entre tablas de la base de datos y los objetos del negocio a través de archivos declarativos, en este caso archivos XML.

Soporta la conexión a una gran variedad de servidores de base de datos, como PostgreSQL, Oracle, SQLServer y otros, permite además adaptarse a una base de datos ya existente, así como generar la base de datos a partir de un modelo objetual. Entre sus funciones se encuentran: permitir transacciones, asociaciones, polimorfismo, herencia, carga mediante referencia, persistencia transitiva. Ofrece también un lenguaje de consulta denominado HQL (en inglés: Hibernate Query Language), siendo este una poderosa vía de comunicación entre el programador y la base de datos, debido a que permite realizar consultas basándose en los objetos del negocio y no en las tablas de la base de datos propiamente dicho, aunque permite la ejecución de consultas SQL (en inglés: Standard Query Language). (14)

Se hace uso de Hibernate Framework en su versión 3.5.

1.6.4.4 Dojo Toolkit

Dojo Toolkit es una colección de scripts estáticos que permiten el desarrollo de aplicaciones web enriquecidas en el cliente, incorpora soporte para el trabajo con la tecnología AJAX (en inglés: Asynchronous JavaScript and XML). Se destaca por permitir desarrollar aplicaciones que pueden funcionar con independencia del navegador web donde se ejecuten, hace transparente el desarrollo para diferentes implementaciones del DOM, ofrece soporte para la internacionalización e incluye un gran cúmulo de componentes visuales basados en XHTML, CSS y JavaScript para enriquecer la interfaz de usuario. Presenta una arquitectura modular donde destacan los módulos: Dojo, Dijit y Dojox. (15)

Se hace uso de Dojo Toolkit en su versión 1.3.

1.6.5 Herramientas de desarrollo

1.6.5.1 Herramienta CASE

Una herramienta CASE (en inglés: Computer Aided Software Engineering) consiste en diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas. Este tipo de herramientas ayuda en todos los aspectos del ciclo de vida del desarrollo del software como la realización de su diseño, generación de código a partir de un diseño dado, documentación, entre otros.

En sentido general estas herramientas intentan dar ayuda automatizada al proceso de desarrollo de software, sirven de apoyo a la metodología de desarrollo empleada.

Visual Paradigm

Herramienta multiplataforma para el modelado UML que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Ayuda a una más rápida construcción de aplicaciones de calidad y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Otra gran utilidad que presenta es la capacidad de integrarse con IDE (en inglés: Integrated Development Environment) como el Eclipse y el Netbeans. Su principal dificultad radica en que posee una licencia muy restringida. (16)
Se estará haciendo uso de Visual Paradigm en su versión 6.4.

1.6.5.2 Control de versiones

Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas. Un sistema de control de versiones debe proporcionar un mecanismo de almacenaje de los elementos que deba gestionar y un registro histórico de las acciones realizadas con cada elemento o conjunto de elementos entre otros aspectos.

Subversion

Es un sistema de control de versiones libre y de código abierto conocido habitualmente como SVN que se ha expandido en gran medida dentro de la comunidad del desarrollo de software. Maneja ficheros y directorios a través del tiempo y la información radica en un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Subversion puede acceder al repositorio a través de redes lo que le permite ser usado por personas en ordenadores distintos fomentando la colaboración que deriva en la

reducción del tiempo de desarrollo. Existen diferentes clientes para el Subversion ya sean programas independientes como el TortoiseSVN y el Subeclipse para integrarlo con Eclipse. (17)
Se hace uso de Subversion en su versión 1.6.6.

1.6.5.3 Entorno integrado de desarrollo

Eclipse IDE

Eclipse es un entorno de desarrollo integrado de código abierto y multiplataforma desarrollado por IBM. Emplea plug-ins para proporcionar toda su funcionalidad, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. La arquitectura de plug-ins permite integrar diversos lenguajes, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo, tales como: herramientas UML, editores visuales de interfaces, ayuda en línea para librerías, entre otros. (18)
Se estará haciendo uso de Eclipse IDE en su versión 3.4.

1.6.5.4 Servidor de aplicaciones web

Tomcat

Tomcat es un contenedor de servlets bajo la filosofía del código abierto licenciado con Apache Sofware License que presenta la ventaja de ser multiplataforma. Implementa las especificaciones de Servlets 2.5 y JSP (en inglés: Java Server Pages) 2.1. Con frecuencia se presenta en combinación con el servidor web Apache aunque puede realizar esta función por sí mismo. En la actualidad es utilizado como un servidor web autónomo en entornos donde existe un alto nivel de tráfico y alta disponibilidad. (19)
Se hace uso de Tomcat en su versión 6.

1.6.5.5 Servidor de Base de Datos

Microsoft SQL Server 2005

SQL Server es un servidor de base de datos basados en el modelo relacional. Se caracteriza por brindar soporte para transacciones y procedimientos almacenados. Incluye un potente entorno gráfico para administración, permite además trabajar en modo cliente-servidor y promueve la escalabilidad y

seguridad de la información. Sus lenguajes de consulta son el SQL y el T-SQL (en inglés: Transact-SQL), siendo este último el principal medio de programación y administración del servidor. Requiere para su funcionamiento un sistema operativo Microsoft Windows, representando esto un inconveniente para su utilización. (20)

1.7 Procesos bancarios

Toda economía por muy grande o pequeña que sea debe contar en su infraestructura con un ente fundamental; los bancos, influenciada en gran medida por la existencia de estos. La importancia que se les concede viene dada porque son entidades que canalizan los recursos financieros en la economía a través de la captación de depósitos y el otorgamiento de préstamos, para ello utilizan como principal instrumento las tasas de interés, las cuales son un incentivo tanto para ahorrar dinero como para pedir prestado.

1.7.2 Depósitos

Los depósitos se pueden definir como el dinero depositado en un banco para que éste proceda a su custodia. Existen diversos tipos de depósitos según las características del valor depositado y las condiciones que se fijaron para realizarlo. Los depósitos a la vista son los que se realizan en el área de cuentas corrientes, están a disposición de los titulares para poder retirarlos. Los depósitos a plazo son aquellos que sólo pueden ser retirados luego de un término determinado y con ellos se gana un interés mayor que el de las simples cuentas de ahorro, pero quedan inmovilizados para el depositante durante el plazo fijado. También pueden depositarse en los bancos bienes o valores físicos que son guardados en cajas de seguridad. Los bancos cobran una suma fija por el alquiler de dichas cajas. Los depósitos recibidos pueden ser de clientes o de bancos, estos son usados para efectuar préstamos y otras transacciones. Los depósitos recibidos aumentan la liquidez del banco. (21)

1.7.3 Préstamos

Todo préstamo se efectúa entre un prestamista, quien da el dinero y un prestatario, quien lo recibe, originando una deuda de este último ante el primero. Los bancos y otras instituciones financieras asumen generalmente el papel de prestamistas, captando recursos que luego ofrecen a los interesados.

También pueden prestarse bienes físicos, aunque en este caso suele hablarse por lo regular de un contrato de arrendamiento, donde el pago del alquiler correspondiente sustituye a los intereses. Los prestatarios son generalmente empresas que requieren recursos de capital para mantener, desarrollar o ampliar sus actividades, personas que desean disponer de sumas relativamente grandes con respecto a sus ingresos para la adquisición de bienes o gobiernos que buscan recursos para el pago de sus compromisos más allá del límite de sus ingresos corrientes. La economía moderna se basa en gran medida en la existencia de un enorme número de préstamos de diversos tipos y magnitudes. Ellos serían imposibles si no existiesen instituciones que, como los bancos, realizan la función especializada de concentrar y hacer circular el capital. (21)

1.8 Sistemas Informáticos Bancarios

Hoy día todos los sectores de la sociedad se han visto beneficiados de una forma u otra con el empleo de las tecnologías de la información. Los sistemas informáticos bancarios son un ejemplo claro de los beneficios que trae aparejada la informática y los bancos mediante su uso han logrado mejorar en gran medida la eficiencia y eficacia de su quehacer diario.

Los sistemas informáticos bancarios son programas contables destinados a sistematizar y simplificar las tareas de contabilidad; préstamos, depósitos, gestión de cuentas, análisis financieros, entre otros. Sólo necesitan que sea ingresada la información requerida y posteriormente el programa se encarga de hacer las operaciones necesarias. A continuación serán descritos y caracterizados algunos sistemas de este tipo, tanto nacionales como internacionales, con el objetivo de buscar puntos que contribuyan a la solución del problema.

1.8.2 Sistema Bancario Financiero BYTE

Es un conjunto de módulos independientes e integrables, que permiten la automatización de todos los departamentos, según sean las necesidades. Este sistema contempla todas las operaciones que una institución financiera realiza, tanto administrativas como operativas y esto se logra a través de la interacción de los módulos que lo integran y que constituyen además su eje central:

✓ **Clientes:** Permite una visión global de las operaciones que un cliente ha realizado con la institución, además de poder agruparlos por grupos o sectores.

✓ **Captaciones:** Permite las operaciones de cuenta corriente, ahorros, depósitos a plazo, cuentas de traslado automático, cheques certificados, cuentas over night, administración de valores.

✓ **Colocaciones:** Realiza las operaciones de evaluación de sujetos de créditos, análisis financiero, control de solicitudes y presión de créditos, préstamos, documentos descontados, créditos en cuenta corriente, valuación de activos, control de recuperación, análisis de morosidad.

Se caracteriza por ser una aplicación completamente WEB que ha sido desarrollada siguiendo el modelo MVC, implementada completamente en el lenguaje Java, presenta soporte para bases de datos MS SQL Server, Oracle y DB2. (22)

1.8.3 BANTOTAL-Core Bancario

Es un software desarrollado por la compañía “De Larrobla & Asociados Internacional”, el mismo comprende el procesamiento integral de todas las operaciones de una entidad financiera. Este sistema incorpora como parte de sus funcionalidades la gestión de depósitos y préstamos, cuentas corrientes, cajas de ahorro, custodia y administración de valores, ordenes de pagos y transferencias, cajas de seguridad, cartas de crédito de importación y exportación, SWIFT, entre otras.

El sistema BANTOTAL puede ejecutarse en arquitecturas Java o .NET además de ser una aplicación implementada sobre la WEB y emplea el patrón MVC. En cuanto al almacenamiento y procesamiento de los datos la misma fue desarrollada con la capacidad de soportar bases de datos IBM i-Series, MS SQLServer, DB2 y Oracle. (23)

1.8.4 Sistema Automatizado para la Banca Internacional de Comercio (SABIC)

El SABIC es un sistema diseñado y desarrollado por la Dirección de Sistemas Automatizados del Banco Central de Cuba para satisfacer las necesidades de procesamiento de datos de bancos e instituciones no bancarias, utilizando los medios técnicos de computación disponibles en el mercado.

El sistema ha sido modificado en varias ocasiones en busca de satisfacer los requerimientos de las operaciones propias del Banco Nacional, logrando que el personal que opere mediante él pueda tramitar las operaciones sin necesidad de recurrir a archivos ni a la actividad manual, incrementando así la productividad, la eficiencia y seguridad en el trabajo.

Entre las características fundamentales que presenta el sistema se encuentran: la contabilización en tiempo real; que permite mantener actualizados los archivos contables en todo momento, la contabilización en varias monedas; para contabilizar los activos y pasivos en sus monedas de origen sin tener que realizar las conversiones correspondientes y la operación con transacciones; permitiendo realizar las operaciones usando transacciones que crean asientos automáticamente. Como otra de sus características está además su estructura modular que permite que le sean incorporados otros módulos de acuerdo a las particularidades de las instituciones.

El SABIC fue desarrollado empleando el lenguaje de programación FOXPRO, y su lógica de negocio se encuentra en su mayoría implementada en forma de procedimientos almacenados en la Base de Datos, realizada en SQLServer, además de ser ejecutado sobre MS DOS.

Independientemente de los beneficios que ha representado el SABIC para el sistema bancario cubano, presenta incompatibilidad con el sistema de mensajería SISCOM que funciona sobre Windows y es el que emplea el Banco Nacional, presenta una pobre gestión de los reportes y la realización de los procesos de préstamos y depósitos se hace muy engorrosa para el personal que labora con el sistema.

Existe otra versión del SABIC, en el lenguaje de programación Visual FoxPro, que se ejecuta sobre Windows, con Base de Datos SQLServer realizada para el Banco Central de Cuba, la cual mejora la situación del mismo, pero sus características no se ajustan a las del Banco Nacional. (24)

1.8.5 Valoración del estado del arte

En sentido general los sistemas antes expuestos constituyen buenas soluciones informáticas para el mundo financiero, debido a la gama de funcionalidades que brindan, posibilitando la gestión automática de las operaciones bancarias.

Decir de los sistemas extranjeros antes mencionados que están sustentados por tecnologías que hoy en día marcan un elevado nivel en la industria de software, atendiendo al número de aplicaciones desarrolladas sobre las mismas que han alcanzado gran prestigio internacionalmente. En estas soluciones informáticas se destaca como otra de sus características la multimoneda, elemento clave para la actividad económica y financiera a nivel mundial, que es a su vez de vital importancia para el Banco Nacional de Cuba debido al papel que este desempeña en el país como banco comercial, que interactúa directamente con el exterior. Un aspecto a tener en cuenta en el sistema bancario cubano es la dualidad monetaria; característica de la economía del país resultante de las diversas transformaciones que se han venido desarrollando y que en el ámbito internacional es un término poco difundido, para lo cual no fueron concebidos estos sistemas. Por otro lado las licencias de dichos productos constituyen una limitante; teniendo en cuenta que son privativas y que poseen altos precios, lo que trae consigo que se dificulte tanto la adquisición como el uso y mantenimiento de los mismos, a partir de que Cuba no está en condiciones de invertir en la compra de estas licencias, independientemente de esto pueden servir de base para la definición de tecnologías, herramientas y plataformas para el desarrollo de soluciones informáticas relacionadas con esta área de conocimientos.

Por su parte el sistema SABIC, hoy en explotación en el BNC, además de las deficiencias que se presentaron al caracterizarlo anteriormente, no cuenta con lo referente a la realización de un calendario de pago; haciéndose este de forma manual, lo que dificulta la ejecución de los préstamos y los depósitos bancarios, pero sin embargo constituye una guía para la definición de la funcionalidades claves del sistema Quarxo.

1.9 Conclusiones del capítulo

En el BNC se hace necesario un sistema que gestione la información contable de manera eficiente debido a que el actual sistema informático no cumple en su totalidad con sus exigencias, viéndose afectados notablemente la realización de los procesos de préstamos y depósitos bancarios.

La utilización de los frameworks, lenguajes y herramientas estudiados como parte del ambiente de desarrollo, facilitarán entre otros aspectos un desarrollo ágil, con el menor costo posible y la obtención de un producto como resultado final con la calidad requerida, apto para realizar su función en el Banco Nacional de Cuba.

Capítulo 2: Diseño de la solución.

2.1 Introducción

La transformación de los requisitos funcionales en el diseño del futuro sistema, que permita un mejor entendimiento de los mismos, en aras de conformar una entrada adecuada para la posterior implementación, tomando en cuenta la arquitectura definida por la dirección del proyecto, constituye la principal aspiración del presente capítulo. Partiendo de aquí, se derivarán un grupo de artefactos que responden al flujo de trabajo Diseño de la metodología utilizada, los cuales son claves para el correcto desarrollo de las etapas venideras, como son: modelo de diseño, diagramas de paquetes, diagramas de clases, diagramas de secuencia, el modelo de datos, entre otros.

2.2 Diseño de la solución

La Especificación de Requisitos está presente como uno de los artefactos más importantes obtenidos durante el flujo de trabajo de Requisitos, elemento clave para el diseño y la posterior implementación de la solución. En este caso se hará énfasis y se tomarán como entrada los requisitos funcionales definidos previamente para los subsistemas de Créditos y Depósitos por el grupo de analistas del proyecto.

A continuación se presentan los requisitos funcionales organizados por subsistemas:

Subsistema Depósitos:

- ✓ Gestionar Depósito a Plazo.
  - ✓ Registrar Depósito a Plazo.
  - ✓ Actualizar Depósito a Plazo.
  - ✓ Consultar Depósito a Plazo.
  - ✓ Buscar Depósito a Plazo.
  - ✓ Cancelar Depósito a Plazo.

Subsistema Créditos:

- ✓ Gestionar Préstamo Concedido.

- ✓ Actualizar Préstamo Concedido.
- ✓ Registrar Préstamo Concedido.
- ✓ Consultar Préstamo Concedido.
- ✓ Buscar Préstamo Concedido.

✓ Gestionar Préstamo Recibido

- ✓ Actualizar Préstamo Recibido.
- ✓ Registrar Préstamo Recibido.
- ✓ Consultar Préstamo Recibido.
- ✓ Buscar Préstamo Recibido.

Los requerimientos anteriormente expuestos presentan una adecuada organización y documentación, caracterizados por estar correctamente descritos y presentar prototipos de interfaz de usuario completos que han sido esenciales para el entendimiento del equipo de desarrollo. La descripción es precisa, clara y completa debido a que los requisitos son consistentes, no dejan margen a la ambigüedad ni a malas interpretaciones y satisfacen el nivel de detalle requerido para el diseño.
En sentido general la especificación de requerimientos cumple con la calidad requerida y constituye el elemento fundamental para el diseño de los subsistemas Créditos y Depósitos.

Es importante destacar que los subsistemas en cuestión están compuestos por módulos, en el caso de Créditos presenta a Préstamo Recibido y Préstamo Concebido que responden a las requisitos funcionales Gestionar Préstamo Recibido y Concebido respectivamente y un módulo Common que contiene los aspectos comunes para los demás módulos, por otra parte Depósitos está compuesto por el módulo Gestionar Depósito a Plazo que responde al requisito funcional del mismo nombre y un módulo Common que contendrá los aspectos comunes con otros módulos que puedan aparecer en un futuro.

2.2.1 Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso cendrándose en cómo los requisitos funcionales junto a otro grupo de restricciones relacionadas con el entorno de implementación tienen impacto en el sistema, está compuesto por otros artefactos como los diagramas de paquetes y diagramas de clases, además sirve como abstracción a la implementación y se empleará como entrada fundamental de las actividades de esta etapa. (3)

2.2.1.1 Diagrama de paquetes

Durante el desarrollo de software resulta muy conveniente agrupar clases y ficheros por diferentes criterios que ayudarán a una fácil comprensión de la aplicación, resultando conveniente el desarrollo de los diagramas de paquetes, los cuales muestran como está dividido el sistema en agrupaciones lógicas mostrando las dependencias entre estas. En el caso de los subsistemas Créditos y Depósitos la agrupación se realizó en dependencia del rol que desempeñan estos ficheros en el sistema.

A continuación se muestra la estructura y dependencia de paquetes del módulo Gestionar Depósito a Plazo del Subsistema Depósitos y una explicación de la composición de cada uno. El diagrama de paquetes que comprende los módulos asociados a los requisitos de Préstamos Recibidos y Concedidos del Subsistema Crédito se comporta de manera similar.

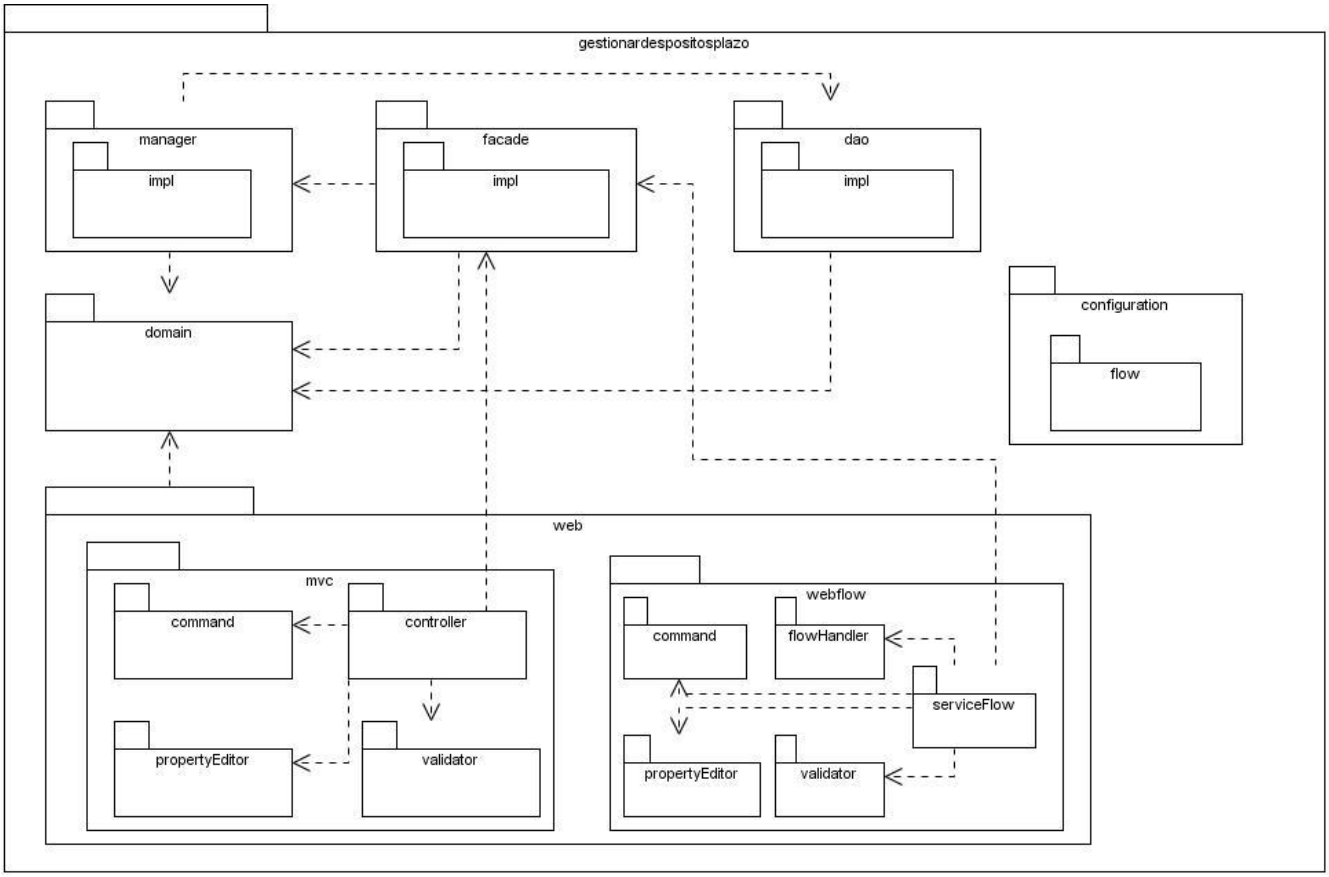

Figura 2 : Diagrama de paquetes del módulo Gestionar Depósitos a Plazos

Paquete *configuration*: En este paquete se encuentran los ficheros de configuración en formato XML de los diferentes contextos de Spring para el módulo, son estos:

- ✓ -servlet.xml: Define el contexto de Spring MVC.
- ✓ -bussiness.xml: Define el contexto para el negocio.
- ✓ -webflow.xml: Define el contexto para Spring WebFlow.
- ✓ -dataaccess.xml: Define el contexto para acceso a datos.

Paquete *flow*: En este paquete están presente los archivos XML que definen los flujos para Spring WebFlow.

Paquete *facade*: En este paquete se encuentran la interfaz y su respectiva implementación, encargadas de brindar las funcionalidades que serán usadas por la presentación, así como las que se consumen por otros módulos.

Paquete *manager*: En este paquete se hallan las interfaces y las implementaciones que comprenden la lógica de negocio que existe en el módulo, que será brindada a las capas superiores.

Paquete *dao*: En el paquete dao se encuentran igualmente las interfaces y las implementaciones encargadas de brindar los servicios de comunicación con la base de datos, además de hallarse también los ficheros de mapeo utilizados por Hibernate correspondientes al módulo.

Paquete *domain*: Aquí se localizan las clases relacionadas con el dominio del módulo en cuestión.

Paquete *web*: El paquete web agrupa un conjunto de clases y paquetes que son los encargados de realizar todo lo referente a la presentación en el lado del servidor.

Paquete *mvc*: En este subpaquete se agrupa todo lo referente a la lógica de presentación cuando se hace uso de SpringMVC.

Paquete *webflow*: En este subpaquete se agrupa todo lo referente a la lógica de presentación cuando se hace uso de Spring WebFlow.

Paquete *controller*: En este paquete se hallan las clases que extienden de los controladores propuestos por Spring MVC, encargadas de responder a las peticiones realizadas por el cliente.

Paquete *serviceFlow*: Contiene las clases encargadas de establecer la comunicación entre el flujo y la fachada del módulo.

Paquete flowHandler: Clases utilizadas para personalizar el trabajo con el WebFlow.

Paquete *command*: Se encuentran las clases que representan los datos que son introducidos y mostrados en los formularios de la interfaz de usuario.

Paquete *propertyEditor*: Agrupa las clases que actúan como intermediarias para convertir datos de la interfaz de usuario en objetos y viceversa.

Paquete *validator*: Cuenta con las clases encargadas de realizar la validación de datos en el lado del servidor.

2.2.1.2 Diagramas de clases de diseño

Una realización de caso de uso del diseño es una colaboración del modelo de diseño que describe como se realiza un caso de uso específico y como se ejecuta en término de clases de diseño y sus objetos. Una realización contiene diagramas de clases que muestran sus clases de diseño participantes y diagramas de interacción que muestran la realización de un flujo o escenario en concreto, en términos de interacción entre objetos. (3)

Específicamente los diagramas de clases de diseño son muy útiles porque muestran a través de atributos y métodos la estructura de las clases que después serán escritas en algún lenguaje de programación.

A continuación se muestran los diagramas de clases realizados para el módulo Gestionar Depósito a Plazo, en el cual se hizo uso de Spring WebFlow mayormente.

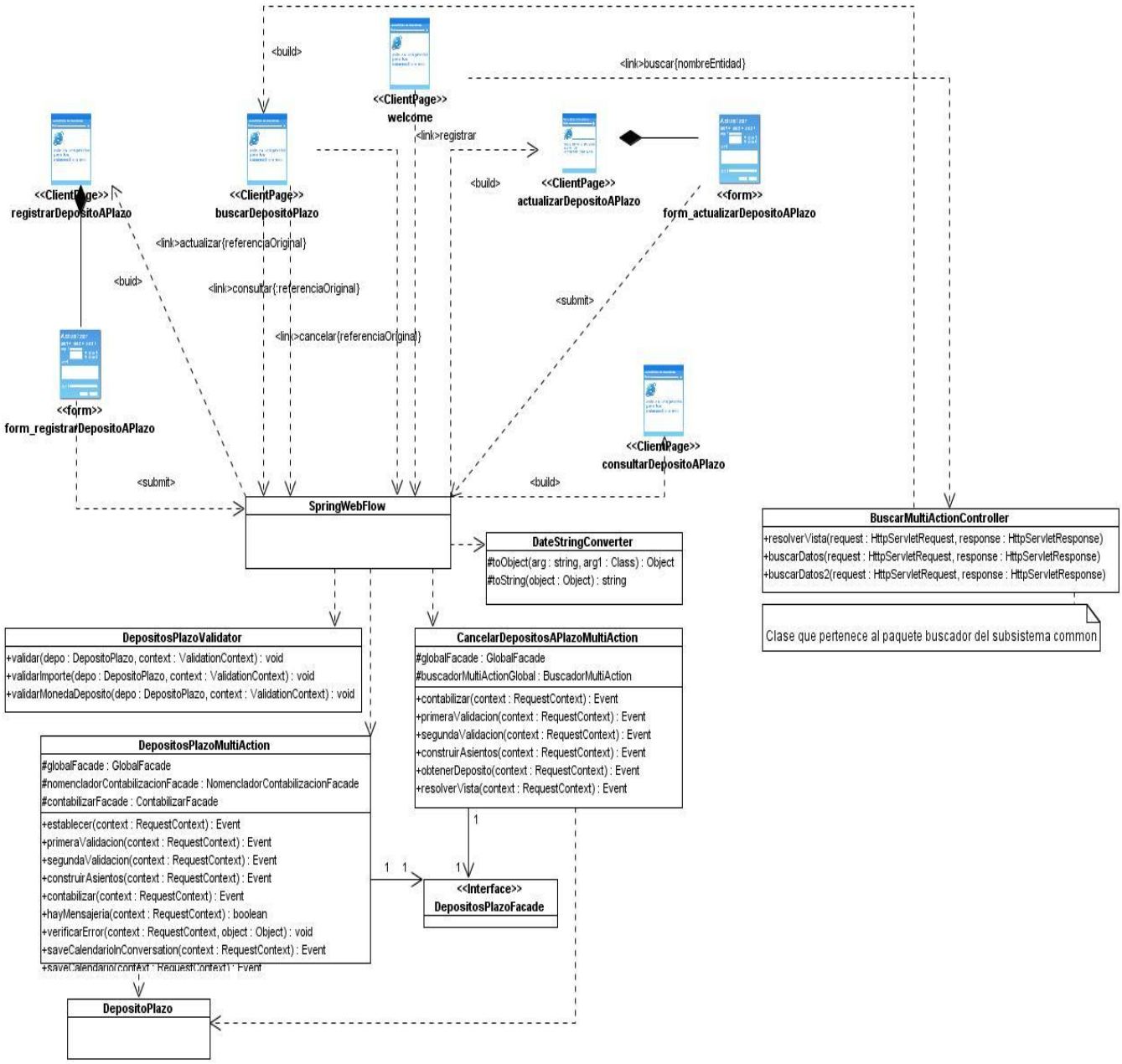

Figura 3: Capa de presentación del Diagrama de clase de diseño del módulo Gestionar Depósitos a Plazos.

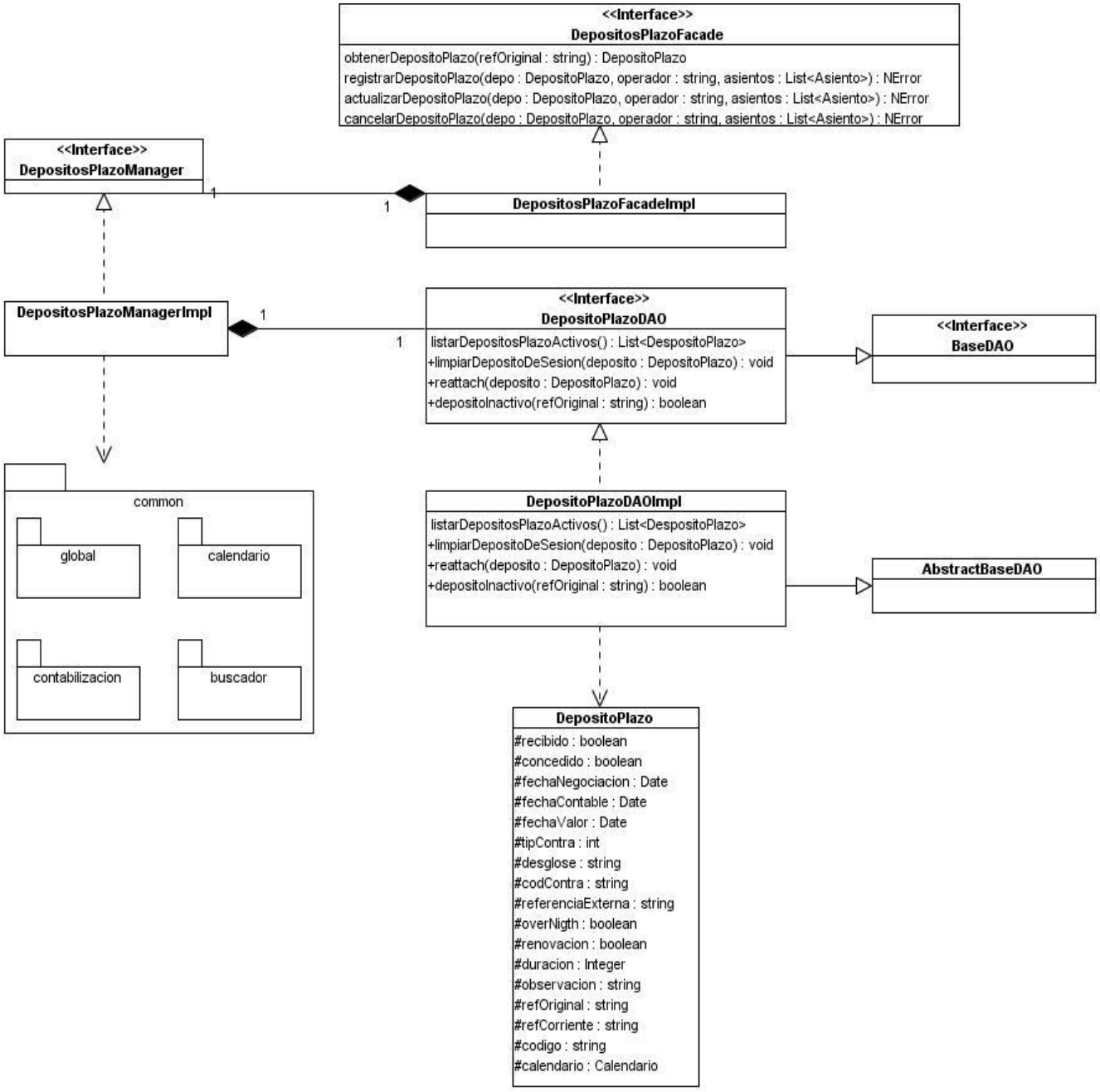

Figura 4: Capa de negocio y acceso a datos del Diagrama de clase de diseño Gestionar Depósitos a Plazos.

Debido a la utilización de frameworks y los estándares definidos por el grupo arquitectura del proyecto, en la realización de los diagramas mostrados con anterioridad, fue necesario diseñar determinadas

clases que pudieran crear confusión al tratar de comprender dichos diagramas, a continuación se brinda una breve descripción de estas para una mejor compresión de los mismos:

Clases ClientPage: Páginas web encargadas de mostrar los formularios e información al usuario.

SpringWebFlow: Clase que representa el mecanismo interno con el que Spring WebFlow gestiona las peticiones realizadas desde el cliente y mediante el cual se controlan los flujos por los que son guiados los usuarios en el sistema.

CancelarDepositosAPlazoMultiAction: Es la clase encargada de gestionar las acciones asociadas al flujo cancelar, SpringWebFlow interactúa con ella para su funcionamiento.

DepositosPlazoMultiAction: Clase que gestiona las acciones asociadas a los flujos de registrar, actualizar y consultar Depósito a Plazo, SpringWebFlow interactúa con ella para su funcionamiento.

DepositosPlazoFacade: Interfaz encargada de brindar las funcionalidades del módulo a la capa de presentación y a otros módulos que la requieran.

DepositosPlazoFacadeImpl: Clase encargada de implementar la lógica de programación de la Interfaz DepositosPlazoFacade.

DepositosPlazoManager: Clase que contiene las funcionalidades que se deben implementar en la clase DepositosPlazoManagerImpl para dar respuesta a las acciones que se solicitan desde DepositosPlazoFacadeImpl.

DepositoPlazoDAO: Interfaz de comunicación que se encarga de brindar las funcionalidades de la capa de Acceso a Datos.

DepositoPlazoDAOImpl: Clase que implementa las funcionalidades de la capa de Acceso a Datos.

BaseDAO y AbstractBaseDAO interfaz y clase abstracta respetivamente, brindadas por el DAO genérico utilizado en la aplicación, ellas brindan un conjunto de funcionalidades básicas que se realizan con las clases persistentes, ya sea buscar por identificador, listar, persistir, actualizar y eliminar.

Básicamente solo se necesita que las interfaces del acceso a datos implementen BaseDAO y las implementaciones hereden de AbstractBaseDAO.

DepositoPlazo: Clase que representa la información que persiste en la Base de Datos asociada a los Depósitos a Plazo.

Paquete common: Paquete de carácter general que contiene los componentes y estos a su vez las funcionalidades que son comunes a diferentes subsistemas.

Los diagramas de Clases del diseño correspondientes al Subsistema Créditos específicamente para los préstamos recibidos aparecen en:
Anexo 1: Capa de presentación del diagrama de clases de diseño de Gestionar Préstamos Recibidos.
Anexo 2: Capa de negocio del diagrama de clases de diseño de Gestionar Préstamos Recibidos.
Anexo 3: Capa de AD del diagrama de clases de diseño de Gestionar Préstamos Recibidos.

2.2.1.3 Diagramas de interacción

Los diagramas de interacción son utilizados para modelar los aspectos dinámicos de los sistemas, muestran la realización de un flujo o escenario concreto de un caso de uso en términos de interacción entre objetos del diseño. Se les atribuye una gran importancia debido a la compresión del sistema a la que puede llegarse a través de ellos.

Debido a esto se realizaron diagramas de secuencia, que son un ejemplo de diagramas de interacción que destacan principalmente el orden temporal de los mensajes para varios escenarios de los subsistemas en cuestión, a continuación se presenta los diagramas de secuencia asociados al escenario Registrar Depósito a Plazo.

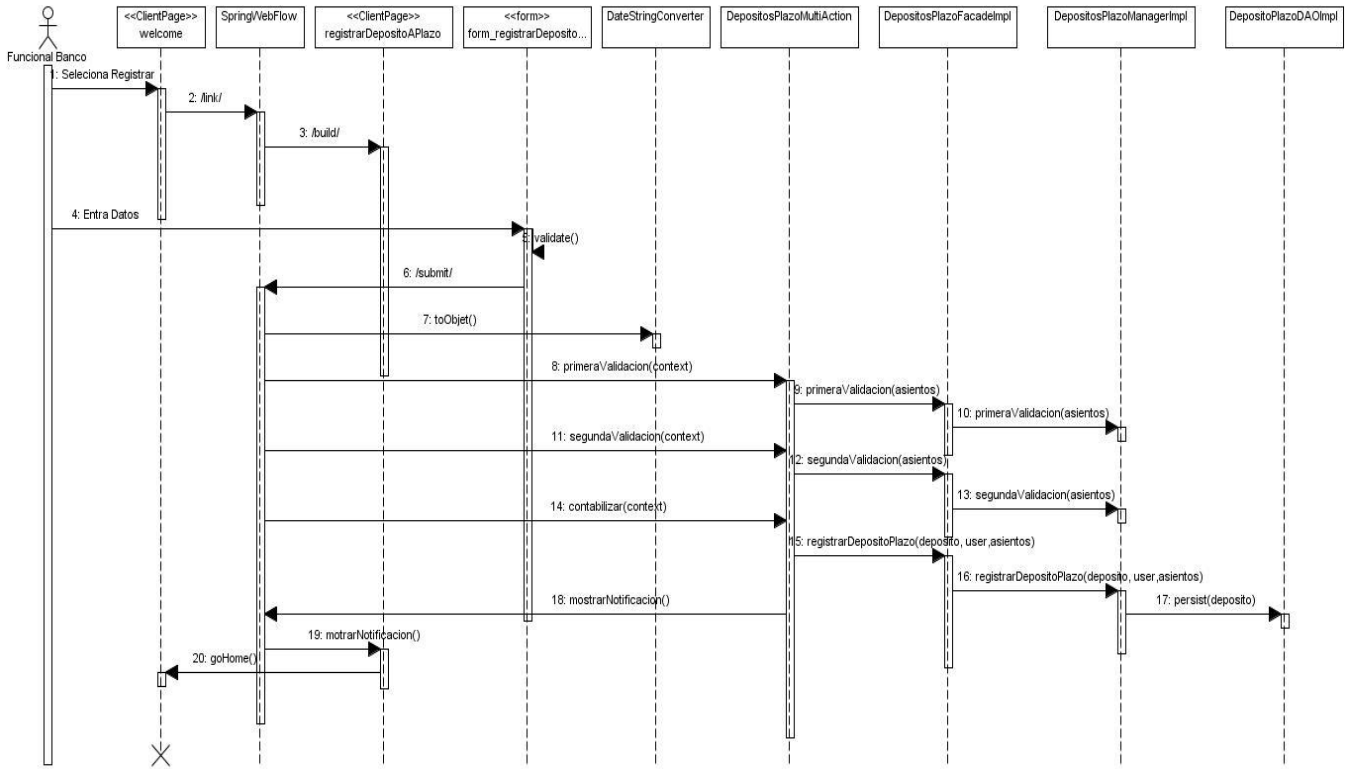

Figura 5: Diagrama de interacción del escenario Registrar Depósito a Plazo.

2.2.2 Modelo de datos

Un modelo de datos es un modelo abstracto que describe cómo deben ser representados y usados los datos. Sirve para describir la estructura de la base de datos, así como los datos, sus relaciones y las restricciones que deben cumplirse entre ellos.

El modelo de datos propuesto para lo solución, está dividido en dos partes, el desarrollado para el subsistema Depósitos que comprende el requisito Gestionar Depósito a Plazo y subsistema Créditos que agrupa a Gestionar Préstamo Recibido y Concedido. Vale destacar que para su construcción se tuvo en cuenta la reducción a la mínima expresión de los campos nulos.

Teniendo en cuenta el requisito funcional Gestionar Depósito a Plazo y los que forman parte de él, se crea la tabla **o_deposito_a_plazo** en la que se almacenan los datos de los depósitos unas vez que son contabilizados, ésta a su vez posee relación de no identificación de multiplicidad uno a uno con la tabla **o_calendario** definida para el componente Calendario.

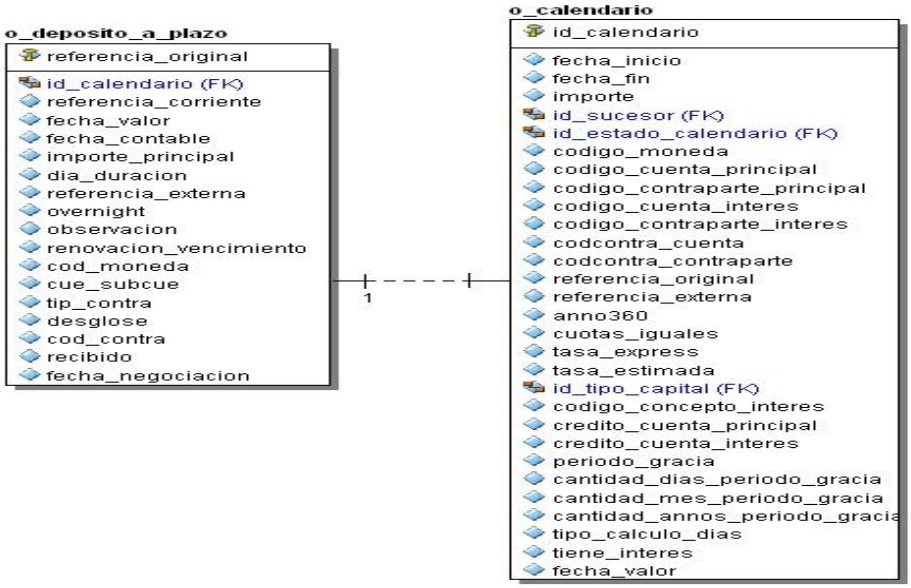

Figura 6: Modelo de datos del Subsistema Depósitos.

Atendiendo a los requisitos funcionales Gestionar Préstamo Recibido y Gestionar Préstamo Concedido y los que los que los componen a ellos, se crea primeramente la tabla **o_prestamo**, esta forma parte como padre de una relación de herencia, que incluye como hijas a las tablas **o_prestamo_concedido** y **o_prestamo_recibido**. Como la relación lo indica **o_prestamo** será utilizado para persistir los datos que son comunes a los dos tipos de préstamos y el caso de datos particulares se almacenarán en las especializaciones mencionadas con anterioridad. Además presenta relación de no identificación de multiplicidad uno a uno con la tabla **o_calendario** definida para el componente Calendario así como una relación del mismo tipo de multiplicidad uno a muchos con el nomenclador **n_tipo_respaldo**.

n_tipo_prestamista surge por una necesidad de BNC de contabilizar en diferentes cuentas el préstamo recibido en dependencia del tipo de prestamista pudiendo ser este cualquier banco, banco Central, el Estado y Sindicado para los préstamos de múltiples prestamistas.

c_medpag, contiene los medios de pago que se usan en BNC, ejemplos de ellos son las transferencias, cheques, débito automático.

n_tipo_prestamo, se crea por la necesidad de BNC de contabilizar en diferentes cuentas el préstamo concedido en dependencia de la forma del préstamo que se concede, pudiendo ser: concedido como la forma típica, recibido/concedido, para aquellos préstamos donde inmediatamente que se reciben los fondos se conceden y sindicado para cuando intervienen múltiples bancos en la concesión del préstamo.

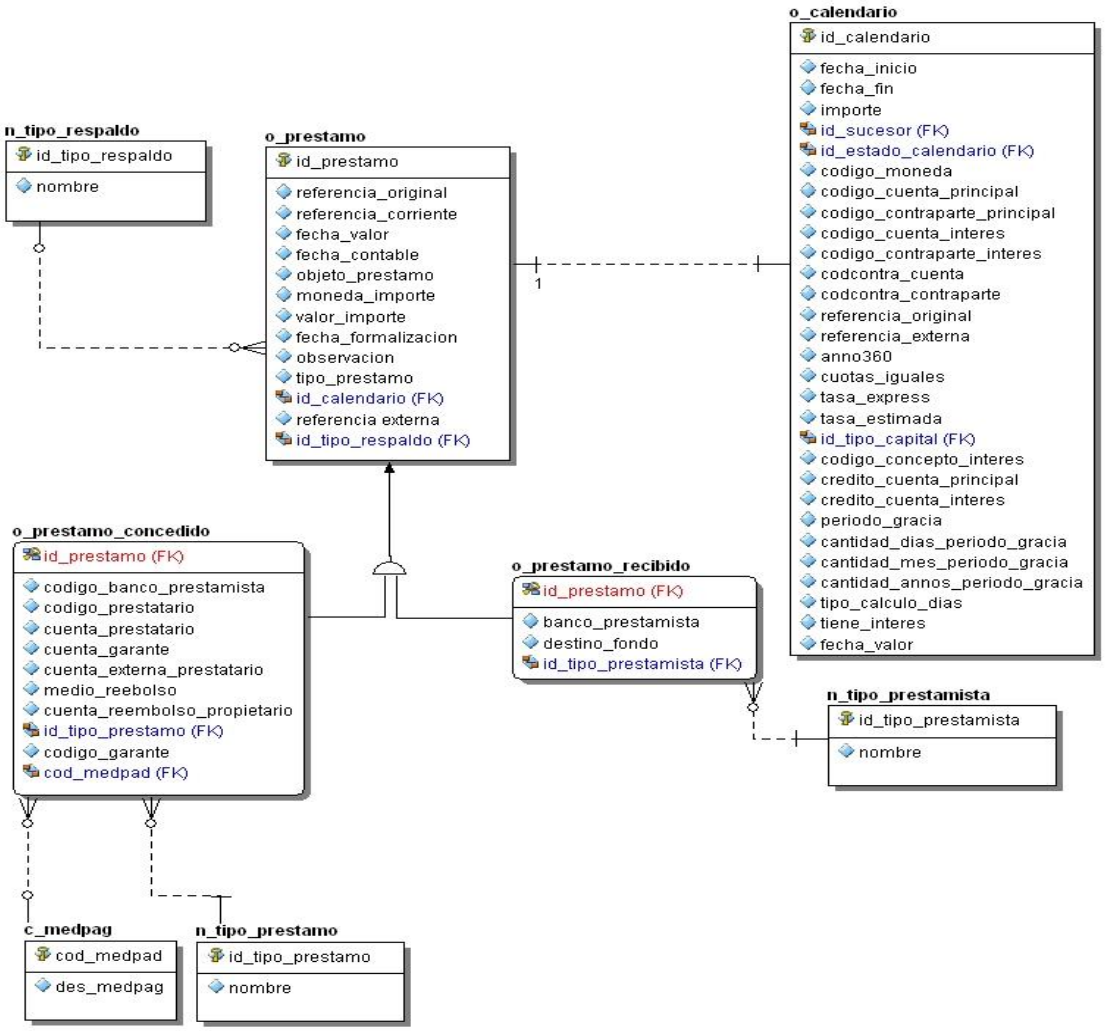

Figura 7: Modelo de datos del Subsistema Créditos.

2.2.3 Patrones de diseño empleados

El diseño fue elaborado siguiendo patrones basados en la experiencia, que de manera general

constituyen soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Los patrones que se utilizaron son los siguientes:

✓ **Controlador:** La clase controladora CargarDatosMultiAction, constituye un ejemplo de la aplicación de este patrón, la misma tendrá en cuenta la responsabilidad de manejar los eventos que consisten en cargar los datos que serán mostrados al usuario.

✓ **Experto:** Dicho patrón es evidenciado en la definición de las clases de acuerdo a las funcionalidades que deben realizar a partir de la información que manejan. Específicamente: la clase DepositoPlazoDAO, será la responsable de efectuar las operaciones que conciernen a las funciones: insertar, eliminar, actualizar y consultar los depósitos. Sobre este mismo principio se realiza el diseño de las restantes funcionalidades.

✓ **Alta cohesión:** Este patrón fue utilizado en el diseño del componente de manera general; donde se agruparon las clases en dependencia de los requerimientos (Gestionar Depósitos a Plazo, Gestionar Préstamos Recibidos y Concedidos) a los que se les debía dar respuesta, según la premisa de que cada clase debe implementar las operaciones que estén sobre la misma área funcional.

✓ **Bajo acoplamiento:** Este patrón se evidencia con la definición de interfaces e implementaciones, como puede ser la interfaz DepositosPlazoFacade y su implementación, que permiten que CargarDatosMultiAction y DepositosPlazoMultiAction, clases de la presentación se relacionen únicamente con ellas para realizar sus operaciones, reduciendo el impacto de cambios posteriores en el negocio del sistema.

✓ **Fachada:** La utilización de este patrón se evidencia en la definición de la interfaz DepositosPlazoFacade y su implementación responsables de la comunicación entre la presentación y el grupo de clases e interfaces más complejas que se encargan de la lógica de negocio y el acceso a datos.

✓ **DAO:** Su utilización se evidencia con la definición de la interfaz DepositoPlazoDAO y la clase DepositoPlazoDAOImpl en las que se proveen funcionalidades específicas a realizar sobre la base de

datos, de manera que se aísla al negocio de los posibles cambios que se puedan producir en la lógica de acceso a datos y la fuente de datos.

2.3 Conclusiones del capítulo

La Arquitectura Base definida y los artefactos generados como resultado de la especificación de los requisitos correspondientes a los procesos préstamos y depósitos, fueron la base para la realización del presente capítulo, a partir de los cuales fue posible llevar a cabo un diseño flexible y escalable mediante el correcto uso de patrones, con el fin de responder a las funcionalidades clave en la gestión de las operaciones asociadas a dichos procesos y proporcionar una entrada apropiada como punto de partida a las actividades de implementación.

Capítulo 3: Implementación y validación de la solución.

3.1 Introducción

El presente capítulo estará enfocado a dar cumplimiento a dos flujos de gran importancia en el proceso de desarrollo de software. En un primer momento se llevará a cabo la implementación de los subsistemas Créditos y Depósitos, donde se especificarán los estándares de codificación definidos para la escritura del código, se representará la interacción de los componentes mediante el Diagrama de Componentes y serán detallados los métodos y atributos de las clases más importantes del modelo de diseño presentado en el capítulo anterior, seguido de una breve descripción del funcionamiento de Spring WebFlow a partir de un escenario del sistema.

Para finalizar el capítulo se dará paso a la validación de la solución mediante la aplicación de las pruebas de unidad en aras de localizar errores que imposibiliten el cumplimiento de los requisitos funcionales y las perspectivas del cliente.

3.2 Implementación

La implementación constituye uno de los flujos de trabajo más importantes propuestos por RUP, en ella se toma como punto de partida lo arrojado como resultado en el diseño y se implementa el sistema en términos de componentes como ficheros de código binario, código fuente, scripts, ejecutables, entre otros. Su importancia se debe a que se obtiene como consecuencia un sistema ejecutable, siendo uno de los principales objetivos en el desarrollo de software.

3.2.1 Estándares de codificación

Los estándares de codificación se definen por el equipo de desarrollo para lograr estandarización en la programación del software. Estos se basan en la estructura y apariencia física de un programa con el fin de facilitar la lectura, comprensión, mantenimiento del código, reutilización a lo largo del proceso de desarrollo de un software y no en la lógica del programa. La generalización de aspectos tan simples como el trato de las mayúsculas, ayuda a eliminar conflictos de funcionalidades implementadas con nombres iguales y guían de forma clara el proceso de desarrollo.

3.2.1.1 Convenciones de nomenclatura

La nomenclatura de las clases está definida por la utilización de la notación Pascal Casing, la cual define que los nombre e identificadores pueden estar compuestos por múltiples palabras juntas y la primera letra de cada palabra irá siempre en mayúsculas además se obvia el uso de artículos.

Ejemplo: DepositosPlazoManager. En este caso el nombre de clase está compuesto por 3 palabras iniciadas cada una con letra mayúscula.

También se tomó en cuenta para el nombrado de las clases el tipo que esta posee, entiéndase como tipo el rol que ellas desempeñan en el sistema. A continuación se presentan las nomenclaturas organizadas por los paquetes a los que pertenecen las clases.

controller: Las clases incluidas en este paquete, después del nombre se le incorpora el nombre del controlador de Spring del cual hereda. Ejemplo: CargarDatosMultiActionController.

command: Las clases que se ubican dentro de este paquete se nombran con el nombre de la clase más la palabra Command.

Ejemplo: PrestamoConcedidoCommand.

validator: En este paquete la nomenclatura de las clases está determinada por el nombre de estas más la palabra Validator.

Ejemplo: DepositosPlazoValidator.

propertyEditor: Las clases que se encuentran dentro de propertyEditor después del nombre se le agrega la palabra PropertyEditor. Ejemplo: MonedaPropertyEditor.

flowHandler: En este paquete a las clases después del nombre se les coloca la palabra FlowHandler. El nombre responde a lo que realiza el flujo que se quiere personalizar. Ejemplo: ConsultarDepositosFlowHandler

serviceFlow: Al nombre de las clases que están dentro de dicho paquete se le agrega la palabra Action o MultiAction en dependencia de cuál de las dos clases herede.

Ejemplo: DepositosPlazoMultiAction.

facade: Las clases incluidas en este paquete, después del nombre se le agrega la palabra Facade y en el caso de subpaquete impl tendrán el mismo nombre de la interfaz que implementen más la palabra Impl.

Ejemplo: DepositosPlazoFacade, DepositosPlazoFacadeImpl.

manager: Las clases que se encuentran dentro de manager después del nombre llevan la palabra Manager y en el caso de subpaquete impl tendrán el mismo nombre de la interfaz que implementen más la palabra Impl. Ejemplo: DepositosPlazoManager y DepositosPlazoManagerImpl.

dao: Las clases incluidas en el paquete dao, después del nombre se le incorpora la abreviatura DAO y en el caso de subpaquete impl tendrán el mismo nombre de la interfaz que implementen más la palabra Impl.

Ejemplo: DepositoPlazoDAO, DepositoPlazoDAOImpl.

De manera general el nombre de los métodos y los atributos de las clases se escriben con la inicial del identificador en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing, que es muy similar a Pascal Casing con la excepción de que la letra inicial comienza con minúsculas. Lo mismo se aplica a los nombres de ficheros de código JavaScript y sus funciones y variables internas.

Los comentarios deben ser lo más claros y precisos posibles de forma tal que se entienda el propósito de lo que se está desarrollando. En caso de ser una función complicada se debe comentar para lograr una mejor comprensión del código.

3.2.2 Modelo de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Los componentes

representan todos los tipos de elementos del software que entran en la fabricación de aplicaciones informáticas. (25)

El Modelo de componentes que se muestra a continuación se ha elaborado de forma tal que muestra las relaciones existentes entre los subsistemas Créditos y Depósitos y el resto de componentes dentro del sistema.

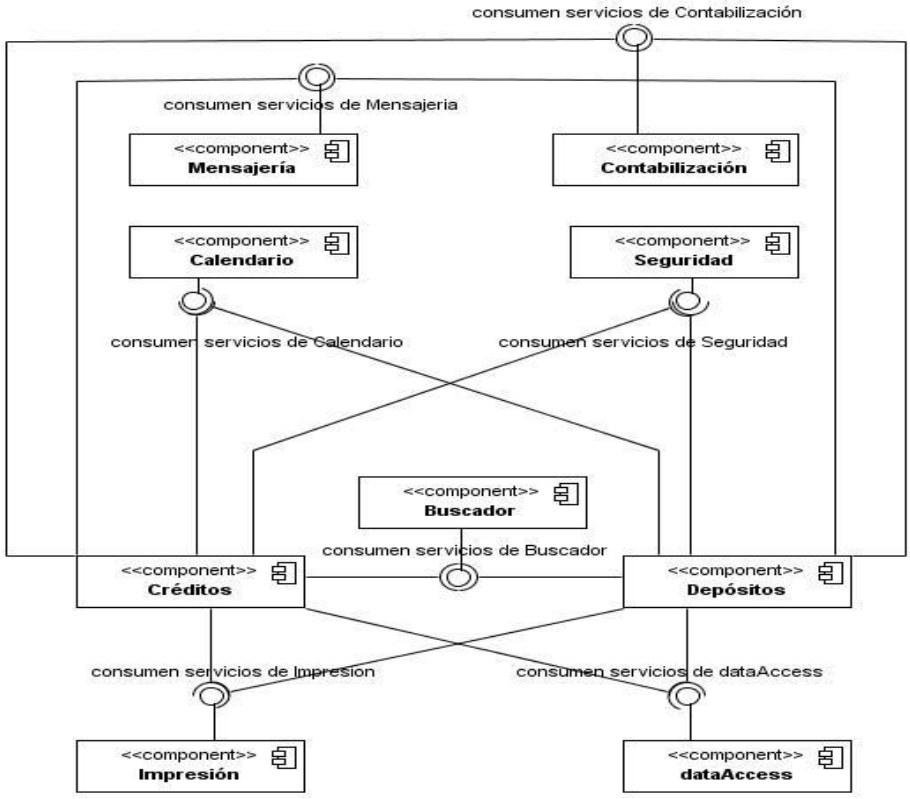

Figura 8: Modelo de componentes.

A continuación se explican de manera general cada uno de los componentes.

El componente **Calendario** brinda un conjunto de clases además de una interfaz gráfica, responsables de gestionar la creación de las formas de pago y los vencimientos tanto para los pagos principales como para los intereses, relacionada con una determinada operación bancaria. Las formas de pago, específicamente, definen los períodos en los que se harán los pagos ya sean diarios, mensuales, trimestrales, etc. y los vencimientos, por su parte, determinan los montos y la fecha en que se

realizarán. Es necesaria la utilización de este componente para precisar el tiempo, los períodos y los montos en los que serán pagados los préstamos y los depósitos concedidos o recibidos por el Banco Nacional de Cuba.

El componente **Buscador** ofrece un conjunto de clases que constituyen el motor de búsqueda, presentando una interfaz gráfica mediante la cual se solicitan los diferentes conceptos en dependencia del módulo donde se emplee. Para la correcta gestión de la información en el subsistema Créditos se hace necesaria la búsqueda tanto de préstamos recibidos como concedidos y en Depósitos se procede de igual forma para los depósitos a plazos.

En el caso del componente **Contabilización** se proponen un grupo de clases necesarias para realizar la contabilización de determinadas operaciones que así lo requieran y otro grupo de clases y una interfaz gráfica para el cobro de comisiones que se asocien a determinada operación. La realización de préstamos y depósitos constituyen procesos que requieren contabilización, además de ser posible la asociación de comisiones, de aquí la relación con este componente.

Mensajería es el componente encargado de conceder las clases necesarias para el envío de mensajes SWIFT tras la realización de operaciones bancarias que requieran la notificación a los bancos implicados en ella. Los subsistemas Créditos y Depósitos realizan operaciones que por su naturaleza requieren de la notificación mediantes mensajes por lo cual es necesario el empleo del componente.

El componente **Seguridad** como su nombre lo indica es el encargado, entre otros aspectos, de mantener la seguridad en todo el sistema a través de la comprobación de las peticiones y sus permisos en dependencia del usuario que la realice, de aquí que sea necesaria la comunicación con el mismo para garantizar la seguridad de los subsistemas Créditos y Depósitos.

Impresión consiste en un componente que tiene como función brindar las clases que contienen las funcionalidades mediante las cuales es posible imprimir determinada información y de forma automática las operaciones que realicen contabilización. Específicamente los subsistemas Créditos y Depósitos engloban funcionalidades que requieren de la contabilización.

Por otro lado el **dataAccess** tiene la función de brindar todos los elementos necesarios para llevar a cabo la conexión a la base de datos y por ende todos los subsistemas dependen de él para poder realizar las funcionalidades que engloban.

3.2.3 Descripción de las clases y las funcionalidades

Teniendo en cuenta el diseño de las clases correspondientes al subsistema Depósito realizado en el capítulo anterior, a continuación serán descritos los atributos y métodos de aquellas que son más importantes para el sistema desde el punto de vista funcional.

Tabla 1: Descripción de la clase DepositosPlazoMultiAction.

Nombre: DepositosPlazoMultiAction	
Tipo de clase: Controladora	
Atributo	**Tipo**
globalFacade	GlobalFacade
nomencladorContabilizacionFacade	NomencladorContabilizacionFacade
contabilizarFacade	ContabilizarFacade
depositoPlazoFacade	DepositosPlazoFacade
Para cada responsabilidad:	
Nombre:	**Descripción**:
establecer(RequestContext context)	Permite obtener el objeto Depósito a Plazo que servirá de modelo a la vista, en caso de que este se vaya a registrar, actualizar o consultar.
primeraValidacion(RequestContext context)	Permite invocar la primera validación de los asientos a contabilizar y en caso de obtener algún mensaje de error o alerta indicárselo al flujo.
segundaValidacion(RequestContext context)	Invoca una segunda validación de los asientos e indica al flujo la presencia o no de mensajes de

	error o alerta.
contabilizar(RequestContext context)	Envía a contabilizar los asientos y en caso de existir errores de contabilización alerta al flujo de la presencia de estos.
hayMensajeria(RequestContext context)	Permite conocer si existe mensajería en la operación que se está realizando.
verificarError(RequestContext context, Object object)	Permite conocer si existieron errores en el proceso de mensajería.
saveCalendarioInConversation(RequestContext context)	Salva el calendario de pagos que tiene el objeto Depósito a Plazo de forma tal que el componente calendario pueda acceder a este.
saveCalendario(RequestContext context)	Permite actualizarle el calendario de pagos al Depósito a Plazo una vez fue modificado por el componente Calendario.
construirAsientos(RequestContext context)	Construye los asientos contables de la operación en curso.

Tabla 2: Descripción de la clase DepositosPlazoManagerImpl.

globalFacade	GlobalFacade
nomencladorFacade	NomencladorFacade
contabilizarFacade	ContabilizarFacade
calendarioFacade	CalendarioFacade
depositoPlazoDAO	DepositoPlazoDAO
Para cada responsabilidad:	
Nombre:	**Descripción**:
obtenerDepositoPlazo(String refOriginal)	Obtiene un Depósito a Plazo según la referencia original

primeraValidacion(RequestContext context)	Solicita a ContabilizarFacade que realice la primera validación de los asientos.
segundaValidacion(RequestContext context)	Solicita a ContabilizarFacade que realice la segunda validación de los asientos.
actualizarDepositoPlazo(DepositoPlazo depo, String operador,	Actualiza un Depósito a Plazo y en caso de tener asientos contables los contabiliza
esCortoPlazo(DepositoPlazo depo)	Indica si el Depósito a Plazo es a corto plazo.
crearTransaccionRegistrarDepositoAPlazo(DepositoPlazo depo, String codOper)	Agrupa la información necesaria y solicita a CalendarioFacade que construya los asientos contables para registrar un Depósito a Plazo
registrarDepositoPlazo(DepositoPlazo depo, String operador, List<Asiento> asientos)	Registra un Depósito a Plazo y contabiliza los asientos contables del mismo.
cancelarDepositoPlazo(DepositoPlazo depo, String operador, List<Asiento> asientos)	Cancela un Depósito a Plazo.
contabilizar(List<Asiento> asientos, String codigoOperador)	Solicita a ContabilizarFacade que contabilice los asientos.
crearTransaccionActualizarDepositoAPlazo(DepositoPlazo depo, String codOper)	Agrupa la información necesaria y solicita a CalendarioFacade que construya los asientos contables para actualizar un Depósito a Plazo.
crearTransaccionCancelarDepositoAPlazo(DepositoPlazo depo, String codOper)	Agrupa la información necesaria y solicita a CalendarioFacade que construya los asientos contables para cancelar un Depósito a Plazo.
calzarMonedas(List<Asiento> asientos)	Encargado de solicitar a ContabilizacionFacade que calce las monedas de los asientos si existe descuadre en ellas.
obtenerEstadoSistema()	Obtiene el estado en el que se encuentra el sistema.

obtenerReferenciaCorriente(String Letra)	Obtiene la referencia corriente dada una letra, desde ContabilizaFacade.
incrementarDesglose(String refCorriente)	Incrementa el desglose de la referencia corriente para poder reutilizar la misma.
listarMonedas()	Lista las monedas desde NomnecladoFacade.
obtenerMoneda(String codigo)	Obtiene una moneda dado el código de esta, desde Nomenclador Facade.
listarCuentasDepositoPlazo(boolean recibido)	Lista las cuentas para en dependencia del si se recibe o se concede el Depósito a Plazo.
obtenerCuenta(String codigo)	Obtiene una cuenta dado el código de esta, desde Nomenclador Facade.
obtenerCentroCosto()	Obtiene el centro de costo desde ContabilizarFacade.

Tabla 3: Descripción de la clase DepositoPlazoDAOImpl

Nombre: DepositoPlazoDAOImpl	
Tipo de clase: Data Access Object	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**Descripción**:
listarDepositosPlazoActivos()	Se encarga de recuperar de la base de datos los Depósitos a Plazo que estén en estado inactivos.
limpiarDepositoDeSesion(DepositoPlazo deposito)	Retira de la sesión de Hibernate el objeto Depósito a Plazo.
reattach(DepositoPlazo deposito)	Bloquea en la base de datos el Depósito a Plazo con el que se esté trabajando.
depositoInactivo(final String refOriginal)	Permite conocer dada la referencia original si el

	Depósito a Plazo está inactivo.

3.2.4 Aspectos Principales de la Implementación

Utilización de Spring WebFlow Framework

A lo largo del desarrollo del presente trabajo, se ha hecho énfasis en la utilización de Spring WebFlow por las facilidades que brinda en el desarrollo, principalmente en escenarios donde existen flujos complejos de vistas e información. Para el desarrollo de los subsistemas Créditos y Depósitos se emplea este framework en todas las funcionalidades que sus módulos comprenden, promovido por la necesidad de interactuar con una serie de componentes como es el caso de Calendario y Comisiones dentro Contabilización.
A continuación se describe el funcionamiento del mismo desde el módulo Gestionar Depósitos a Plazo:

El principal aspecto de este subsistema es la existencia de un flujo común capaz de redireccionar a diferentes estados de vista y trabajar con el mismo objeto DepositoPlazo en diferentes fases, al mismo tiempo agrupa la implementación de casos de uso agrupados por el patrón CRUD.

Por cada funcionalidad del CRUD: Registrar, Actualizar y Consultar se creó un flujo que inicia con la siguiente declaración:

```
<var name="depositosaplazos"
class="cu.uci.finixubnc.depositosaplazos.gestionardepositosaplazo.domain.DepositoPlazo" />

<on-start>
<evaluate expression="depositosPlazoMultiAction.establecer" />
</on-start>

<subflow-state id="gestionarDepositos-flow" subflow="gestionarDepositos-flow">
    <input name="depositosaplazos" />
    <input name="action" />
    <transition on="end" to="end"/>
</subflow-state>
```

Se declara la variable ***despositosaplazos*** que representa el modelo del que se estará haciendo uso en la vista, el cual será modificado en dependencia de la funcionalidad que se esté realizando con la llamada al método ***establecer(RequestContext context)*** en la etiqueta ***on-start***.

Todas las funcionalidades presentes en el flujo son implementadas en la clase DepositosPlazoMultiAction, la cual se describió con anterioridad. Nótese que estas funcionalidades

declaradas en el flujo no necesitan declararse con los parámetros correspondientes, debido a que sí reciben el RequestContext como único parámetro, Spring WebFlow es capaz de reconocerlo.

Posteriormente se hace entrada al flujo común ***gestionarDepositos-flow*** mediante la etiqueta ***subflow*** al cual es necesario transferirle las variables ***despositosaplazos*** y ***action*** que se encuentra en el contexto y define la operación que se está realizando.

Una vez dentro del flujo común se ejecutan los estados de decisión ***esRegistrar***, ***esActualizar*** los cuales son los encargados de decidir a qué estado de vista se moverá el flujo en dependencia de la variable ***action***.

```
<decision-state id="esRegistrar">
    <if test="flowScope.action.equals('registrar')" then="gestionarDepositos"
        else="esActualizar" />
</decision-state>

 <decision-state id="esActualizar">
    <if test="flowScope.action.equals('actualizar')" then="actualizarDepositos"
        else="consultarDepositos" />
</decision-state>
```

En el caso de que se vaya a registrar un Depósito a Plazo el flujo se detiene en el estado de vista ***gestionarDepositos***, el cual muestra al usuario la página cliente ***registrarDepositoAPlazo*** al mismo tiempo que contiene un grupo de transiciones que entrarán en acción a partir de los posibles eventos que se puedan generar por el usuario.

```
<view-state id="gestionarDepositos"  view="registrarDepositoAPlazo" model="depositosaplazos">
    <transition on="salvar" to="validar"/>
    <transition on="calendarioCobro" to="saveCalendario" />
    <transition on="verAsientos" to="construirAsientos" />
    <transition on="aceptarM" to="hayMensajeria"/>
    <transition on="terminar" to="end" />
    <transition on="cancelar" to="end"/>
    <transition on="actualizarDatosComisiones" validate="false" bind="false" to="gestionarDepositos">
        <evaluate expression="comisionesMultiAction.actualizarDatosComisiones" />
    </transition>
    <transition on="gestionarComision" validate="false" to="gestionarComisionSubflow">
    <evaluate expression="comisionesMultiAction.registrarCallMetodo"></evaluate>
    <evaluate expression="depositosPlazoMultiAction.setear"></evaluate>
    </transition>
</view-state>
```

La generación del evento ***calendarioCobro*** indica que habrá interacción con el componente Calendario, en ese momento se realiza la acción ***saveCalendario*** encargada de invocar al método

saveCalendarioInConversation y redireccionar las acciones para el flujo ***calendario-flow*** brindado por el componente Calendario.

```
<action-state id="saveCalendario">
       <evaluate expression="depositosPlazoMultiAction.saveCalendarioInConversation"/>
        <transition on="success" to="calendarioCobro"/>
</action-state>

<subflow-state id="calendarioCobro" subflow="calendario-flow">
  <transition on="end" to="esRegistrar">
    <evaluate expression="depositosPlazoMultiAction.saveCalendario"/>
  </transition>
</subflow-state>
```

El flujo que brinda el calendario es global para la aplicación y su definición se encuentra dentro del componente, por lo que es necesario configurar su registro en el contexto de SpringWebFlow del módulo, esto es válido para todos los flujos externos al módulo, a continuación se muestra la definición:

```
<webflow:flow-registry id="calendarioflowRegistry"
    flow-builder-services="calendarioflowBuilderServices" parent="comisionesFlowRegistry">
    <webflow:flow-location-pattern
        value="classpath:cu/uci/finixubnc/common/calendario/configuration/flow/calendario-flow.xml" />
</webflow:flow-registry>
```

A partir del resultado de la petición que se le realiza al componente Calendario, y si fuese necesaria al componente Comisiones, se reúne parte de la información que debe comprender el depósito a plazo a registrar.

Si desde la página que se muestra al usuario se ejecuta el evento ***salvar*****,** se desencadenaran una serie de estados de acción y de decisión que manejaran las validaciones de los asientos, la contabilización y la mensajería, que culminaran en un estado de finalización; ***end-state.***

Una vez finalizadas las operaciones en el flujo común el mando es devuelto al flujo que lo haya invocado, estos a su vez pasan a su estado de finalización donde está definida la página cliente con la que se dará por terminada la funcionalidad.

3.3 Validación de la implementación de los subsistemas

La calidad de un producto de software; conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia; se ha convertido en un elemento estratégico de las grandes organizaciones debido

a su fuerte impacto en la competitividad de las empresas. Durante el proceso de desarrollo de software las posibilidades de errores son múltiples por lo que se hace necesario detectar a tiempo las imperfecciones e irregularidades y proporcionar una visión objetiva de la madurez y calidad de los procesos asociados.

El aspecto fundamental que rige esta etapa es determinar mediante las pruebas, cómo y en qué sentido los subsistemas Crédito y Depósitos cumplen con las expectativas del cliente, a partir de los requisitos establecidos y las restricciones impuestas. En ese ámbito se trazan un conjunto de objetivos dentro de los que se sitúan:

- ✓ Verificar la implementación de los subsistemas.
- ✓ Verificar que todos los requisitos se han implementado correctamente.
- ✓ Identificar los errores y asegurar que estos sean corregidos de la mejor manera.

3.3.1 Pruebas de Software

Conjunto de técnicas experimentales para la búsqueda de fallos en los programas, que determinan en cierto grado la calidad de un producto.

Las pruebas que se le realizaron a los subsistemas corresponden al nivel de Unidad, las cuales están enfocadas al código fuente de los componentes para verificar todos los flujos de control, probando de manera individual las partes del sistema que han sido desarrolladas. Adecuadas a este nivel se aplicaran los métodos de prueba Caja Blanca; para analizar la lógica interna del programa y Caja Negra con el objetivo de verificar que las funciones son operativas a través de la interfaz del software, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, manteniendo la integridad de la información externa.

3.3.1.1 Aplicación de la Prueba de Caja Blanca o Estructural

Esta prueba consiste específicamente en diseñar los Casos de prueba [1]atendiendo al comportamiento interno y la estructura del programa, examinándose la lógica interna sin considerar los aspectos de rendimiento. Asociadas a este método existen cuatro técnicas de prueba; Condición, Flujo de Datos, Bucles y Camino Básico, de las cuales será aplicada esta última, debido a que permite obtener una medida de la complejidad lógica del diseño y usar la misma como guía para la definición de un conjunto de caminos básicos, garantizando que durante la prueba se ejecute al menos una vez cada sentencia del programa.

Para ello es necesario conocer el número de caminos independientes de un determinado algoritmo mediante el cálculo de la complejidad ciclomática. Se debe comenzar por un análisis del código, posteriormente son enumeradas cada una de las instrucciones, se construye el grafo de flujo asociado y según las fórmulas pertinentes se calcula dicha complejidad.

A continuación se analizan y enumeran las sentencias de código de uno de los métodos contenidos en la clase PrestamoRecibidoManagerImpl, específicamente: estimarCuentas, este por su parte determina las cuentas que serán utilizadas en la contabilización en dependencia de las características del Préstamo Recibido. La funcionalidad es invocada para contribuir a la construcción de los asientos contables en esa propia clase.

[1] Casos de prueba: especifican una forma de probar el componente, incluye: la entrada, las condiciones bajo las cuales ha de probarse y los resultados esperados.

```
private List<Cuenta> estimarCuentas(PrestamoRecibidoCommand pr,
        StringBuffer codigo) {
    List<Cuenta> cuentas = new ArrayList<Cuenta>(); //1
    Cuenta contrapartePrincipal = nomencladorContabilizacionFacade.consultarCuenta("1210"); //1
    Cuenta contraparteIntereses = nomencladorContabilizacionFacade.consultarCuenta("5100"); //1
    Cuenta cuentaPrincipal; //1
    Cuenta cuentaIntereses; //1
    if (calcularDias(pr.getCalendario().getFechaFin(), pr.getCalendario().getFechaInicio()) < 365) { //2
        if (pr.getTipoPrestamista().equals("1")) {  //3
            cuentaPrincipal = nomencladorContabilizacionFacade.consultarCuenta("3411");  //4
            cuentaIntereses = nomencladorContabilizacionFacade.consultarCuenta("3412"); //4
            codigo.append("0101");//4
        }
        else
            if (pr.getTipoPrestamista().equals("2")) {  //5
            cuentaPrincipal = nomencladorContabilizacionFacade.consultarCuenta("3413"); //6
            cuentaIntereses = nomencladorContabilizacionFacade.consultarCuenta("3414"); //6
            codigo.append("0104"); //6
        }
            else
        {
            cuentaPrincipal = nomencladorContabilizacionFacade.consultarCuenta("3431"); //7
            cuentaIntereses = nomencladorContabilizacionFacade.consultarCuenta("3432"); //7
            codigo.append("0101"); //7
        }
    } //8
    else {
        if (pr.getTipoPrestamista().equals("1")) { //9
            cuentaPrincipal = nomencladorContabilizacionFacade.consultarCuenta("3421"); //10
            cuentaIntereses = nomencladorContabilizacionFacade.consultarCuenta("3422"); //10
            codigo.append("0101"); //10
        }
        else
            if (pr.getTipoPrestamista().equals("2")) { //11
            cuentaPrincipal = nomencladorContabilizacionFacade.consultarCuenta("3425"); //12
            cuentaIntereses = nomencladorContabilizacionFacade.consultarCuenta("3426"); //12
            codigo.append("0104"); //12
        }
            else if (pr.getTipoPrestamista().equals("3")) { //13
            cuentaPrincipal = nomencladorContabilizacionFacade.consultarCuenta("3441"); //14
            cuentaIntereses = nomencladorContabilizacionFacade.consultarCuenta("3442"); //14
            codigo.append("0101"); //14
        }
            else {
            cuentaPrincipal = nomencladorContabilizacionFacade.consultarCuenta("3461"); //15
            cuentaIntereses = nomencladorContabilizacionFacade.consultarCuenta("3462"); //15
            codigo.append("0103"); //15
        }
    }//16
    cuentas.add(cuentaPrincipal); //17
    cuentas.add(contrapartePrincipal); //17
    cuentas.add(cuentaIntereses); //17
    cuentas.add(contraparteIntereses); //17
    return cuentas; //17

}
```

Figura 9: Método estimarCuentas.

Después de este paso, es necesario representar el grafo de flujo asociado al código antes presentado a través de nodos, aristas y regiones, en ese caso:

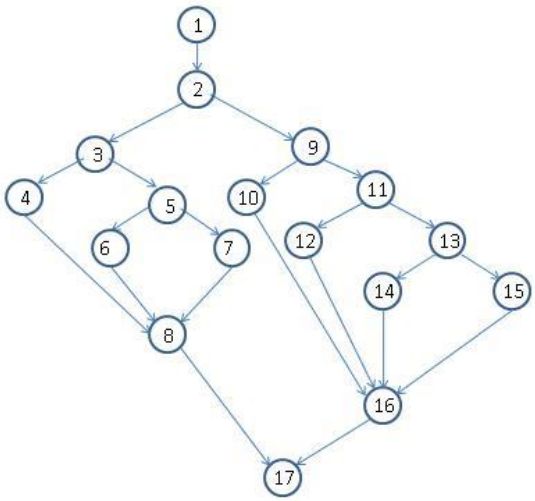

Figura 10: Grafo de flujo asociado al algoritmo estimarCuentas.

Una vez construido el grafo de flujo asociado al procedimiento anterior se determina la complejidad ciclomática, el cálculo es necesario efectuarlo mediante tres vías o fórmulas de manera tal que quede justificado el resultado, siendo el mismo en cada caso:

1. V (G) = (A – N) + 2

Siendo “A” la cantidad total de aristas y “N” la cantidad total de nodos.

V (G) = (22 -17) + 2

V (G) = 7.

2. V (G) = P + 1

Siendo “P” la cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas).

V (G) = 6 + 1

V (G) = 7.

3. V (G) = R

Siendo “R” la cantidad total de regiones, se incluye el área exterior del grafo, contando como una región más.

V (G) = 7

El cálculo efectuado mediante las fórmulas antes presentadas muestran una complejidad ciclomática de valor 7, de manera que existen siete posibles caminos por donde el flujo puede circular, este valor representa el número mínimo de casos de pruebas para el procedimiento tratado.

Seguidamente es necesario especificar los caminos básicos que puede tomar el algoritmo durante su ejecución.

Camino básico #1: 1 – 2 – 3 – 4 – 8 – 17
Camino básico #2: 1 – 2 – 3 – 5 – 6 – 8 - 17
Camino básico #3: 1 – 2 – 3 – 5 – 7 – 8 – 17
Camino básico #4: 1 – 2 – 9 – 10 – 16 – 17
Camino básico #5: 1 – 2 – 9 – 11 – 12 – 16 – 17
Camino básico #6: 1 – 2 – 9 – 11 – 13 – 14 – 16 – 17
Camino básico #7: 1 – 2 – 9 – 11 – 13 – 15 – 16 – 17

Se procede a ejecutar los casos de pruebas para cada uno de los caminos básicos determinados en el grafo de flujo. Para definir los casos de prueba es necesario tener en cuenta:

Descripción: Se describe el caso de prueba y de forma general se tratan los aspectos fundamentales de los datos de entrada.

Condición de ejecución: Se especifica cada parámetro para que cumpla una condición deseada y así ver el funcionamiento del procedimiento.

Entrada: Se muestran los parámetros que serán la entrada al procedimiento.

Resultados Esperados: Se expone el resultado esperado que debe devolver el procedimiento después de efectuado el caso de prueba.

De forma general:

El tipo de prestamista tiene cuatro variantes: Banco, Banco Central, Sindicado, Estado.

Tipo Prestamista = 1 significa que es Banco.

Tipo Prestamista = 2 significa que es Banco Central.

Tipo Prestamista = 3 significa que es Sindicado.

Tipo Prestamista = 4 significa que es Estado.

Las descripciones y las condiciones de ejecución coincidirán en cada uno de los casos de prueba.

1- Caso de prueba para el camino básico # 1.

Descripción: Se deben determinar con exactitud las cuentas que estarán implicadas en la construcción de los asientos contables y posteriormente en la contabilización.

Condición de ejecución: Para el algoritmo es necesario que las fecha de inicio y fin del calendario asociadas al préstamo sean valores de tipo **Date** de JAVA, asimismo los campos correspondientes al Tipo de Prestamista no pueden estar vacíos.

Entrada:
Préstamo Recibido con calendario que contiene fechas de inicio y fin:
Fecha Inicio = 14/4/2011
Fecha Fin = 24/4/2011
Tipo de Prestamista = 1
Resultados esperados: Teniendo en cuenta los datos pasados por parámetro se espera que las cuentas estimadas sean (3411, 1210, 3412, 5100).
El resultado obtenido fue correcto.

2- Caso de prueba para el camino básico # 2.
Entrada:
Préstamo Recibido con calendario que contiene fechas de inicio y fin:
Fecha Inicio = 14/4/2011
Fecha Fin = 24/4/2011
Tipo de Prestamista = 2
Resultados esperados: Teniendo en cuenta los datos pasados por parámetro se espera que las cuentas estimadas sean (3413, 1210, 3414, 5100).

El resultado obtenido fue correcto.

3- Caso de prueba para el camino básico # 3.

Entrada:

Préstamo Recibido con calendario que contiene fechas de inicio y fin:

Fecha Inicio = 14/4/2011

Fecha Fin = 24/4/2011

Tipo de Prestamista = 3

Resultados esperados: Teniendo en cuenta los datos pasados por parámetro se espera que las cuentas estimadas sean (3431, 1210, 3432, 5100).

El resultado obtenido fue correcto.

4- Caso de prueba para el camino básico # 4.

Entrada:

Préstamo Recibido con calendario que contiene fechas de inicio y fin:

Fecha Inicio = 14/5/2009

Fecha Fin = 24/5/2011

Tipo de Prestamista = 1

Resultados esperados: Teniendo en cuenta los datos pasados por parámetro se espera que las cuentas estimadas sean (3421, 1210, 3422, 5100).

El resultado obtenido fue correcto.

5- Caso de prueba para el camino básico # 5.

Entrada:

Préstamo Recibido con calendario que contiene fechas de inicio y fin:

Fecha Inicio = 14/5/2009

Fecha Fin = 24/5/2011

Tipo de Prestamista = 2

Resultados esperados: Teniendo en cuenta los datos pasados por parámetro se espera que las cuentas estimadas sean (3425, 1210, 3426, 5100).

El resultado obtenido fue correcto.

6- Caso de prueba para el camino básico # 6.

Entrada:

Préstamo Recibido con calendario que contiene fechas de inicio y fin:

Fecha Inicio = 14/5/2009

Fecha Fin = 24/5/2011

Tipo de Prestamista = 3

Resultados esperados: Teniendo en cuenta los datos pasados por parámetro se espera que las cuentas estimadas sean (3441, 1210, 3442, 5100).

El resultado obtenido fue correcto.

7- Caso de prueba para el camino básico # 7.

Entrada:

Préstamo Recibido con calendario que contiene fechas de inicio y fin:

Fecha Inicio = 14/5/2009

Fecha Fin = 24/5/2011

Tipo de Prestamista = 4

Resultados esperados: Teniendo en cuenta los datos pasados por parámetro se espera que las cuentas estimadas sean (3461, 1210, 3462, 5100).

El resultado obtenido fue correcto.

3.3.1.2 Aplicación de la Prueba de Caja Negra o Funcional

Este tipo de prueba se centra en lo que se espera del software, es decir, los casos de prueba pretenden demostrar que las funciones del sistema son operativas, que los valores de entradas se aceptan adecuadamente y que se produce una salida correcta, sin preocuparse de lo que pueda estar haciendo el software internamente, es decir, todas las pruebas se realizan sobre la interfaz de usuario.

La prueba de Caja Negra no es una alternativa a las técnicas de prueba de la Caja Blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la Caja Blanca.
Con este tipo de pruebas se intenta encontrar:

- ✓ Funciones incorrectas o ausentes.
- ✓ Errores de interfaz.
- ✓ Errores en estructuras de datos o en accesos a las bases de datos externas.
- ✓ Errores de rendimiento.
- ✓ Errores de inicialización y terminación.

Para el correcto desarrollo de las pruebas existen diferentes técnicas: Partición de equivalencia, Análisis de valores límites y Grafos de Causa-Efecto**.**

Para la realización de las pruebas de caja negra se empleó la técnica Partición de Equivalencia que representa una de las más efectivas pues permite examinar los valores válidos e inválidos de las entradas existentes en el software, descubre de forma inmediata una clase de errores que de otro modo requerirían la ejecución de muchos casos antes de detectar el error genérico. Dirige la definición de casos de pruebas que descubran clases de errores, reduciendo el número de clases de prueba que hay que desarrollar.

Los resultados obtenidos a través de la realización de los métodos de prueba expuestos con anterioridad como parte de las pruebas internas realizadas a los subsistemas, fueron satisfactorios desde el punto de vista interno y funcional, atendiendo al correcto comportamiento del mismo ante diferentes situaciones.

Los subsistemas Créditos y Depósitos una vez probados y liberados por parte del departamento de calidad de la universidad (CALISOFT), entraron a las pruebas de aceptación correspondientes en el Banco Nacional de Cuba, donde fueron aprobados por parte del cliente. De manera general dichos subsistemas se encuentran en completa capacidad para su explotación como parte del sistema Quarxo.

3.4 Conclusiones del capítulo

Una vez desarrolladas las tareas que dan por cumplido este capítulo, se concluye que la solución implementada y posteriormente validada, desde el punto de vista funcional, permite llevar a cabo los procesos relacionados con los préstamos y depósitos bancarios en el BNC, de manera que han sido incorporados los subsistemas Créditos y Depósitos al sistema Quarxo, contribuyendo a la ejecución de las operaciones correspondientes. La solución exhibe valor técnico a partir de la utilización de Spring WebFlow como elemento fundamental, el cual brinda recursos potentes que permitieron entre otros aspectos la integración con componentes de carácter general dentro del sistema.

Conclusiones generales

Una vez terminado el presente trabajo de diploma se puede concluir que se desarrollaron todas las tareas a fin de cumplir los objetivos propuestos y para ello:

✓ Se analizaron ventajas y deficiencias de sistemas informáticos tanto nacionales como internacionales vinculados a la gestión de préstamos y depósitos, donde la dificultad principal para su utilización resultó ser el alto costo de sus licencias. Evidenciándose de esta manera la necesidad de desarrollar una solución informática capaz de ejecutar dichas funcionalidades, que cumpliera con los requisitos establecidos a nivel nacional.

✓ Se realizó el diseño y la implementación de los subsistemas Créditos y Depósitos a partir de los requisitos definidos, con el objetivo de llevar a cabo los procesos de préstamos y depósitos en el BNC mediante los subsistemas del sistema Quarxo para contribuir a erradicar los problemas del SABIC y fusionar sus mejores prácticas.

✓ Se evaluó la viabilidad de los subsistemas Créditos y Depósitos a través de pruebas de software efectuadas para el nivel de unidad, las cuales arrojaron resultados favorables posibilitando dar cumplimiento a las funcionalidades previstas para el mismo.

La propuesta exhibe valor técnico; se destaca la incorporación de principios por los que se mide la factibilidad de un diseño de software como lo es la utilización de patrones, lo que posibilitó la reutilización, la sostenibilidad y el mantenimiento del sistema. La solución es además novedosa, su importancia radica en la realización de los procesos referentes a la gestión de préstamos y depósitos en el BNC a través de subsistemas que funcionan rápido y correctamente, permitiendo el ahorro de tiempo y recursos, así como el control eficiente de los cambios y la información manipulada.

Recomendaciones

- ✓ Aprovechar los estudios realizados en esta investigación para mejorar la solución presentada en futuras versiones que se realicen del sistema.
- ✓ Incrementar el uso de las tecnologías y herramientas presentadas en la investigación en el desarrollo de otros sistemas de gestión bancaria.

Referencias bibliográficas

1- Universidad de las Americas Puebla. [En línea] [Citado el: 20 de febrero de 2011.] http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/marquez_a_bm/capitulo5.pdf.

2- **Pérez, Pedro Piñero.** Metodologías Ágiles y Robustas. 2009.

3- **Ivar Jacobson, Grady Booch, James Rumbaugh**. El Proceso Unificado de Desarrollo de Software. s.l.: Pearson Educación, S.A, 2000. 84-7829-036-2.

4- **Larman, Craig.** UML y Patrones. s.l. : Prentice Hall, 1999. 970-17-0261-1.

5- Desarrollo Web. Usabilidad y arquitectura del software. [En línea] [Citado el: 4 de febrero de 2011.] http://www.desarrolloweb.com/articulos/1622.php.

6- **Iglesias, Adolfo Miguel.** Documento de Arquitectura, Modernización Sistema del Banco Nacional de Cuba. 2009.

7- **Torrijos, Ricard Lou.** Programación en Castellano. *Catálogo de Patrones de Diseño J2EE.* [En línea] [Citado el: 15 de marzo de 2011.] http://www.programacion.com/articulo/catalogo_de_patrones_de_diseno_j2ee__y_ii:_capas_de_negocio_y_de_integracion_243/8.

8- **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** Design Patterns. s.l. : Addison-Wesley Pub Co, 1995.

9- Desarrollo Web. Que es Java? [En línea] [Citado el: 15 de febrero de 2011.] http://www.desarrolloweb.com/articulos/497.php.

10- Sun MicroSystems. Sun MicroSystems. [En línea] [Citado el: 27 de febrero de 2011.] http://java.sun.com/javaee/5/docs/tutorial/doc/.

11- Manual de XHTML. [En línea] [Citado el: 15 de febrero de 2011.] http://manual-xhtml.blogspot.com/2006/05/primer-documento-xhtml.html.

12- Codebox. Glosario. [En línea] [Citado el: 2 de marzo de 2011.] http://www.codebox.es/glosario.

13- **Seth Ladd, Keith Donald.** Expert Spring MVC and Web Flows.s.l.: Apress, 2006. 978-1-59059-584-8.

14- **Patrick Peak, Nick Heudecker.** Hibernate Quickly. s.l. : Manning Publications Co., 2006.

15- **A.Russell, Matthew.** Dojo The Definitive Guide. 2008. 978-0-596-51648-2.

16- Visual Paradigm. [En línea] [Citado el: 15 de marzo de 2011.] http://www.visual-paradigm.com.

17- Control de versiones con Subversion. [En línea] [Citado el: 4 de marzo de 2011.] http://svnbook.red-bean.com/nightly/es/svn-ch-1-sect-1.html.

18- EclipseWiki. [En línea] [Citado el: 20 de marzo de 2011.] http://wiki.eclipse.org/Main_Page.

19- Apache Tomcat. [En línea] [Citado el: 25 de marzo de 2011.] http://tomcat.apache.org/.

20- Microsoft SQL Server 2005. [En línea] [Citado el: 5 de abril de 2011.] http://www.microsoft.com/sqlserver/2005/en/us/product-information.aspx.

21- **Yoan Antonio López, Rodríguez Yulier Matías León.** DEFINICIÓN DE LOS REQUERIMIENTOS FUNCIONALES DEL MÓDULO TESORERÍA, PRÉSTAMOS Y DEPÓSITOS DEL PROYECTO BANCO NACIONAL. 2008.

22- Sistema Bancario Financiero Byte. [En línea] [Citado el: 1 de febrero de 2011.] http://www.bytesw.com/new/sistema_bancario_financierobyte.asp.

23- Bantotal. [En línea] [Citado el: 1 de febrero de 2011.] http://www.bantotal.com/home/core_bancario_banking_solution_system.asp.

24- **Mesa, Lissett Diaz.** Proyecto Técnico para el Sistema Automatizado para la Gestión Bancaria(SAGEB). 2009.

25- Universidad de Castilla-La Mancha. Escuela Politécnica Superior de Albacete, Ingeniería de Software. [En línea] [Citado el: 26 de abril de 2011.] http://www.dsi.uclm.es/asignaturas/42530/pdf/M2tema12.pdf.

Bibliografía consultada

Galíndez, Rodrigo. Control de Versiones Usando Subversion. [En línea] [Citado el: 15 de marzo de 2011.] http://www.rodrigogalindez.com/files/14.pdf.

Scribd. Conceptos de RUP. [En línea] http://es.scribd.com/doc/7844685/CONCEPTOS-DE-RUP.

Desarrollo Web. Arquitectura cliente-servidor. [En línea] [Citado el: 10 de febrero de 2011.] http://www.desarrolloweb.com/articulos/arquitectura-cliente-servidor.html.

Debug Mode On. El patrón MVC. [En línea] http://es.debugmodeon.com/articulo/el-patron-mvc.

Pérez, Javier Eguíluz. Introducción a JavaScript. 2008.

Apache. Documentación del Servidor HTTP Apache 2.0. [En línea] http://httpd.apache.org/docs/2.0/es/.

Pressman, Roger. Ingeniería de Software. Un Enfoque Práctico. 1998.

Rawld Gill, Craig Riecke, Alex Russell. Mastering Dojo. s.l.: Pragmatic Bookshelf, 2008. -934356-11-5

DISEÑO DE BASES DE DATOS RELACIONALES. [En línea] http://usuarios.multimania.es/cursosgbd/UD4.htm.

Craig Walls, Ryan Breindenbach. Spring In Action. s.l.:Manning Publications Co, 2005. 1-932394-35-4

Craig Walls, Ryan Breidenbach. Spring in Action Second Edition. s.l.: Manning Publications Co, 2008. 1-933988-13-4.

Christian Bauer, Gavin King. Hibernate in Action. s.l.: Manning Publications Co, 2005. 1932394-15-X.

Jason Hunter, Willian Crawford. Java Servlet Programing. s.l.: O'Reilly & Associates, Inc, 1998. 1-56592-391-X.

Pruebas de Software. [En línea] [Citado el: 8 de abril de 2011.] http://lsi.ugr.es/~ig1/docis/pruso.pdf.

Glosario de términos

Swift: Swift(en inglés: Society for Worldwide Interbank Financial Telecommunication) es una organización que posee una red internacional de comunicaciones financieras entre bancos y otras entidades financieras.

Servlet: Son objetos que están presentes dentro del contexto de un contenedor de servlets como el Tomcat, es comúnmente utilizado para generar páginas web de forma dinámica a partir de parámetros de una petición de un navegador web.

JSP: Es una tecnología Java para crear contenido dinámico para web en forma de documentos HTML o XML.

CRUD: Es el acrónimo de Crear, Obtener, Actualizar y Borrar (en inglés: Create, Read, Update and Delete)**.**

IDE: Un entorno de desarrollo integrado constituye un programa informático compuesto por un conjunto de herramientas para la programación.

Framework: Es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Código abierto (en inglés Open Source): Término con el que se conoce al software distribuido y desarrollado libremente.

Garbage Collector: El recolector de basura es un mecanismo de gestión de memoria existente en algunos lenguajes de programación.

WWW: Es un sistema de distribución de información basado en hipertexto enlazados y accesibles a través de internet.

Navegador: Software que permite al usuario recuperar y visualizar documentos de hipertexto desde servidores web a través de internet.

API: Aplication Programming Interface, es un conjunto de funciones y procedimientos que ofrece determinada biblioteca para ser utilizada por otro software.

Algoritmo: Cualquier procedimiento computacional bien definido mediante un conjunto de reglas que da solución a instancias de un problema (entrada) produciendo un valor o conjunto de valores (salida).

XML: Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium que permite definir la gramática de lenguajes específicos. XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades

HTML: Lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

GoF: Gang of Four es el nombre con el que se conoce generalmente a los autores del libro Design Patterns.

Anexos

Anexo 1: Capa de presentación del diagrama de clases de diseño de Gestionar Préstamos Recibidos.

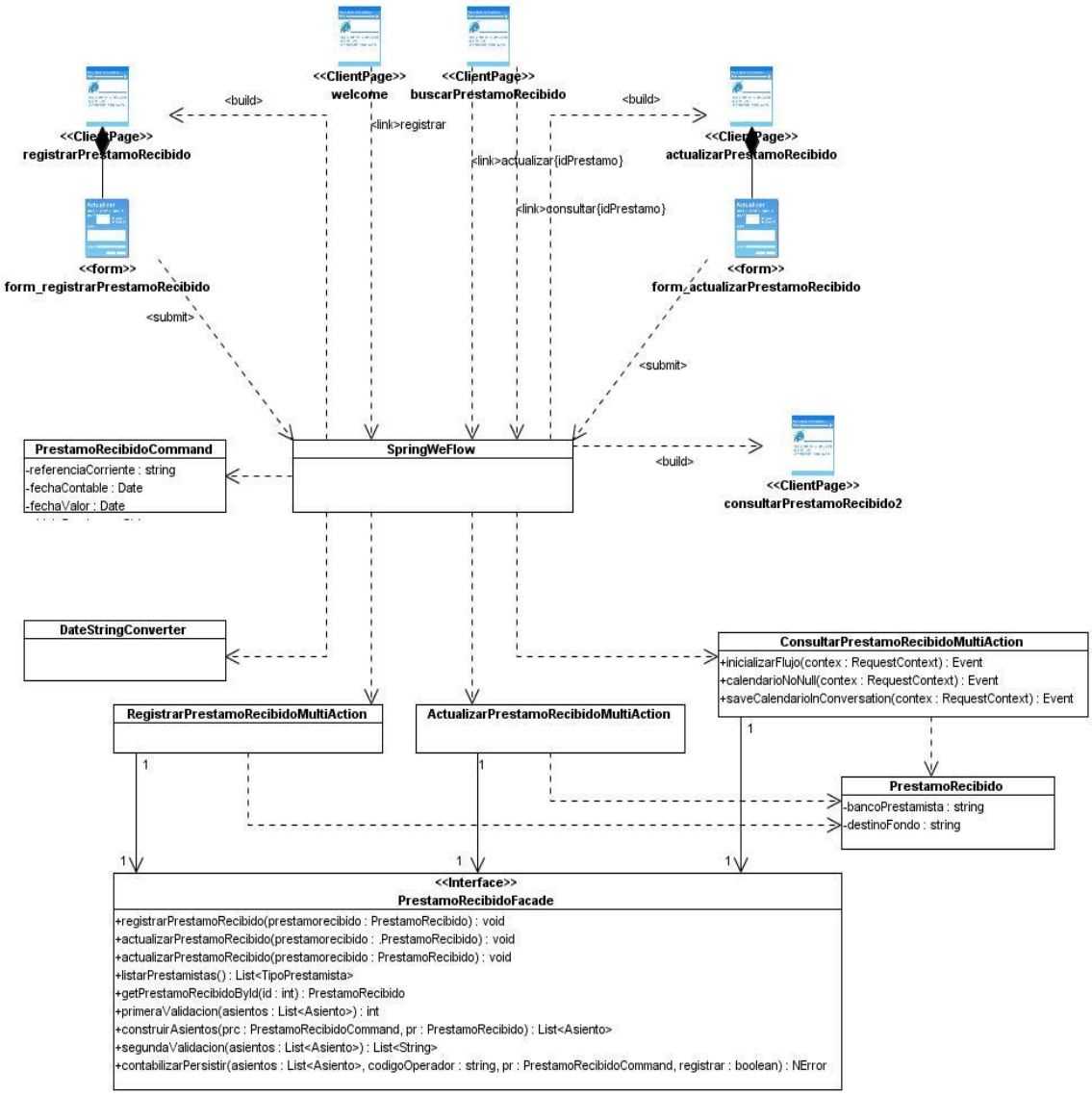

Figura 11: Capa de presentación del diagrama de clases de diseño de Gestionar Préstamo Recibido.

Anexo 2: Capa de negocio del diagrama de clases de diseño de Gestionar Préstamos Recibidos.

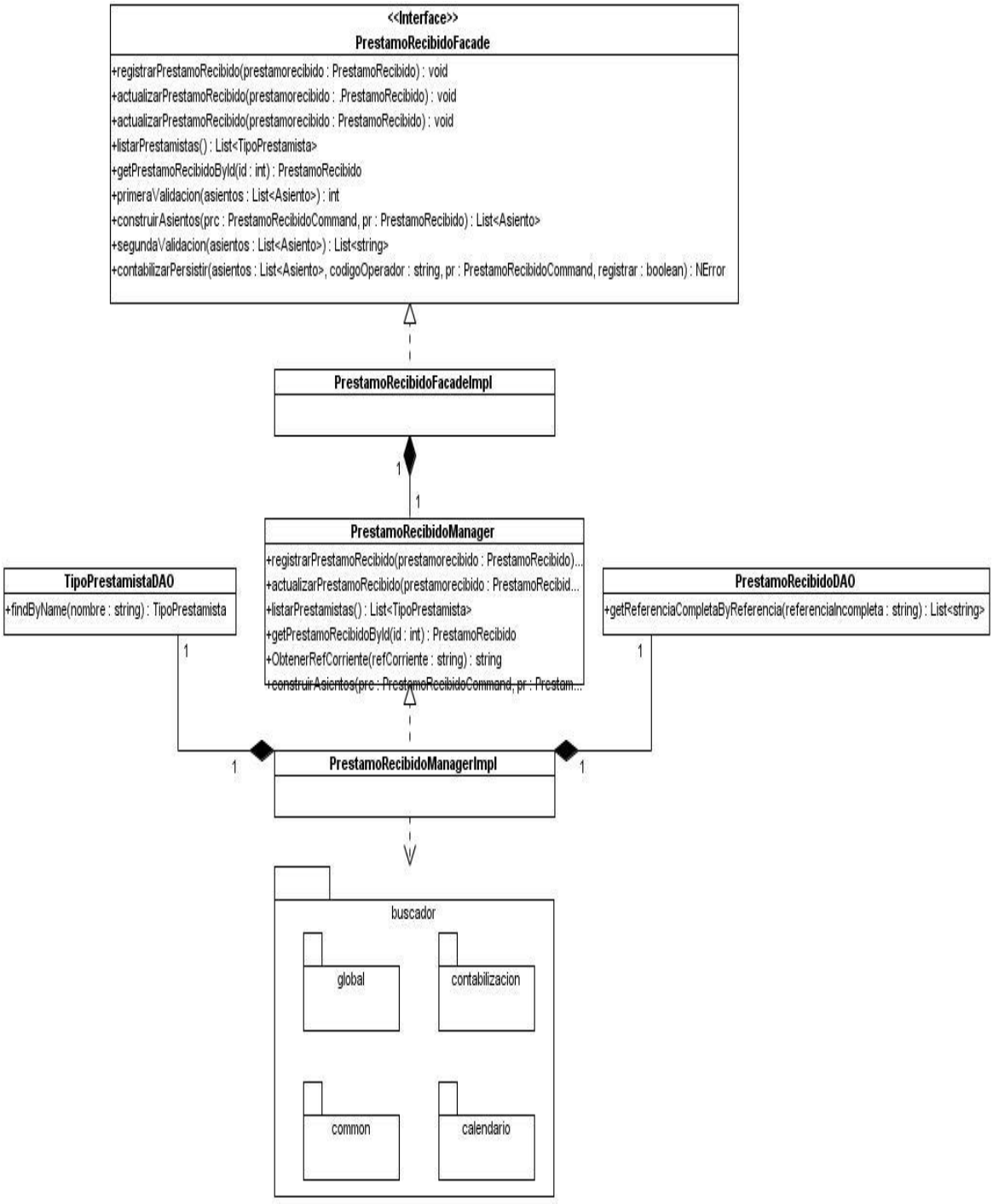

Figura 12: Capa de negocio del diagrama de clases de diseño de Gestionar Préstamo Recibido.

Anexo 3: Capa de AD del diagrama de clases de diseño de Gestionar Préstamos Recibidos.

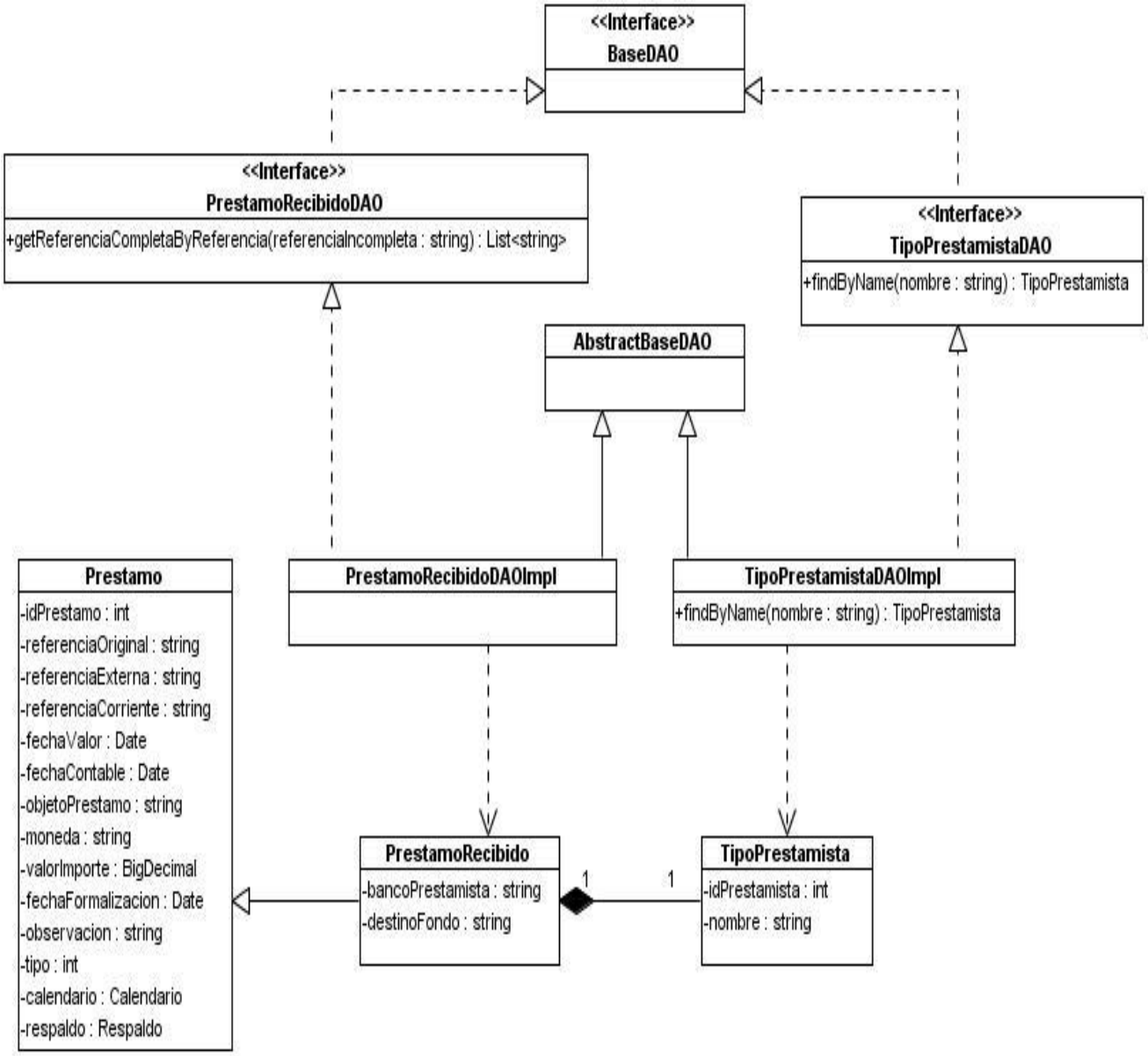

Figura 13: Capa de acceso a datos del diagrama de clases de diseño de Gestionar Préstamo Recibido.