Universidad de las Ciencias Informáticas
Facultad 3

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

Título: "Desarrollo de la tercera fase del subsistema de Cartas de Crédito del sistema Quarxo."

Autores: Yeniersy Garcia Hernández.

Luis Carlos Guerra Rodríguez.

Tutor: Ing. Alejandro Leandro Sosa.

La Habana, curso 2012-2013

"Año 55 de la Revolución".

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores del trabajo y autorizamos a la Facultad 3 de la Universidad de las Ciencias Informáticas hacer uso del mismo en su beneficio. Para que así conste firmamos la presente a los ___ días del mes de Junio del año 2013.

Yeniersy Garcia Hernández

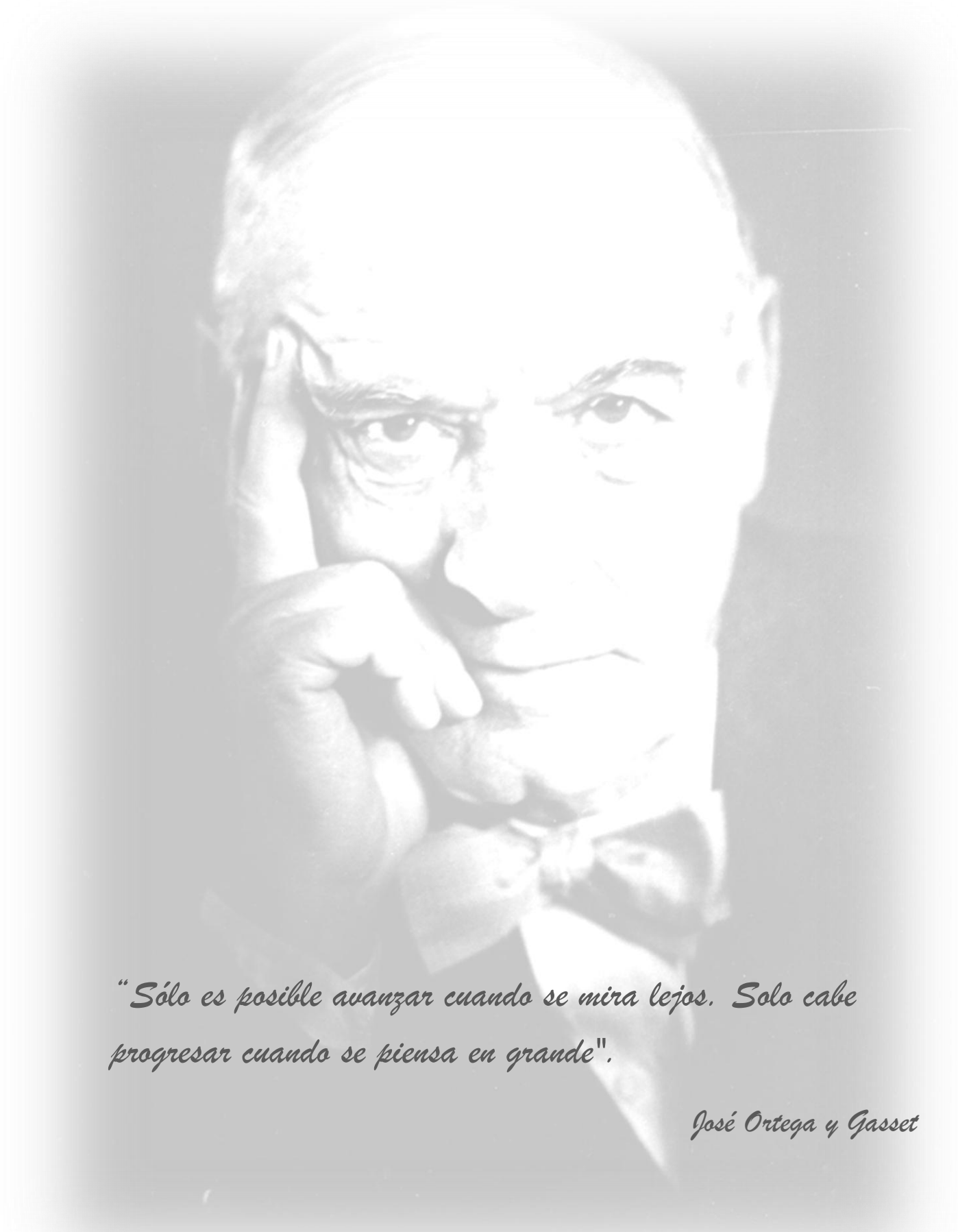
_____
Firma del Autor

Luis Carlos Guerra Rodríguez

_____
Firma del Autor

Alejandro Leandro Sosa

_____
Firma del Tutor



"Sólo es posible avanzar cuando se mira lejos. Solo cabe progresar cuando se piensa en grande".

José Ortega y Gasset

Dedicatoria

Dedico con toda mi alma esta quimera al mayor obsequio que me ha dado la vida "mi familia".

A mi madre por estar siempre pendiente, por ser mi luz y cuidarme en cada instante, por enseñarme lo bueno y malo de la vida, por depositar su confianza y apoyarme en todo momento, por darme fuerzas y ganas de triunfar, por ser la mejor madre y amiga del mundo.

A mi padre por inculcarme sus valores por hacer de mí una mejor persona, por apoyarme y hacerme sonreír en mis peores momentos.

A mi hermano por ser mi estrellita de béisbol, por ser un niño grande de corazón, por darme tanto cariño y por representar mi mayor alegría porque tenerte en mis brazos me lleno de regocijo.

A mi abuela Guille por ser otra madre para mí, a mi tía Margará por estar siempre en la buenas y las malas, a mis primas por compartir tantos recuerdos de nuestra infancia, a mi tío Oscar por ser maravilloso e incondicional, a todos por tener la complacencia de poder contar con ustedes.

A Richar por compartir su vida junto a mí, por apoyarme y comprenderme, pero sobre todo por hacerme muy feliz.

A Aslan por formar parte de mi vida y de mi felicidad.

Yeniersy

Le dedico este gran logro de mi vida a una persona que siempre quiso estar aquí conmigo en este momento pero la vida no lo permitió así, a mi padre por todo lo que me dio y por guiarme por el buen camino en todo instante.

A mi madre Julia por dármelo todo en la vida y siempre darme buenos consejos que me han servido de mucho.

A mi hermanita Lily por saber cómo alegrarnos el día aun cuando todo estuviera nublado y siempre buscar lo bueno en todo momento.

A mi abuela Conchi y mi tía Lisy que siempre me han apoyado en todas mis decisiones.

A toda mi familia que es lo más preciado que tengo en la vida, su cariño y su amor.

Luis Carlos

Agradecimientos

A mis padres Elina y Oreste por estar presentes en mi vida, les doy gracias por apoyarme en cada segundo, sin ustedes no hubiera sido posible alcanzar la meta.

A Elieser por ser un amigo especial e incondicional, gracias por estar siempre que te necesito.

A mis compañeros de estudio le agradezco infinitamente por compartir todos estos años juntos y las experiencias vividas.

A la universidad y a todos los profesores por su aporte en conjunto a nuestra formación profesional.

A mi compañero Luis Carlos gracias por el apoyo, fue maravilloso compartir este logro junto a ti.

A Alejandro Leandro, nuestro tutor, por su gran ayuda y la confianza depositada.

A Yoan Antonio, por ser más que un tutor para nosotros. Gracias por la dedicación y paciencia, por enseñarnos y apoyarnos.

A Manuel, Bello y todos los profesores del proyecto por apoyarnos siempre que necesitamos de su ayuda.

A Fidel y la revolución por la oportunidad que nos ofrece.

Gracias a todos los que de una forma u otra han hecho posible la realización de esta tesis.

Yeniersy

A mi padre a mi madre y mi hermana por hacer de mí la persona que soy. A mis abuelas Conchi y Aida. A mis tías Lisy, Evy, Flor, Nidia y Eva. A mis primos por todo su apoyo y ayuda. A Aslin por ser como mi otra hermana, mi confidente.

A Zory y Raichelis, gracias por acogerme como uno más de la familia.

A Rainier por ser más que un amigo un hermano. A mis amigos: Leo, Keniel, Leonardo, Silvio, Rada, Jorge Ricardo, Ramón, Jorgito, Félix, Leandro, Diógenes, Raidel, Yenquiel, Karel y el Bode.

A Karen Natalia por llegar para alegrarme la vida.

A Yamilet gracias por todo el tiempo que estuviste conmigo y todo lo que aprendí junto a ti.

A Yeniersy mi compañera por todo su apoyo y consejos.

A Alejandro nuestro tutor por su preocupación y ayuda en todo momento.

A Yoan, Bello, Manuel, Héctor, Dariel, Yulier y todos los demás profesores del proyecto que siempre estuvieron ahí cuando necesitamos su ayuda. Gracias a todos aquellos que donde quiera que nos veían nos preguntaban:" ¿Cómo va la tesis?".

Luis Carlos

RESUMEN

Como parte de la colaboración que mantiene la Universidad de las Ciencias Informáticas con el resto de las instituciones del país, produciendo aplicaciones y sirviendo de soporte a la industria cubana del software, se implantó desde mayo del 2012 en el Banco Nacional de Cuba la primera versión del sistema Quarxo. Desde su implantación se acordó realizar futuras versiones del sistema que respondieran a mejoras y cambios que surgieran en lo adelante. Dentro de los subsistemas con que cuenta el sistema Quarxo se encuentra Cartas de Crédito, del mismo fue preciso, por la urgencia de los cambios solicitados, realizar una fase intermedia conocida como Segunda Fase. A pesar de ello, persistieron deficiencias en la ejecución de los procesos de emisión, negociación y discrepancias, que provocaban retrasos en la gestión de las cartas de crédito. Para darle solución a esas deficiencias se desarrolló una tercera fase del subsistema Cartas de Crédito manteniendo salvo algunas mejoras, la línea arquitectónica de las fases anteriores.

La tercera fase del subsistema Cartas de Crédito resuelve un grupo de problemas y solicitudes de cambios relacionados con los módulos emisión, negociación y discrepancias, dentro del ellos: la actualización de las cartas de crédito en todos sus estados, la impresión de cartas a enviar a la empresa ante la negociación de documentos discrepados, la inserción automática de las uniones, la gestión de las cartas de crédito con varios tipos de deudas y formas de aseguramiento.

Palabras clave: Cartas de Crédito, Emisión, Discrepancia, Negociación.

ÍNDICE

INTRODUCCIÓN

Para el año 2008, como parte de la modernización del Sistema Bancario Cubano se decide realizar un sistema informático que mejore la situación existente en el Banco nacional de Cuba (BNC).

Los sistemas informáticos presentes en el BNC para el año 2008 son:

- ✓ SABIC: Sistema Automatizado para la Banca Internacional de Comercio, en su versión MS-DOS.
- ✓ SLBTR: Sistema de Liquidación Bruta en Tiempo Real.
- ✓ SISCOM: Sistema de Comunicación de mensajería SWIFT, sobre Windows.

El SABIC como sistema puramente contable, permitía la gestión de los principales procesos llevados a cabo en el BNC. El SLBTR, es una aplicación desarrollada por los especialistas e informáticos del Banco Central de Cuba (BCC), teniendo en cuenta la ISO 15022; pero realizándole algunas modificaciones a la estructura de los mensajes propuesta por el estándar, creando así un nuevo estándar de comunicación: el SLBTR. El objetivo principal de este sistema no es más que garantizar la automatización de las transacciones realizadas por el BNC y otros bancos del país, así como asegurar la irrevocabilidad y precisión de los movimientos de fondos interbancarios. El SISCOM es un producto para el procesamiento y enrutamiento de mensajería SWIFT a través de la red SWIFT FIN (1).

La situación existente en el BNC estaría dada entonces por un sistema de gestión de los procesos contables (SABIC) sin puntos de integración con el sistema utilizado para enviar y recibir los mensajes SWIFT (SISCOM), por la diferencia entre las plataformas, donde los operadores del banco deben introducir los mismos datos tanto en un sistema como en el otro; necesitando así un tiempo considerable y el doble de esfuerzo para realizar las operaciones bancarias que llevan consigo la generación y recepción de mensajes (1).

Las deficiencias propias que presenta la plataforma MS-DOS, como la monotarea, dan lugar a que los operadores tengan que reiniciar frecuentemente la máquina, con las afectaciones al hardware que eso conlleva, para trabajar en el sistema de mensajería, en el correo o en otras aplicaciones. Asimismo debido a lo complejo que se tornaban las actividades, la contabilidad no era realizada en tiempo real y por tanto no podía utilizarse la información almacenada por el sistema en algunas áreas vitales para el BNC como Tesorería. Esas áreas tendrían que mantener documentos en aplicaciones ofimáticas sobre el estado de las cuentas del BNC a emplear en los procesos de toma de decisiones.

Al BNC le urge entonces eliminar esos problemas y ante el hecho de que las versiones mejoradas realizadas al sistema Sabic como la versión Sabic para Windows realizada por BCC, no se

ajustaban a sus necesidades, solicita la colaboración de la Universidad de las Ciencias Informáticas (UCI) para la realización de un sistema informático (1). Se desarrolla entonces el sistema Quarxo y es implantado en el BNC a mediados de mayo del 2012.

EL sistema Quarxo desarrollado en la UCI por el proyecto SAGEB[1] resolvería en su primera versión la mayor parte de las necesidades del BNC. Al mismo tiempo a raíz de su implantación se acordó resolver cambios y mejoras que surgieran en lo adelante, en futuras fases que se realizarían del sistema.

Dentro de los subsistemas con que cuenta el sistema Quarxo se encuentra Cartas de Crédito. Dicho subsistema contiene los módulos: Acuerdo, Capacidad Financiera, Emisión, Discrepancia, Documento de Embarque y Negociación. Para el subsistema Cartas de Crédito fue preciso, por la urgencia de los cambios solicitados, realizar una fase intermedia conocida como Segunda Fase. A pesar de ello, persistieron deficiencias en la ejecución de los procesos de emisión, negociación y discrepancias, que provocaban retrasos en la gestión de las cartas de crédito (CC). De forma detallada se plantean a continuación las deficiencias del subsistema por cada uno de sus módulos.

Emisión: el módulo tiene como objetivo principal la gestión de las CC desde principio a fin, este es un proceso complejo que transita por varios estados y requiere ser rectificado en cualquier instante. El sistema no permite actualizar el operativo con nuevos datos, después de ser contabilizada la CC, lo que provoca lentitud y requiere mayor esfuerzo para los operadores. Por otro lado el sistema no es capaz de cancelar e identificar cuándo un crédito ha sido devuelto antes de ser contabilizado lo que genera gran polémica respecto a la existencia o no de la CC. Al registrar una CC el sistema no permite definir el banco de apertura de la CC (*conocido del inglés como please open)*, igualmente no es posible que las cartas de crédito financiadas puedan ser aseguradas y no aseguradas o tener varios tipos de deudas a la vez, de manera que los usuarios del sistema deben hacer estas operaciones de forma manual.

Discrepancia: el módulo tiene como objetivo principal gestionar aquellos documentos de embarques que presentan problemas de cualquier índole. El mismo tiene incorporadas un grupo de funcionalidades pero no incluye la impresión de las cartas de discrepancias que serán enviadas a las empresas y el envío del mensaje 750 al registro de la discrepancia.

Negociación: el módulo tiene como objetivo principal la gestión de las negociaciones que se realizan sobre las cartas de crédito al recibo de los documentos de embarque. Las deficiencias detectadas en este módulo están relacionadas con el registro de las negociaciones impidiendo el correcto funcionamiento del mismo. Al registrar una negociación el sistema no permite definir el tipo

[1] *Sistema Automatizado de Gestión de Entidades Bancaria*.

de negociación si será asegurada o no asegurada, no envía una serie de mensajes (799, 754, 999, 734, 750) indispensables en el registro de la negociación, el respaldo del pagaré[2] se realiza por la cuenta principal impidiendo el respaldo por la cuenta de interés, de la misma forma se dificulta el respaldo de la letra de cambio[3], las transacciones de negociación no permiten distintos tipos de deuda de forma simultánea y el sistema no es capaz de registrar automáticamente en la base datos las negociaciones compuestas.

Para darle solución a esas deficiencias se desarrolló una tercera fase del subsistema Cartas de Crédito.

El presente trabajo se enfoca en el desarrollo de la tercera fase del subsistema Cartas de Crédito. A raíz de las deficiencias encontradas, se identifica como **problema a resolver:** las limitaciones en la ejecución de los procesos de emisión, discrepancia y negociación en el Banco Nacional de Cuba provocan retrasos en la gestión de las cartas de crédito.

Siendo identificado como **objeto de estudio:** la gestión de cartas de crédito en las entidades financieras bancarias. Proponiendo como **objetivo general**: el desarrollo de la tercera fase del subsistema Cartas de Crédito incorporando funcionalidades y mejoras a los módulos de Emisión, Discrepancia y Negociación.

Dentro de dicho objeto de estudio se enmarca el siguiente **campo de acción:** la gestión de cartas de crédito en el Banco Nacional de Cuba.

Como **idea a defender** de la investigación se tiene que con el desarrollo de la tercera fase del subsistema Cartas de Crédito para el sistema Quarxo se agilizará el proceso de gestión y toma de decisiones en las operaciones con las cartas de crédito en el Banco Nacional de Cuba.

Se definen como **objetivos específicos**:

1. Elaborar el marco teórico de la investigación para la formulación de los principales conceptos de cartas de crédito.
2. Realizar el análisis y diseño de la tercera fase del subsistema Cartas de Crédito para comprender las necesidades del cliente y elaborar los diagramas correspondientes al modelado de diseño.

[2] *Documento financiero en virtud del cual una persona natural o jurídica (deudor) se compromete a pagar al acreedor una suma definida de dinero a su demanda o en fecha determinada en el futuro.*

[3] *Documento financiero que se define como una orden incondicional, por escrito, dirigida por una persona natural o jurídica (girador), a otra persona (girado), firmada por la persona que la emite, requiriendo a la persona a la que está dirigida (girado) a pagar a la vista o a un tiempo fijo o futuro determinado, una suma definida de dinero a la orden de pago, a una persona específica, o al portador (beneficiario).*

3. Implementar las funcionalidades para la tercera fase del subsistema Cartas de Crédito bajo la tecnología propuesta por el proyecto SAGEB para satisfacer las necesidades del cliente.
4. Validar la solución propuesta para asegurar la calidad del software y la aceptación del cliente.

Aportes de la tesis:

1. Incorporación de nuevas funcionalidades al subsistema Cartas de Crédito de Quarxo.
2. Mejoras en la gestión de las cartas de crédito en el Banco Nacional de Cuba.

Durante la investigación, se utilizarán varios métodos científicos para lograr un mayor entendimiento del tema. Los mismos se describen y clasifican a continuación:

Métodos Teóricos: los métodos teóricos permiten revelar las relaciones esenciales del objeto de investigación que no son observables directamente. Posibilitan la interpretación conceptual de los datos empíricos encontrados, además de explicar los hechos y profundizar en las cualidades fundamentales de los procesos, hechos y fenómenos (2).

1. Método Histórico – Lógico: utilizado para la realización de un estudio detallado de los sistemas contables que gestionan las cartas de crédito desarrollados a nivel nacional e internacional.
2. Método Analítico – Sintético: empleado en la realización del estudio teórico de la investigación y el análisis previo sobre los proceso de cartas de crédito.
3. Método de Modelación: utilizado para la creación de modelos de diseño y componentes.

Métodos Empíricos: los métodos empíricos revelan y explican las características fundamentales del objeto a través de procedimientos prácticos (2).

1. Entrevistas: se utilizará con el fin de obtener información, para ello será preciso establecer diálogos planificados con los especialistas del Banco Nacional de Cuba en aras de obtener los requisitos funcionales.

La memoria escrita de la tesis presenta la siguiente estructura:

Capítulo 1: Fundamentación teórica

Aborda la fundamentación teórica del tema, se realiza un análisis del estado del arte a nivel nacional e internacional describiendo algunos de los sistemas existentes que incluye el objeto de estudio. Se expone una disertación de los principales conceptos del dominio del problema, se describen las tecnologías a considerar, la metodología, los lenguajes de programación y herramientas de desarrollo a utilizar en la elaboración de la solución.

Capítulo 2: Análisis y diseño de la solución

Se centra fundamentalmente en el análisis y diseño de la solución, se caracterizan los patrones de diseño en la búsqueda de soluciones a problemas comunes del desarrollo para implementar las funcionalidades, modela los procesos de negocios, realiza la especificación de requisitos, describe la arquitectura que presenta el sistema y el modelo de diseño. Se exponen los artefactos generados como: los diagramas de clases de diseño, diagramas de paquetes, diagramas de secuencias y de flujo 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: Implementación y validación

Se enfoca directamente en la construcción de la solución explicando los aspectos principales de la implementación y la validación de los requisitos propuestos. Se realiza una descripción de las clases y funcionalidades, así como de los diagramas de interacción, de componentes y la relación de los estándares de codificación utilizados.

Finalmente se procede a las conclusiones generales, las recomendaciones y al glosario de término.

Capítulo I: Fundamentación teórica

1.1. Introducción

En el presente capítulo se abordan los conceptos de cartas de crédito necesarios para la comprensión del negocio, haciendo énfasis en los tipos, funcionamientos y las partes que intervienen en este proceso. De igual forma se realiza un estudio de los sistemas contables que operan las cartas de crédito para la gestión de sus transacciones en el mundo y en Cuba. Por último se especifica la metodología empleada en el proyecto, así como la descripción de las herramientas, tecnologías y lenguajes que se utilizarán en el desarrollo del producto.

Para facilitar la comprensión del trabajo se deben entender primeramente los términos que se emplean en las instituciones bancarias, motivo por el cual se especifican algunos conceptos importantes tratados durante la investigación.

1.2. Sistemas bancarios

El sistema bancario, o también conocido como banca, es el conjunto de entidades financieras que operan dentro de la misma economía. Según la literatura revisada (3) dentro de un sistema bancario se puede identificar distintos tipos de bancos, estos son:

- ✓ Bancos corrientes: son las instituciones financieras que ofrecen servicios de ahorro y disposición de cheques para personas físicas. También ofrecen préstamos y pueden ser vistos como intermediarios entre personas con excedentes de dinero y personas con necesidades de dinero.
- ✓ Bancos prestamistas especializados: son instituciones que otorgan préstamos o créditos con fines específicos, como pueden ser los Bancos Hipotecarios, entre otros.
- ✓ Bancos de Emisión: se les denomina así a los Bancos Centrales o Bancos Nacionales, son las únicas instituciones bancarias con la función de emitir dinero ya sea en papel o en moneda.

El sistema bancario se compone finalmente de estos tres tipos de bancos los cuales participan dentro de los mismos mercados, cada uno cumpliendo funciones distintas para atender las necesidades financieras de las personas físicas y morales de la economía (3).

En sentido general los sistemas bancarios constituyen una estructura organizacional dentro de la cual se mueve el conjunto de instituciones bancarias, respondiendo a las directrices que le marca la autoridad superior; incluye la banca nacional, banca central, banca privada y mixta.

1.2.1. Sistema bancario cubano

El sistema bancario cubano está encabezado por el Banco Central de Cuba y constituido por 9 bancos comerciales, 15 instituciones financieras no bancarias, 11 oficinas de representación de bancos extranjeros en Cuba y 4 oficinas de representación de instituciones financieras no bancarias (4).

El BNC es un banco comercial o de emisión que desempeña un papel fundamental para el estado cubano, dentro de sus principales funciones se encuentra: obtener y otorgar créditos en moneda nacional y en moneda libremente convertible a clientes jurídicos, centralizar las relaciones oficiales con las entidades extranjeras de seguro de crédito oficial a la exportación, según decida el Banco Central de Cuba, constituir entidades de seguro de crédito oficial a la exportación con arreglo a la legislación vigente en materia de seguros y mantener el registro, control, servicio y atención de la deuda externa que el estado cubano y el BNC tienen contraída con acreedores extranjeros hasta la fecha de entrada en vigor del Decreto Ley No. 172, de 1997, del Banco Central de Cuba (5).

1.2.2. Contabilidad bancaria

Previamente antes de definir la contabilidad bancaria se debe concretar qué se entiende por contabilidad; según el organismo rector de las finanzas de Cuba especifica que: “La contabilidad registra, clasifica y resume en forma propia y en términos monetarios, las operaciones que acontecen en una entidad y por medio de ella, se interpretan los resultados obtenidos. No constituye un fin en sí misma, sino que representa un medio para llegar a uno o más fines” (6).

La contabilidad bancaria según la define la literatura consultada (7): “Es aquella que tiene relación con la prestación de servicios monetarios y registra todas las operaciones de cuentas en depósitos o retiros de dinero que realizan los clientes, ya sea de cuentas corrientes o ahorros. También registran los créditos, giros tanto al interior o exterior, así como otros servicios bancarios.”

Los bancos como instituciones financieras, encargadas de regular las relaciones monetarias entre personas e instituciones, necesitan de la contabilidad bancaria para conocer su situación económica y registrar todas las operaciones que realizan dentro de sus funciones.

1.2.3. Instrumentos de pago

Se denomina instrumento o medios internacionales de pago aquellas modalidades operativas por medio de los cuales el importador puede cancelar el compromiso de pago contraído con el exportador, en virtud de una relación comercial instrumentada mediante un contrato de compraventa internacional (8).

Según la literatura consultada (8) existen diferentes instrumentos de pagos, pero los más utilizados en el comercio internacional son:

- ✓ Cheque internacional.
- ✓ Orden de pago o transferencia internacional.
- ✓ Cobranza internacional.
- ✓ Cartas de crédito.

Los bancos juegan un papel importante en el comercio tanto nacional como internacional. La contabilidad bancaria genera la necesidad de efectuar pagos y cobros a distancia entre partes ubicadas en distintos municipios, provincias y países, utilizando para ello los medios de pago. La carta de crédito es el medio de pago por excelencia usado por las empresas para pagarles a los proveedores de las mercancías en el extranjero.

1.2.4. Cartas de crédito

Las cartas de crédito son un instrumento de pago, sujeto a regulaciones internacionales, mediante el cual un banco (Banco Emisor) obrando por solicitud y conformidad con las instrucciones de un cliente (ordenante) debe hacer un pago a un tercero (beneficiario) contra la entrega de los documentos exigidos, siempre y cuando se cumplan los términos y condiciones del crédito. Este instrumento es uno de los documentos más sencillos en su forma y de los más complejos en cuanto a su contenido. Llamada también “Crédito Comercial”, “Crédito Documentario” y en algunas ocasiones simplemente crédito (9).

Es un instrumento solicitado por el comprador a favor del vendedor. En donde se menciona todos los documentos que deben aportar el vendedor y la forma en que debe entregarlos al banco para su negociación (8).

Una carta de crédito es un efecto bancario que se emite para proteger al beneficiario (exportador) en una transacción. Técnicamente una carta de crédito es una garantía de pago, una obligación al pago de un beneficiario, es decir, el beneficiario sólo podrá girar sobre la misma mediante la presentación de los documentos de embarques o los documentos comerciales exigidos en la apertura de la carta de crédito, para asegurar el cumplimiento del contrato por parte del vendedor.

Los documentos de embarques son documentos representativos de una transacción comercial internacional y del embarque efectuado en particular, que se exigen para demostrar que se ha cumplido con los requisitos del envío de las mercancías vendidas y para exigir el pago de las mismas por parte del comprador (11). Cualquier diferencia o anormalidad que tengan los documentos de embarque respecto a lo establecido en los términos del crédito, se denomina discrepancia de los documentos de embarque**.**

Si existe discrepancias, supuestamente el exportador no podrá hacer efectivo el crédito hasta tanto el banco emisor (y obviamente el importador) autoricen a pagarlo a pesar de ellas, para lo cual se puede aplicar diferentes procedimientos. A esto se le denomina técnicamente "levantar las reservas". El levantar o no las reservas es un derecho que le asiste al importador quien, por tanto, deberá hacerlo constar por escrito. En este procedimiento el banco emisor es el encargado de notificar a la empresa (importador) las discrepancias de los documentos de embarques mediante la carta de discrepancia, la empresa deberá informar al banco emisor su aprobación para que puedan ser confirmados o rechazados los documentos al banco extranjero en un término de cinco días hábiles bancarios.

1.2.4.1 Partes que intervienen en una carta de crédito

Según la literatura consultada (12) las partes que intervienen en una carta de crédito son:

- ✓ **Ordenante o tomador:** es aquella persona natural o jurídica, que participa como comprador en el contrato de compraventa y ordena a su banco a abrir las cartas de crédito en favor del exportador. El tomador también recibe el nombre de ordenante de cartas de crédito.
- ✓ **Beneficiario o acreedor de las cartas de crédito:** es la persona natural o jurídica a cuyo beneficio se abre la carta de crédito y generalmente será el exportador o vendedor.
- ✓ **Banco emisor:** es el "banco del importador" que recibe el encargo de emitir las cartas de crédito y avisa el acreditado al beneficiario por intermedio de otro banco establecido en el domicilio o residencia del beneficiario de la carta de crédito.
- ✓ **Banco avisador o banco notificador, o banco receptor:** el banco corresponsal en el lugar en que reside el beneficiario, avisa la apertura del crédito al acreedor de la carta de crédito.
- ✓ **Banco confirmador:** usualmente, una sucursal del banco emisor. La entidad se encarga de abonar las cartas de crédito al beneficiario, mayormente utilizado cuando no son suficientes las garantías por parte del banco emisor o cuando el beneficiario quiere asegurar el pago.
- ✓ **Banco pagador:** banco encargado de pagar o comprometerse al pago inmediato a la presentación de los documentos conformes a los términos estipulados y el crédito, generalmente en el mismo país del exportador.
- ✓ **Banco aceptador:** prácticamente cumple con la función del banco pagador con la diferencia que acepta un efecto al vencimiento, no el compromiso del pago. En este caso el beneficiario dispone de la letra de cambio que es un compromiso de pago documental.
- ✓ **Banco negociador:** su función es similar a los dos anteriores sin embargo es el encargado de negociar el descuento al exportador. Si el exportador cobra a la vista, se le descontarán los intereses en dependencia del crédito. Este puede ser el banco confirmador.

Generalmente en cada compra-venta median dos bancos para todo lo relacionado a la documentación que acompaña a la mercancía, uno de los bancos es el del importador (en este caso, la empresa cubana) desempeñando el rol de emisor y el otro el del proveedor que comúnmente es el banco corresponsal en el país del exportador, que en dependencia de la función que realice asume uno o varios roles de los anteriores.

1.2.4.2 Tipos de cartas de crédito

Existen diferentes tipos de cartas de crédito en dependencia del objetivo con que sea emitida, según la bibliografía consultada (12) se clasifican por:

El flujo de su mercancía en:

- ✓ **Cartas de crédito de importación:** es aquella emitida por un banco emisor a un beneficiario por la adquisición de bienes y/o servicios del extranjero, ya sea a petición de un ordenante o en su propio nombre.
- ✓ **Cartas de crédito de exportación:** es la que solicita un comprador en el extranjero a su banco emisor a favor de un proveedor (beneficiario).

Su compromiso de pago:

- ✓ **Irrevocable:** es cuando sólo puede ser modificada o cancelada con el acuerdo de todas las partes involucradas.
- ✓ **Notificada:** es cuando el banco avisador/confirmador notifica o avisa al exportador, la existencia de un crédito documentario a su favor, pudiendo el banco notificador pagar una vez que éste reciba los fondos del banco emisor.
- ✓ **Confirmada:** es cuando el banco avisador/confirmador, asume el compromiso de pagar una vez que el beneficiario ha cumplido en tiempo y forma con los términos acordados.

La carta de crédito deberá indicar la forma en que ésta le será pagada al vendedor / exportador, la misma se podrá realizar de la siguiente forma:

- ✓ **A la vista:** se paga de manera inmediata a la presentación de los documentos que cumplan con los términos y condiciones.
- ✓ **Con pago diferido:** se paga al vencimiento del plazo convenido entre el comprador y el exportador.
- ✓ **Con plazo proveedor o aceptación:** se paga al vencimiento de la letra de cambio aceptada por el banco emisor o confirmador.

Existen algunas modalidades especiales de las cartas de crédito como son:

- ✓ **Cartas de crédito transferibles:** permite que el exportador nombre a sus proveedores o a terceras personas como segundos beneficiarios, logrando con ello, brindar a sus proveedores la seguridad de pago una vez que ellos cumplan con lo establecido.
- ✓ **Standby:** se denomina así a la CC mediante la cual, los bancos garantizan el pago al beneficiario en caso de incumplimiento por parte de un obligado a realizar una acción previamente comprometida, ante el no pago de préstamos o importes entregados como anticipo e inclusive, ante la ocurrencia o no ocurrencia de alguna otra contingencia.
- ✓ **De garantía**: es el instrumento que acompaña contratos internacionales y suministro de bienes de toda clase, de prestación de servicios, de realización de mano de obras y otros contratos de préstamos. La emite el banco garantizando el pago al beneficiario en caso de que su cliente no efectúe el mismo.

En la apertura de un CC se debe definir bien claro todos los términos entre el ordenante y el banco emisor. Según el tipo de CC es la obligación de los sujetos a cumplir lo determinado en el plazo acordado, para ello se realizan negociaciones estableciendo los pagos a la vista o a futuros en dependencia de lo pactado en el contrato.

Según la literatura consultada (13) realizar negociaciones sobre una CC significa hacer entrega del valor del/de los instrumento/s de giro y/o documento/s por parte del banco autorizado a negociar. El simple examen de los documentos sin hacer entrega de su valor no constituye una negociación.

1.2.4.3 Proceso de cartas de crédito

El proceso de cartas de crédito para el BNC se inicia cuando la empresa cubana solicita la apertura de una carta de crédito a la institución. El área operativa de emisión y la DGN[4] son las encargadas de realizar los procesos de registro y autorizo de la CC, la misma es contabilizada en cuentas contingentes en espera de la llegada de las mercancías al puerto. El proceso de emisión de CC incluye la contabilización y el envío de mensajes a los bancos corresponsales en el exterior. A la llegada de las mercancías, el BNC registra los documentos de embarques y verifica que no presente discrepancias para comenzar el proceso de gestión de negociación definiendo los pagos a futuros y plazos de vencimiento según lo pactado.

1.3. Sistemas contables informáticos

Los sistemas contables informáticos son los programas de contabilidad o paquetes contables, destinados a sistematizar y simplificar las tareas de contabilidad. El software contable registra y procesa las transacciones históricas que se generan en una empresa o actividad productiva: las funciones de compra, ventas, cuentas por cobrar, cuentas por pagar, control de inventarios,

[4] *Direcciones de Gestión de Negocios*

balances, producción de artículos, nóminas, etc. Para ello es necesario ingresar la información requerida, como las pólizas contables: ingresos y egresos, para que el programa realice los cálculos necesarios (15).

De forma general constituyen aplicaciones capaces de gestionar la información contable de una empresa o de una entidad bancaria llevando a cabo el control de todas las actividades financieras. El sistema contable de un banco o de una empresa, independientemente del software que utilice, se deben ejecutar en tres pasos básicos. Se deben: registrar, clasificar y resumir los datos, sin embargo los procesos realizados en cada entidad son los que diferencian los sistemas informáticos de una empresa a los de un banco involucrando procesos tales como: gestión de cartas de crédito, préstamos, análisis de riesgos, depósitos, vencimientos, mensajería, conciliaciones, títulos valores, tesorería y otros más en dependencia del tipo de banco. Toda institución debe disponer de un sistema de información contable que se adecue a sus características.

1.3.1 Sistemas contables informáticos internacionales

Llevar el control de las actividades y el registro de toda la información contable es un objetivo primordial en cualquier entidad, para ello se han desarrollado sistemas capaces de simplificar estas tareas. Una de las operaciones fundamentales en las instituciones bancarias es el manejo de las cartas de crédito. A continuación se mostrarán diferentes sistemas que gestionan este proceso en el mundo.

CARD CRED es un sistema desarrollado por la compañía venezolana La Sistema[5] con más de 20 años de trascendencia en el mercado financiero de ese país, el cual permite mantener el control y seguimiento de todos los procesos asociados a las cartas de crédito desde su apertura hasta el momento de cancelación final. La aplicación se basa en una codificación de: Clientes, Corresponsales, Aduanas, Cartas de Crédito. Emite la relación contable diaria en reporte y archivo de textos y genera las llamadas débitos/créditos de las operaciones efectuadas. Este sistema se encuentra dividido en seis módulos. El módulo Cartas de Crédito posibilita el registro de los instrumentos de pago utilizados en el comercio internacional (16).

COBIS SCI es el Sistema Integral de Comercio Internacional desarrollado con tecnología de punta y arquitectura cooperativa cliente-servidor. Concebido y orientado a las operaciones de comercio internacional bajo normas establecidas relativo a los créditos documentarios de la Cámara de Comercio Internacional. El sistema permite informar al cliente el estado de sus operaciones, la generación de mensajes SWIFT de forma automática, manejo de cobranzas, discrepancias. Este realiza las funciones de asignación, mantenimiento y control de las líneas de créditos de bancos

[5] *Fábrica Venezolana de Tecnología Financiera.*

corresponsales, con el propósito de canalizar las cartas de crédito en forma ágil y oportuna, provee facilidades para el manejo del proceso de negociación de la misma, con posibilidad de registrar discrepancias, indicándose los detalles de embarques y formas de pago, liquidación, abono y cancelación, así como los costos a cobrar y puede contabilizar las transacciones financieras generadas automáticamente. Dispone de consultas que le indican al usuario los posibles estados que puede tomar una operación y sus embarques, realiza financiamientos a operaciones de Comercio Internacional. COBIS SCI es un sistema totalmente funcional, flexible y puede ser adaptado a las necesidades únicas de cada institución financiera para mejorar la productividad, rentabilidad y el manejo de integral de la información (17).

Sistema de Administración Comercial Global TradeMasterQW (*en inglés TradeMasterQW Global Trade Management System*) es otro sistema contable desarrollado por QuestaWeb, compañía que ha automatizado el proceso de cartas de crédito permitiendo a los importadores administrar y controlar el mismo de principio a fin. El sistema automáticamente crea y rellena los registros de datos, realiza todas las conciliaciones, tramita enmiendas, alerta a las piezas claves de la documentación qué falta o términos de negocio no satisfechos, mantiene los totales acumulados de las cantidades de cartas de crédito con transacción efectiva, contiene una historia completa de cada cambio y un registro en tiempo real de cómo el dinero de la empresa se dispersa a nivel mundial, es basado en web por lo que los bancos pueden tener acceso a la carpeta de los documentos asociados a discreción de la empresa o utilizar el intercambio electrónico de datos, eliminando la necesidad del intercambio de los documentos de papel, además brinda la posibilitad de eliminar virtualmente los errores y discrepancias (18).

Con el estudio de los sistemas contables informáticos expuestos anteriormente que gestionan los procesos de cartas de crédito, se puede afirmar que constituyen buenas soluciones para las finanzas empresariales. Los mismos brindan una serie de funcionalidades que garantizan la eficiencia en las operaciones bancarias. Estos sistemas extranjeros han alcanzado un gran prestigio internacionalmente, desarrollados sobre tecnología de punta, por lo que hoy en día marcan un elevado nivel en la industria de software. 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 rol que presenta como banco comercial, interactuando con diferentes países. Una de las razones que dificulta la utilización de estos programas es la dualidad monetaria, una característica muy particular del sistema bancario cubano, para lo cual no fueron concebidos. Por otro lado una limitante lo constituye las licencias privativas y altos precios, dificultando tanto el uso como el mantenimiento del mismo.

1.3.2 Sistemas contables informáticos nacionales

Los sistemas contables informáticos en Cuba están enfocados fundamentalmente a mejorar la eficiencia en la gestión contable empresarial (15), por lo que el estudio se centrará en los sistemas utilizados por el BNC.

SABIC es un sistema automatizado para la Banca Internacional de Comercio, diseñado y desarrollado por la Dirección de Sistemas Automatizados del BCC[6] para satisfacer las necesidades de procesamiento de datos de bancos e instituciones no bancarias, utilizando los medios técnicos de computación disponibles en el mercado. Está desarrollado en el lenguaje de programación FoxPro para el sistema operativo MS-DOS. También existe una versión en Visual FoxPro para Windows con Servidor de Base de Datos: Microsoft SQLServer. Entre las características fundamentales del SABIC expuestas en (19) se encuentran:

- ✓ Contabilización en tiempo real: permite la actualización de los ficheros contables justo en el momento de hacer las operaciones.
- ✓ Contabilización en varias monedas: permite contabilizar los activos y pasivos en sus monedas de origen, sin tener que realizar las conversiones correspondientes, lo cual aumenta en exactitud la información sobre la posición financiera de la entidad.
- ✓ Operación con transacciones: permite que las operaciones puedan realizarse usando transacciones tipificadas que crean asientos automáticamente.
- ✓ Estructura modular: permite que además de los módulos que contiene el sistema y que pueden ser enriquecidos, puedan incorporarse otros módulos nuevos, según las demandas que tenga cada entidad en particular.

Debido a la evolución de las actividades que se llevan a cabo dentro de las instituciones financieras bancarias cubanas se ha hecho necesario la integración de otras funcionalidades al sistema por lo que existen varias versiones del SABIC:

- ✓ La primera versión utiliza el MS-DOS como sistema de explotación lo cual constituye una limitación al ser este último monotarea, es decir, el microprocesador solo puede atender un único proceso. Esta primera versión fue realizada en FoxPro y utiliza un servidor de ficheros lo que conlleva a un tráfico excesivo dentro de la red. Por esta razón se decidió realizar una nueva versión en un ambiente cliente-servidor (20).
- ✓ La segunda versión fue en el año 2000 y se realizó bajo la filosofía de tener en un corto tiempo un sistema que utilizara las ventajas de la técnica cliente-servidor; pero sin realizar un nuevo diseño del mismo, por lo que siguieron persistiendo problemas de inadaptabilidad con todos los procesos que se llevan a cabo dentro de un banco. Utiliza como lenguaje Visual FoxPro que funciona sobre Windows con el servidor de base de datos a Microsoft SQLServer. Esta aplicación

[6] *Banco Central de Cuba.*

se ejecuta en cada máquina cliente de forma local para aprovechar la capacidad de procesamiento de cada una de ellas y no recargar al servidor innecesariamente. La lógica del negocio de esta aplicación se encuentra en forma de procedimientos almacenados en la base de datos. Esta versión no se ajustó a las características del BNC, pues este requería de nuevas funcionalidades (20).

El SABIC era el sistema que se utilizaba en el BNC para el año 2008, sus limitaciones dieron lugar al desarrollo del sistema Quarxo que hoy se usa en dicha entidad. El Sabic resultaba incompatible con el sistema de mensajería (SISCOM, que funciona sobre Windows). Presentaba una serie de limitaciones tales como: no permitía obtener y crear reportes dinámicos, no permitía gestionar información estadística de procesos de negocios, era un sistema puramente contable.

QUARXO v1.0.0 es un sistema desarrollado en la UCI por el proyecto SAGEB para el BNC. Está basado en web y contiene un conjunto de subsistemas que garantizan los procesos fundamentales tales como: cartas de crédito, préstamos, depósitos, cheques, cartas de remesas, gestión de cobros y pagos, contabilidad, inventario, vencimientos y mensajería interbancaria. Posibilita la generación automática de la mayoría de las transacciones contables a partir de los datos de entrada del negocio, genera los reportes que muestran información específica sobre la actividad contable. Posee integración con el sistema de mensajería y ofrece el tratamiento requerido a la validación de los datos de entrada, facilitando la detección de forma temprana de cualquier irregularidad.

Quarxo está integrado por varios subsistemas, dentro de ellos el subsistema Cartas de Crédito creado con el objetivo fundamental de llevar a cabo la gestión de los procesos de las cartas de crédito. El mismo está compuesto por seis módulos, de ellos, los módulos Emisión, Documento de Embarque, Negociación y Discrepancia, resuelven básicamente la gestión de las cartas de crédito y la negociación de las mercancías. El módulo Capacidad Financiera es usado para gestionar las capacidades financieras otorgadas a los clientes a modo de garantías y un sexto módulo nombrado Acuerdo resuelve la gestión de los respaldos de las cartas de crédito.

Emisión

Dentro del módulo emisión el usuario puede registrar la carta de crédito, para ello debe insertar una serie de datos entre los cuales se encuentran: el ordenante, el importe, la moneda, el proveedor y la descripción de la mercancía. Una vez insertada la carta de crédito en la base de datos, el usuario puede haciendo uso del buscador, consultar dicha carta de crédito, actualizarla en caso necesario o pasar a autorizarla, operación que solo puede realizar el personal autorizado en el banco. En el autorizo de una carta de crédito el usuario deberá insertar el banco corresponsal, el respaldo, el tipo de cambio, la tasa de interés y la forma de financiamiento. Luego de autorizada la carta de crédito corresponde su emisión y para ello el usuario insertará las fechas de vencimiento y de la última fecha de embarque, a la vez que configurará los mensajes que serán enviados al banco

corresponsal. Antes de emitir la carta de crédito el usuario tiene la posibilidad de observar la transacción contable que se generará. Con la emisión, la carta de crédito queda contabilizada y los mensajes de la misma quedan enviados a los bancos correspondientes. Otra de las operaciones que se realizan en el módulo es la enmienda, que consiste en modificar una carta de crédito luego de haber sido emitida, asimismo el módulo ofrece posibilidades de realizar pagos de la carta de crédito cuando se requieran y de cancelarla finalmente. A pesar de ello el módulo presentan un grupo de dificultades tales como: en el registro no se permite guardar el banco en el cual se registró previamente la CC, la funcionalidad actualizar solo se puede utilizar para las CC que no han sido contabilizadas, haciéndose necesario que el administrador de la base de datos realice manualmente modificaciones sobre CC. Por otra parte, no es posible autorizar las CC como asegurada y no asegurada o con varios tipos de deudas a la vez.

Documentos de Embarque

En el módulo Documentos de Embarque el usuario puede gestionar todo lo referente a la mercancía una vez que haya sido recibida. Cada documento de embarque contiene la información de la mercancía que acompaña. La información abarca las fechas y la carta de crédito a la cual se relaciona la mercancía. Dentro del módulo, el usuario puede registrar un nuevo documento de embarque y haciendo uso del buscador, puede modificar, eliminar o consultar uno existente.

Discrepancias

Existen mercancías que a su recibo presentan ciertas discrepancias con lo pactado en los contratos de la carta de crédito, las mismas se registran en documentos de embarque de manera habitual y a través del módulo Discrepancias reciben tratamiento. En el módulo el usuario puede registrar una nueva discrepancia y para ello el usuario deberá seleccionar el documento de embarque y el banco beneficiario, por medio del buscador el usuario puede además consultar, actualizar o cancelar una discrepancia previamente registrada. A pesar de ello el módulo no permite imprimir automáticamente las cartas de discrepancias a enviar a las empresas y se ha solicitado por el BNC incluir el envío de varios mensajes nuevos en el registro de la discrepancia.

Negociación

El módulo Negociación fue realizado precisamente para negociar las mercancías recibidas de las cartas de crédito una vez que ya han sido registradas en el sistema a través de los documentos de embarques. El usuario debe seleccionar el documento de embarque, configurar el calendario en caso de que la carta de crédito de la negociación sea financiada y entrar el importe. En el módulo Negociación el usuario puede además, haciendo uso del buscador, consultar, actualizar y unir las negociaciones previamente registradas. A pesar de ello el módulo presenta un grupo de deficiencias tales como: en el registro de una negociación sobre una CC financiada no es posible definir la misma como asegurada o no asegurada ni con el tipo de deuda, no es posible realizar el respaldo del pagaré ni de la letra de cambio por las cuenta principal e interés y no se registran

automáticamente las negociaciones compuestas. Asimismo el BNC ha solicitado incluir el envío de varios mensajes nuevos en el registro de las negociaciones.

Acuerdo

El módulo Acuerdo les permite a los usuarios de las Direcciones de Gestión de Negocios (DGN) gestionar los acuerdos tomados donde se establecen todos los procedimientos y condiciones para tramitar las operaciones bajo financiamiento. En el módulo Acuerdo el usuario puede registrar un nuevo acuerdo y haciendo uso del buscador puede también modificar, eliminar o consultar uno existente.

Capacidad Financiera

El módulo Capacidad Financiera tiene como objetivo principal el control de las operaciones que se realizan con respaldo, siendo en el caso del BNC, Cuenta Única. En el módulo Capacidad Financiera el usuario puede registrar una capacidad financiera, reservar una parte de la capacidad financiera para ser utilizada en una carta de crédito y haciendo uso del buscador puede también modificar y consultar las capacidades existentes.

El subsistema Cartas de Crédito de Quarxo ha contribuido significativamente a aumentar la rapidez y eficiencia de los procesos relacionados con las cartas de crédito en el BNC, reduciendo el tiempo dedicado a las mismas en un 65% (*ver Anexo1*), Con él no solo se logra mayor rapidez en los procesos, sino también, el almacenamiento de un conjunto de información estadística de gran valor para la toma de decisiones en la entidad. El subsistema Cartas de Crédito de Quarxo unido a la solución de las deficiencias y solicitudes presentadas anteriormente, representa la mejor solución para el BNC. Será indispensable entonces sobre la base de la versión existente, realizar el desarrollo de una nueva fase.

1.4. Metodología de desarrollo de software

Una metodología para el desarrollo de software es un conjunto de filosofías, frases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de sistemas informáticos (21).

Una metodología de desarrollo de software en general constituye un marco de trabajo basado en un conjunto de modelos, herramientas y métodos aplicados a la Ingeniería de Software usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información. Actualmente no existe una metodología universal que le haga frente con éxito a cualquier proyecto de desarrollo de software, puesto que para la selección de la metodología se ha de tener en cuenta las características del equipo de desarrollo. Es muy importante para lograr un producto con calidad escoger bien la metodología, las más utilizadas actualmente son: RUP (*en inglés: Rational Unified Process*), XP (*en inglés: Extreme Programming*) y MSF (*en inglés: Microsoft Solution Framework*).

RUP se adapta mejor a los proyectos de largo plazo, dividiendo el desarrollo de software en cuatro fases desarrolladas iterativamente y abarcando seis disciplinas ingenieriles y tres de apoyo; XP se recomienda para proyectos cortos y tiene por particularidad contar con el usuario final como parte del equipo y MSF se adapta a cualquier tipo de proyecto y se centra en los modelos de procesos y de equipo (16).

El CEIGE ha definido como metodología de desarrollo de software para sus aplicaciones el modelo de desarrollo software atendiendo a las características de los proyectos del centro y las áreas de procesos del nivel dos de CMMI[7]. El mismo será utilizado para el desarrollo de la solución, puesto que incluye la especificación de las actividades de cada una de las fases del ciclo de vida de los proyectos del centro y define cada uno de los artefactos a generar en cada momento independientemente de las herramientas o métodos que se utilicen para ello.

1.4.1. Modelo de desarrollo de software V1.1

El modelo de desarrollo de software es una guía o modelo estandarizado, que establece las distintas fases para el desarrollo de software en los proyectos del centro, el conjunto de procesos y productos de trabajo a generar en cada una de ellas teniendo en cuenta las áreas de proceso de CMMI nivel 2 para la UCI.

Descripción de las fases del ciclo de vida de los proyectos:

✓ **Inicio o Estudio preliminar:** durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto a un alto nivel, la evaluación de la factibilidad del proyecto y el registro de este. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo, costo y decidir si se ejecuta o no el proyecto.

Los objetivos de la fase son:

- Asegurar la factibilidad del proyecto.
- Establecer un plan para la ejecución del proyecto.

Hitos:

- Plan de desarrollo de software.
- Acta de inicio del proyecto firmada.

✓ **Desarrollo**: en esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura.

[7] *Capability Maturity Model Integration. Modelo de Capacidad de Madurez del Software.*

Durante el desarrollo se refinan los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.

El objetivo de esta fase es:

- Obtener un sistema que satisfaga las necesidades de los clientes y usuarios finales.

Hito:

- Producto liberado por entidad certificadora de calidad.

En esta fase se ejecutan las disciplinas Modelado de negocio, Requisitos, Análisis y diseño, Implementación, Pruebas internas y Pruebas de liberación (*ver Figura 1.1*) (22).

Figura 1.1. Ciclo de vida del proyecto SAGEB

Los artefactos generados para cada una de estas disciplinas son los siguientes:

- Modelado de negocio
  - Descripción de procesos de negocio.
  - Modelo Conceptual.
  - Reglas del negocio.
- Requisitos
  - Especificación de requisitos de software.
  - Descripción de requisitos de software.
  - Documento de salida del sistema.

- Análisis y diseño
  - Modelo de Diseño.
  - Diseño de casos de pruebas.
- Implementación
  - Modelo de implementación.
  - Ficheros de código.
- Pruebas internas
  - Acta de liberación de calidad.
- Pruebas de liberación
  - Certificado de aceptación del BNC.

1.5. Lenguajes

Un lenguaje informático es un idioma usado por ordenadores por lo que en ocasiones se refiere como lenguaje de ordenadores, diseñados para transmitir información mediante equipos de cómputo. La clasificación de lenguaje informático engloba diferentes tipos tales como los lenguajes de programación, modelado, consulta, sonido, gráfico, marcas y pseudocódigos. En esta sección se describirán los diferentes lenguajes de modelado y de programación utilizados en el desarrollo del producto.

1.5.1. Lenguaje de modelado y notación

UML[8] es el estándar industrial de mayor utilización en la actualidad, patrocinado por el Grupo de Gestión de Objetos (*OMG, del inglés Object Management Group*), usado fundamentalmente para visualizar, especificar, construir y documentar los artefactos de un sistema de software (24). El mismo incluye reglas para describir un plano del sistema (modelo), así como aspectos conceptuales tales como: procesos de negocio y funciones del sistema, aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables (25). UML utiliza herramientas de modelado visual facilitando la gestión de dichos modelos, permitiendo ocultar o exponer detalles cuando sea necesario. Ayuda a mantener la consistencia entre los artefactos del sistema: requisitos, diseños e implementaciones. En resumen, ayuda a mejorar la capacidad del equipo para gestionar la complejidad del software (26).

BPMN[9] es una notación gráfica que describe la lógica de los pasos de un proceso de Negocio. Esta notación ha sido especialmente diseñada para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes de las diferentes actividades. Proporciona un lenguaje común para que las partes involucradas puedan comunicar los procesos de forma clara, completa y eficiente. De esta forma BPMN define la notación y semántica de un Diagrama de Procesos de

[8] *Unified Modeling Language. Lenguaje de modelado unificado.*
[9] ***B**usiness **P**rocess **M**anagement **N**otation. Notación para el modelado de procesos de negocio.*

Negocio (*BPD, del inglés Business Process Diagram*) (27). El modelado en BPMN se realiza mediante diagramas muy simples con un conjunto muy pequeño de elementos gráficos. Con esto se busca que para los usuarios del negocio y los desarrolladores técnicos sea fácil entender el flujo y el proceso.

UML será usado en el modelado del sistema por su facilidad y rigor en la especificación. Con su utilización se podrá automatizar determinados procesos y generar código a partir de los modelos y a la inversa (a partir del código fuente generar los modelos) permitiendo que el modelo y el código estén actualizados, pudiéndose además mantener la visión en el diseño y efectuar la verificación y validación del modelo realizado. La notación BPMN será usada en la confección de BPD[10] por su facilidad y eficiencia para modelar los procesos de negocio. Dicha notación proporciona elementos que posibilitará la representación clara de la interacción entre las entidades de negocio, las secuencias de actividades y patrones de intercambios de mensajes entre los participantes.

1.5.2. Lenguajes del lado del servidor

Se les clasifica así a los lenguajes que son reconocidos, interpretados y ejecutados en el servidor web, justo antes de que se envíe la página a través de Internet al cliente. Las páginas que se ejecutan en el servidor pueden realizar accesos a bases de datos, conexiones en red y otras tareas para crear la página final que verá el cliente (28).

Existen diferentes lenguajes de gran popularidad destacándose por la cantidad de software que se han desarrollado sobre ellos y los resultados obtenidos con las aplicaciones, algunos de estos son: C# desarrollado y estandarizado por Microsoft que soporta el paradigma orientado a objetos, PHP diseñado para la creación de páginas web dinámicas siendo además multiparadigma y Java lenguaje de código abierto y multiplataforma. A continuación se presentan las principales características de los lenguajes del lado del servidor que serán usados en el desarrollo de la propuesta de solución.

Java 6 es un lenguaje de programación orientado a objetos. Inspirado en el lenguaje C++, proyectado con la finalidad de obtener grandes productos en entornos empresariales. Es simple y portátil sobre diferentes plataformas y sistemas operativos, lo que significa que los programas Java pueden ser ejecutados sobre cualquier computadora en la cual esté instalada la máquina virtual. Es usado en la web para la carga y ejecución de programas en el ordenador cliente (29). Java es un lenguaje de programación orientado a objetos, robusto y fácil de aprender, diseñado para crear software altamente fiable, permite la codificación de la propuesta de solución de una manera rápida y flexible, al integrarse con los diferentes marcos de trabajos de desarrollo, permite programar

[10] *Business Process Diagram. Diagrama de procesos de negocio.*

páginas web dinámicas con accesos a bases de datos utilizando XML con cualquier tipo de conexión de red entre cualquier sistema.

Java fue seleccionado por el equipo de arquitectura para su empleo en el desarrollo atendiendo a las ventajas que este ofrece y al conocimiento y preparación del equipo de desarrollo con que se contaba al iniciar el proyecto, basándose además en los resultados obtenidos por el proyecto SIGEP[11], el cual usó este lenguaje y desarrolló varias librerías para el mismo con el objetivo de llevar a cabo un desarrollo mucho más rápido logrando montar una arquitectura robusta que se adaptaba a las necesidades del sistema Quarxo.

1.5.3. Lenguajes del lado del cliente

Un lenguaje del lado cliente es totalmente independiente del servidor, lo cual permite que la página pueda ser albergada en cualquier sitio, para su correcta visualización es necesario que la computadora cliente tenga instalados los plugins adecuados. Una de las desventajas de los lenguajes del lado del cliente es que el código tanto del hipertexto como de los scripts, es accesible a cualquiera y ello puede afectar a la seguridad (28). Seguidamente se presentan las principales características de los lenguajes del lado del cliente que serán usados en el desarrollo de la propuesta de solución.

Javascript es un lenguaje de programación que realiza acciones dentro del ámbito de una página web. Su compatibilidad con la mayoría de los navegadores modernos, lo posiciona como el lenguaje de programación del lado del cliente más utilizado. El código es ejecutado en el cliente por lo que el navegador (browser) se encarga de interpretar las instrucciones Javascript y ejecutarlas para realizar estos efectos e interactividades (28). Es un lenguaje interpretado, que no requiere compilación y es utilizado comúnmente para la construcción de páginas web en combinación con el HTML, para el desarrollo de aplicaciones cliente-servidor dentro de Internet. Brinda respuesta a eventos locales dentro de la página y posibilita la realización de cálculos en tiempo real. Permite la validación de formularios dentro de una página, la personalización de la página por el usuario y la inclusión de datos del propio sistema (19).

JSP[12] es un lenguaje para la creación de sitios web dinámicos, orientado a desarrollar páginas web en Java y es multiplataforma. Desarrollado por Sun Microsystems creado para programar aplicaciones web potentes. Posee un motor de páginas basado en los servlets de Java y necesita tener instalado un servidor Tomcat (28).

En este caso se definió en el proyecto el uso del Javascript para la validación de los datos de entrada y JSP por su integración al lenguaje base Java, como es precisamente atendiendo a una

[11] *Sistema de Gestión Penitenciaria*
[12] *Java Server Pages. Páginas Servidoras de Java*

de sus principales características la posibilidad de crear páginas web con programación en Java que permitiendo mezclar HTML estático con HTML generado dinámicamente.

1.6. Herramientas

En informática, las herramientas son un conjunto de programas que se usan para crear un determinado producto normalmente otro programa o sistema informático. Resulta fundamental después de haber definido la metodología que guiará el proceso de desarrollo del software y los lenguajes que se utilizarán para la realización del diseño e implementación del sistema definir las herramientas adecuadas sobre la que se deberá desarrollar el producto.

1.6.1. Herramientas de modelado CASE[13]

Las herramientas CASE son un complemento de la caja de herramientas del ingeniero del software. Ayudan a asegurar la calidad de un producto desde su diseño antes de construirlo (19). La principal ventaja de la utilización de una herramienta CASE, es la mejora de la calidad de los desarrollos realizados y, en segundo término, el aumento de la productividad. Existen varias herramientas que permiten una modelación con calidad, dentro de las más usadas se encuentran: Rational Rose, herramienta de modelado visual para el análisis y diseño de sistemas basados en objetos que se utilizan para modelar un sistema antes de proceder a construirlo y el Visual Paradigm que contiene características gráficas muy cómodas, que facilitan la realización de los diagramas de modelado que sigue el estándar de UML. A continuación se presentan las principales características de la herramienta que será usada en el modelado de la solución.

Visual Paradigm para UML 8.0 es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue. Ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación (19). Además es multiplataforma.

Luego de un análisis realizado por parte de la dirección del proyecto sobre las posibles CASE a utilizar se decidió llevar a cabo el modelado del sistema con el Visual Paradigm La herramienta se seleccionó por las características que presenta y las facilidades que brinda al integrarse con la notación BPMN y UML.

[13] *Computer Aided Software Engineering. Ingeniería de Software Asistida por Computadora*

1.6.2. Entorno de desarrollo integrado

El entorno de desarrollo integrado (*IDE, por sus siglas en inglés Integrated Development Environment*) es un programa compuesto por un conjunto de herramientas para un programador. El mismo puede dedicarse a un solo lenguaje de programación o a varios. Además puede formar parte de aplicaciones existentes o ser una aplicación por sí solo. Estos sistemas informáticos han sido empaquetados como programas de aplicaciones. A continuación se presentan algunas características del IDE que será para el desarrollo de la aplicación.

Eclipse 3.5 JEE GALILEO IDE: es un entorno de desarrollo integrado, de código abierto y multiplataforma. Fue creado por la IBM (en inglés: International Business Machines) bajo la filosofía de software libre. Aunque Eclipse se usa como IDE para java, la arquitectura de plugins que posee permite integrar diversos lenguajes sobre un mismo IDE, además de introducir otras aplicaciones que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces y ayuda en línea para librerías, entre otros (16).

Plataforma J2EE: *(conocido del inglés: Java 2 Platform, Enterprise Edition*) define un estándar para el desarrollo de aplicaciones empresariales multicapa. La plataforma J2EE ofrece mejores perspectivas de desarrollo al basar su arquitectura en productos de software libre. J2EE entre sus principales ventajas se encuentra que permite soporte de múltiples sistemas operativos, marcos de trabajos y patrones de programación que permiten responder de una forma robusta y flexible a todas las demandas de este tipo de aplicaciones (19).

Después de un análisis se decidió por la dirección de proyecto en un inicio el usó de este IDE sobre la plataforma J2EE por las caractrerísticas que presenta y la experiencia de los desarrolladores con los mismos, comprobándose que en su mayoría usaban Eclipse.

1.6.3. Contenedor web

Como soporte para el funcionamiento de las aplicaciones desarrolladas en Java existen varios entornos de ejecución como Jboss Server que combina una arquitectura orientada a servicios revolucionaria con una licencia de código abierto, GlassFish que implementa la plataforma J2EE y soporta las últimas versiones de tecnologías como: JSP, JSF, servlets, arquitectura Java para enlaces XML y Apache Tomcat, el cual no es considerado como un servidor de aplicaciones, sino como un contenedor de servlets, pero es capaz de soportar la mayoría de las aplicaciones desarrolladas en dicha tecnología.

Apache Tomcat 6.0.35 es un contenedor web basado en el lenguaje Java que actúa como motor de Servlets y Java Server Pages. Desarrollada bajo el proyecto Jakarta de la Fundación Apache (*en*

inglés: Apache Software Foundation). Está desarrollado en un entorno abierto Open Source por lo que su implementación es libre. Es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Teniendo en cuenta las características expuestas, se selecciona Apache Tomcat como servidor de aplicaciones. Debido a que todos los servidores analizados tienen mucha similitud en lo referente a sus funcionalidades, se utilizó la experiencia del equipo de desarrollo y los bajos recursos que este demanda como factores fundamentales en la toma de esta decisión.

1.6.4. Gestor de base de datos

Una base de datos se le llama a un conjunto de datos informativos organizados en un mismo contexto para su uso y vinculación. Los gestores de base datos son un conjunto de programas que permiten almacenar la información, modificarla y extraerla de una base de datos, además de proporcionar herramientas para añadir, borrar modificar y analizar los datos. Es decir se entiende por base de datos un conjunto de datos y como gestor de base de datos una aplicación capaz de manejar este conjunto de datos de manera eficiente y cómoda.

Existen numerosos gestores de base datos, entre los más utilizados se encuentran: PostgreSQL un sistema de gestión de base de datos relacional orientada a objetos y libre, publicada bajo licencia BSD, un programa de código abierto, multiplataforma, diseñado para ambientes de alto volumen, MySQL es muy sencillo, su arquitectura lo hace fácil de usar e increíblemente rápido, es uno de los motores de base de datos más usados en Internet, la principal razón de esto es que es gratis para aplicaciones no comerciales, por otro lado tenemos a Oracle uno de los sistemas más completos, destacando su soporte de transacciones, estabilidad, escalabilidad y soporte multiplataforma, su dominio en el mercado de servidores empresariales ha sido casi total, recientemente sufre la competencia del Microsoft SQL Server, que es una plataforma de base de datos y de análisis de datos, posee alta disponibilidad y escalabilidad, ofrece flexibilidad, se utiliza en el procesamiento de transacciones en línea a gran escala, en el almacenamiento de datos y las aplicaciones de comercio electrónico (19). Se seleccionó como gestor de base de datos a utilizar Microsoft SQL Server a continuación se presentan algunas de sus características.

Microsoft SQL Server 2005 es un sistema para la gestión de bases de datos producido por Microsoft basado en el modelo relacional. Entre sus principales características se encuentra: la escalabilidad, estabilidad y seguridad, es un sistema de reserva de memoria muy rápido, proporciona sistemas de almacenamiento transaccional y no transaccional, funciona en diferentes plataformas, es relativamente sencillo de añadir otro sistema de almacenamiento y administrar información de otros servidores de datos. Sus lenguajes de consulta son el SQL y el T-SQL Apache

(*en inglés: Transact-SQL*), siendo este último el principal medio de programación y administración del servidor. Probado con un amplio rango de compiladores diferentes, escrito en C y en C++. Requiere para su funcionamiento el sistema operativo Microsoft Windows, representando esto un inconveniente para su utilización (30).

Este gestor de base datos aunque es un software privativo, se seleccionó tomando en cuenta la petición del cliente, por la familiarización que presenta con la herramienta y considerando los procedimientos que ya estaban almacenados en este y que por cuestiones de tiempo no podrían ser implementados en otro gestor.

1.7. Marcos de trabajo

Los marcos de trabajos son básicamente una estructura de soporte que determina la arquitectura sobre el cual puede ser desarrollado un software. Diseñados fundamentalmente para acelerar el proceso de desarrollo, reutilizar código y promover buenas prácticas de desarrollo como el uso de patrones. Existen diversos tipos de marcos de trabajos en la industria del software, con propósitos distintos, algunos están orientados al desarrollo de aplicaciones web. El entorno de desarrollo utilizado J2EE incluye diferentes marcos de trabajos de utilidad como Spring, Hibernate, Dojo, Tapestry, WebWork, entre otros. Existen marcos de trabajos que son empleados específicamente para los lenguajes anteriormente seleccionados (19).

1.7.1 Marcos de trabajo del lado del servidor

Existe un conjunto de marcos de trabajos utilizados en el servidor que hacen posible la aplicación del patrón MVC y que a su vez poseen una elevada aceptación a nivel mundial, ellos son: Struts framework el cual es libre, permite reducir el tiempo de desarrollo y es compatible con todas las plataformas en las que Java Enterprise esté disponible, JBoss Seam framework desarrollado por JBoss donde se puede acceder a cualquier componente EJB (*en inglés: Enterprise JavaBeans*) desde la capa de presentación refiriéndose a él mediante su nombre de componente seam y Spring Framework incluido por la plataforma J2EE. De las tecnologías anteriores se seleccionó Spring Framework para su utilización del lado del servidor, a continuación se presentan sus principales características.

Spring Framework 2.5.6

Spring es un popular marco de trabajo de aplicaciones de código abierto, contiene en conjunto de librerías de clases que brindan una estructura conceptual que hace más fácil el desarrollo de aplicaciones basadas en la plataforma Java/J2EE. Ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas que permiten desarrollar un software seguro y robusto.

Su funcionamiento se basa en inversión de control e inyección de dependencia, apoyándose en el API de Java Reflection (31).

El marco de trabajo Spring fue seleccionado por el equipo de arquitectura del proyecto SAGEB debido a que este es ideal para aplicaciónes de gran tamaño, posee una comunidad de gran prestigio que facilita la retroalimentación, se acopla perfectamente a la plataforma J2EE y su uso propicia la aplicación del patrón de arquitectura MVC. Además de las facilidades que brinda fue considerada su uso por la experiencia en proyectos exitosos como SIGEP del cual fue tomado la arquitectura base de Quarxo.

Spring WebFlow 2.0.9

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 especifican las reglas (32).

Spring WebFlow será usado para los flujos complejos que no pueda ser manejado por los controladores definidos en Spring Web MVC, permitiendo de una manera sencilla definir en archivos XML de configuración el desarrollo total del flujo. Su uso se debe a la fácil integración con la plataforma de Spring Web MVC para la definición de un lenguaje declarativo de flujos.

1.7.2 Marcos de trabajo del lado del cliente

En las interfaces de usuario se utilizan marcos de trabajos debido a la importancia que estas poseen en los sistemas informáticos, teniendo en cuenta su interacción con los usuarios. Existen diversas librerías desarrolladas para JavaScript que enriquecen las interfaces de usuario y brindan un ambiente de trabajo más limpio para el uso de Ajax, (*en inglés: Asynchronous JavaScript And XML*), entre ellas se destacan: Qooxdoo de código abierto para el desarrollo de aplicaciones web interactivas que permite al desarrollador abstraerse del HTML, CSS (en *inglés: Cascading Style Sheets*) y DOM (*en inglés: Document Object Model*), ExtJS creado con el fin de desarrollar aplicaciones web interactivas usando tecnologías AJAX y DOM que puede ejecutarse como una aplicación independiente y Dojo Toolkit permite realizar peticiones asincrónicas al servidor e incorpora soporte para el trabajo con la técnica AJAX (14). De las tecnologías anteriores se seleccionó Dojo Toolkit para su uso del lado del cliente, a continuación se presentan sus principales características.

Dojo Toolkit 1.3

Dojo Toolkit contiene APIs (*en inglés: Application Programming Interface*) y controles (*en inglés: widgets*) que ayudan al desarrollo de aplicaciones web enriquecidas en el cliente, empaquetando los efectos visuales de la interfaz de usuario, abstracción de eventos, almacenamiento de APIs en el cliente, e interacción de APIs con AJAX. Destacado por permitir el desarrollo de aplicaciones que pueden funcionar con independencia del navegador web donde se ejecuten, ofrece soporte para la internacionalización e incluye un gran cúmulo de componentes visuales basados en HTML, CSS y Javascript para enriquecer la interfaz de usuario (33).

Dojo fue seleccionado por el equipo de arquitectura debido a su fácil integración con Spring MVC y teniendo en cuenta que el proyecto SIGEP desarrolló una gran cantidad de componentes y funciones las cuales se podían reutilizar en el desarrollo del sistema Quarxo.

1.7.3 Marcos de trabajo para el acceso a datos

Los marcos de trabajo ORM[14] se utilizan para el trabajo con el acceso a datos de una aplicación, entre ellos se puede mencionar a: IBATIS una solución híbrida con las mejores ideas de las tecnologías ORM y Mapeador de datos (*del inglés: Data Mapper*) y soporta procedimientos almacenados, SQL dinámico, SQL en línea y el Mapeo Objeto Relacional, Spring JDBC es un módulo perteneciente al framework Spring el cual posee algunas librerías de clases para trabajar con base de datos a través del API de Java JDBC y brinda soporte para invocar funciones almacenadas y procedimientos e Hibernate Framework una solución para el lenguaje de programación Java, concebido bajo la filosofía del código abierto. De las tecnologías anteriores se seleccionó Hibernate Framework para su uso en el acceso a datos, a continuación se describen sus principales características.

Hibernate 3.5

Hibernate es un marco que abstrae el trabajo con la base de datos, soportando la manipulación de los datos persistentes objetualmente, a través de ficheros que mapean clases java contra tablas. 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. 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*

[14] *Object Relational Mapping. Mapeo objeto-relacional.*

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*) (34).

Fue seleccionado principalmente por la sólida integración con Spring Framework, la facilidad que brinda en el control sobre las sesiones y el gran número de métodos para el trabajo con los datos que incluye la clase *HibernateTemplate.*

1.8. Patrones de software

Los patrones de software estandarizan principios y buenas prácticas en la solución de sistemas informáticos. Estos han permitido agrupar soluciones a problemas existentes en la construcción de aplicaciones y generar una respuesta común que resuelva de forma genérica las principales deficiencias en la creación de software.

1.8.1 Patrones de arquitectura

Los patrones de arquitectura ofrecen soluciones a problemas de arquitectura en la ingeniería de software. Estos expresan un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. Describen los elementos, relaciones y el conjunto de restricciones sobre su empleo.

Las arquitecturas más utilizadas para el desarrollo de software son: Monolítica, Cliente-Servidor y Tres Capas. Esta última constituye una especialización de la arquitectura Cliente-Servidor, donde la carga se divide en tres capas, con un reparto claro de funciones: una capa para la presentación otra para el modelado el negocio y una tercera para el almacenamiento, una capa solamente tiene relación con la siguiente. La arquitectura del sistema Quarxo está basada en tres capas empleando el patrón de arquitectura MVC, seguidamente se presentan algunas de sus características.

MVC es un patrón de arquitectura que pertenece a la familia de los estilos arquitectónicos de llamada y retorno. Separa los datos de una aplicación, la interfaz de usuario y la lógica de negocio en tres componentes distintos: Modelo, Vista y Controlador. El modelo encapsula los datos y la funcionalidad de la aplicación. La vista despliega la información contenida en el modelo. El controlador está asociado a cada vista, recibe entradas que se traduce en invocaciones de métodos del modelo o de vista. De ahí que su empleo garantiza una mayor claridad en el diseño y facilita una mayor escalabilidad. Además fomenta la reutilización de los componentes y simplifica el mantenimiento de los sistemas.

La arquitectura definida para el desarrollo del sistema Quarxo está basada en una arquitectura de tres capas lógicas fundamentales: Capa de Presentación, Capa de Lógica de Negocio, Capa de Acceso a Datos y una capa transversal a las otras con los objetos del dominio. Para esta arquitectura se definió por el equipo desarrollo utilizar el patrón MVC, que lo incluye el framework Spring, siendo este parte de su infraestructura extensible y de gran soporte para el desarrollo de aplicaciones web.

1.8.2 Patrones de diseño

Los patrones de diseño son la base de las soluciones a problemas comunes en el desarrollo de software. Estos brindan una solución probada y documentada de estos problemas en contextos similares. Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular (Buschman, 1996) (36).

Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. Estos brindan soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos y basadas en la experiencia. Para que una solución sea considerada un patrón se debe haber comprobado su efectividad resolviendo problemas similares con anterioridad además debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias. A continuación se mencionan algunos de los patrones que son utilizados en el desarrollo del diseño.

1.8.2.1. Patrones GRASP[15]

Los patrones GRASP describen las buenas prácticas de aplicación recomendable en el diseño de software y los principios fundamentales de la asignación de responsabilidades a objetos expresados en forma de patrones.

Experto: es un patrón que su usa para asignar la responsabilidad a objetos, indica que la creación de un objeto debe recaer sobre la clase que conoce toda la información necesaria para crearlo. Al emplear este patrón se garantiza que los objetos tengan su propia información para hacer lo que se les pide de esta manera se conserva el encapsulamiento. Soporta bajo acoplamiento lo que favorece por otro lado a desarrollar sistemas más robustos y de fácil mantenimiento. El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases "sencillas" y más cohesivas que son más fáciles de comprender y de

[15] *General Responsibility Assignment Software Patterns. Patrones Generales de Software para Asignar Responsabilidades*

mantener de esa manera se brinda soporte a una alta cohesión (24). Este patrón será usado para indicar la creación de objetos a las clases de acuerdo con la información que posea cada una.

Creador: es un patrón que guía la asignación de responsabilidades relacionadas con la creación de objetos identificando quien debe ser el responsable de la creación o instancia de nuevos objetos o clases. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento (24). El patrón se usará en la asignación de instancias solamente a las clases que requierán la creación del objeto.

Alta cohesión: es un patrón que caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme, lo que no las hace susceptibles a constantes cambios (20). Plantea que la información que almacena una clase debe ser coherente y estar en la mayor medida relacionada con la clase (21). Este patrón se tendrá en cuenta para el diseño de clases de manera que sean distribuidas por las capas y le sean asignadas las responsabilidades necesarias y bien delimitadas para garantizar una alta cohesión generando por ende bajo acoplamiento.

Bajo acoplamiento: es un patrón que estimula asignar una responsabilidad de modo que su colocación no incremente tanto el acoplamiento para que no se produzcan los resultados negativos propios de un alto acoplamiento. Soporta el diseño de clases más independientes, que reducen el impacto de los cambios y también más reutilizables, que acrecientan la oportunidad de una mayor productividad (24). Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y disminuyendo la dependencia entre las clases (17). El patrón será usado en la definición de interfaces e implementaciones de manera que los controladores se relacionen exclusivamente con ellas para realizar sus operaciones, disminuyéndose de esta forma el impacto de cambios posteriores en el negocio del sistema.

Controlador: es el intermediario entre una interfaz y la implementación de la lógica de las operaciones del sistema. Se responsabiliza por controlar el flujo de eventos del sistema. Asigna la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas, facilitando la centralización de las actividades (24). Este patrón será usado en la creación de clases controladoras que dividan las interfaces de usuarios de las clases del negocio.

1.8.2.2. Patrones estructurales GOF

Los patrones estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.

Fachada: el patrón de diseño Fachada (*en inglés: Facade*) sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Reduce el número de objetos con los que tiene que interactuar un cliente proporcionando un menor acoplamiento y facilitando el cambio de componentes sin afectar a sus clientes. Es el único punto de entrada para los servicios de un subsistema; la implementación y otros componentes del subsistema son privados y no pueden ver los componentes externos. Proporciona protección frente a los cambios en las implementaciones de un subsistema (24). El patrón será usado en la creación de clases fachadas de manera que divida la vista de la lógica de negocio.

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. Este patrón será usado para delegar responsabilidades cuando se solicita información desde la vista que se encuentra en la base de datos, permitiendo que la información sea atendida por las clases controladoras, luego por la fachada, el manager y finalmente las clases de acceso a datos que se encargaran de obtener la información en la base de datos.

1.8.2.3. Patrón de acceso a datos (DAO)

El patron de acceso a datos tiene como objetivo fundamental abstraer y encapsular todos los accesos a la fuente de datos aislando los objetos de negocio de una implementación particular de la persistencia. La ventaja de usar objetos de acceso a datos es que cualquier objeto de negocio no requiere conocimiento directo del destino final de la información que manipula y que se puede cambiar fácilmente de fuente de datos (24). Este patrón será utilizado en la creación de clases de acceso a datos con la función de servir como intermediarias entre las clases del negocio y la fuente de datos.

1.8.2.4. Patrón de presentación

Vistas compuestas: (*en inglés: Composite View*) gestiona los diferentes elementos de la vista por medio de una plantilla que hace la representación de las vistas más manejable. Este patrón es con el propósito de crear vistas compuestas de varias sub-vistas de forma modular, flexible y extensible para construir vistas de páginas JSP para aplicaciones J2EE. Por ejemplo, una página JSP que incluye otras páginas JSP y HTML usando la directiva *include*. Cualquier cambio realizado en una sub-vista es reflejado automáticamente en cada vista compuesta que la utilice. La vista compuesta

también maneja la disposición de las sus sub-vistas y la plantilla proporciona una apariencia consistente y facilidades a la hora de modificarla y mantenerla a lo largo de toda la aplicación (19). El uso de este patrón se tendrá en cuenta en la creación de los ficheros JSP de los módulos con la utilización de la etiqueta <jsp: include>para incluir líneas de código separados en otros archivos y que sean comunes para varias páginas.

1.9. Conclusiones parciales

En el presente capítulo se efectuó la fundamentación teórica de la investigación a través de la formulación de los principales conceptos respecto a los sistemas bancarios y el estudio de los sistemas contables informáticos existentes tanto a nivel nacional como internacional que incluye el manejo de las cartas de crédito dentro de sus funcionalidades. Conjuntamente se efectuó una caracterización de la metodología, lenguajes, tecnologías y herramientas a utilizar en el desarrollo de la solución. Dicha investigación arrojó los siguientes resultados:

✓ De los sistemas contables informáticos, el subsistema Cartas de Crédito de Quarxo representa la mejor solución para gestionar los procesos de cartas de crédito del BNC evidenciando la necesidad de desarrollar sobre la base una nueva fase que incluya funcionalidades y mejoras a los procesos correspondientes de los módulos de Emisión, Discrepancia y Negociación con el objetivo de enmendar las debilidades detectadas.

✓ La metodología seleccionada para guiar el desarrollo de la solución fue el modelo de desarrollo de software V1.1 propuesto por CEIGE dado que el mismo define las actividades y artefactos a generar en cada una de las fases del ciclo de vida de los proyectos del centro.

✓ Se utilizarán como lenguaje de modelado UML para el modelado del sistema y la notación BPMN para el modelado de procesos de negocio, JSP y JavaScript como lenguaje del lado del cliente y el lenguaje base Java V6.0.

✓ Será usado como herramienta de modelado Visual Paradigm V8.0, Ecplise como entorno de desarrollo integrado sobre la plataforma J2EE con el contenedor web Apache Tomcat V6.0. y SQL Server 2005 como gestor de base de datos.

✓ Se empleará una arquitectura en tres capas: modelo, vista y controlador, aplicando el patrón MVC y los marcos de trabajos seleccionados: Spring Framework V2.5.6 como marco base junto al Spring MVC V2.5.6, para los flujos de trabajos más complejos Spring WebFlow V2.0.9, Hibernate V3.5 como marco de trabajo para acceso a datos y DojoToolkit V1.3 para la decoración de los componentes.

✓ El desarrollo de la solución se desarrollarán con herramientas y tecnologías en su mayoría libres y multiplataforma, lo cual trae como resultado una reducción notable del costo del sistema por concepto de licencias de software y soporte.

Capítulo II: Análisis y diseño de la solución

2.1 Introducción

En el siguiente capítulo se exponen los artefactos generados por las disciplinas de modelado de negocio, requerimientos, análisis y diseño del modelo de desarrollo de software propuesto por CEIGE. En esta sección se plasma el modelo de procesos de negocio y la especificación de los requisitos funcionales a incorporar en el subsistema. Además se realiza una breve descripción de los elementos fundamentales de la arquitectura utilizada en el desarrollo, guiada por la arquitectura base del proyecto SAGEB. En la disciplina análisis y diseño se construye la estructura de la solución mediante el modelo de diseño donde se obtendrán una serie de artefactos que contribuirán a una visión detallada sobre cada uno de los requerimientos. De igual forma se ejemplifican los patrones utilizados en el diseño de la solución.

2.2 Modelado de negocio

Es la fase del modelo de desarrollo de software destinada a entender la estructura y dinámica de la organización. Se comprende cómo funciona el negocio que se desea automatizar para tener garantías de que el software desarrollado va a cumplir su propósito (19). Su objetivo fundamental radica en asegurar que los clientes, usuarios finales y desarrolladores tienen un entendimiento común de la organización.

2.2.1 Modelo de procesos de negocio

Un proceso de negocio es un conjunto estructurado de actividades, diseñado para producir una salida determinada o lograr un objetivo específico. Cada proceso de negocio tiene sus entradas, funciones y salidas. Las entradas son requisitos que deben tenerse antes de que una función pueda ser aplicada. Cuando una función es aplicada a las entradas de un método, tendremos ciertas salidas resultantes. Los procesos describen como es realizado el trabajo en la organización y se caracterizan por ser observables, medibles, mejorables y repetitivos (38).

Con el objetivo de comprender los procesos de negocio e identificar las necesidades reales de los clientes en el BNC se realizaron varias entrevistas. La misma es una técnica utilizada para recopilar la información necesaria y analizar las características de la institución generalmente son preguntas realizadas por los analistas de forma verbal a los usuarios del sistema o aquellos que proporcionan algún dato, no es una interrogación sino una forma de dialogar y llegar a un consenso sobre cómo se realizan los procesos en la organización. A continuación se enumeran los procesos de negocios definidos en el BNC para la gestión de las CC:

1. Actualizar CC.
2. Emitir cartas de discrepancias.

3. Poner en estado devuelta la CC.
4. Registrar las CC asegurada y no asegurada a la vez.
5. Establecer el respaldo del pagaré en el registro de negociaciones.
6. Enviar mensajes 799, 754, 999, 734,750 en negociaciones.
7. Enviar mensaje 750 en el registro de discrepancia.
8. Registrar automáticamente en la base datos las negociaciones compuestas.
9. Registrar las CC please open.
10. Establecer el respaldo de la letra de cambio en el registro de negociaciones.
11. Modificar la transacción de negociaciones para que permita distintos tipos de deuda a la vez.

Al realizar el análisis de dichos procesos se decidió por parte del equipo de desarrollo efectuar el modelado de negocio solo a los tres primeros por la complejidad que estos presentan en la organización, los restantes son procesos ya identificados que incluyen cambios y mejoras, de los cuales se generaron una serie de artefactos que conducen a la captura de requisitos del cliente tales como: modelo de procesos de negocio, modelo conceptual, descripción de los procesos de negocio y reglas de negocio. Dichos artefactos fueron validados posteriormente por los funcionales entrevistados.

De los procesos expuestos anteriormente se decidió seleccionar Emitir cartas de discrepancias con el objetivo de exponer de forma simple cada fase del modelo de desarrollo de software. Los artefactos generados por los restantes procesos, para cada una de estas disciplinas, podrán ser consultados en los anexos del documento de tesis en formato digital (*disponibles a partir de la página 90*). A continuación se mostrarán los artefactos que generó el proceso seleccionado en la disciplina modelado de negocio.

2.2.1.1. Descripción del proceso

Para apoyar el estudio de este proceso de negocio se consultaron a los especialistas funcionales en el tema de la gestión de discrepancias. Dicho estudio propició el conocimiento necesario sobre las cartas de discrepancias, de la cual se concluyó que son documentos de disconformidad emitidos por el BNC a la empresa a la llegada de los documentos de embarque de la mercancía. Si estos documentos en la revisión son encontrados inconsistentes con la especificación de la carta de crédito se dice que hay discrepancias por los errores existentes, omisiones y/o detalles que no están de acuerdo con los términos y condiciones establecidas en la apertura de la CC.
El resultado de la entrevista realizada se utilizó como base para la descripción de dicho proceso (*ver Tabla 1*).

Objetivo	Emitir las cartas de discrepancias a la empresa.
Evento(s) que lo genera(n)	No conformidad con los documentos de embarque.
Precondiciones	La carta de crédito y los documentos de embarque deben estar registrados.
Marco legal	Reglas y usos uniformes relativos a los créditos documentarios, publicación No.600 del 2007 de la CCI.
Entradas	Cartas de crédito. Documentos de embarque.
Flujo de eventos	
Flujo básico Emitir discrepancia	
1.	Registrar DCP[16]: El Operador DCP registra en el sistema Quarxo la discrepancia.
2.	Crear documento DCP: El Operador DCP elabora el Documento DCP de forma manual llenando los campos: • número de CC. • número de contrato. • importe de los documentos. • observaciones.
3.	Emitir discrepancia: El Operador DCP envía al área de correspondencia, 3 copias del documento DCP.
Pos-condiciones	
1.	Se ha emitido la discrepancia.
Salidas	
Entidades DCP (Persistida). Documento DCP (Word).	

Tabla 1. Descripción del proceso de negocio emitir cartas de discrepancias.

2.2.1.2. Diagrama del proceso de negocio

Los diagramas de procesos son representaciones gráficas que se obtienen de cualquier tipo de proceso paso a paso. Básicamente describen los pasos que se siguen en toda una secuencia de actividades, dentro de un proceso o un procedimiento.

Actualmente en el sistema Quarxo es posible registrar la discrepancia pero resulta un proceso engorroso para el operador o usuario, el cual debe confeccionar de forma manual la carta de discrepancia que se le enviará a la empresa, consultando los datos registrados en la aplicación. El diagrama de proceso *(ver Figura 2.1)* muestra lo descrito anteriormente.

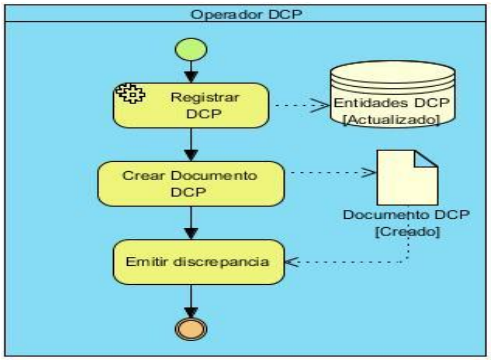


Figura 2.1. Diagrama del proceso emitir cartas de discrepancias.

[16] *Documento de discrepancia.*

2.2.2 Modelo conceptual

Un modelo conceptual también conocido como modelo de dominio es una representación visual de las clases conceptuales del dominio del problema, explica (a sus creadores) los conceptos significativos indicando las características del proceso en la organización y cómo se relacionan los conceptos más relevantes en la descripción del problema.

Para la elaboración del modelo conceptual relacionado al proceso emitir cartas de discrepancia se identificaron las clases conceptuales del negocio, los atributos y las relaciones existentes entre dichas clases. A continuación se muestra el modelo conceptual *(ver Figura 2.2)* con las clases identificadas y los atributos correspondientes.

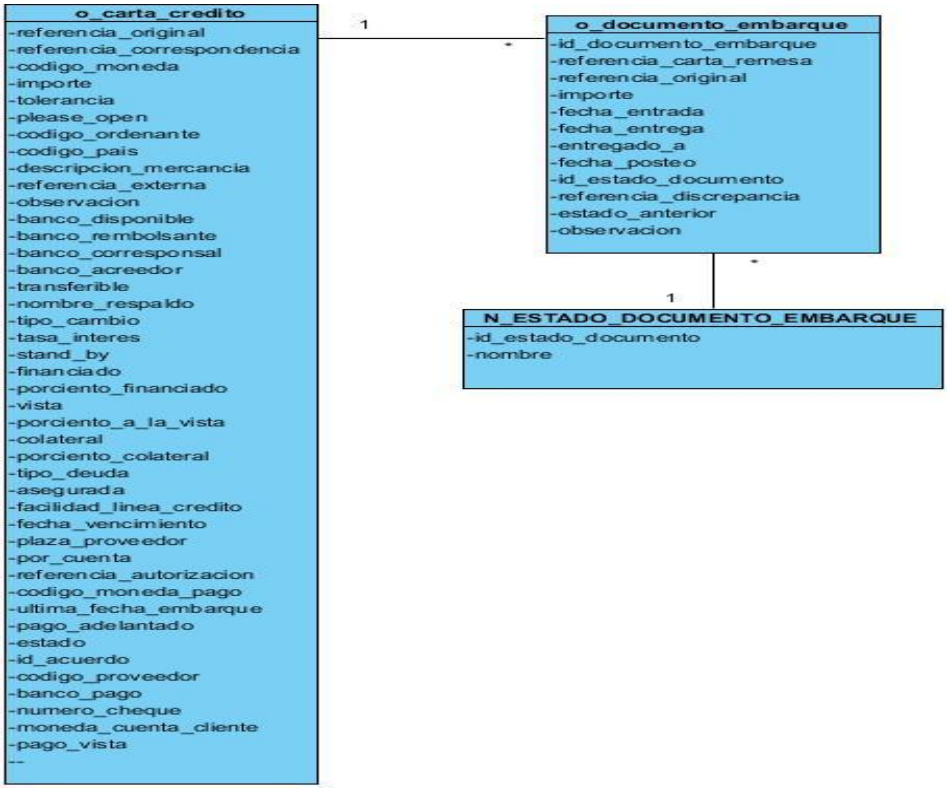


Figura 2.2. Modelo conceptual.

2.2.3 Reglas del negocio

Las reglas son esenciales para los modelos de negocio, son restricciones de comportamiento y/o proporcionan soporte para la dirección de las actividades de negocio se aplican a lo largo de los procesos y procedimientos. Una regla de negocio define o limita un aspecto del negocio con el objetivo de establecer una estructura. Las mismas deben ser explícitas y escritas en un lenguaje natural, expresadas en términos sencillos para su total comprensión.

Existen diferentes tipos de reglas dentro de las que se encuentran:

- ✓ Reglas Textuales
- ✓ Reglas del Modelo de Datos
- ✓ Reglas de Relación
- ✓ Reglas de Derivación

Las reglas de negocios se podrán consultar en el documento de tesis en formato digital (*Anexo7*).

2.3 Requisitos

Los requisitos son una condición o capacidad que debe cumplir o poseer un sistema o componente de un sistema para satisfacer un contrato, estándar, especificación u otro documento formalmente impuesto o puede ser también una restricción en el proceso de desarrollo de un sistema. La definición de requisitos es una actividad fundamental para modelar el sistema que consiste en identificar qué es lo que realmente se necesita. Esto se realiza con el objetivo de establecer la comunicación entre el cliente y el equipo de desarrollo.

A continuación se muestran los artefactos generados en la disciplina de requisitos del modelo de desarrollo de software propuesto por CEIGE evaluados en dos aspectos fundamentales en cuanto a la criticidad y complejidad.

2.3.1. Requisitos funcionales

Los requisitos funcionales son los que definen las funciones que el sistema será capaz de realizar. Estos describen las transformaciones que el sistema realiza sobre las entradas para producir salidas. Los mismos al tiempo que avanza el proyecto de software se convierten en los algoritmos, la lógica y gran parte del código del sistema (39).

Requisitos funcionales (RF):

Módulo emisión

RF 01. Registrar CC.
El sistema permitirá registrar las CC especificando el banco please open.
Prioridad: alta.
Complejidad: media.
RF 02. Actualizar CC.
El sistema permitirá modificar los datos de la CC independientemente del estado en que se encuentre la misma.
Prioridad: alta.
Complejidad: alta.
RF 03. Autorizar CC.
El sistema permitirá autorizar las CC financiadas aseguradas y no aseguradas a la vez.

Prioridad: alta.

Complejidad: alta.

RF 04. Poner en estado devuelta la CC.

El sistema permitirá cancelar la CC antes de ser autorizada.

Prioridad: alta.

Complejidad: alta.

Módulo discrepancia

RF 05. Registrar discrepancia.

El sistema deberá registrar los datos y generar de forma automática la carta de discrepancia.

Prioridad: alta.

Complejidad: alta.

Este requisito incluye el siguiente:

RF 05.1 Enviar mensajes 750.

El sistema deberá realizar el envío del mensaje 750 al registro de la discrepancia.

Prioridad: alta.

Complejidad: alta.

Módulo negociación

RF 06. Registrar negociación.

El sistema deberá realizar el registro de los datos de las negociaciones

Prioridad: alta.

Complejidad: alta.

Este requisito incluye los siguientes:

RF 06.1 Registrar negociación asegurada y no asegurada.

El sistema deberá realizar el registro de los datos de las negociaciones aseguradas o no aseguradas.

Prioridad: alta.

Complejidad: alta.

RF 06.2 Enviar mensajes 799, 754, 999, 734,750.

El sistema deberá enviar los mensajes 799, 754, 999, 734,750 al registro de negociaciones.

Prioridad: alta.

Complejidad: alta.

RF 06.3 Registrar en la base de datos las negociaciones compuestas.

El sistema deberá registrar automáticamente las negociaciones compuestas en la tabla O_NEGOCIACION_UNION una vez que han sido registradas por el sistema en la tabla O_NEGOCIACION.

Prioridad: alta.

Complejidad: alta.

RF 06.4 Registrar negociación con pagaré.

El sistema permitirá registrar las negociaciones con pagaré especificando lo que será respaldado por la cuenta principal e intereses.

Prioridad: media.

Complejidad: media.

RF 06.5 Registrar negociación con letra de cambio.

El sistema permitirá registrar las negociaciones con letra de cambio especificando lo que será respaldado por la cuenta principal e intereses.

Prioridad: media.

Complejidad: media.

RF 06.7 Modificar la transacción de negociaciones.

El sistema deberá modificar la transacción de negociaciones para que permita distintos tipos de deuda a la vez.

Prioridad: media.

Complejidad: alta.

De los requisitos anteriores se decidió seleccionar Registrar discrepancia en correspondencia al procesos Emitir cartas de discrepancias del módulo discrepancia para exponer cada uno de los artefactos generados en esta fase. Los restantes podrán ser consultados en los anexos del documento de tesis en formato digital (*disponible a partir de la página 97*).

2.3.2. Requisitos no funcionales

Los requisitos no funcionales son exigencias de cualidades que se le imponen al proyecto. Especifican criterios que pueden usarse para calificar las funcionalidades de un sistema en lugar de sus comportamientos específicos. Definen propiedades o características ya sea del sistema, del proyecto o del servicio de soporte, que no son requeridas junto con la especificación del sistema pero que no se satisface añadiendo código, sino cumpliendo con esta como una restricción del software (41).

De acuerdo con las características sobre las cuales se desarrolla este subsistema se toman en cuenta los requisitos no funcionales (RNF) correspondientes al sistema Quarxo. Su descripción puede consultarse en el documento de tesis en formato digital (*Anexo33*).

2.3.3. Descripción de requisitos

A continuación se muestra la descripción del requisito funcional (*ver Tabla 2*) seleccionado anteriormente.

Precondiciones	El usuario se ha identificado y autenticado ante el sistema y tiene permisos para ejecutar esta acción. Deberá estar registrada la carta de crédito y el documento de embarque.
Flujo de eventos	
Flujo básico Emitir cartas de discrepancias	
4.	Se introducen los datos de la discrepancia. Ver prototipo no funcional: registrar discrepancia. • Referencia corriente • Referencia original • Fecha contable • Fecha valor • Referencia de la carta de remesa • Fecha de vencimiento de la carta crédito • Banco Beneficiario • Importe • Observaciones
5.	El sistema valida los datos introducidos.
6.	Si los datos son correctos el sistema registra los datos y genera el documento de discrepancia.
7.	El sistema confirma el registro de los datos.
8.	Concluye el requisito.
Pos-condiciones	
2.	Se registra la discrepancia en el sistema.
2	Se genera el documento de discrepancia.
Flujos alternativos	
Flujo alternativo 1.a Información errónea	
1	El sistema señala los datos erróneos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso 2 del flujo básico.
Pos-condiciones	
1	No se realizan cambios en la aplicación.
Flujo alternativo 1.b Información incompleta	
1	El sistema señala los datos vacíos y permite corregirlos.
2	El usuario corrige los datos.
3	Volver al paso 2 del flujo básico.
Pos-condiciones	
1	No se realizan cambios en la aplicación.
Flujo alternativo 1.c El usuario cancela la acción	
1	Concluye el requisito.
Pos-condiciones	
1	No se registra la discrepancia en el sistema.
2	No se genera el documento de discrepancia.

Validaciones		
1	Se valida según lo establecido en las Reglas del negocio: CIG-QF2-N-CAR - i1702	
2	Se validan los datos según lo establecido en el Modelo conceptual: CIG-QF2-N-CAR - i1302.	
Relaciones	**Requisitos Incluidos**	Contabilizar transacción. Configurar comisiones.
Conceptos	Documento de discrepancia.	Atributos del concepto que se utilizan en el requisito: Empresa Fecha Número de la carta de crédito Contrato Importe de los documentos Observaciones

Tabla 2. Descripción del requisito registrar discrepancias.

2.3.4. Prototipos de interfaz de usuario

Los prototipos de interfaz de usuario son modelos presentados por el equipo de desarrollo a los especialistas funcionales para que los mismos conciban la simulación del sistema y validen que la interfaz contempla las necesidades reales. Además son la base para que los desarrolladores implementen las interfaces de usuario finales. Los prototipos de interfaces de usuario presentados en el trabajo de diploma fueron revisados por los especialistas del BNC, lo que constituye una validación para los requisitos funcionales.

El siguiente prototipo (*ver Figura 2.3*) muestra la interfaz Registrar discrepancias del requisito en cuestión.

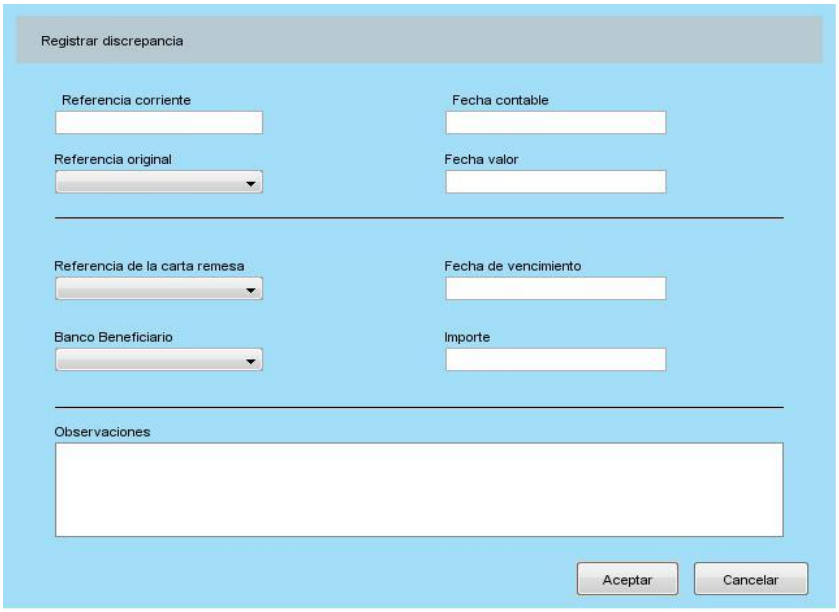


Figura 2.3. Prototipo de interfaz de usuario registrar discrepancia

2.3.5. Salidas del sistema

La salida es la capacidad de un sistema de información para procesar los datos almacenados o seleccionar los datos de entrada y enviarlos al exterior. En el diseño de las salidas del sistema la información es representada mediante documentos, informes u otros formatos, en ellos se guarda la información que se necesita extraer del sistema. Las especificaciones de las salidas representan unas de las características más importantes para el usuario, pues si la salida no responde a sus expectativas al usuario le resulta un sistema inútil.

2.3.5.1 Prototipo de salida del sistema

Los prototipos de salidas del sistema son construidos con la finalidad de establecer una representación clara y realista de los requerimientos del usuario. Al construir un prototipo se deben seguir los estándares para los datos que emplea la organización (dimensiones, fuente, longitud de datos, características, entre otros).

El prototipo que se muestra seguidamente (*ver Figura 2.4*) refleja la salida de los documentos de discrepancias, generados automáticamente por el sistema, una vez que han sido introducidos los datos por el usuario en la interfaz Registrar discrepancias.

BANCO NACIONAL DE CUBA

Empresa:_____

Fecha:_____

Asunto: DOCUMENTOS CON DISCREPANCIAS

C/C No._____ Ctto.: _____ Mda.:____ Importe:_____

Hemos recibido documentos bajo la carta de crédito del asunto con las siguientes discrepancias:

Discrepancias:

Según lo establecido por las Reglas y Usos Uniformes relativos a los Créditos Documentarios, publicación No.600 del 2007 de la CCI, debemos confirmar o rechazar los documentos al banco extranjero en un término de 5 días hábiles bancarios, así como la discrepancia detectada e importe a descontar al pago del vencimiento. **Solicitamos su aceptación o reparo en un término no mayor de 5 días hábiles bancarios** a partir de la fecha de la firma en que recibe esta comunicación cuyo acuse de recibo consta al pie de esta carta.

Le enviamos con el abajo firmante sólo copias no negociables de la factura y del conocimiento de embarque, sin validez alguna ante las autoridades aduanales.

Sin otro asunto.

Atentamente,

Banco Nacional de Cuba
Firmas Autorizadas

Para uso de la empresa sobre
la aceptación o reparo

Recibe:

Empresa:_____

Nombre y Apellidos (Firma)

Figura 2.4. Prototipo de salida del sistema carta de discrepancia.

2.3.6. Validación de los requisitos

Esta actividad se realizó con el objetivo de garantizar que los requisitos fueran descritos correctamente y cumplieran con las necesidades del cliente. Se desarrolló mediante las siguientes técnicas de validación:

- ✓ La revisión técnica por el equipo de analistas principales.

El objetivo de este paso fue verificar la construcción correcta de los artefactos correspondientes en la ingeniería de requisitos.

- ✓ La revisión funcional por los especialistas del BNC.

Se realizó con el propósito de validar que las funcionalidades descritas cumplieran con las necesidades y aspiraciones del cliente. En esta revisión se realizó en el BNC con los especialistas funcionales donde se presentaron los prototipos elaborados durante la especificación de requisitos. Para ello, se desarrollaron varios escenarios posibles con el auxilio de juegos de datos, de forma tal que se visualizaron las diferentes funcionalidades. Las no conformidades fueron documentadas y corregidas.

- ✓ Revisión técnica de artefactos por el equipo de calidad.

Se realizaron revisiones de los artefactos por parte del equipo de calidad del proyecto y se corrigieron las no conformidades identificadas hasta ser liberada la documentación.

2.4 Análisis y diseño

El análisis y diseño es una de las etapas de construcción de un sistema informático, que consiste en relevar la información actual, proponer los rasgos generales de la solución futura y realizar el modelado del sistema para que soporte todos los requisitos. Esto contribuye a una arquitectura sólida y estable que se convierte en un plano para la próxima fase**.** Su propósito radica fundamentalmente en generar los artefactos que permiten estructurar el sistema y establecer las bases para la implementación del software.

2.4.1. Modelo de datos

Un modelo de datos es una colección de conceptos bien definidos que ayudan a expresar las propiedades de una aplicación con un uso de datos intensivo. Sirve para describir la estructura de la base de datos, así como los datos, sus relaciones y las restricciones que deben cumplirse entre ellos (40).

Es necesario resaltar que el subsistema Cartas de Crédito forma parte del sistema Quarxo el cual ya posee una base de datos. De dicho subsistema se mostrará el diagrama de modelo de datos (ver *Figura 2.5*) perteneciente al módulo de negociación, documento de embarque y discrepancia en relación al requisito expuesto. El modelo de dato del subsistema completo se puede consultar en el documento de tesis en formato digital (*Anexo34*).

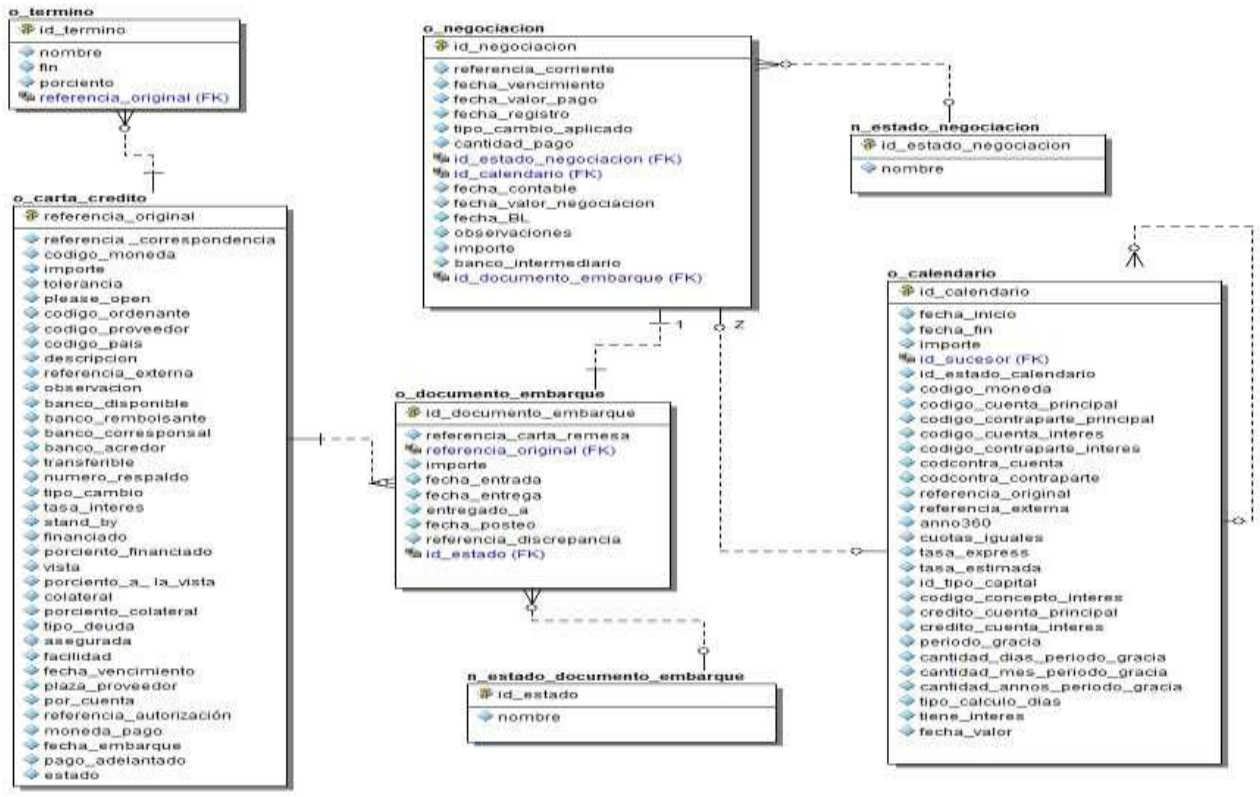


Figura 2.5. Modelo de datos de los módulos documentos de embarque, negociación y discrepancia.

2.4.2. Diccionario de datos

El diccionario de datos describe las características lógicas de las entidades y sus atributos, contenidos en el modelo conceptual incluyen el nombre de la entidad, descripción de la entidad, nombre del atributo, descripción del atributo, tipo del atributo, restricciones del atributo. A continuación se presenta el diccionario de datos (*ver Tabla 3*) con los conceptos que tienen mayor relevancia en el modelo conceptual expuesto anteriormente.

Descripción	La entidad o_documento_embarque contiene toda información referente al documento de embarque de la mercancía.					
Atributos						
Nombre	**Descripción**	**Tipo**	**¿Puede ser nulo?**	**¿Es único?**	**Restricciones de clases Válidas**	**No válidas**
id_documento_embarque	Identificador secuencial de la BD	int	NO	Si	Numérico	Cadenas de caracteres
referencia_carta_remesa	Nombre de la mercancía	varchar	NO	NO	Cadena de caracteres	Caracteres extraños
referencia_original	Referencia de la carta de crédito	varchar	NO	NO	Cadena de caracteres	Caracteres extraños
importe	Importe del documento de la mercancía	decimal	NO	NO	Numérico > 0	Cadenas de caracteres

fecha_entrada	Fecha de entrada al área de correspondencia.	datetime	NO	NO	Date	Cadenas de caracteres
fecha_entrega	Fecha de entrega al negociador	datetime	NO	NO	Date	Cadenas de caracteres
entregado_a	Operador que negociará el documento	varchar	NO	NO	Cadena de caracteres	Caracteres extraños
fecha_posteo	Fecha de registro en el sistema	datetime	SI	NO	Date	Cadenas de caracteres
id_estado_documento	Identifica al estado del documento de embarque	int	SI	NO	Numérico	Cadenas de caracteres
referencia_discrepancia	No se utiliza	varchar	SI	NO	Cadena de caracteres	Caracteres extraños
estado_anterior	Estado anterior del documento de embarque	int	SI	NO	Numérico	Cadenas de caracteres
observación	Observaciones sobre la mercancía	varchar	SI	NO	Cadena de caracteres	Caracteres extraños

Tabla 3. Diccionario de datos de la entidad o_documento_embarque.

2.4.3. Arquitectura del sistema

La arquitectura definida para el desarrollo del sistema Quarxo está basada en una arquitectura de tres capas lógicas fundamentales: capa de presentación, capa de lógica de negocio, capa de acceso a datos y una capa transversal a las otras con los objetos del dominio.

2.4.3.1 Capa de presentación

En esta capa se desarrollará la lógica de presentación, es la encargada de interactuar con el usuario obtiene todos los pedidos de la interfaz, valida y envía los datos al servidor y además, controla la apariencia visual que tendrá la aplicación. En el lado del cliente se utilizará la librería Dojo Toolkit para generar las interfaces que interactuarán con el usuario. En el lado del servidor se utilizará 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 reutilizables de la aplicación. La capa de presentación estará relacionada con la capa de negocios y la capa de dominio (41).

2.4.3.2 Capa de lógica de negocio

Esta capa es la encargada de modelar la lógica de negocio, contiene todas las entidades del dominio y los componentes con la responsabilidad de manejar la lógica de negocio de los mismos. Para ello 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.

La capa de negocio está divida en dos subcapas: Facade y Manager. La Facade será el punto de intercambio entre la capa de presentación y la capa de negocio. Esta capa no tendrá lógica de negocio; sino que agrupará las funcionalidades según su naturaleza para que pueda ser invocada desde la capa de presentación. La subcapa Facade delegará a la subcapa Manager la realización de la lógica del negocio. Por otro lado, la subcapa Manager tendrá la jerarquía de clases suficiente para implementar el negocio de la aplicación. Esta subcapa utilizará 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 envolverán transaccionalmente las funcionalidades de la aplicación para evitar inconsistencia en los datos. Se utilizará para esto, las políticas de transacciones que propone Spring Framework (41).

2.4.3.3 Capa de acceso a datos

En esta capa se realizarán todas las operaciones básicas relacionadas con la base de datos tales como: persistir, consultar, actualizar y eliminar los objetos de dominio recibidos de la capa lógica y la invocación a funciones y procedimientos almacenados. La capa de negocio interactuará con esta capa a través de sus interfaces. Para desarrollar esta capa se utilizará el patrón DAO[17].

Las interfaces de los DAOs contienen básicamente las operaciones de inserción, modificación, eliminación y de localización de un objeto a partir de su llave primaria. Se utilizará 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 la integración con Hibernate y la utilización del patrón DAO.

Cuando la interacción con la base de datos se realiza a través de los procedimientos almacenados se utilizará Spring JDBC. El mismo se utilizará mediante el componente desarrollado por el proyecto para facilitar las invocaciones a funciones y procedimientos almacenados. Este componente permite que de una misma clase DAO se pueda acceder a varias funciones y procedimientos de forma transparente al desarrollador (41).

Dichas capas están bien delimitadas una de la otra, una capa superior interactúa con la inferior mediante interfaces que definen las funcionalidades que la misma debe brindar.

[17] *Data Acces Object*. *Objetos de Acceso a Datos*

2.4.4. Diagrama de paquetes

Los diagramas de paquetes constituyen una representación jerárquica del sistema o subsistema. Los cuales son utilizados precisamente para mostrar las agrupaciones lógicas en las que se encuentra dividida la aplicación. Estos propician un medio para organizar los artefactos del análisis en fragmentos más fáciles de manejar. Por otro lado ayudan a los desarrolladores mostrando una abstracción más concreta de lo que sería la composición de las diferentes clases en los paquetes.

El siguiente diagrama de paquetes (*ver Figura 2.6*) contiene la estructura del módulo de negociación que comprende internamente el módulo de discrepancias, al cual pertenece el requisito Registrar discrepancia.

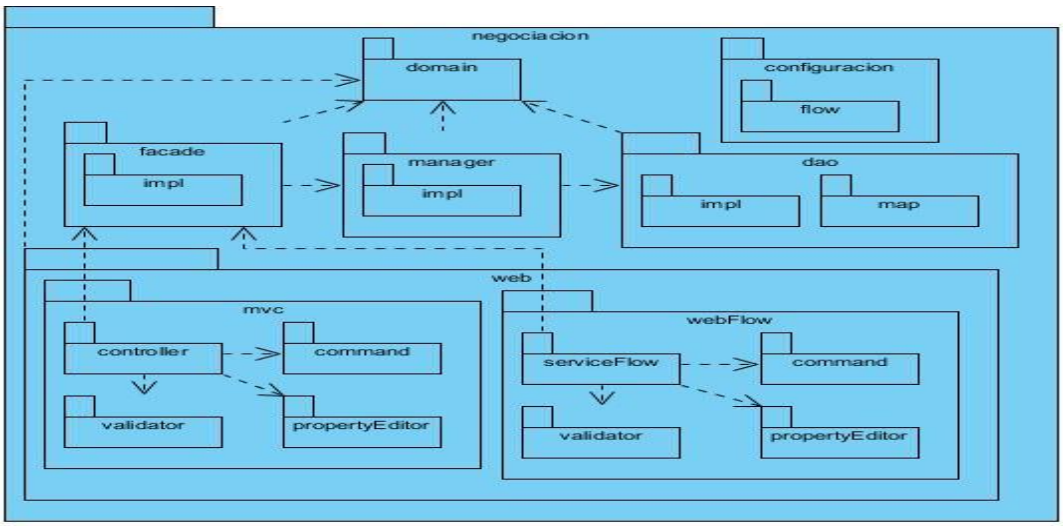


Figura 2.6. Diagrama de paquetes del módulo negociación.

2.4.4.1 Descripción de paquetes

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

- ✓ 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 facade: en este paquete se encuentran las interfaces y las implementaciones, 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 serán brindadas a las capas superiores.

Paquete dao: es donde 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: es donde se localizan las clases relacionadas con el dominio del módulo en cuestión.

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 sub-paquete se agrupa todo lo referente a la lógica de presentación cuando se hace uso de SpringMVC.

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 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.

2.4.5. Diagrama de clases

El diagrama de clases del diseño es un artefacto que permite representar los conceptos en un dominio del problema. Estos diagramas son utilizados durante el proceso de análisis y diseño del software. Describen la estructura del sistema mostrando sus clases, atributos y las relaciones entre ellos. Constituyen el diseño conceptual de la información que se manejará en el sistema y los componentes que se encargan del funcionamiento y la relación entre uno y otro.

Los siguientes diagramas (*ver Figura 2.7, Figura 2.8, Figura 2.9*) representan el diseño para la capa de presentación, lógica de negocio y acceso a datos de los módulos negociación y discrepancia atendiendo la arquitectura definida por el proyecto.

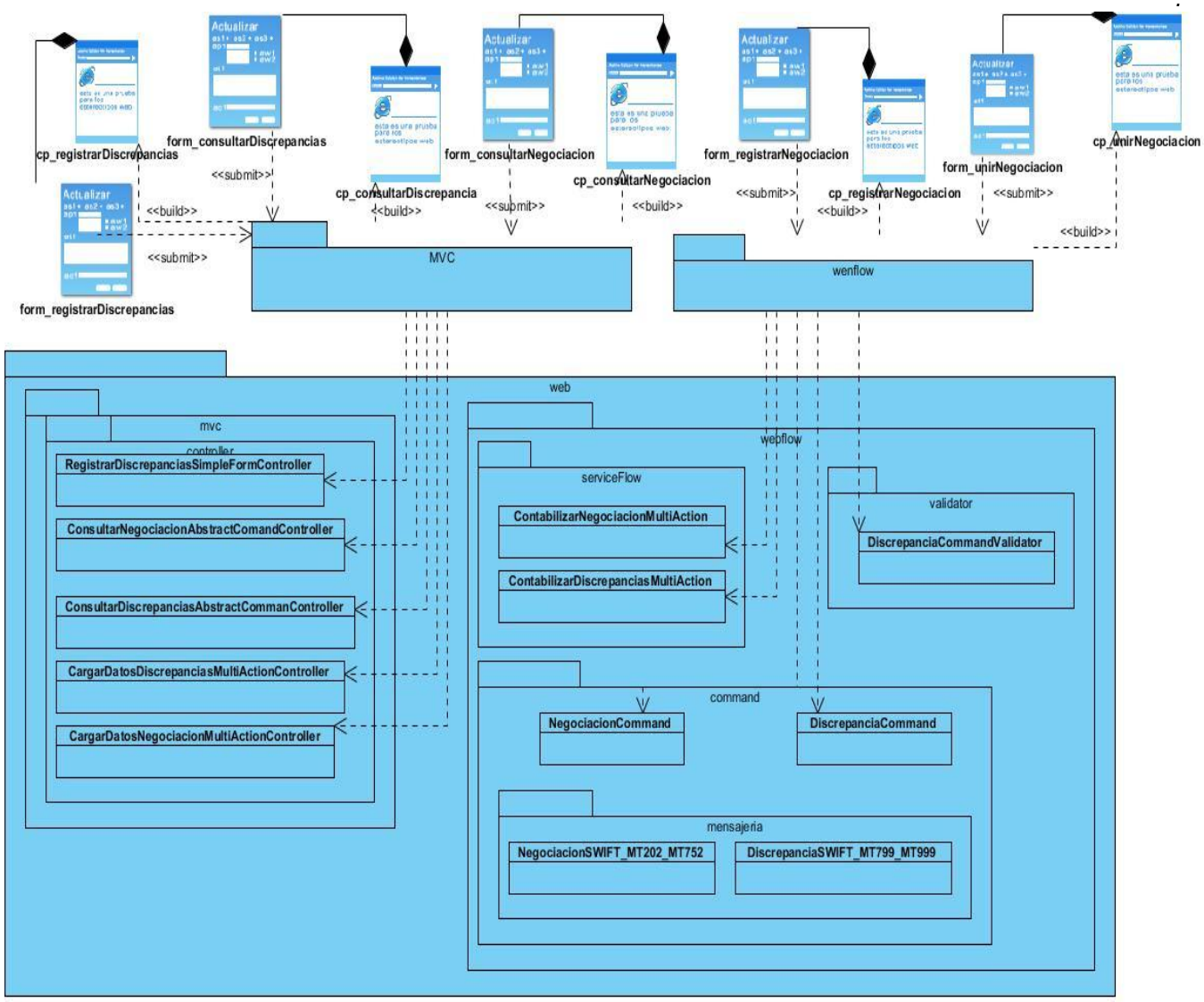


Figura 2.7. Diseño de la capa de presentación de los módulos discrepancias y negociación.

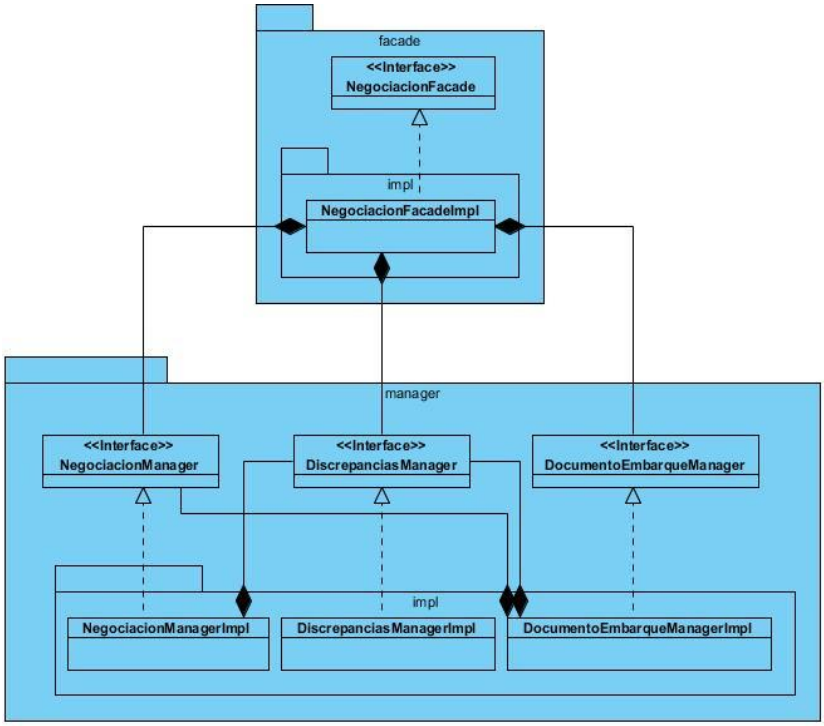


Figura 2.8. Diseño de la capa de negocio de los módulos discrepancias y negociación.

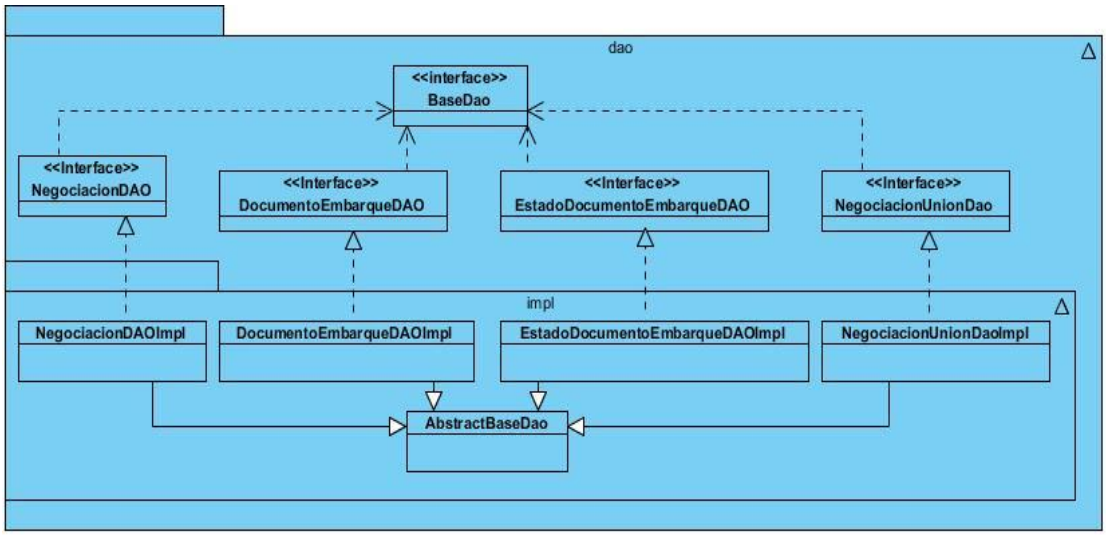


Figura 2.9. Diseño de la capa de acceso a datos de los módulos discrepancias y negociación.

2.4.6. Diagrama de interacción

Los diagramas de interacción son utilizados para modelar los aspectos dinámicos de un sistema. Describen la manera en que interactúan un grupo de objetos entre sí. Existen dos tipos de diagramas de interacción: el diagrama de secuencia y el diagrama de colaboración.

En la etapa del diseño del software se utiliza fundamentalmente el diagrama de secuencia. Este indicará los módulos o clases que forman parte del sistema y las llamadas que se hacen en cada uno de ellos para una tarea determinada. Se realizan los diagramas de secuencia para definir las acciones que se pueden ejecutar y para mostrar las interacciones ordenadas en secuencia entre los

objetos de un escenario. Si los requisitos a modelar tienen varios flujos o sub-flujos distintos, suele ser útil crear un diagrama de secuencia para cada uno de ellos. A continuación se muestran los tres diagramas de secuencias (*ver Figura 2.10, Figura 2.11, Figura 2.12*) generados por el requisito en cuestión.

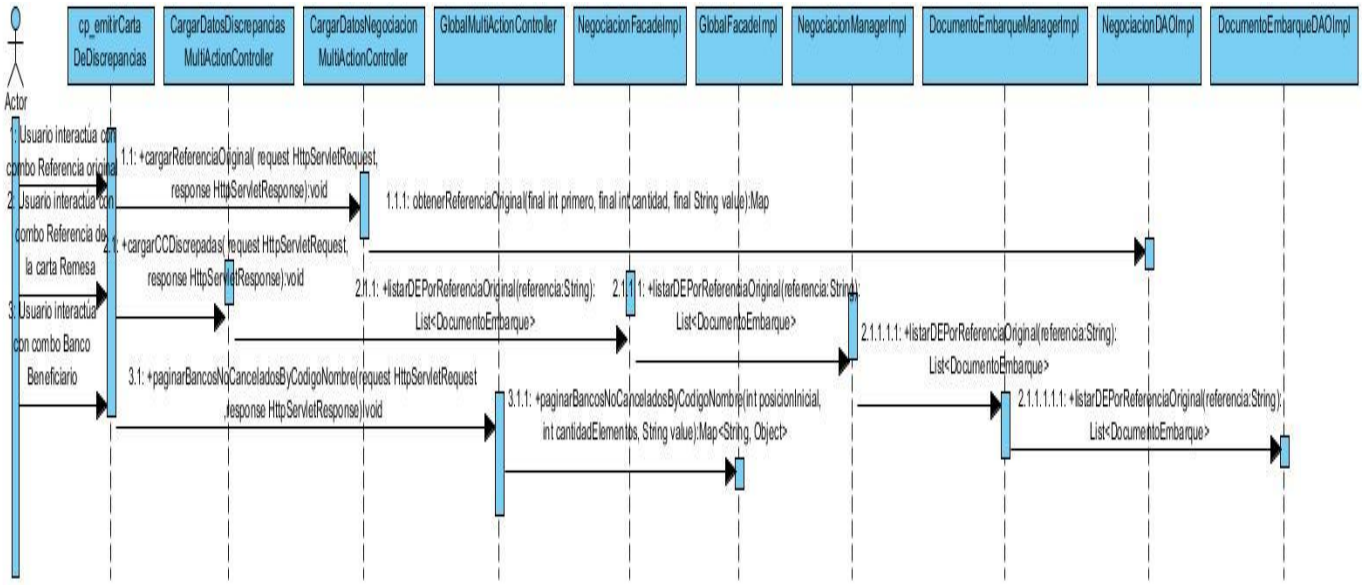


Figura 2.10. Diagrama de secuencia del requisito Registrar discrepancia primer escenario.

Este diagrama describe la interacción del usuario con la interfaz y el flujo de peticiones para cargar los datos del escenario Registrar discrepancias, a través de los campos de selección. Comienza cuando Spring MVC construye la página cliente *cp_emitirCartaDeDiscrepancia*. Una vez cargada esta, solicita a *CargarDatosNegociacionMultiActionController* cargar la referencia original, la cual utiliza directamente *NegociacionDaoImpl*. De igual forma el usuario solicita cargar la referencia de la carta remesa a la página cliente *cp_emitirCartaDeDiscrepancia* y esta le pide las cartas discrepadas a *CargarDatosDiscrepanciasMultiActionController*, la cual solicita la lista de los documentos de embarques de la clase *NegociacionFacadeImpl* ésta a su vez utiliza a *NegociacionManagerImpl* y este llamando a *DocumentoEmbarqueManagerImpl* que obtiene los datos de la lista de la clase *DocumentoEmbarqueDAOImpl,* que interactúa directamente con la información de la base de datos. Igualmente el usuario se relaciona con el campo de selección banco beneficiario cargando los bancos no cancelados de la clase *GlobalMultiActionController,* este a su vez le solicita la información al *GlobalFacadeImpl*. Así concluye este escenario resultando lista la página para continuar con el flujo de sucesos del requisito Registrar discrepancias.

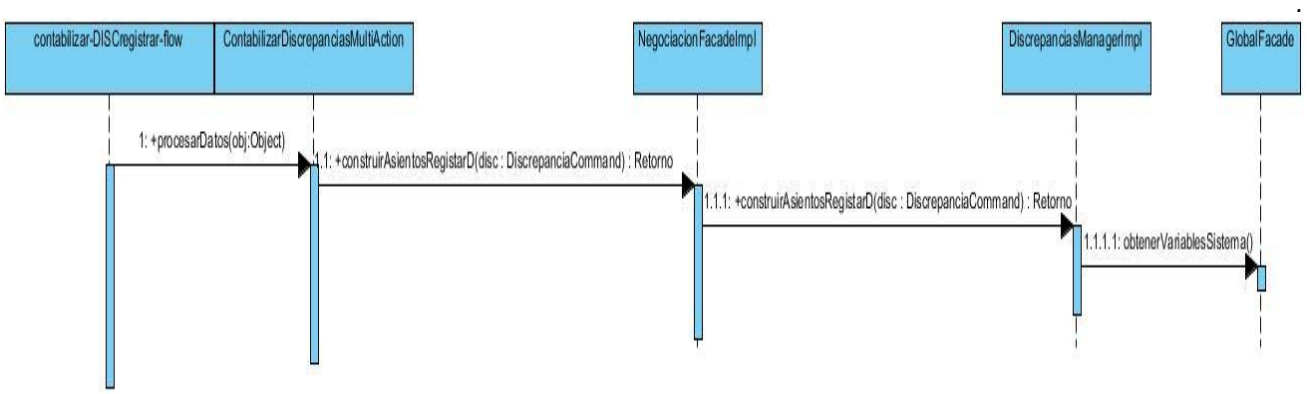


Figura 2.11. Diagrama de secuencia del requisito Registrar discrepancia segundo escenario.

El diagrama describe el flujo de datos partiendo que el usuario ha introducido toda la información en el formulario de la página cliente y ha presionado el botón aceptar. Estos datos son validados por el mismo formulario y enviados al motor de SpringWebFlow, quien a su vez hace uso del propertyEditor *DateStringConverter* para convertir las fechas en objetos. Debido a que este requisito requiere contabilización se genera un subflujo el cual es representado en este escenario. El sub-flujo contabilizar contempla la construcción de los asientos para el registro de la discrepancia para ello el flujo de datos se comporta de la siguiente manera: la clase *contabilizarDISCregistar-flow* solicita procesar los datos a *ContabilizarDiscrepanciasMultiAction* antes de construir los asientos por *NegociacionFacadeImpl* que a su vez delega su función en *DiscrepanciasManagerImpl,* la cual obtiene las variables necesaria a través de la interfaz GlobalFacade.

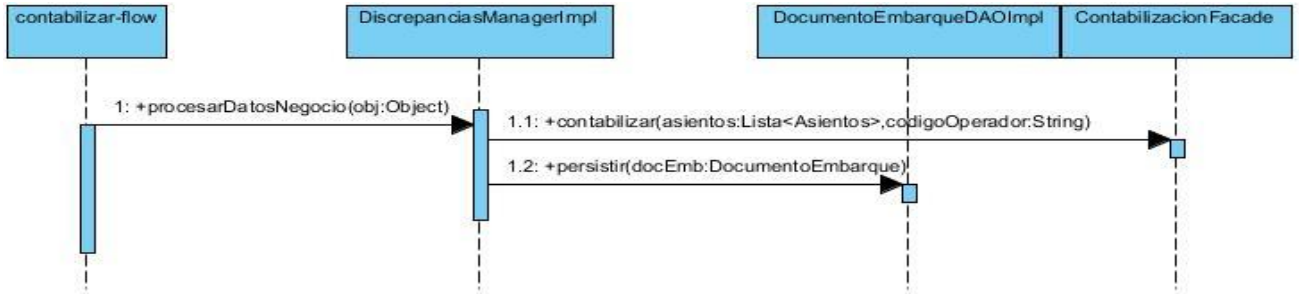


Figura 2.12. Diagrama de secuencia del requisito Registrar discrepancia tercer escenario.

Este escenario describe el flujo de datos después de pasar por el sub-flujo contabilizar. La clase *DiscrepanciasManagerImpl* es la encargada de ordenar a *ContabilizacionFacade* la contabilización de los asientos creados anteriormente y la persistencia de los documentos de embarque a la clase *DocumentoEmbarqueDAOImpl*.

2.4.7. Diagrama de flujo

Un diagrama de flujo es una representación gráfica de cada paso de un proceso, descrito por diferentes símbolos. Los diagramas de flujo son una manera de visualizar el flujo de datos a través

de sistemas de tratamiento de información. Estos describen que operaciones y en que secuencia se requieren para solucionar un problema dado.

La siguiente representación corresponde al diagrama de flujo web (*ver Figura 2.13*) relacionado al requisito analizado anteriormente.

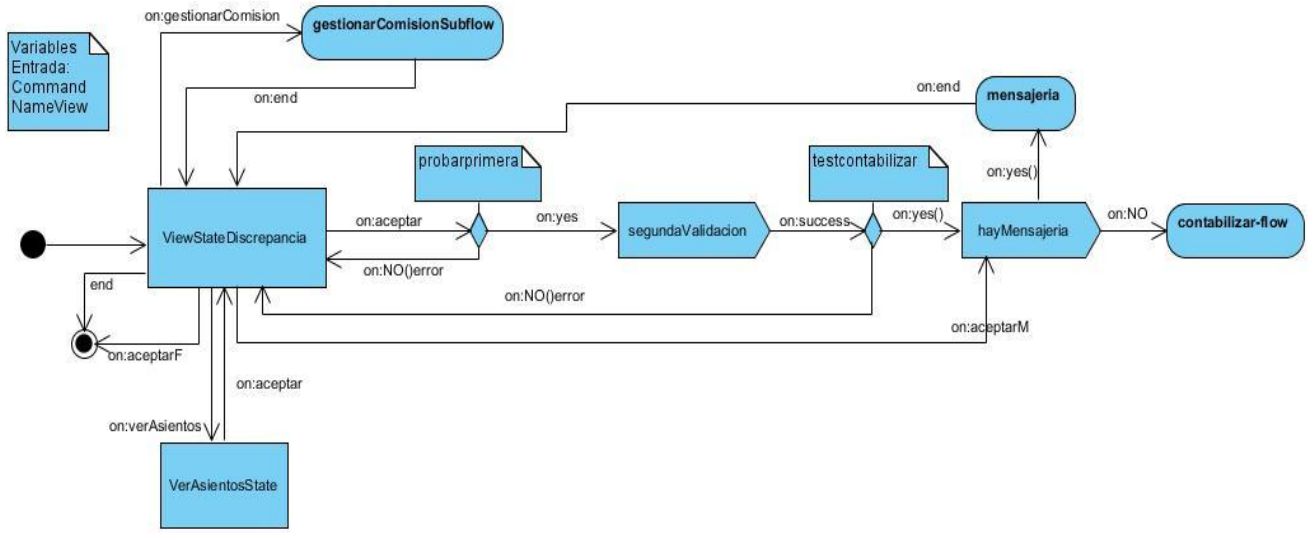


Figura 2.13. Diagrama del flujo web del requisito Registrar discrepancia.

2.4.8. Diseño de casos de pruebas

Los casos de pruebas son un conjunto de condiciones o variables bajo las cuales se determinará si el requisito de una aplicación es parcial o completamente satisfactorio. Se pueden realizar muchos casos de prueba para determinar que un requisito es completamente satisfactorio. Con el propósito de comprobar que todos los requisitos de una aplicación son revisados, debe haber al menos un caso de prueba para cada requisito. La descripción de los casos de prueba puede consultarse en el documento de tesis en formato digital (*disponibles a partir de la página 127*).

2.4.9. Patrones de diseño empleados

En el capítulo anterior se realizó un estudio detallado sobre los patrones de diseño, puesto que son una solución estándar para un problema común de programación resuelto con anterioridad. Uno de los pasos a tener en cuenta cuando se decide desarrollar un proyecto de software es identificar qué patrones pueden ser utilizados. A continuación se describe el uso de los patrones aplicados en el diseño de los requisitos a incorporar en el subsistema Cartas de Crédito.

De los patrones GRASP, encargado de describir los principios fundamentales de la asignación de responsabilidades, se emplearon específicamente:

✓ Experto: se evidencia en la asignación de las funcionalidades a las clases de acuerdo con las información que manejan cada una, ejemplo de la utilización del patrón se encuentra en la clase

DocumentoEmbarqueDAO responsable de efectuar las operaciones persistir, listar, consultar y eliminar los documentos concernientes al registro de las discrepancias.

✓ Controlador: un ejemplo de la utilización de este patrón lo representa la clase *CargarDatosNegociacionMultiActionController* debido a que funciona como intermediaria entre la interfaz de usuario y el negocio, presenta la responsabilidad de contener los métodos que cargan los datos que serán mostrados al usuario.

✓ Alta Cohesión: los módulos están estructurados con la menor cantidad posible de clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.

✓ Bajo Acoplamiento: este patrón se evidencia con la definición de interfaces e implementaciones, como puede ser la interfaz *DevolverFacade* y su implementación, que permiten que *DevolverCCMultiActionController* clase de la presentación se relacionen únicamente con ella para realizar sus operaciones, reduciendo el impacto de cambios posteriores en el negocio del sistema.

De los patrones estructurales GoF, encargado de describir como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades, se utilizó el patrón:

✓ Fachada: se evidencia en la utilización de las interfaces en los distintos requisitos ejemplo *DevolverCCFacade* responsable 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, reduciendo las dependencias en el sistema entre la parte del servidor y la parte del cliente.

✓ Cadena de Responsabilidad: cuando desde la vista se solicita información que se encuentra en la base de datos, esta es atendida primeramente por los *Controller*, luego por la *Facade*, el *Manager* y finalmente el *DAO*, evidenciándose de esta manera la utilización de dicho patrón.

Patrón de acceso a datos utilizado:

✓ DAO: se evidencia con la definición de las clases *NegociacionDAO* y *DocumentoEmbarqueDAO* efectuando su uso a través del framework Hibernate para almacenar los datos de la discrepancia.

Patrón de presentación utilizado:

✓ Vistas Compuestas: se evidencia en el fichero *registrarDiscrepancias.jsp* que contiene código HTML para construir la estructura de la interfaz mostrada al usuario y para dar cumplimiento a la funcionalidad hace uso de la etiqueta "include" para contener otra vista en su estructura.

Patrón de arquitectura empleado:

✓ MVC: se evidencia en el subsistema Cartas de Crédito y en todos los módulos contenidos en él de forma general, pues utiliza a SpringMVC como framework núcleo de la aplicación, lo cual obliga al mismo a utilizar el patrón MVC para su correcto funcionamiento. Además forma parte de la arquitectura utilizada en el proyecto SAGEB dividida en tres capas: presentación, lógica de negocio y acceso a datos.

2.5 Validación del diseño

Validación del diseño se le llama a las actividades que buscan asegurar si el software construido se ajusta a los requisitos del cliente. Durante y después del proceso de implementación, el programa en desarrollo debe ser comprobado, para asegurar que satisface su especificación y entrega la funcionalidad esperada por el cliente (42). Para una correcta validación del diseño se utilizaron métricas.

2.5.1. Métricas para la evaluación del diseño

El IEEE (*Standard Glosary of Software Engering Terms*) define como métrica: una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado. Se dice que la medición es esencial, si es que se desea realmente conseguir la calidad en software. Es por eso que existen distintos tipos de métricas para poder evaluar, mejorar y clasificar al software final, en donde serán manejadas dependiendo del entorno de desarrollo del software al cual pretendan orientarse (43).

Entre las métricas más conocidas se encuentran las propuestas por Lorenz y Kidd los cuales dividen las basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos. Las orientadas a tamaños para una clase se centran en cálculos de atributos y de operaciones para una clase individual, para luego promediar los valores para el sistema en su totalidad. Las basadas en herencia se centran en la forma en que se reutilizan las operaciones en la jerarquía de clases. Las métricas para valores internos de clase examinan la cohesión y asuntos relacionados con el código así como las métricas orientadas a valores externos examinan el acoplamiento y la reutilización (44).

De estas métricas analizadas se propone la **Métrica de Tamaño Operacional de Clase** (TOC) como métrica a utilizar para la validación del diseño.

La aplicación de la métrica TOC define los siguientes atributos de calidad:

✓ Responsabilidad: consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.

✓ Complejidad de implementación: consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.

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

Para evaluar con las métricas es necesario definir los valores de los umbrales para los parámetros de calidad (*ver Tabla 4).* Algunos especialistas plantean umbrales para esta métrica basándose en el promedio de operaciones por clases obtenidos, estos valores fueron los aplicados en el diseño del sistema.

Atributos de Calidad	Clasificación	Umbrales de Medición
Responsabilidad	Baja	<=Promedio
	Media	>Promedio y <= 2*Promedio
	Alta	>2*Promedio
Complejidad de implementación	Baja	<=Promedio
	Media	>Promedio y <= 2*Promedio
	Alta	>2*Promedio
Reutilización	Alta	<=Promedio
	Media	>Promedio y <= 2*Promedio
	Baja	>2*Promedio

Tabla 4. Umbrales de medición.

En el siguiente diagrama (*ver Figura 2.14*) se muestra la distribución de los procedimientos en las clases del diseño realizado separados por rangos.

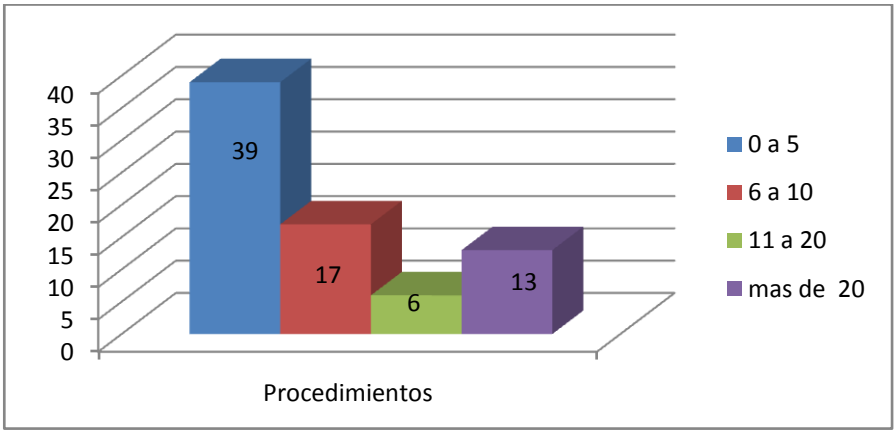


Figura 2.14. *Representación del nivel de procedimientos de las clases.*

A continuación se representan los estados de los atributos de calidad: responsabilidad (*ver Figura 2.15)*, complejidad de implementación (*ver Figura 2.16)* y reutilización (*ver Figura 2.17*) respectivamente.

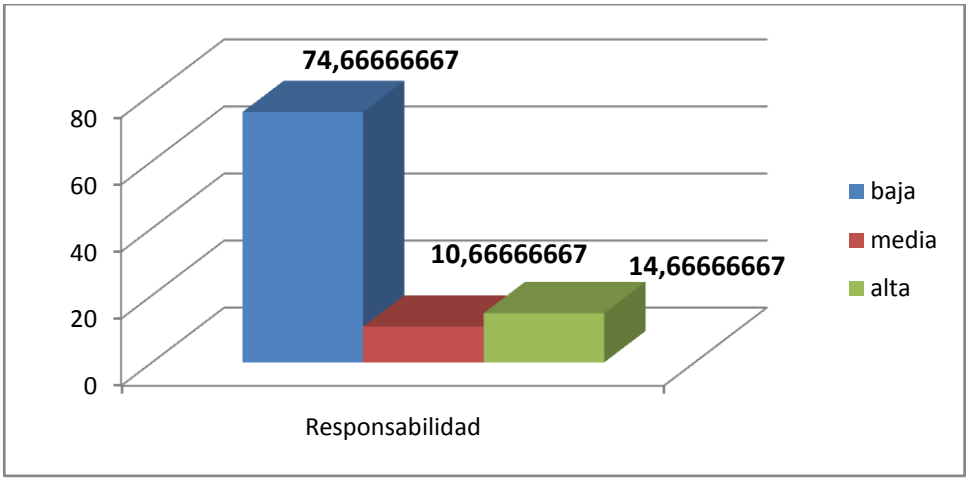


Figura 2.15. Representación de las clases según la responsabilidad.

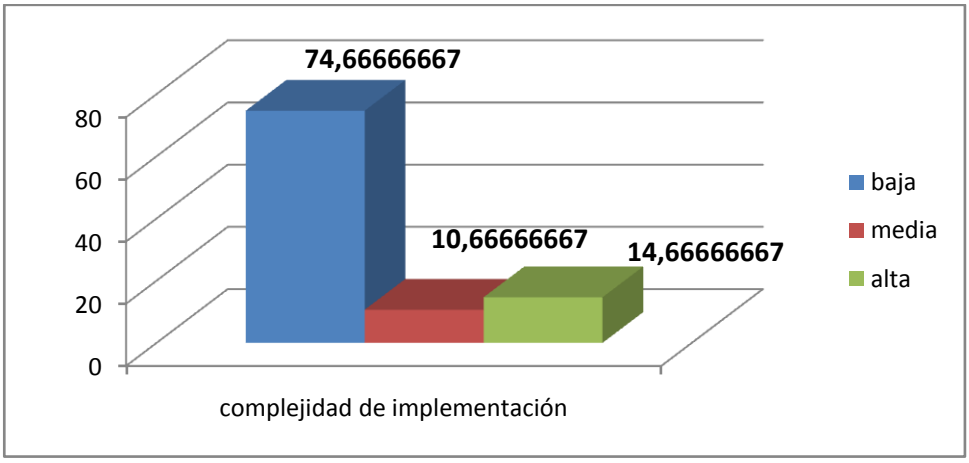


Figura 2.16. Representación de las clases según la complejidad de implementación.

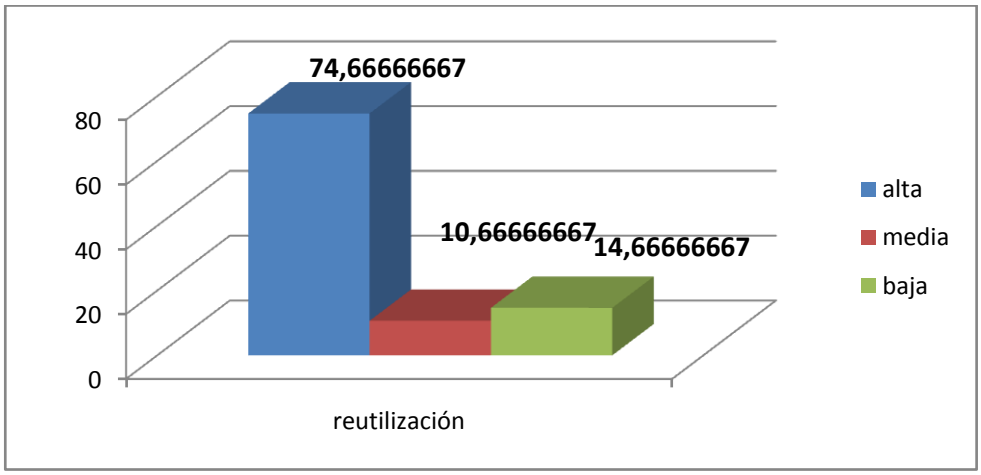


Figura 2.17. Representación de las clases según la reutilización

Después del análisis realizado basándose en los atributos de calidad definidos se llega a la conclusión de que existe una baja responsabilidad de las clases validando que un 74.7% de los procedimientos son menores que el promedio de procedimientos por clases, esto influye en que exista una baja complejidad de implementación contando solo con alta complejidad el 14.7% de los

procedimientos, teniendo solamente con poca capacidad de reutilización un 14.6% de los procedimientos lo que influye a que los atributos de calidad no se vean afectados en ningún momento.

2.6 Conclusiones del capítulo

En este capítulo se llevó a cabo las tres primeras disciplinas del ciclo de vida del proyecto SAGEB atendiendo a la metodología defina en el capítulo anterior. Las mismas arrojaron los siguientes resultados:

- ✓ La realización del modelado de negocio permitió obtener una visión detallada de los procesos negocio de la entidad. Mediante las entrevistas efectuadas a los especialistas del BNC se obtuvieron los artefactos correspondientes a esta disciplina como efecto se logró la identificación y descripción de los procesos de negocios, la confección de los diagramas de procesos utilizando la notación BPMN, el modelo de dominio usando como lenguaje de modelado UML y las descripción de las reglas de negocio de cada uno de los procesos identificados.
- ✓ La disciplina de requisitos posibilitó la identificación y descripción de todos los requisitos funcionales a incorporar en la solución logrando el entendimiento claro por parte del equipo de desarrollo de las necesidades del cliente, se generó artefactos como las interfaces de usuario, la cuales permitieron validar que las funcionalidades descritas cumplieran con las aspiraciones del cliente además de las revisiones técnicas por parte del equipo de analistas principales y el equipo de calidad.
- ✓ El análisis y diseño permitió la confección de los artefactos correspondientes a la disciplina donde se obtuvieron: el modelo de datos y el diccionario de datos logrando la descripción de las tablas y atributos de la base de datos del subsistema, la arquitectura del sistema utilizando el patrón de arquitectura MVC, los diagramas de paquetes que propiciaron una visión de la estructura de los módulos Emisión, Discrepancia y Negociación, de los diagramas de secuencias se obtuvo como efecto el flujo de datos y su relación entre las clases, de los diagramas de clases del diseño se obtuvo las clases, atributos, métodos y sus relaciones para llevar a cabo el desarrollo de la solución así como la utilización de los patrones estudiados en el anterior capítulo. Se realizó la validación del diseño a través de la métrica TOC resultando una baja responsabilidad de las clases, una baja complejidad de implementación y una alta capacidad de reutilización, demostrando que el diseño propuesto presenta buena calidad.
- ✓ Esta sección incide de manera notable en el siguiente capítulo, proporcionando una entrada apropiada y un punto de partida para las actividades de implementación, capturando las interfaces y clases del dominio, logrando además una uniformidad y organización en el proceso de desarrollo.

Capítulo 3: Implementación y validación

3.1 Introducción

En el presente capítulo expone la disciplina de Implementación del modelo de desarrollo propuesto por CEIGE y los artefactos generados tales como: el modelo de componentes, modelo de despliegue, los estándares de codificación, la descripción de clases, métodos y atributos implementados más relevantes del modelo de diseño propuesto en el capítulo anterior. Se muestran además las clases que intervienen en los principales flujos de la implementación, así como los métodos principales. Además se efectúan la validación del sistema para dar cumplimiento a los requisitos especificados.

3.2 Modelo de implementación

El modelo de implementación describe como los elementos del modelo del diseño y las clases se implementan en términos de componentes, como ficheros de código fuente o ejecutables. El modelo de implementación describe también como se organizan los componentes de acuerdo con los mecanismos de estructuración disponibles en el entorno de implementación los lenguajes de programación utilizados y cómo dependen los componentes unos de otros (19).

3.2.1. Diagrama de componentes

Los diagramas de componentes modelan la vista estática de un sistema. Se representa como un grafo de componentes de software unidos por medio de relaciones de dependencia (compilación, ejecución). Los mismos permiten visualizar con más facilidad la estructura general del sistema y el comportamiento del servicio que estos componentes proporcionan y utilizan a través de las interfaces. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes, por lo cual cada diagrama describe un apartado del sistema**.**

Para el desarrollo del sistema Quarxo se crean un grupo de componentes de los que consumirán servicios los subsistemas y módulos de la aplicación. Se muestra a continuación el diagrama (*ver Figura 3.1*) que evidencia la interacción del subsistema Cartas de Crédito con los componentes definidos y una descripción de la función que realiza cada uno de ellos.

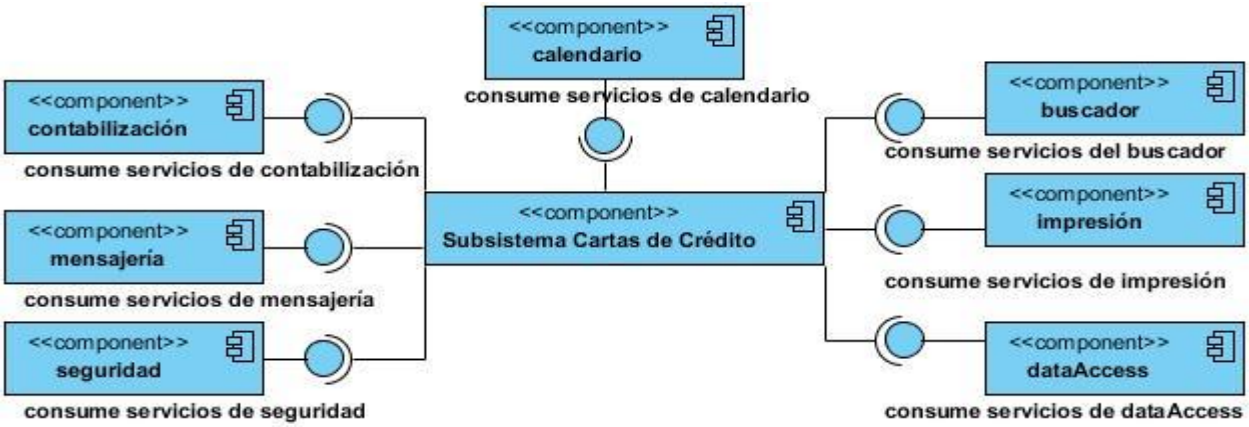


Figura 3.1. Modelo de componentes del subsistema Cartas de Crédito.

El componente **mensajería** brinda un conjunto de clases y funcionalidades que garantizan el envío de mensajes SWIFT tras la realización de operaciones bancarias que requieran la notificación a los bancos, la recepción y el envío automático de los mensajes resultantes de la contabilidad, disminuyéndose el esfuerzo de los usuarios del sistema para enviar o recibir dichos mensajes. Su utilización se evidencia en la negociación y emisión de una carta de crédito.

El componente **seguridad** se encarga de mantener la seguridad en todo el sistema a través de la comprobación de las peticiones y permisos. Gestiona la información de los usuarios, roles y sus permisos para lograr la seguridad requerida en el sistema. Es utilizado en el subsistema para controlar el acceso del usuario a las posibles operaciones a realizar sobre las cartas de crédito.

El componente **contabilización** brinda un conjunto de clases y funcionalidades necesarias que permiten realizar la contabilización de determinadas operaciones; constituye el de mayor consumo por parte del subsistema.

El componente **dataAccess** es imprescindible para el funcionamiento de todo el sistema, porque se encarga de proporcionar los elementos necesarios para la conexión a la base de datos.

El componente **buscador** brinda un conjunto de clases que constituyen el motor de búsqueda, así como una interfaz gráfica encargada de la búsqueda de diferentes conceptos.

El componente **impresión** contiene las funcionalidades necesarias mediante las cuales es posible la impresión de la información referente a la contabilidad.

El componente **calendario** ofrece un conjunto de clases además de una interfaz gráfica encargadas de gestionar los elementos necesarios para la creación de los cronogramas de compromisos de pago por cada funcionalidad para la gestión de las formas de pago que contienen los vencimientos.

3.3 Estándares de codificación

Un estándar de codificación comprende todos los aspectos relacionados con la generación de código, de tal manera que sea prudente, práctico y entendible para todos los programadores. Estos

no están enfocados a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código (16).

3.3.1 Convenciones de nomenclatura

El proyecto SAGEB decidió definir para la nomenclatura de las clases la notación PascalCasing, la cual define que los nombres deben comenzar por letra mayúscula, en caso de estar formado por palabras compuestas el inicio de cada palabra deberá tener mayúscula, se obvia el uso de artículos y se tiene en cuenta el tipo de clase que representa, entiéndase como tipo el rol que ellas desempeñan en el sistema.

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

Para la nomenclatura de las variables y los métodos se definió la notación CamelCasing, que es muy similar a PascalCasing con la excepción de que la letra inicial siempre es minúscula, en caso de estar compuesta por más de una palabra, estas empezarán con mayúscula exceptuando la primera.

Ejemplo: *idDocumentoEmbarque*. Lo mismo se aplica a los nombres de ficheros de código javascript y sus funciones y variables internas. Ejemplo: *registrarDiscrepancia***.**

Los nombres de los paquetes deben ser con minúscula y deben ser sustantivos que describan de alguna forma que tienen dentro. Ejemplo: *negociación*, *web*, *dao*.

3.3.2 Convenciones en la capa de presentación

Las clases pertenecientes a los sub-paquetes mvc y webflow del paquete web se nombrarán de la siguiente manera:

- ✓ Paquete controller: [Nombre de la clase] + [Nombre del controlador de Spring que se hereda]. Ejemplo:*CargarDatosNegociacionMultiAction.*
- ✓ Paquete command: [Nombre de la clase] + [Command]. Ejemplo: *NegociacionCommand*.
- ✓ Paquete validator: [Nombre de la clase] + [Validator]. Ejemplo: *DiscrepanciaValidator*.
- ✓ Paquete propertyEditor: [Nombre de la clase] + [PropertyEditor].Ejemplo: *EstadoDocumentoPropertyEditor***.**
- ✓ Paquete flowHandler: [Nombre de la clase] + [FlowHandler]. Ejemplo: *NegociacionFlowHandler***.**
- ✓ Paquete serviceFlow: [Nombre de la clase] + [Action o de MultiAction en dependencia de cuál de las dos clases herede]. Ejemplo: *ContabilizarNegociacionMultiAction***.**

3.3.3 Convenciones en la capa de negocio

Las clases pertenecientes a los paquetes facade y manager y sus sub-paquetes impl respectivamente se nombrarán de la manera siguiente:

✓ Paquete facade: para las Interfaces [Nombre del módulo] + [Facade]. Ejemplo: *EmisionFacade*. y las clases que implementan las interfaces [Nombre del módulo] + [FacadeImpl]. Ejemplo: *EmisionFacadeImpl.*

✓ Paquete manager: para las Interfaces [Nombre del módulo] + [Manager]. Ejemplo: *CartaCreditoManager*. y las clases que implementan las interfaces Nombre del módulo] + [ManagerImpl]. Ejemplo: *CartaCreditoManagerImpl*.

3.3.4 Convenciones en la capa de acceso a datos

Las clases pertenecientes al paquete dao y su sub-paquete impl se nombrarán de la manera siguiente:

✓ Paquete dao: para las Interfaces [Nombre de la entidad] + [DAO]. Ejemplo: *NegociacionDAO* y las clases que implementan las interfaces [Nombre de la entidad] + [DAOImpl]. Ejemplo: *NegociacionDAOImpl.*

3.4 Descripción de las clases y funcionalidades

Para un mayor entendimiento de los módulos del subsistema Cartas de Crédito se describirán las clases, atributos y métodos de mayor peso para el módulo Negociación, ya que los restantes siguen una estructura y comportamiento similar.

Se describirá la clase *ContabilizarNegociacionMultiAction* (*ver Tabla 5*) que es la encargada de manejar el flujo del módulo Negociación, la clase *Negociacion* (*ver Tabla 6*) que recoge la información necesaria de una negociación y la clase *NegociacionDAOImpl* (*ver Tabla 7*) encargada de obtener todo la información referente a una negociación desde la base de datos.

Nombre: ContabilizarNegociacionMultiActionController	
Tipo de clase: Controladora.	
Atributo	**Tipo**
globalFacade	GlobalFacade
negociacionFacade	NegociacionFacade
calendarioFacade	CalendarioFacade
Para cada responsabilidad:	
getComand(RequestContext contex)	Guarda en memoria el objeto negociación en un estado determinado, sirviendo como modelo a la vista.
getNameView(RequestContext contex)	Reconoce a que vista tiene que responder el flujo.
construirComandUnir(RequestContext contex)	Construye el command con los datos de la unión de las negociaciones que fueron elegidas.
construirComand(RequestContext contex)	Construye el command con los datos de la negociación que fue elegida.
construirComandCancelar(RequestContext contex)	Se encarga de seleccionar una negociación para cancelar, construir el command de esta negociación para enviarla al flujo.
compararCalendarioconNegociacion(RequestContext contex)	Verifica que los datos del componente Calendario

	sean iguales a los de la vista.
mostrarMensajeCalendario(RequestContext contex)	Notifica los datos del componente Calendario que no se corresponde con la vista.
primeraValidacion(RequestContext contex)	Permite invocar la primera validación de los asientos a contabilizar y en caso de obtener algún mensaje de error o alerta indicárselo al flujo.
contabiliza(RequestContext contex)	Envía a contabilizar la lista de asientos y en caso de existir errores de contabilización alerta al flujo de la presencia de estos.
saveCalendarioInConversation(RequestContext contex)	Salva el calendario de pagos que tiene el objeto Negociación de forma tal que el componente calendario pueda acceder a este.
saveCalendario(RequestContext context)	Permite actualizarle el calendario de pagos a la Negociación una vez que fue modificado por el componente Calendario.
comprobarDatosCalendario(RequestContext contex)	Valida que los datos del calendario sean correctos.
segundaValidacion(RequestContext contex)	Realiza la segunda validación de los asientos contables y notifica al flujo la presencia o no de mensajes de error o alerta.
formarMensajes(mensajes:List<String>)	Se encarga de unir los mensajes utilizando la lista de mensajes que envía le negociación.
construirAsientos(RequestContext contex)	Construye los asientos en función del requisito.
armarPagarCommand(RequestContext contex)	Se encarga de formar el command con los datos de la negociación a pagar.
esConMensajeria(RequestContext context)	Permite conocer si la vista lleva mensajería o no.
prepararDatosMensajeria(RequestContext context)	Permite conformar los datos que son necesarios para la mensajería.
checkError(RequestContext context, Object object)	Verifica si se produjo algún error en el proceso de mensajería.

Tabla 5. Descripción de la clase ContabilizarNegociacionMultiAction.

Nombre: Negociacion	
Tipo de clase: Modelo	
Atributo	**Tipo**
idNegociacion	Integer
referenciaCorriente	String
idDocumentoEmbarque	Integer
fechaVencimiento	java.util.Date
tipoCambioAplicado	Double
cantidadPago	Integer
idEstadoNegociacion	Integer

calendario	Calendario
fechaContable	java.util.Date
fechaValorNegociacion	java.util.Date
fechaBl	java.util.Date
observaciones	String
importe	java.math.BigDecimal
clientePleaseOpen	String

Tabla 6. Descripción de la clase Negociación.

Nombre: NegociacionDAOImpl	
Tipo de clase: Data Access Object	
Atributo	**Tipo**
Para cada responsabilidad:	
obtenerReferenciaOriginal(final int primero, final int cantidad, final String value)	Se encarga de obtener de la base de datos la referencia original de la carta de crédito que va a ser negociada.
obtenerCodigoBancos()	Permite obtener de la base de datos una lista con el código de los bancos.
obtenerNombreBancos()	Permite obtener de la base de datos una lista con el nombre de los bancos.
ExisteNegociacion(String referenciaCorriente)	Permite conocer dada la referencia corriente si existe la negociación.

Tabla 7. Descripción de la clase NegociacionDAOImpl.

3.5 Modelo de despliegue

Los diagramas de despliegue son un modelo de objetos que describen las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. Se utiliza como entrada fundamental en las actividades de diseño e implementación. Este representa una correspondencia entre la arquitectura del software y la arquitectura del sistema. A continuación se muestra la distribución de los nodos en el diagrama de despliegue del sistema Quarxo (*ver Figura 3.2*).

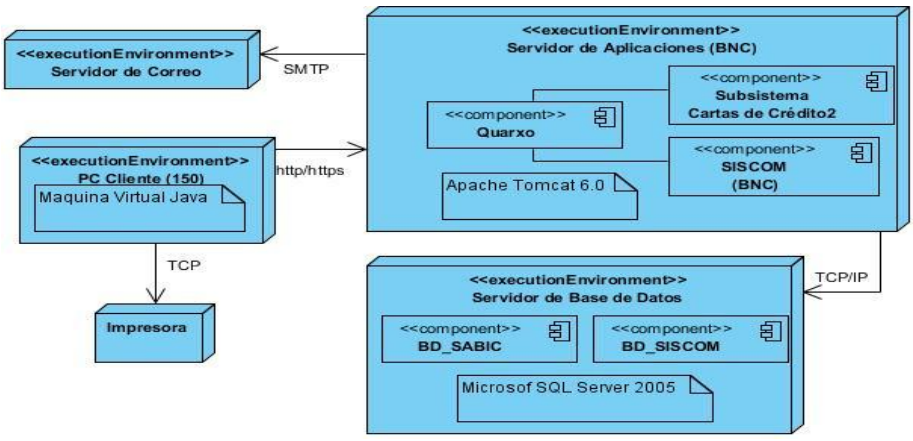


Figura 3.2. Diagrama de despliegue del sistema Quarxo.

3.6 Validación de la solución

Durante la implementación del software las posibilidades de cometer errores son múltiples por lo que se hace necesario realizar pruebas que contribuyan a detectar las imperfecciones e irregularidades de la aplicación. Las pruebas realizadas al subsistema Cartas de Crédito son las correspondientes al nivel de Unidad, Integridad y Aceptación.

3.6.1 Pruebas unitarias

Las pruebas de unitarias 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. Las pruebas de unitarias se derivan en pruebas funcionales (caja negra) y pruebas estructurales (caja blanca).

3.6.1.1 Pruebas funcionales o caja negra

El método de caja negra es aplicado con el fin 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.

Estas pruebas funcionales se realizaron en dos iteraciones, la primera se llevó a cabo por parte del grupo de calidad del CEIGE con el objetivo de descubrir errores como: 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 la realización de estas pruebas se utilizaron los diseños de casos de prueba elaborados en la disciplina de análisis y diseño. En la primera iteración fueron detectadas 14 no conformidades (*ver Anexo 51*) en dos pruebas de regresión que fueron corregidas logrando el resultado de 0 no conformidades para la segunda iteración. Para avalar la aplicación desde el punto de vista funcional fue firmada por el especialista del laboratorio de calidad del CEIGE y el responsable del equipo de desarrollo el acta de liberación (*ver Anexo 52*).

La segunda iteración fue realizada por el departamento de calidad de la universidad (CALISOFT) con el objetivo de validar y valorar la aceptabilidad de los límites operacionales de un sistema bajo carga de trabajo variable, mientras el sistema bajo prueba permanece constante, y de estrés: enfocada a evaluar cómo el sistema responde bajo condiciones anormales (extrema sobrecarga e insuficiente de memoria). Conjuntamente fue sometida la aplicación a las pruebas de aceptación del cliente en el BNC con el objetivo de verificar que el software está listo para el despliegue y cumple satisfactoriamente con los requisitos del cliente. Como resultado CALISOFT manifiesta su aprobación en el acta de liberación del sistema (*ver Anexo 53*).

3.6.1.2 Pruebas estructurales o caja blanca

El método de caja blanca es aplicado con el objetivo de analizar la lógica interna del programa, para ello se realiza diseños de casos de prueba atendiendo al comportamiento interno y la estructura del programa, examinándose la lógica interna sin considerar los aspectos de rendimiento. Estas pruebas estructurales se realizaron mediante el marco de trabajo JUnit el cual permite la automatización de las pruebas de aplicaciones en Java integrándolo al IDE de desarrollo Eclipse. El mismo consta de un conjunto de clases que fueron usadas para construir los casos de prueba y ejecutarlos automáticamente.

A continuación se muestran las imágenes que corresponden a la realización de las pruebas de Caja Blanca aplicando el *framework JUnit* al método *existeReferencia (ver figura 3.3)* de la clase controladora *CargarDatosMultiActionController* dentro del módulo emisión del subsistema Cartas de Crédito.

```
public boolean existeReferencia(HttpServletRequest request,
        HttpServletResponse response) {

    String refOriginal ="";
    Boolean existeReferencia=false;

    JSONObject json = new JSONObject();
    json.put("identifier", -123445);
    refOriginal= request.getParameter("ref");
    if(refOriginal!=null){

        CartaCredito cc = emisionFacade.getCartaCreditoByReferenciaOriginal(refOriginal);
        if(cc!=null)
            existeReferencia=true;
    }
    if(existeReferencia)
        json.put("refCorriente", "1");
    else
        json.put("refCorriente", "2");

    try {
        ResponseUtil.escribirDatosEnElResponse(response, json);
    } catch (IOException e) {
        e.printStackTrace();
    }
    return existeReferencia;
}
```

Figura 3.3. Método existeReferencia de la clase CargarDatosMultiactiónController del módulo emisión.

Para realizar las pruebas a dicho método primeramente se crea un objeto de la clase donde se encuentra el método a probar y a continuación se da paso a crear tres *mocks*, debido a que el método recibe como parámetros dos objetos de tipo *HttpServlet(request y response)*. Esto lo posibilita la librería *EasyMock* que trabaja en la creación de un *proxy* dinámico para dicho simulacro, el cual es controlado a través de un objeto de tipo *MockControl.*

```
public class TestJUnit extends TestCase {

    CargarDatosMultiActionController cargarDatos;
    public static final String KEY = "cartacredito";
    @SuppressWarnings({ "deprecation", "unchecked" })
    private MockControl controlHttpServlet;
    private HttpServletRequest mockhttpServletRequest;
    @SuppressWarnings({ "deprecation", "unchecked" })
    private MockControl controlHttpResponse;
    private HttpServletResponse mockhttpServletResponse;
    @SuppressWarnings({ "deprecation", "unchecked" })
    private MockControl controlHttpSession;
    private HttpSession mockHttpSession;
    @SuppressWarnings("deprecation")
    protected void setUp()throws Exception {
        cargarDatos = new CargarDatosMultiActionController();
        ApplicationContext context = new ClassPathXmlApplicationContext(
        "classpath:cu/uci/finixubnc/junit/cartacredito context.xml");
        CartaCreditoManagerImpl manager = (CartaCreditoManagerImpl) context.getBean("CartaCreditoManager");
        EmisionFacadeImpl facade = new EmisionFacadeImpl();
        facade.setCartaCreditoManager(manager);
        cargarDatos.setEmisionFacade(facade);
        controlHttpServlet= MockControl.createControl(HttpServletRequest.class);
        mockhttpServletRequest=(HttpServletRequest) controlHttpServlet.getMock();
        controlHttpResponse= MockControl.createControl(HttpServletResponse.class);
        mockhttpServletResponse=(HttpServletResponse) controlHttpResponse.getMock();
        controlHttpSession= MockControl.createControl(HttpSession.class);
        mockHttpSession = (HttpSession) controlHttpSession.getMock();
        super.setUp();
    }
```

Figura 3.4. Declaración de la clase TestJUnit para la prueba.

En el método *setUp,* de la propia clase, se inicializan las variables antes de cada prueba. Debido a la arquitectura del subsistema se debe crear un objeto por cada capa que haga énfasis en el método a probar. Estos pasos se van a configurar en el fichero *cartacredito-context.xml,* el cual va a ser el encargado de declarar y mapear los objetos creados por cada nivel de la aplicación.

```
55
56      @SuppressWarnings("deprecation")
57      protected void tearDown() {
58          controlHttpServlet.verify();
59      }
60      public void testExisteReferenciaCC() throws Exception{
61
62          mockhttpServletRequest.getParameter("ref");
63          controlHttpServlet.setReturnValue("CC00000188000");
64
65          controlHttpServlet.replay();
66
67          assertEquals(true,cargarDatos.existeReferencia(mockhttpServletRequest, mockhttpServletResponse));
68
69
70
71      }
72
```

Figura 3.5. Método tearDown y testExisteReferenciaCC de la clase TestJUnit.

En el método *tearDown* es donde se verifican si las llamadas a los métodos se realizaron correctamente, es el encargado de informar la existencia de los errores en la realización de las pruebas. En este caso verifica los objetos que son pasados por el *request* de forma automática para todos los test definidos en la clase *TestJUnit.*

Durante la realización de la prueba al método, se preparan los parámetros de pruebas necesarios para el método *existeReferencia.* Dichos parámetros serán pasados por el *mockHttpServletRequest*. Posteriormente se realiza la condición de prueba a través del método *assertEquals* el cual compara un valor con la respuesta del método.

Según el resultado del mismo JUnit muestra una ventana en correspondencia con este: si es satisfactorio, mediante una línea de color verde, en caso contrario de color rojo. A continuación se muestra el caso favorable para dicha prueba.

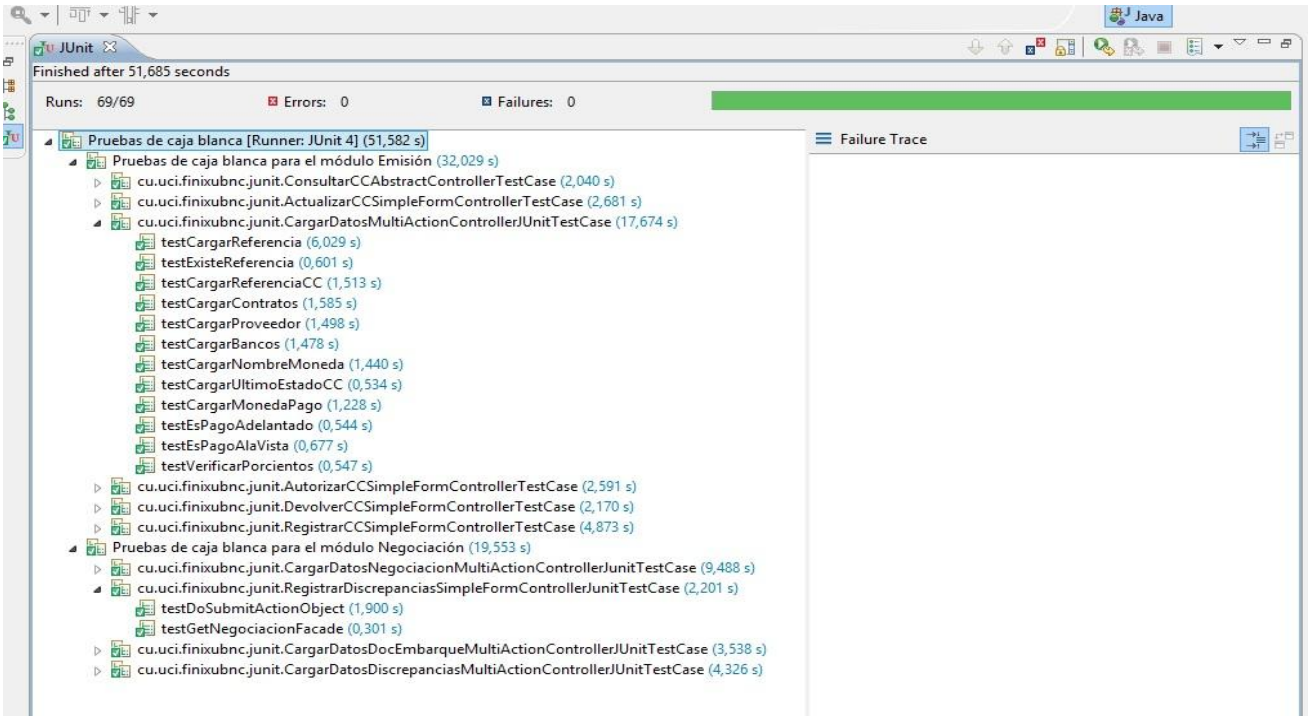


Figura 3.6. Consola de JUnit con el resultado de la prueba realizada.

3.6.2 Validación de la variable de investigación

La investigación desarrollada plantea como idea a defender que con el desarrollo la tercera fase del subsistema Cartas de Crédito para el Sistema Quarxo se agilizará el proceso de gestión y toma de decisiones en las operaciones con las cartas de crédito en el Banco Nacional de Cuba. A continuación se evaluará la variable tiempo en las operaciones con cartas de crédito.

Tiempo de ejecución de los procesos en la gestión de las cartas de crédito.

Operación	**Tiempo en minutos en el subsistema Cartas de Crédito (Fase 2)**	**Tiempo en minutos en el subsistema Cartas de Crédito (Fase 3)**
Registrar CC	10	6

Actualizar CC	30	10
Autorizar CC	20	10
Poner en estado devuelta la CC	30	2
Registrar discrepancia	30	5
Registrar Negociación	30	10

Tabla 8. Comparación del tiempo de realización para los procesos de CC.

3.7 Conclusiones del capítulo

En este capítulo se llevó a cabo las últimas disciplinas del ciclo de vida del proyecto SAGEB atendiendo a la metodología defina en el primer capítulo. Las mismas arrojaron los siguientes resultados:

✓ En la disciplina de implementación se obtuvo el diagrama de componentes del cual se logró como resultado la estructura general del sistema y el comportamiento de los servicios de estos componentes, se estableció los estándares de codificación utilizados en el proyecto a través de la notación PascalCasing y CamelCasing propiciando las reglas de escritura del código fuente para las capas de presentación, negocio y acceso a datos, se obtuvo la programación de todos los requisitos identificados en el capítulo anterior y la descripción de sus clases, atributos y funcionalidades, se construyó además el modelo de despliegue del cual resultó una vista física de la distribución de los componentes en los nodos del sistema conectado en el BNC.

✓ En la disciplina de pruebas se obtuvo la validación de la solución a través del acta de liberación emitida por el equipo de calidad por las pruebas efectuadas y el certificado de aceptación emitido por el BNC con la conformidad del cliente. Las pruebas realizadas demostraron que el software cumple con los requerimientos para satisfacer las necesidades del BNC garantizando la gestión eficiente de los módulos Emisión, Discrepancia y Negociación.

CONCLUSIONES

Una vez culminado el presente trabajo de diploma se puede afirmar que se logró alcanzar el objetivo general propuesto, tras cumplir todas las tareas trazadas al inicio de la investigación. El desempeño de los objetivos específicos permitió arribar a las siguientes conclusiones:

✓ La investigación realizada sobre los sistemas contables informáticos que gestionan los procesos de cartas de crédito manifestó la necesidad desarrollar la tercera fase del subsistema Cartas de Crédito del sistema Quarxo para satisfacer las necesidades del BNC.

✓ El estudio y caracterización de las tecnologías y herramientas definidas para el desarrollo de la aplicación contribuyeron a su correcta utilización garantizando la disminución de costos por concepto de licencias y soporte.

✓ Las disciplinas de Modelado de Negocio y Requerimientos del modelo de desarrollo propuesto por CEIGE contribuyeron a una mayor comprensión del negocio de la entidad y entendimiento de las necesidades del cliente; permitiendo realizar la captura, especificación y validación de los requisitos.

✓ La generación de los artefactos durante el desarrollo de la disciplina de Análisis y Diseño, proporcionaron la base para la fase de Implementación incidiendo notablemente en la estructura, organización y reutilización de código a través del diagrama de clases del diseño y de los patrones de diseños empleados.

✓ La disciplina de Implementación proporcionó la creación de la tercera fase del subsistema Cartas de Crédito y la integración satisfactoria del mismo al sistema Quarxo, lo que implica una gestión eficiente de los procesos de cartas de crédito en el BNC.

✓ Las pruebas efectuadas por el equipo de calidad interna y las pruebas de aceptación por parte del cliente, realizadas en el BNC avalaron el correcto funcionamiento de los requisitos implementados y la estabilidad necesaria para ser desplegado.

✓ Con el desarrollo del presente trabajo se consolidaron los conocimientos adquiridos durante la carrera que permitieron la realización del mismo. Se logró una buena interrelación con los clientes del BNC, lo que permitió alcanzar el objetivo general y cumplir las tareas definidas para el desarrollo de la solución.