Universidad de las Ciencias Informáticas

Facultad 3

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

Título: Diseño e Implementación del Subsistema Carta de Créditos Segunda Fase del Sistema QUARXO

Autor(es): Rolando Choy Piñeiro

Tutor(es): Ing. Yoan Antonio López Rodríguez

La Habana, 28 de Junio de 2012

"Año del 54 Aniversario del Triunfo de la Revolución"

DECLARACIÓN DE AUTORÍA

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

_____	_____
Rolando Choy Piñeiro	Ing. Yoan Antonio López Rodríguez
Autor	Tutor

DATOS DE CONTACTO

Ing. Yoan Antonio López Rodríguez: Ingeniero en Ciencias Informáticas graduado en la Universidad de las Ciencias Informáticas en Julio de 2008, Título de Oro, instructor. En la producción ejerce como Jefe de Equipo en el proyecto SAGEB (Sistema Automatizado para la Gestión Bancaria) perteneciente al Dpto. de Soluciones Financieras del Centro CEIGE. Docentemente se desempeña como profesor de Teleinformática de la Facultad 3.

FRASE

"La posibilidad de realizar un sueño es lo que hace que la vida sea interesante".

Paulo Coelho

DEDICATORIA

A mis padres, por siempre apoyarme, por su dedicación y enseñanza. A ellos les dedico esta tesis y todos mis logros personales. Ustedes son las personas que me impulsan a seguir mis sueños.

A toda mi familia, por ser el soporte de mi vida en especial a mis abuelas María y Ana. Por brindarme su apoyo incondicional y su gran amor.

AGRADECIMIENTOS

A mis padres por el apoyo que siempre me han dado, por ayudarme a emprender mi propio camino. Hoy les quiero agradecer por hacer de mí quien soy…Los amo. Esto es para ustedes.
A mi hermana Aylen por compartir todos estos años conmigo y marcar momentos felices en mi vida.
A mi abuela Ana por ser el ángel de mi guarda. Te quiero abuelita siempre y no sabes cuánto te extraño.
A mi abuela María y mi tía Barbarita por ser parte importante en mi vida y sentirse siempre orgullosas de mí.
A mis primos y primas en especial a Deisy, Félix, Anabel y Anisleidys por los momentos vividos junto a ustedes desde mi infancia.
A Yanni el amor de mi vida, por los maravillosos seis años que vivimos, por ser mi mejor complemento. Por compartir mis alegrías y angustias, contigo ha sido más liviano. Gracias por tu gran amor. Aunque estés lejos y la distancia nos separe hoy en día, esto es tan tuyo como mío.
A mi mejor amiga Dailiet (la Michi mía) que siempre ha estado a mi lado, mi conciencia, mi consejera y mi más fiel compañera. Porque hemos crecido juntos. Quiero darte las gracias por estar siempre a mi lado cuando lo he necesitado, te quiero mucho.
A Adi (la Puti mía) una persona que ocupa un gran espacio en mi corazón. Gracias por estar junto a mí, por las peleas de adolescentes, por esos buenos momentos que pasamos sin saber y por tu sonrisa, te quiero mucho.
A Eileen (la Sexicienta) por contagiarme contigo desde que te conocí, por las noches locas vividas que solo tú sabes, por compartir todos los momentos de alegría y sufrimiento. Siempre ha sido divertido estar contigo, te quiero también.
A Dicsan, Javier y Roberto Carlos porque más que amigos son mis hermanos, por crecer juntos desde la infancia, porque a pesar del tiempo y la distancia siguen ahí presentes.
A todas mis amistades de la UCI, en especial a Ania y Susana por las risas y los momentos vividos, por compartir estos cinco años, las extrañaré muchísimo.
A los amigos del grupo 4104 y 4306, a mis compañeros del proyecto SAGEB Yake, Puig, Vladimir y Eduardo que siempre me ayudaron cuando lo necesité.
A mi tutor Yoan por toda la ayuda brindada y por su aporte a mi formación profesional.
A la Universidad de las Ciencias Informáticas por permitir formarme como Ingeniero en Ciencias Informáticas.

RESUMEN

Como parte de la búsqueda de soluciones internas factibles a problemas sociales y económicos, y en aras de satisfacer las actuales necesidades operacionales del Banco Nacional de Cuba (BNC) con eficiencia, seguridad y mínimo costo, se desarrolla en la Universidad de las Ciencias Informáticas (UCI) por el proyecto Sistema Automatizado para la Gestión Bancaria (SAGEB) un sistema informático llamado Quarxo.

A raíz de la implantación de Quarxo se detectaron un grupo de operaciones relacionadas con los procesos de cartas de crédito, que por cuestiones de tiempo de desarrollo quedaron fuera del alcance del sistema y que aún no pueden realizarse por Quarxo. Teniendo en cuenta la complejidad e importancia del proceso de gestión de cartas de créditos en el BNC, se decide realizar una segunda fase de desarrollo e incorporar las nuevas funcionalidades claves dentro de las operaciones bancarias.

Con el propósito de incorporar dichas funcionalidades al sistema Quarxo, el presente trabajo de diploma comprende el Diseño e Implementación del subsistema Cartas de Crédito Segunda Fase, con el objetivo de satisfacer las necesidades del cliente.

La solución abarca la caracterización de herramientas y tecnologías de la plataforma Java por sus potencialidades como software libre para el desarrollo de aplicaciones web, así como la elaboración de artefactos propios de los flujos Diseño e Implementación como parte del proceso de desarrollo de software.

Palabras claves: BNC, Cartas de Crédito, Diseño, Implementación, Quarxo, SAGEB.

ÍNDICE

INTRODUCCIÓN

Las Tecnologías de la Información y las Comunicaciones (TIC) son un eslabón importante para las empresas, atendiendo a que facilitan todo tipo de actividades que se realizan en ellas. Las ventajas del manejo de las mismas a través de sistemas computarizados, son significativamente superiores a las de dicho procedimiento realizado de forma manual. Estos sistemas informatizan los procesos operativos y suministran los datos necesarios para la toma de decisiones, contribuyendo al incremento de su capacidad de organización y eficiencia.

Los bancos como entidades financieras no son la excepción ya que la complejidad de sus actividades, exige que cuenten con sistemas contables en función de la informatización de sus procesos, permitiendo agilizar su toma de decisiones, incrementar el control de sus operaciones y elevar la calidad del servicio ofrecido a sus clientes, dejando clara la necesidad de perfeccionamiento y evolución de sus soluciones informáticas.

En correspondencia con los avances tecnológicos logrados en el mundo, las instituciones bancarias cubanas han ido renovando sus sistemas informáticos. En nuestro país el Sistema Bancario Cubano y en específico el Banco Nacional de Cuba (BNC) ha incorporado las tecnologías para prestar servicios de manera más eficiente a sus clientes. En el mismo se realiza la mayor parte de las negociaciones del estado, además de la amortización de la deuda externa. La gestión de las cartas de créditos como instrumento indispensable de pago, ocupa más del sesenta por ciento del tiempo de los trabajadores de la entidad.

Actualmente se desarrolla para el BNC una segunda fase del sistema recién implantado denominado Quarxo en colaboración con la Universidad de las Ciencias Informáticas (UCI). El mismo en una primera fase fue concebido para llevar a cabo las funcionalidades críticas de la entidad y en una segunda fase se pretende incorporar un grupo de funcionalidades que aunque no fueron consideradas críticas, permitirán lograr mayor rapidez en la ejecución de los procesos.

Quarxo en una primera fase de desarrollo posee un conjunto de subsistemas que pueden estar relacionados entre sí, entre ellos, el subsistema Cartas de Crédito. A raíz de su implantación se detectaron un grupo de operaciones relacionadas con los procesos de cartas de crédito y de cambios introducidos en el sector bancario y otros, que por cuestiones de tiempo de desarrollo quedaron fuera del alcance de la primera fase y que aún no pueden realizarse por el sistema.

Estas operaciones de cartas de crédito se realizan por la Transacción General de Contabilidad (TGC), que permite ejecutar cualquier operación bancaria construyendo de forma manual cada registro o asiento contable involucrado. El uso de la TGC puede ser complejo y lento cuando se requiere contabilizar un proceso de negocio, debido a que la transacción solo posee validaciones generales de contabilidad y no propias del proceso de negocio que se esté realizando. Otro inconveniente de la TGC es el hecho de que no permite actualizar las tablas operativas con la información contable de forma automática y tampoco se encuentra integrada con los subsistemas de mensajería impidiéndose el envío de mensajería SLBTR y/o SWIFT de forma automática. Teniendo en cuenta que las operaciones sobre las cartas de crédito que se realizan por esta transacción pueden involucrar gran cantidad de asientos contables, exige la actualización de operativos y el envío de mensajes SLBTR y SWIFT en algunos casos. Por tanto, la utilización de la TGC para los procesos identificados representa una solución lenta y compleja.

El presente trabajo se centra en el diseño y la implementación del Subsistema Carta de Créditos Segunda Fase. Basándose en los argumentos antes expuestos se plantea como **problema a resolver**: Las deficiencias en el proceso de Cartas de Crédito en el BNC provocan retrasos en la emisión y negociación de las cartas de crédito.

A partir del problema anteriormente planteado se determina como **objetivo general**: Realizar el diseño e implementación del subsistema Cartas de Crédito Segunda Fase para el sistema Quarxo que agilice la emisión y negociación de las cartas de créditos en el BNC.

Teniendo como **objetivos específicos**:

1. Fundamentar la investigación mediante la elaboración del Marco Teórico.
2. Definir las funcionalidades necesarias a agregar en el subsistema Cartas de Crédito de Quarxo.
3. Diseñar el subsistema Cartas de Crédito Segunda Fase de Quarxo.
4. Implementar el subsistema Cartas de Crédito Segunda Fase de Quarxo.
5. Validar el subsistema Cartas de Crédito Segunda Fase de Quarxo.

El **objeto de estudio** del presente trabajo estará enfocado hacia la informatización de la gestión de las cartas de crédito en entidades bancarias y el **campo de acción** hacia la gestión de las cartas de crédito en el BNC.

Idea a defender: Con el desarrollo del subsistema Cartas de Crédito Segunda Fase para el sistema de gestión bancaria Quarxo se agilizará el proceso de gestión y toma de decisiones en las operaciones de cartas de crédito en el BNC.

Dando cumplimiento a los objetivos específicos del trabajo se trazaron las siguientes **tareas investigativas**:

1. Análisis de las metodologías, lenguajes y herramientas definidas en el proyecto.
2. Análisis de las tecnologías y patrones de diseño establecidos en el proyecto.
3. Análisis de los procesos de Cartas de Crédito en el BNC.
4. Análisis de los sistemas informatizados existentes en el mundo para realizar los procesos de Cartas de Crédito.
5. Análisis del subsistema Cartas de Crédito del sistema Quarxo.
6. Definición de las funcionalidades a agregar en el subsistema Cartas de Crédito de Quarxo en su segunda fase de desarrollo.
7. Obtención del Modelo de diseño para el subsistema Cartas de Crédito Segunda Fase.
8. Realización de la implementación del subsistema Cartas de Crédito Segunda Fase.
9. Realización de pruebas de unidad, integración y aceptación al subsistema Cartas de Crédito Segunda Fase.
10. Documentación de la solución.

Posibles resultados:

1. Incorporación de nuevas funcionalidades al subsistema Cartas de Crédito de Quarxo.
2. Mejoras en la gestión y toma de decisiones en las operaciones de cartas de crédito en el BNC.

Estructura del documento

Capítulo 1: Fundamentación teórica

Se realiza un análisis del estado del arte a nivel nacional e internacional, permitiendo la fundamentación teórica del tema. Se mencionan y describen algunos sistemas existentes que incluyan el objeto de estudio y se hace hincapié en el estudio del subsistema Cartas de Crédito del Sistema Quarxo. También se describen las tecnologías a considerar, el lenguaje de programación que se utiliza para la solución, las metodologías y herramientas de desarrollo.

Capítulo 2: Diseño de la solución

En este capítulo se describe la arquitectura del subsistema, se explica el modelo de diseño, así como los patrones utilizados para dar respuesta a la solución planteada. Se realiza la estructuración del subsistema, evidenciando los diagramas de clases de diseño, diagramas de paquetes y diagramas de secuencias para conformar el modelo de diseño, según la arquitectura base y los requerimientos del sistema, así como la realización del modelo de datos.

Capítulo 3: Implementación y validación

Se enfoca en la construcción de la solución explicando los aspectos principales de la implementación. Se realiza una descripción de las clases y funcionalidades del módulo Negociación del subsistema Cartas de Crédito, quedando establecida la validación de la solución propuesta, la implementación del producto y además la aplicación de pruebas unitarias para la comprobación de su buen funcionamiento.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

El presente capítulo pretende en un primer momento realizar el estado del arte acerca de los conceptos fundamentales del tema para la comprensión del negocio y del resto de la investigación. Se incluye además, un análisis sobre los sistemas que automatizan el proceso a nivel mundial y a nivel nacional. Por último, se realiza un estudio relacionado con la Ingeniería de Software haciéndose énfasis en las metodologías, herramientas y lenguajes que se utilizarán para modelar el subsistema, con el fin de lograr de forma productiva los objetivos propuestos.

1.2 Conceptos fundamentales en entidades bancarias

1.2.1 Los bancos

Los bancos son entidades que canalizan los recursos financieros (dinero) en la economía a través de la captación de depósitos y el otorgamiento de préstamos, para ello utilizan como principal instrumento las tasas de interés, las cuales son un incentivo tanto para ahorrar dinero como para pedir prestado [1].

Se puede señalar que el banco moderno tiene que cumplir tres grandes funciones:

- ✓ La intermediación del crédito.
- ✓ La intermediación de los pagos.
- ✓ La administración de los capitales.

1.2.2 Los bancos cubanos

La política económica cubana ha evolucionado sustancialmente hacia un proceso de ajuste conforme a las nuevas circunstancias en que se desarrollan sus relaciones y en cuyo contexto, la banca juega un nuevo e importante papel. Dentro de estas, el BNC desempeña un papel fundamental para el estado cubano llevando a cabo funciones de obtener y otorgar créditos en moneda nacional y libremente convertible.

Centraliza las relaciones con las entidades extranjeras de seguro de crédito oficial a la exportación según decida el Banco Central de Cuba (BCC). Constituye una entidad de seguro de crédito oficial a la exportación con arreglo a la legislación vigente en materia de seguros. Mantiene el registro, control, servicio y atención a 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 BCC [2].

1.2.3 La contabilidad

La contabilidad surge por la necesidad que tuvo el hombre de conocer el valor de sus riquezas y deudas. Se ha definido la contabilidad como:

"La ciencia y/o técnica que enseña a clasificar y registrar todas las transacciones financieras de un negocio o empresa para proporcionar informes que sirven de base para la toma de decisiones sobre la actividad". [3].

El Instituto Americano de Contadores Públicos plantea que: "La contabilidad es el arte de registrar, clasificar y resumir en forma significativa y en términos de dinero, las operaciones y los hechos que son cuando menos de carácter financiero, así como el de interpretar sus resultados". [4].

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". [5].

1.2.4 Contabilidad bancaria

"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". [1].

1.2.5 Instrumentos de pago

Un instrumento internacional de pago, es aquel por medio del cual 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. Estos instrumentos establecen un lenguaje internacional estándar para evitar malentendidos entre las partes contratantes porque se sabe que en una operación internacional existen riesgos derivados de las diferencias en idiomas, costumbres, tipos de cambios, distancias, tiempos, y por supuesto también por la inexistencia de un contacto directo con el proveedor o cliente. [32].

Existen diferentes instrumentos de pagos, pero los más utilizados en el comercio internacional son:

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

1.2.5.1 Carta de Crédito

En el comercio exterior existen diferentes instrumentos para efectuar el cobro o pago de las mercancías, tales como el giro bancario, órdenes de pago y las cartas de crédito, siendo esta última, a la que se le reconoce ofrecer mayor seguridad, además de que los términos utilizados en dicho instrumento de pago, son mundialmente aceptados por regirse por las UCP[1] publicadas por la ICC[2]. [23].

La ICC define Carta de Crédito como cualquier convenio, como quiera que se le nombre o describa, por el cual un banco (el "banco emisor"), actuando a solicitud y por instrucciones de un cliente (el "ordenante"), o por su propia cuenta:

- ✓ Hará un pago a un tercero o a su orden (el "beneficiario"), o aceptará y pagará letras de cambio ("giros") giradas por el beneficiario.
- ✓ Autorizará a otro banco a efectuar dicho pago o a aceptar y pagar dichas letras de cambio ("giros").

1 Reglas y Usos Uniformes Relativos a los Créditos Documentarios.

2 Cámara Internacional de Comercio.

- ✓ Autorizará a otro banco a negociar, contra el(los) documento(s) estipulado(s), siempre y cuando los términos y condiciones del crédito sean cumplidos (ICC, 1993). [23].

1.2.5.1 Actores que intervienen en una Carta de Crédito

Ordenante: persona o entidad que solicita la apertura del crédito a su banco, comprometiéndose a efectuar el pago. Suele ser el importador o un agente.

Banco negociador: entidad a cuyo favor se emite el crédito, reseñado mediante la carta de crédito, autorizada a exigir el pago del crédito al presentar la documentación probatoria del cumplimiento de las condiciones pactadas.

Banco emisor: banco elegido por el comprador que confecciona y procede a la apertura del crédito. Efectúa el pago del crédito si se cumplen las condiciones exigidas en el mismo.

Banco corresponsal: banco que obra a petición y de conformidad con las instrucciones del banco emisor obligándose a hacer un pago por la compra de determinada mercancía a un tercero (''beneficiario''), cuando éste efectúe la presentación de los documentos acordados en la Carta de Crédito. [23].

1.2.5.3 Tipos de Cartas de Crédito

Las Cartas de Crédito se pueden clasificar de varias formas dependiendo de la manera o el objetivo con que sean emitidas.

De importación: se refiere a todo acuerdo, cualquiera que sea su denominación o descripción, por el que un banco (Banco Emisor), obrando a petición y de conformidad con las instrucciones de un cliente (Ordenante) o en su propio nombre, se obliga a hacer un pago a un tercero (Beneficiario) por la compra de determinada mercancía.

De exportación: es la que emite un banco cuando el comprador extranjero (Solicitante) solicita a su banco (Banco Emisor) que le emita un crédito documentario a favor de un proveedor (Beneficiario). La Garantía Bancaria constituye un compromiso de pago del banco emisor a favor de un beneficiario.

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. [23].

1.3 Sistemas informáticos contables

"Un sistema contable es una combinación de programas, procedimientos, datos y equipamiento, utilizados de manera coherente en el procesamiento de la información". [6].

Los Sistemas Informáticos Contables 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, balances, producción de artículos, nóminas, etc. Para ello sólo hay que ingresar la información requerida, como las pólizas contables, ingresos y egresos y hacer que el programa realice los cálculos necesarios. [7].

1.3.1 Sistemas informáticos de Carta de Crédito en el mundo

Con el acrecentado desarrollo de la informática en el mundo ha surgido la necesidad de informatizar muchas esferas de la economía, con el fin de no quedar atrás en la era tecnológica, mitigar errores que el ser humano comete sin darse cuenta y acelerar el trabajo en cualquier empresa o institución bancaria. Con los fines anteriormente expuestos se han desarrollado varios sistemas que resuelven los problemas de la banca.

A continuación se mostrarán diferentes sistemas contables en el mundo que gestionan las operaciones relacionadas con las Cartas de Crédito.

CARCRED es un sistema desarrollado por la compañía venezolana La Sistema (Fábrica Venezolana de Tecnología Financiera) 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 Créditos desde su apertura, hasta el momento de cancelación final.
Emite la relación contable diaria en reporte y archivo de textos, genera las llamadas débitos/créditos de las operaciones efectuadas.

Está dividido en seis módulos. El módulo de Cartas de Crédito posibilita el registro de los instrumentos de pago utilizados en el comercio internacional. [8]

TradeMasterQW Global Trade Management System (Sistema de Administración Comercial Global TradeMasterQW)

QuestaWeb es una compañía que permite a importadores por medio del sistema TradeMasterQW Global Trade Management System manejar y controlar el proceso de las Cartas de Crédito de principio a fin. El sistema automáticamente construye cada documento de datos de sistema, automáticamente emite, tramita enmiendas y alerta a las partes de los documentos que fallan o términos (condiciones) de negocio no satisfechos. Además mantiene los totales de acumulado de cantidades que permanecen sobre cada Carta de Crédito después de cada transacción.

Contiene una historia completa de cada cambio y un registro en tiempo real de como el dinero de una empresa está siendo dispersado a escala mundial. Además, los bancos pueden tener acceso al intercambio de datos electrónico, eliminando la necesidad del cambio de documentos de papel. [9].

COBIS SCI es el Sistema Integral de Comercio Internacional, cuyo objetivo principal es brindar soporte a todas las actividades de negocio y operativas de comercio internacional que se llevan a cabo dentro de un Banco o Institución Financiera.
El software permite además la apertura de Cartas de Crédito y el registro de la información correspondiente a la misma.

El sistema controla los distintos estados vinculados a la apertura de una Carta de Crédito, como son: emisión, verificación o confirmación de datos, contabilización, liquidación.

La aplicación permite gestionar Cartas de Crédito de Importación, Cartas de Crédito de Exportación, Garantías, etc. [7].

1.3.2 Sistemas informáticos contables bancarios en Cuba

En Cuba existen varios sistemas contables pero casi todos están enfocados a obtener mayor eficiencia en la gestión contable empresarial. Las operaciones con Cartas de Crédito son procesos puramente bancarios.

SABIC (Sistema Automatizado para la Banca Internacional de Comercio) es un sistema diseñado y desarrollado en lenguaje de programación FoxPro, para el sistema operativo MS-DOS por la Dirección de Sistemas Automatizados del Banco Central de Cuba para satisfacer las necesidades de procesamiento de datos de bancos e instituciones no bancarias, utilizando los medios técnicos de computación disponibles en el mercado. Las características fundamentales del sistema son la contabilización multimoneda, la contabilización en tiempo real, la transaccionalidad y la modularidad. Este sistema permite contabilizar en tiempo real, lo cual se traduce en que mantiene actualizados los ficheros contables en todo momento. Soporta una contabilidad multimoneda que hace innecesario depender del tipo de cambio para registrar los activos y pasivos. Estas características permiten disponer en todo momento de la información, lo más exacta posible, sobre la posición financiera de la institución. Todos los módulos que están incluidos en el sistema tienen acceso limitado, de forma tal que un usuario solo puede acceder a un módulo si la labor que realiza lo requiere y está autorizado, además se pueden integrar más módulos al sistema sin alterar al sistema. [10].

El SABIC contiene en sí los procesos básicos de contabilización pero deja fuera muchos procesos que se realizan en el BNC que se relacionan necesariamente o no con la contabilización. En el proceso de gestión de cartas de crédito la contabilidad es un punto que es imprescindible, pero se llevan a cabo otros procesos de gran importancia como la generación de mensajes SWIFT hacia los bancos exteriores una vez emitida y negociada la carta de crédito. Tampoco permite la generación automática y la configuración de reportes por lo que se hace engorrosa esta actividad. [10].

Tal situación conllevó al desarrollo del subsistema Cartas de Crédito del sistema Quarxo implantado recientemente permitiendo ofrecer una gestión integral en el proceso de emisión y negociación de las cartas de crédito.

QUARXO es una aplicación web realizado en la UCI por el proyecto SAGEB para el BNC. El mismo sustituye al SABIC como solución a los procesos más críticos del BNC y garantiza la gestión de los procesos de una forma más sencilla, segura y eficiente, a través de un conjunto de subsistemas. Posee integración con los sistemas de mensajería y ofrece el tratamiento requerido a la validación de los datos de entrada, posibilitando la detección de forma temprana de cualquier irregularidad. Posibilita generar reportes por el usuario que muestran información específica sobre la actividad contable.

El sistema cuenta con varios subsistemas entre los que se encuentra Cartas de Crédito, el cual cuenta con seis módulos. De ellos, cuatro, resuelven básicamente la gestión de las cartas de créditos y la negociación de las mercancías asociadas.

El módulo de Emisión de Cartas de Crédito es el principal y tiene como objetivo el control y la contabilidad de todas las operaciones que se realizan con las cartas de crédito. El mismo permite registrar una nueva carta de crédito y haciendo uso del buscador se pueden realizar más operaciones como consultar dicha carta de crédito, actualizarla en caso necesario, autorizarla por la Dirección de Gestión de Negocios, operación que solo puede realizar el personal autorizado en el banco. Se podrá emitir la Carta de Crédito contabilizando en tiempo real y enviar los mensajes hacia los bancos extranjeros automáticamente. Se puede enmendar la Carta de Crédito por solicitud de la empresa, 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 créditos cuando se requieran y de cancelarla finalmente.

Un quinto módulo es usado para gestionar las Capacidades Financieras otorgadas a los clientes a modo de garantías y un sexto módulo nombrado Acuerdo que resuelve entre otras cosas, la gestión de los respaldos de las cartas de créditos.

1.4 Metodologías de desarrollo

Una metodología no es más que el estudio de los métodos más apropiados que se emplean para desarrollar software de manera eficiente; o como precisan otros autores, define "Quién debe hacer Qué, Cuándo, y Cómo debe hacerlo. Una metodología define un conjunto de pasos y procedimientos que deben seguirse para desarrollar software". [11].

La importancia de utilizar una metodología radica principalmente en lograr un producto final con calidad. Dentro de las metodologías más utilizadas actualmente se pueden citar: 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.

El proyecto SAGEB desde sus inicios utilizó como metodología de desarrollo de software, la metodología RUP, al constituir un proyecto de alta complejidad y por ser conveniente para asegurar la producción de software con calidad desde el principio y ganar en organización para poder entregar el software en el tiempo planificado. La ventaja principal que brinda la metodología RUP es que se basa en las mejores prácticas que se han intentado y se han probado en el campo.

1.4.1 Proceso Unificado de Desarrollo de Software (RUP)

RUP es una forma disciplinada de asignar tareas y responsabilidades en una organización de desarrollo (quién hace qué, cuándo y cómo). "El proceso de software propuesto por RUP tiene tres características esenciales: está dirigido por los Casos de Uso, está centrado en la arquitectura, y es iterativo e incremental". [11].

RUP divide su trabajo en cuatro fases de desarrollo:

1. Inicio: donde se decide la viabilidad del proyecto.
2. Elaboración: donde se obtiene una arquitectura candidata.
3. Construcción: donde se obtiene la capacidad operacional inicial.
4. Transición: donde se obtiene un release[3] del producto.

RUP además de contar con cuatro fases de trabajo cuenta con nueve flujos de trabajos, los primeros seis son conocidos como flujos de ingeniería y los tres últimos como de apoyo.

Estos flujos son:

1. Modelamiento del negocio: este flujo identifica los procesos de negocio, los que estarán sujetos a automatización y quiénes intervienen en los mismos.
2. Requerimientos: se identifican las restricciones que se imponen y lo que el sistema debe hacer.
3. Análisis y Diseño: se trasladan los requerimientos dentro de la arquitectura de software.
4. Implementación: se crean un software que se ajuste a la arquitectura y que tenga el comportamiento deseado.
5. Pruebas: se asegura que el comportamiento requerido es el correcto y que todo lo solicitado está presente.

[3] **Release:** Nueva versión de una aplicación informática.

6. Configuración y administración del cambio: se guardan todas las versiones del proyecto.
7. Administrando el proyecto: se administran horarios y recursos.
8. Ambiente: se administra el ambiente de desarrollo.
9. Distribución: se hace todo lo necesario para la salida del proyecto.

Teniendo en cuenta los objetivos del presente trabajo es necesario enfocarse en los flujos de Diseño e Implementación.

Análisis y Diseño

Tiene como objetivo principal interpretar y traducir los requisitos a una detallada descripción que indica cómo implementar el sistema. Además de transformar los requisitos al diseño del futuro sistema, desarrollar una arquitectura para el sistema y adaptar el diseño para que sea consistente con el entorno de implementación. [11].

Implementación

En este flujo de trabajo se implementan las clases y objetos en ficheros fuente, binarios, ejecutables y demás, organizándose los mismos en paquetes de implementación. Tiene como objetivo principal dar respuesta a los requisitos funcionales expuestos en el diseño, mediante la implementación, obteniéndose como resultado final el propio software. [11].

1.5 Lenguajes y herramientas de modelado

"El Proceso Unificado utiliza el Lenguaje Unificado de Modelado (Unified Modeling Language, UML) para preparar todos los esquemas de un sistema de software, de hecho, UML es una parte esencial del Proceso Unificado – sus desarrollos fueron paralelos". [12].

1.5.1 UML

UML es un lenguaje de modelado visual que permite visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software, soportando además el paradigma orientado a objetos. Se caracteriza por dividir a los sistemas en una estructura estática y un comportamiento dinámico. [12].

Entre sus ventajas se puede decir que aporta mayor rigor en la especificación, permite realizar una verificación y validación del modelo realizado, posibilita automatizar determinados procesos y permite 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 mantener la visión en el diseño, de más alto nivel, de la estructura de un proyecto.

1.5.2. Visual Paradigm

Visual Paradigm es una herramienta que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Contribuye a que las aplicaciones se construyan con una mejor calidad y menor coste, ya que permite:

- ✓ Aplicar ingeniería inversa.
- ✓ La generación de código.
- ✓ Editar detalles de casos de uso. [28].

1.5.3 Erwin

Erwin es otra de las herramientas de la tecnología CASE, cuyo mayor diferenciador es su simplicidad por generar código para la mayoría de los manejadores de base de datos ya que es completamente abierta. Esta herramienta ofrece una metodología para realizar diagramas entidad-relación y cuenta con una interfaz gráfica altamente intuitiva. Asegura consistencia, reutilización e integración de los datos. [30].

1.6 Ambiente de desarrollo

Para lograr reducir al mínimo el tiempo de desarrollo y lograr un producto con la calidad requerida, se hace necesario seleccionar correctamente las herramientas y tecnologías a utilizar teniendo en cuenta el contexto donde se aplicará el software.

Para el desarrollo del sistema Quarxo se definen un conjunto de tecnologías, herramientas y lenguajes a utilizar, basándose en la disponibilidad e idoneidad de los mismos considerando las características del proyecto. A continuación se expone una breve caracterización de cada uno de estos elementos.

1.6.1 Plataforma J2EE

J2EE (en inglés: Java 2 Platform, Enterprise Edition) define un estándar para el desarrollo de aplicaciones empresariales multicapa. La plataforma J2EE ofrece mejores perspectivas de desarrollo para empresas que quieran basar su arquitectura en productos basados en software libre. Entre las principales ventajas de J2EE se encuentra que permite soporte de múltiples sistemas operativos, frameworks y patrones de programación que permiten responder de una forma robusta y flexible a todas las demandas de este tipo de aplicaciones. [27].

1.6.2 Lenguajes programación

1.6.2.1 Java

Java es un lenguaje de programación orientado a objetos, robusto y fácil de aprender, permite la codificación de la propuesta de solución de una manera rápida y flexible.

Al integrarse con los diferentes frameworks de desarrollo, permite programar páginas web dinámicas con accesos a bases de datos utilizando XML con cualquier tipo de conexión de red entre cualquier sistema.
Es un lenguaje multiplataforma de código abierto, distribuido, que proporciona un conjunto de clases para su uso en aplicaciones de red, permitiendo abrir sockets y establecer conexiones con servidores o clientes remotos. Además de ser poderoso manejando excepciones, Java fue diseñado para crear software altamente fiable. [7].

1.6.2.2 Javascript

Es un lenguaje que no requiere compilación y se utiliza comúnmente para la construcción de páginas web en combinación con el XHTML. Se caracteriza por ser un lenguaje que responde a eventos generados por el usuario o por el navegador, es independiente de la plataforma necesitando solo de un navegador para ejecutar el código, permite un desarrollo rápido y es relativamente fácil de aprender. Es soportado por la mayoría de los navegadores como Internet Explorer, Netscape y Mozilla Firefox. [13].

1.6.3 Entorno integrado de desarrollo

1.6.3.1 Eclipse IDE

Eclipse es un IDE (en español Entorno de Desarrollo Integrado) para Java muy potente. Fue creado por la IBM bajo la filosofía de software libre. Se está convirtiendo en el estándar de puntera de los entornos de desarrollo para Java. Eclipse es un marco de trabajo que está compuesto por componentes que se pueden o no incluir en dependencia de las necesidades del desarrollador, a estos complementos se les llama (plugins). De hecho, existen complementos que permiten usar Eclipse para programar en otros lenguajes aparte del Java como son PHP, Perl, Python, C/C++. [29].

1.6.4 Contenedor web

1.6.4.1 Apache Tomcat

Tomcat es una implementación libre y de código abierto de las tecnologías Servlets y JSP (en inglés: JavaServer Pages) desarrollada bajo el proyecto Jakarta de la Fundación Apache (en inglés: Apache Software Foundation). Es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. [14].

1.6.5 Control de versiones

1.6.5.1 SubVersion

Para el control de las versiones del sistema se utiliza el SubVersion 1.6.6, que está desarrollado sobre tecnología Open Source y se integra con Eclipse mediante el plugin SubEclipse.

Permite recuperar versiones antiguas de los datos registrados y examinar el historial de los mismos, manejando la información de ficheros y directorios a través del tiempo, la cual radica en un árbol de ficheros en un repositorio central. [15]

1.6.6 Base de datos

1.6.6.1 SQL Server 2005

Provee herramientas sólidas y conocidas, reduciendo la complejidad de la creación, despliegue, administración y uso de aplicaciones de datos empresariales. Se caracteriza por brindar soporte para transacciones y procedimientos almacenados. Sus lenguajes de consulta son el SQL y el T-SQL (en inglés: Transact-SQL), siendo este último el principal medio de programación y administración del servidor. Requiere para su funcionamiento el sistema operativo Microsoft Windows, representando esto una de sus principales desventajas.

Se decide utilizar como gestor de base datos Microsoft SQL Server que a pesar de ser una herramienta privativa se hace necesario mantener, a decisión del cliente, ya que se encuentra familiarizado con el trabajo con dicha herramienta, además de contener una considerable cantidad de procedimientos almacenados que por cuestiones de tiempo no es posible implementar en otro gestor de base datos. [16].

1.7 Frameworks

Un framework es "una mini arquitectura reusable que provee una estructura y comportamiento genérico para una familia de abstracciones de software [...]". [19].

Existen diferentes tipos de framework, para diferentes propósitos, algunos orientados al desarrollo de aplicaciones web, o para un determinado sistema operativo o lenguaje. En un sentido muy amplio el framework que se utiliza determina la arquitectura del software.

1.7.1 Spring

Se emplea para el desarrollo de la aplicación el Spring Framework, el cual contiene un conjunto de librerías de clases que brindan una estructura conceptual y tecnológica para el desarrollo de aplicaciones de la plataforma JEE.

Permite crear soluciones bien documentadas, seguras y robustas y ofrece gran libertad a los desarrolladores en Java. Además en su implementación tiene en cuenta las tres capas arquitectónicas: acceso a datos, negocio y presentación. [20].

El framework está dividido en módulos para su correcto funcionamiento, son estos:

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

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

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

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

1.7.2 Spring WebFlow

Spring WebFlow es un framework que permite operar la navegación de la aplicación Web. Al ser limitado el flujo de páginas brindado por los frameworks MVC clásico, surge el framework Spring WebFlow. Los flujos Web en este framework son diseñados para ser autocontrolados, dando la posibilidad de definir reglas de navegación múltiples y complejas. En Spring WebFlow un flujo controla la conversación (entorno nuevo que define el vacío entre Sesión y Petición) completa, desde que inicia hasta que termina, limpiando automáticamente la memoria siempre y cuando se termine el flujo. Permite la creación de flujos reutilizables en toda la aplicación.

1.7.3 Hibernate

Hibernate es una herramienta para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación, mediante archivos declarativos XML (en inglés: Extensible Markup Language) que permiten establecer estas relaciones.

Permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones, y un gran número de tipos de datos. Soporta diferentes entornos como: MySQL, Oracle, DB2 (en inglés: DataBase 2), etcétera. Se adapta a cualquier proceso de desarrollo, ya sea comenzando un diseño desde cero o trabajando con una base de datos existente y apoyará cualquier arquitectura de aplicaciones. [21].

1.7.4 DojoToolkit

Esta herramienta es un framework que contiene APIs[4] y controles para ayudar al desarrollo de aplicaciones web. Incluye un sistema de empaquetado, los efectos visuales de la interfaz de usuario, abstracción de eventos, almacenamiento de APIs en el cliente, e interacción de APIs con AJAX.

Dojo proporciona variadas opciones en una sola biblioteca haciendo un mejor trabajo que sustenta los nuevos y viejos navegadores, resolviendo los problemas de compatibilidad entre los navegadores. Tiene múltiples puntos de entrada, es independiente del intérprete y unifica estándares de codificación. [7].

1.8 Patrones de diseño

Los patrones de diseño de software son una solución probada para un problema general de diseño, en un contexto determinado. Encierran la experiencia de los programadores e ingenieros que han ido adquiriendo en las soluciones de problemas comunes. [17].

1.8.1 Patrones de asignación de responsabilidades. (GRASP)

GRASP (en inglés: General Responsabilty Assignment Software Patterns) es un conjunto de patrones que permite la asignación de responsabilidad en parámetros útiles para el diseño del producto.
Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos expresados en forma de patrones. [12].

Entre los más utilizados se encuentran:

Patrón experto: es el principio básico de asignación de responsabilidades. Indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para ejecutar la tarea. Refuerza el encapsulamiento y esto redunda en bajo acoplamiento. [12].

Patrón creador: guía la asignación de responsabilidades relacionadas con la creación de objetos, de forma tal que una instancia de un objeto solo pueda ser creada por el objeto que contiene la información necesaria para ello.

[4] **API**: Es la abreviatura de Aplication Programming Interface. Un API no es más que una serie de servicios o funciones que se le ofrece al programador.

El uso de este patrón admite crear las dependencias mínimas necesarias entre las clases, lo que beneficia el mantenimiento del sistema y se brindan oportunidades de reutilización. [12].

Patrón controlador: es un intermediario entre la interfaz de usuario y el núcleo de las clases donde reside la lógica de la aplicación. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación para aumentar la reutilización de código y a la vez tener un mayor control. [12].

Patrón bajo acoplamiento: el acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con qué las conoce y con qué recurre a ellas. Asignar una responsabilidad para mantener bajo acoplamiento, menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Reduce el impacto de los cambios y permite crear clases más independientes, más reutilizables, lo que implica mayor productividad. [12].

Patrón alta cohesión: la principal característica de este patrón es asignar responsabilidades, de modo que la cohesión siga siendo alta. La información que almacena una clase debe de ser coherente y debe estar en la medida de lo posible relacionada con la clase. [17].

1.8.2 Patrones estructurales Gang of Four. (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. [18].

Facade (Fachada): el objetivo de este patrón es proveer un intermediario entre dos grupos de objetos, disminuyendo el número de objetos con los que trabaja un cliente y simplificando el acceso de éste a un conjunto de objetos relacionados. Se promueve un acoplamiento débil entre el subsistema y sus clientes, eliminándose o reduciéndose las dependencias. [18].

Patrón de Comportamiento: estudia las relaciones de llamadas entre los diferentes objetos, proporcionan estrategias comprobadas para modelar la manera en que los objetos colaboran entre sí en un sistema. [18].

Patrón de Acceso a Datos (DAO): plantea como solución, utilizar un Data Access Object (DAO) para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos. Actúa como un adaptador entre el componente y la fuente de datos. [18]

Composite View (Vistas Compuestas): gestiona los diferentes elementos de la vista por medio de una plantilla que hace la representación de las vistas más manejable. Por ejemplo, una página JSP que incluye otras páginas JSP y HTML usando la directiva include. La aplicación de este patrón hace que la representación de vistas sea más manejable gestionando los diferentes elementos de una página por medio de una plantilla. [7].

MVC (Modelo Vista Controlador): este patrón propone dividir la aplicación en tres capas el Modelo, la Vista y el Controlador, el modelo es la representación del dominio o datos del sistema, la vista se encarga de presentar la interfaz al usuario, en sistemas web, esto es típicamente HTML, aunque pueden existir otro tipo de formatos. En la vista sólo se deben de hacer operaciones simples y el controlador es el encargado de escuchar los cambios en la vista y enviarlos al modelo, el cual le regresa los datos a la vista como un ciclo. [7].

1.9 Conclusiones del capítulo

Para la búsqueda de una solución a los problemas actuales en la emisión y negociación de las cartas de crédito en el BNC, fue necesario realizar un estudio de los sistemas contables existentes tanto a nivel nacional como internacional que incluyen el manejo de las cartas de crédito dentro de sus funcionalidades, concluyéndose que los mismos no permiten solucionar el problema a resolver.

Se caracterizaron las tecnologías, notaciones, técnicas, metodologías, herramientas CASE y otras herramientas de desarrollo definidos por la dirección del proyecto SAGEB, sobre la premisa del desarrollo de aplicaciones cubanas utilizando software libre y teniendo en cuenta las peticiones del cliente, sentando las bases teóricas necesarias para comenzar el diseño de la solución.

CAPÍTULO 2: DISEÑO DE LA SOLUCIÓN

2.1 Introducción

El presente capítulo enmarca su contenido en explicar la arquitectura del sistema Quarxo, la cual responde a las necesidades planteadas en el proyecto de modernización del sistema bancario nacional. Partiendo de aquí, se derivarán un grupo de artefactos que responden al flujo de trabajo Diseño de la metodología utilizada, los cuales son claves para el correcto desarrollo de las etapas venideras, como son: modelo de diseño, diagrama de paquetes, diagrama de clases, el modelo de datos, así como los diagramas de interacción de las funcionalidades más significativas.

2.2 Descripción de la arquitectura

La arquitectura es el resultado de acoplar elementos arquitectónicos de forma apropiada para satisfacer los requerimientos funcionales y no funcionales de un sistema. Por lo general se adopta una arquitectura conocida en función de sus características, ventajas y desventajas. Entre las más universales se encuentra la Arquitectura de tres niveles, en la cual, como lo indica su nombre, la carga se divide en tres capas: una capa para la presentación (interfaz de usuario), otra para el negocio y otra para el almacenamiento de información (acceso a datos). [22].

La arquitectura que se propuso en el proyecto SAGEB para el sistema Quarxo y sobre la que se realiza el desarrollo del subsistema Cartas de Crédito está compuesta por tres capas arquitectónicas:

- ✓ Capa de Presentación.
- ✓ Capa de Negocio.
- ✓ Capa de Acceso a Datos.

Además se utiliza la capa transversal de Dominio para aquellos objetos del dominio que están presente en el resto de las capas. [22].

Dicha arquitectura fue basada en la complejidad de los procesos y la relación entre ellos, por lo que se organizó de forma jerárquica por subsistemas, módulos y componentes; quedando estructurado de la siguiente forma:

Subsistema: conjunto de módulos relacionados con los procesos que ejecutan.

Módulo: conjunto de casos de uso relacionados con uno o más procesos bancarios estrechamente relacionados.

Componentes: conjunto de funcionalidades comunes que son reutilizados por el resto de los módulos del sistema.

A continuación se muestra una representación de las capas lógicas:

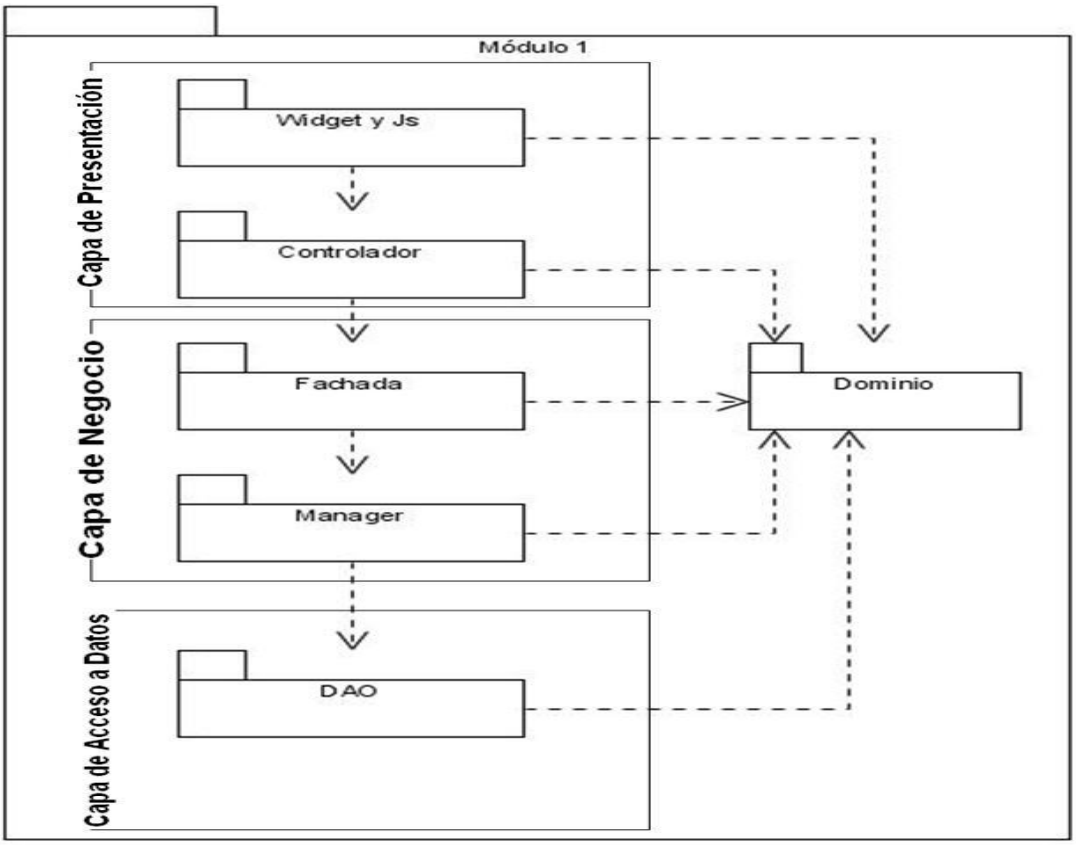

Ilustración 1: Capas lógicas de la arquitectura base de Quarxo. [22]

2.2.1 Arquitectura de la capa de presentación

En esta capa se desarrolla la lógica de presentación, en la cual para recibir, controlar y enviar una respuesta desde el cliente se utiliza el Spring MVC. Para hacer menos complejos los flujos de presentación y lograr una buena reutilización de ellos en los diferentes módulos se utiliza Spring WebFlow con el objetivo de representarlos y controlarlos. Para la generación de las interfaces que interactúan con el usuario se usa la librería Dojo. Esta capa interactúa con la capa de negocio y la capa de dominio.

2.2.2 Arquitectura de la capa de negocio

La arquitectura de la capa de negocio del sistema Quarxo está compuesta por dos subcapas:

- ✓ La subcapa Fachada que es el puente intermediario entre la capa de presentación y la del negocio, donde su función es agrupar las funcionalidades que son invocadas desde la capa de presentación, sin realizar la lógica del negocio.

- ✓ La subcapa Manager donde se realiza la lógica del negocio conteniendo la jerarquía de clases indicadas para su implementación. Esta subcapa hace uso de la capa de acceso a datos para el trabajo con la persistencia de datos y de la capa de dominio para la generación de los objetos del dominio.

2.2.3 Arquitectura de la capa de acceso a datos

Las operaciones que permiten el acceso a los datos de la aplicación se encuentran en esta capa, la cual ejerce la conexión con el gestor de base de datos, permitiendo así la persistencia y el acceso a la información.
La interacción con la capa de negocio se realiza a través de interfaces. Se utiliza el patrón DAO para el desarrollo de la capa, el framework de persistencia Hibernate y los módulos Spring ORM y Spring DAO.

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

2.3 Diseño del subsistema

Durante el desarrollo de un software el diseño se realiza con el fin de traducir los requisitos a una estructura que describe cómo implementar el sistema, siendo lo suficientemente específica para que no existan ambigüedades, y ofreciendo la posibilidad de descomponer los trabajos de implementación en componentes más manejables, visualizándolos mediante una notación común. [11].

A continuación se presentan los requisitos funcionales del subsistema Cartas de Crédito que como parte de la segunda fase de Quarxo se deben desarrollar:

Acuerdo: El módulo tiene como objetivo principal permitir a las Direcciones de Gestión de Negocios gestionar los acuerdos tomados donde se establecen todas las condiciones y procedimientos para tramitar las operaciones bajo financiamiento.

- ✓ Actualizar acuerdo.

Capacidad Financiera: El módulo tiene como objetivo principal el control de las operaciones que se realizan con respaldo de la Cuenta Única.

- ✓ Registrar cuenta de capacidad financiera.
- ✓ Eliminar cuenta de capacidad financiera.
- ✓ Consultar cuenta de capacidad financiera.
- ✓ Buscar cuenta de capacidad financiera.
- ✓ Actualizar capacidad financiera.
- ✓ Eliminar capacidad financiera.
- ✓ Registrar reservación de capacidad financiera.
- ✓ Consultar reservación de capacidad financiera.

Emisión: El módulo tiene como objetivo principal el control de las cartas de créditos.

- ✓ Consultar carta de crédito.
- ✓ Actualizar carta de crédito.
- ✓ Autorizar carta de crédito.
- ✓ Emitir carta de crédito.

Negociación: El módulo tiene como objetivo principal el control de las negociaciones que se realizan sobre las cartas de créditos al recibo de los documentos de la mercancía.

- ✓ Cancelar negociación de carta de crédito.
- ✓ Registrar negociación de carta de crédito.

- ✓ Unir negociación de carta de crédito.

Discrepancias: El módulo tiene como objetivo principal darle tratamiento a aquellos documentos de embarques que presentan problemas de cualquier índole.

- ✓ Cancelar discrepancia.
- ✓ Enviar carta de discrepancia.

2.4 Artefactos de entrada al diseño

El análisis en el proceso de desarrollo de software es una etapa decisiva, y uno de sus propósitos primarios es definir las clases y paquetes que permiten el cumplimiento de todos los requisitos que deben ser cumplidos. Las primeras pautas para la implementación del sistema se definen en esta fase.

En el análisis se razona más sobre los aspectos internos del sistema, se estructuran los requisitos, de manera que facilite su comprensión, su preparación, su modificación, y en general su mantenimiento. [11].

Se trabajaron con artefactos de entrada como:

Diagrama de casos de uso del sistema

Representa como ocurren los procesos en el sistema. Consiste en tener una buena organización en cuanto a la relación actor-caso de uso. [11]. A continuación se muestra el modelo de casos de uso del subsistema a partir de las nuevas funcionalidades como parte de la segunda fase de Quarxo.

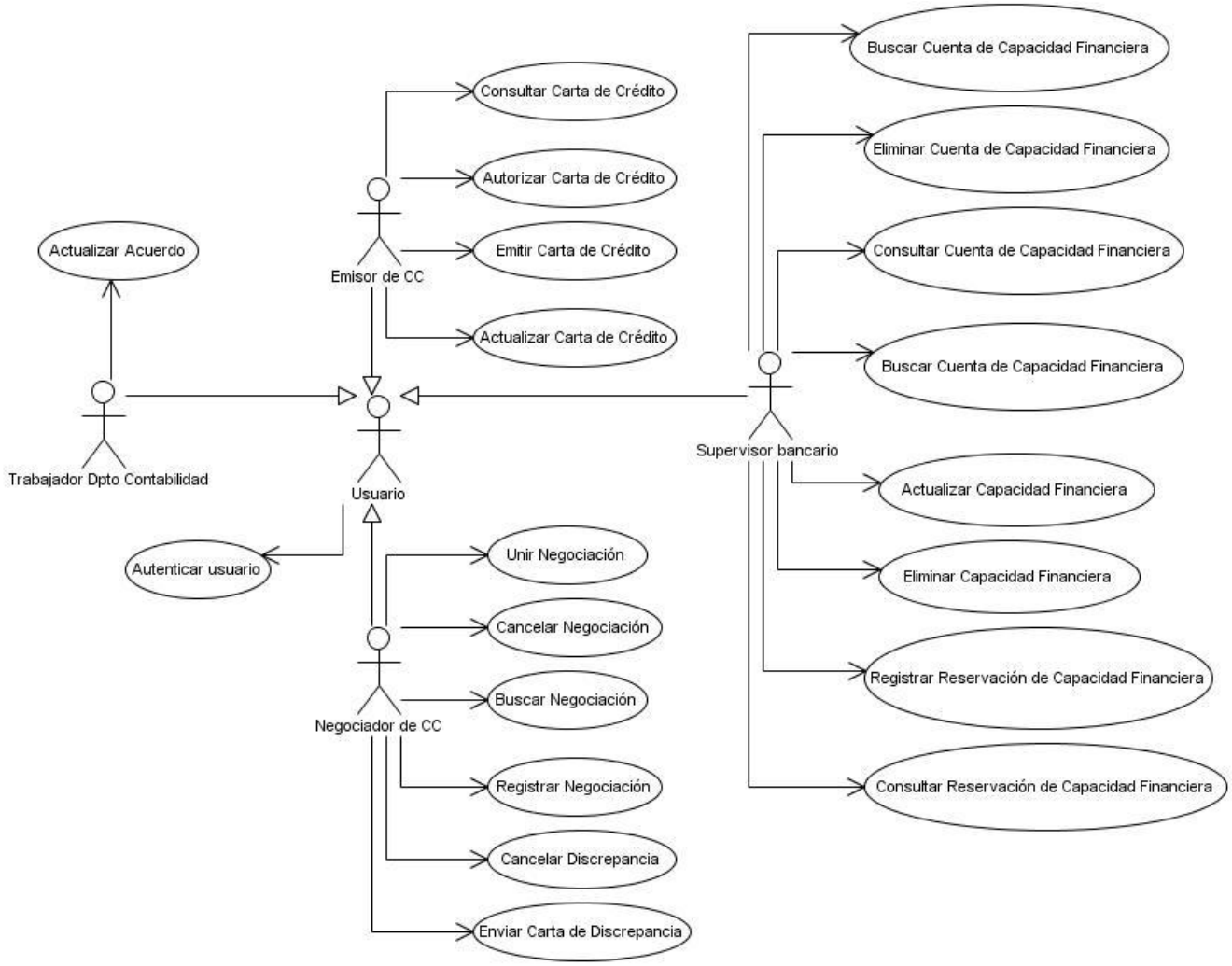

Ilustración 2: Diagrama de casos de uso del sistema.

2.4.1 Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales junto a otro grupo de restricciones relacionadas con el entorno de implementación tienen impacto en el sistema, está compuesto por otros artefactos como los diagramas de paquetes y diagramas de clases, además sirve como abstracción a la implementación y se empleará como entrada fundamental de las actividades de esta etapa. [11].

2.4.1.1 Diagrama de paquetes

Un paquete de diseño es usado para estructurar el modelo de diseño mediante su división en partes más pequeñas. Los paquetes son usados principalmente para la administración de configuración y organización del modelo. [11].

A continuación se muestra la estructura y dependencia de paquetes del módulo Negociación del subsistema Cartas de Crédito y una explicación de la composición de cada uno de los paquetes que lo conforman. De igual manera se comporta para el resto de los módulos del subsistema.

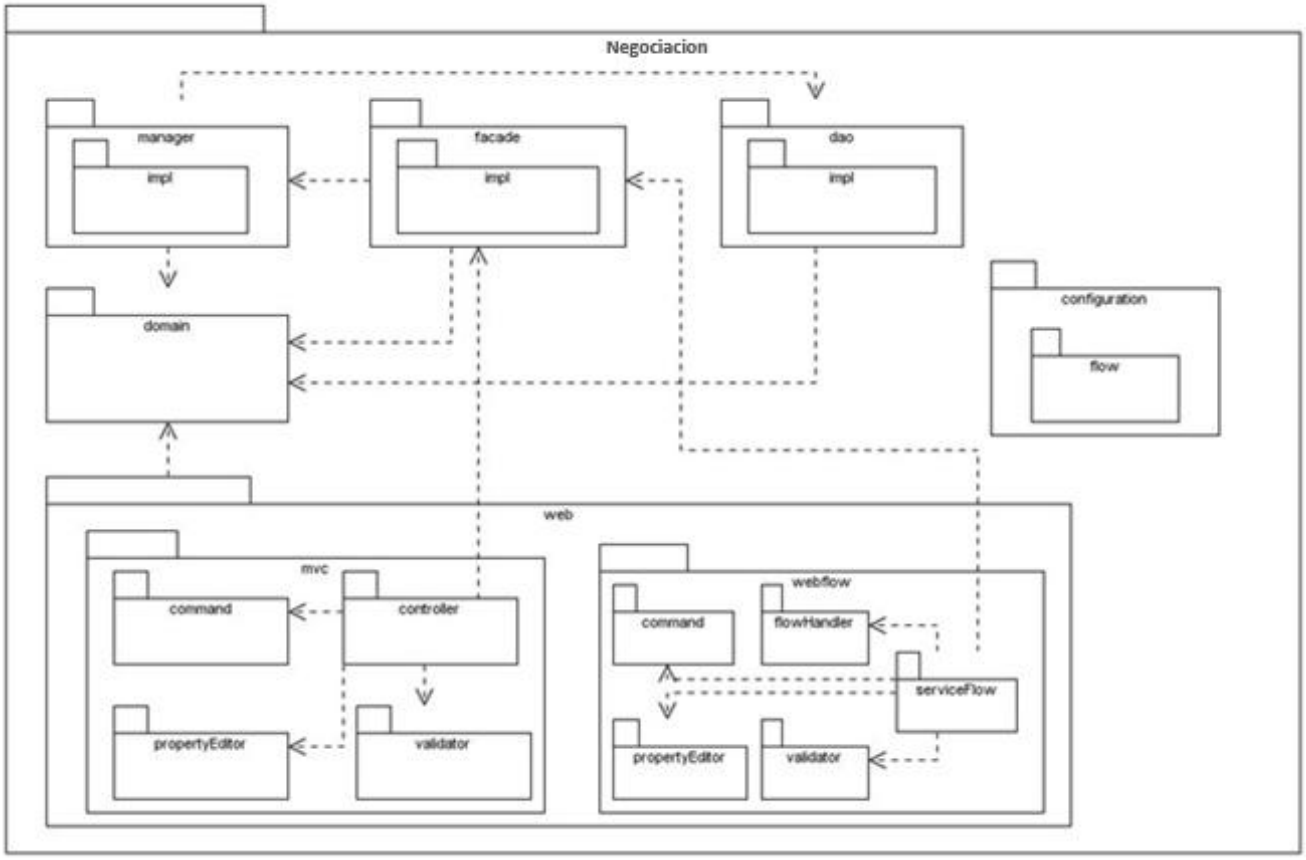

Ilustración 3: Diagrama de paquetes del módulo Negociación del subsistema Cartas de Crédito.

Descripción de cada uno de los paquetes utilizados

Paquete configuration: contiene los ficheros de configuración en formato XML de los diferentes contextos de Spring para el módulo, son estos:

- ✓ -servlet.xml: define el contexto de Spring MVC.
- ✓ -bussiness.xml: define el contexto para el negocio.
- ✓ -webflow.xml: define el contexto para Spring WebFlow.

- ✓ -dataaccess.xml: define el contexto para acceso a datos.

Paquete flow: contiene los archivos XML que definen los flujos para Spring WebFlow.

Paquete facade e impl: contiene la interfaz y su respectiva implementación, encargadas de brindar las funcionalidades que serán usadas por la presentación, así como las que se consumen por otros módulos.

Paquete manager e impl: contiene las interfaces y las implementaciones que comprenden la lógica de negocio que existe en el módulo, que será brindada a las capas superiores.

Paquete dao e impl: contiene 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: contiene las clases relacionadas con el dominio del módulo en cuestión.

Paquete web: contiene 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: contiene todo lo referente a la lógica de presentación cuando se hace uso de Spring MVC.

Paquete webflow: contiene todo lo referente a la lógica de presentación cuando se hace uso de Spring WebFlow.

Paquete controller: contiene las clases que extienden de los controladores propuestos por Spring MVC, encargadas de responder a las peticiones realizadas por el cliente.

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

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

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

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

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

2.4.2. Diagrama de clases del diseño

Durante el diseño, el diagrama de clases se elabora para tener en cuenta los detalles concretos de la implementación del sistema. [11].

En la capa de presentación se hace uso de Spring MVC y Spring WebFlow. En el subsistema Cartas de Crédito, en dependencia de la complejidad de cada módulo, se aplicó el framework correspondiente.

En el caso del módulo Acuerdo sólo se hizo uso de Spring MVC para el caso de uso que no contemplan la contabilización, mientras que para los módulos de Capacidad Financiera, Emisión, Discrepancia y Negociación se hizo necesario aplicar Spring MVC y Spring WebFlow que si contemplan la contabilización siendo un proceso con flujos complejos. Se define el siguiente diagrama de clases del diseño para el módulo Negociación.

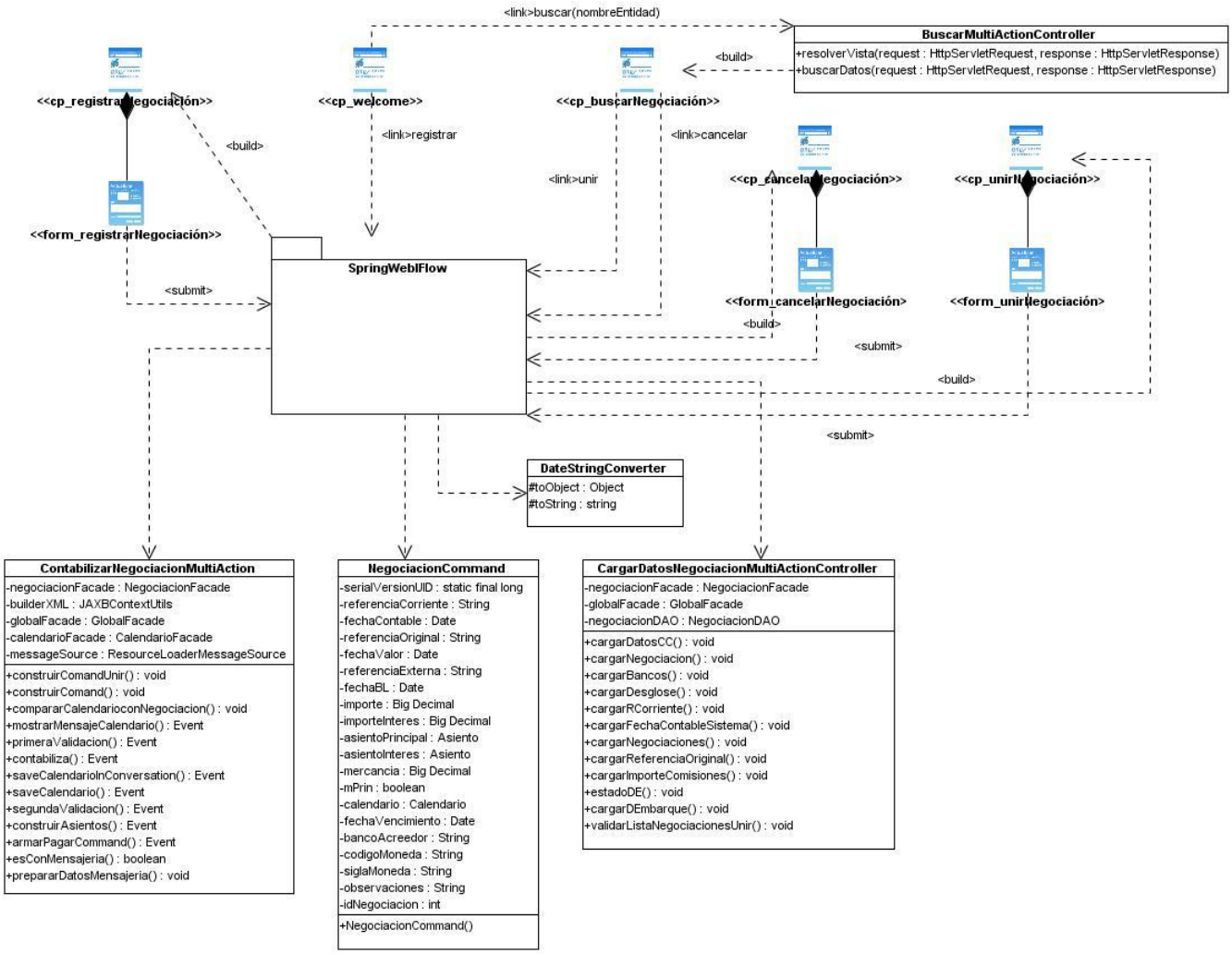

Ilustración 4: Diseño de la capa de presentación del módulo Negociación (Spring Webflow).

Las clases **<<cp_registrarNegociación>>**, **<<cp_cancelarNegociación>>** y **<<cp_unirNegociación>>** son las páginas web encargadas de mostrar los formularios de información al usuario. El paquete **SpringWebFlow** representa el mecanismo interno con el que Spring WebFlow gestiona las peticiones realizadas desde el cliente y mediante el cual se controlan los flujos por los que son guiados los usuarios en el sistema, la clase **ContabilizarNegociacionMultiAction** es la encargada de gestionar las acciones asociadas a los flujos de registrar, unir y cancelar, Spring WebFlow interactúa con ella para su funcionamiento.

2.4.3 Diseño de la capa de negocio

Los módulos del subsistema Cartas de Crédito tienen una fachada, la cual es la encargada de brindar las funcionalidades a la capa de presentación, de esta forma dicha capa no conocerá la verdadera implementación de los métodos. La fachada de un módulo puede interactuar con otros módulos o subsistemas a través de las fachadas de los mismos.

Las clases manager contienen toda la lógica del negocio de la aplicación y es donde se encuentran declaradas las clases que componen la capa de negocio; en la clase NegociacionManagerImpl se encuentran implementadas las funcionalidades para solucionar el negocio.

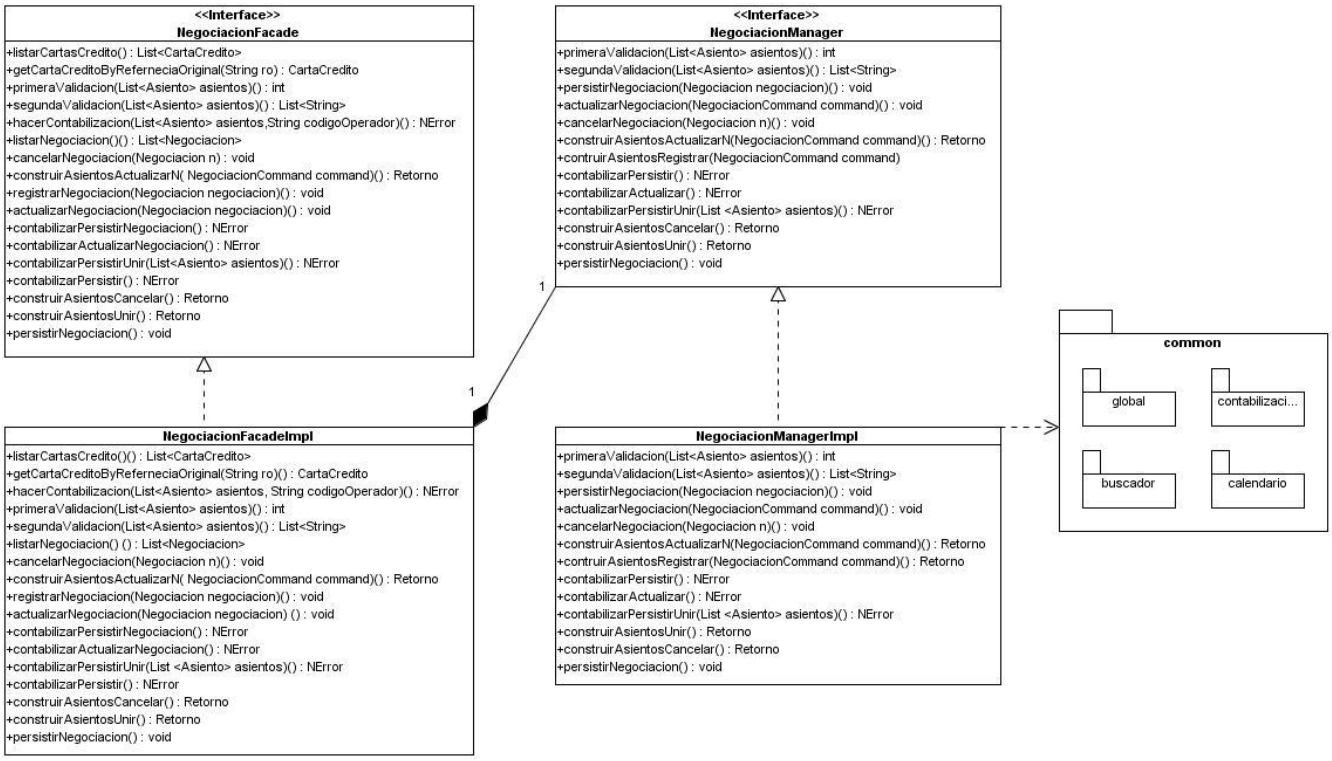

Ilustración 5: Diseño de la capa de negocio del módulo Negociación.

En el diseño de la capa de negocio, la clase **NegociacionFacade** es la interfaz encargada de brindar las funcionalidades del módulo a la capa de presentación y a otros módulos que la requieran, la clase **NegociacionFacadeImpl** es la encargada de implementar la lógica de programación de la interfaz NegociacionFacade, la clase **NegociacionManager** contiene las funcionalidades que se deben implementar en la clase **NegociacionManagerImpl** para dar respuesta a las acciones que se solicitan desde NegociacionFacadeImpl.

El paquete **common** contiene los componentes y estos a su vez las funcionalidades que son comunes a los diferentes subsistemas del sistema Quarxo.

2.4.4 Diseño de la capa de acceso a datos

En dicha capa se hace uso del patrón DAO y de los componentes DAO genéricos utilizados en la arquitectura del proyecto, compuesto por la interface BaseDAO y la clase AbstractBaseDAO. Estas clases ofrecen funcionalidades básicas a realizar con las clases persistentes como son: persistir, actualizar, listar, buscar por identificador, eliminar.

En el siguiente diagrama están declaradas e implementadas las funcionalidades encargadas de interactuar con la base de datos:

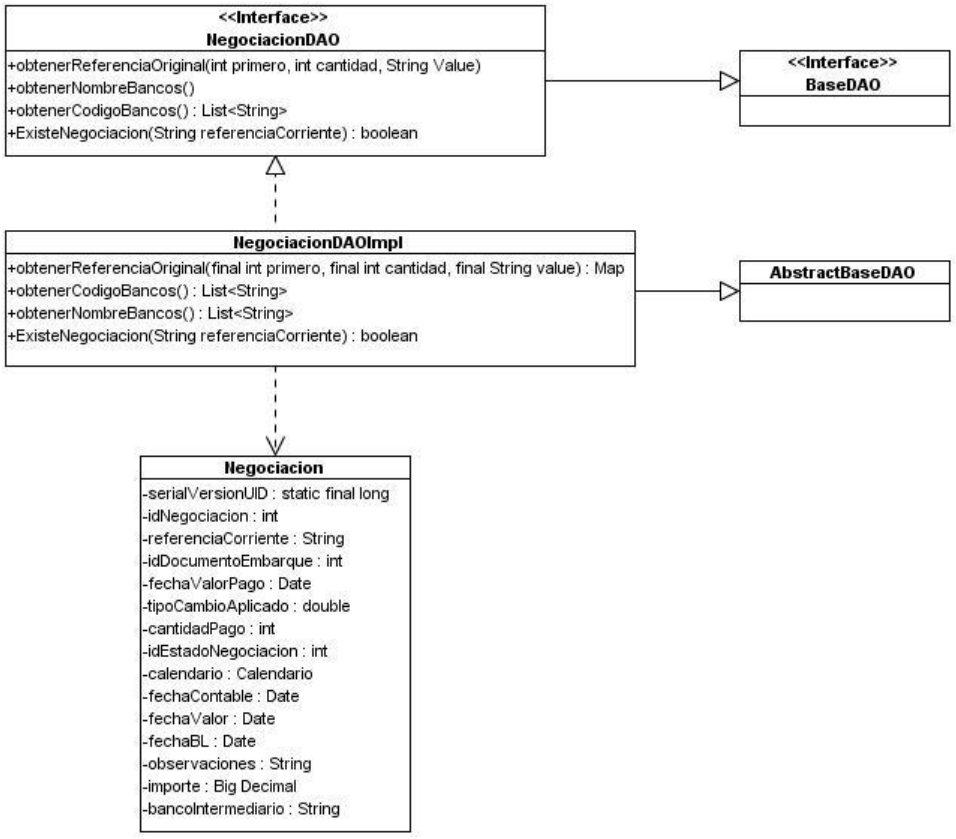

Ilustración 6: Diseño de la capa de acceso a datos del módulo Negociación.

En la capa de acceso a datos, la clase **NegociacionDAO** es la interfaz de comunicación que se encarga de brindar las funcionalidades de la capa de acceso a datos, la clase **NegociacionDAOImpl** implementa las funcionalidades de esta capa, la clase **Negociacion** representa la información que se persiste en la base de datos asociada a las negociaciones, las clases BaseDAO y AbstractBaseDAO son respectivamente una interfaz y una clase abstracta que brindan un conjunto de funcionalidades básicas que se realizan con las clases persistentes, ya sea buscar por identificador, listar, persistir, actualizar y eliminar. Básicamente solo se necesita que las interfaces del acceso a datos implementen BaseDAO y las implementaciones hereden de AbstractBaseDAO.

2.5 Diagramas de interacción

Los diagramas de interacción capturan el comportamiento de los casos de uso mostrando 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. El primero muestra una interacción ordenada según la secuencia temporal de eventos, mientras que el segundo muestra una interacción organizada basándose en los objetos que toman parte en la interacción y los enlaces entre los mismos. [11]. Aunque ambos enfatizan aspectos particulares trabajan sobre la misma información, de manera que se contemplan solamente los diagramas de secuencia.

Como caso de estudio para mostrar en el capítulo se selecciona del módulo Negociación el caso de uso **Cancelar Negociación** y **Unir Negociación**. Como la mayoría de las funcionalidades parten del flujo del buscador de Negociación (implementado en la primera fase de Quarxo), se muestra el diagrama de secuencia del mismo y seguidamente el de los casos de uso Cancelar Negociación y Unir Negociación.

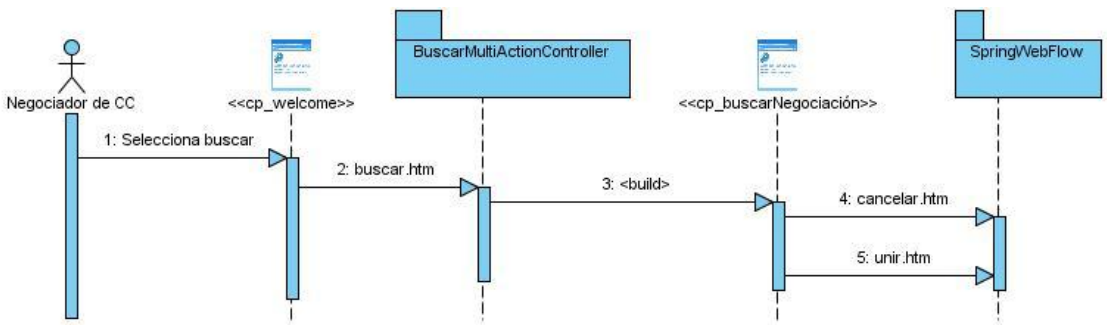

Ilustración 7: Diagrama de secuencia del buscador de Negociación.

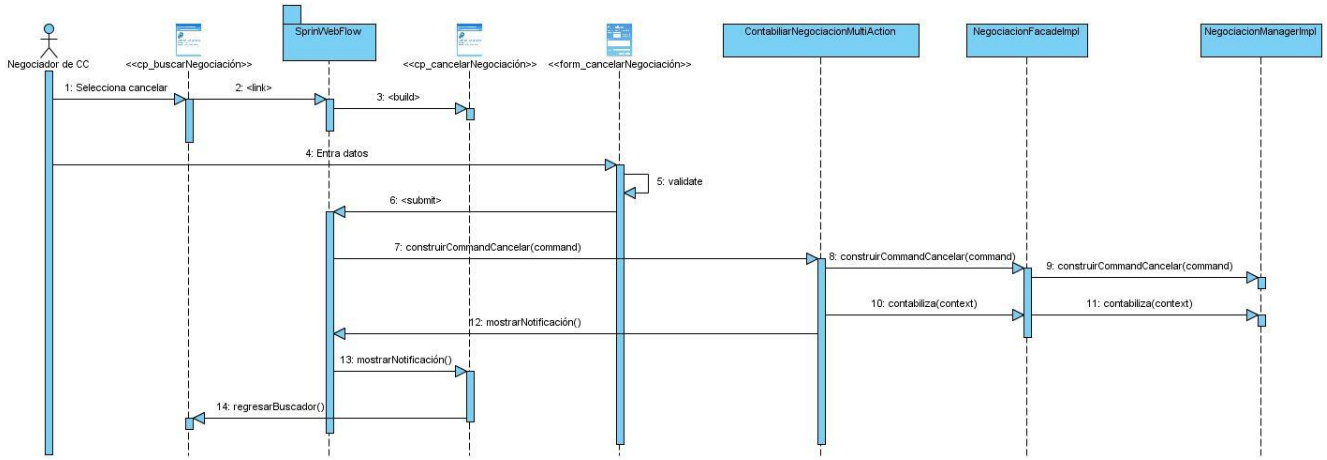

Ilustración 8: Diagrama de secuencia de Cancelar Negociación.

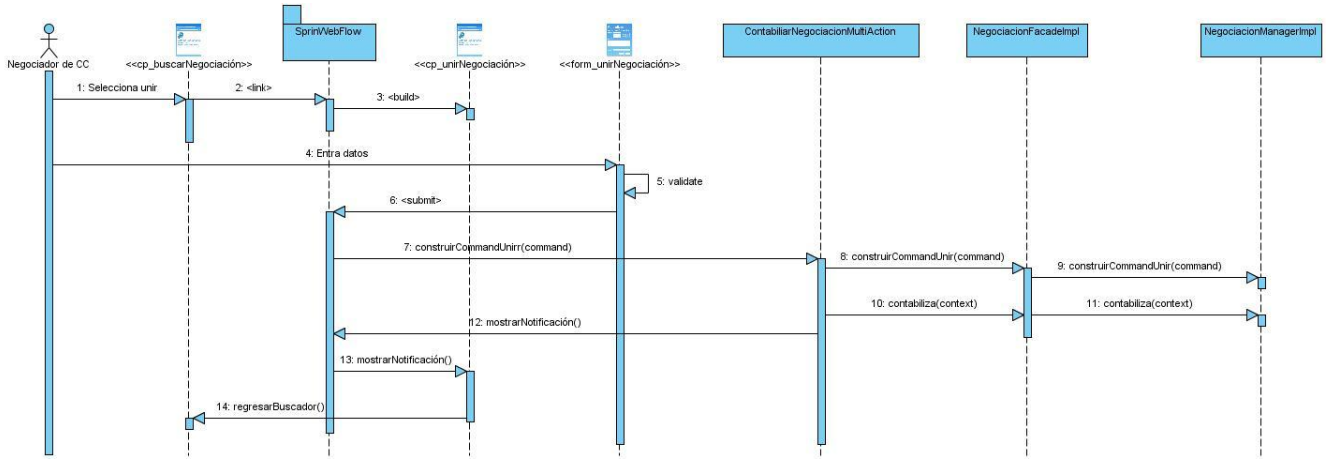

Ilustración 9: Diagrama de secuencia de Unir Negociación.

2.6 Modelo de datos

Un modelo de datos describe la representación lógica y física de los datos persistentes manejados por el sistema. Describe la estructura de la base de datos, así como los datos, sus relaciones y las restricciones que deben cumplirse entre ellos, la cual garantiza el almacenamiento y la recuperación de la información de manera adecuada además de cumplir las restricciones identificadas por el sistema.

El modelo de datos puede recibir cambios mientras transcurra el flujo de implementación si se detecta la necesidad de persistir nuevos elementos o se decida la utilización de procedimientos almacenados, vistas, restricciones y/o funciones definidas en el gestor de base de datos.

Es necesario resaltar que el subsistema Carta de Crédito forma parte del sistema Quarxo desarrollado sobre el núcleo del SABIC, el cual es un sistema que ya posee una base de datos, sin embargo sólo se muestra el modelo de datos que responde al subsistema en cuestión.

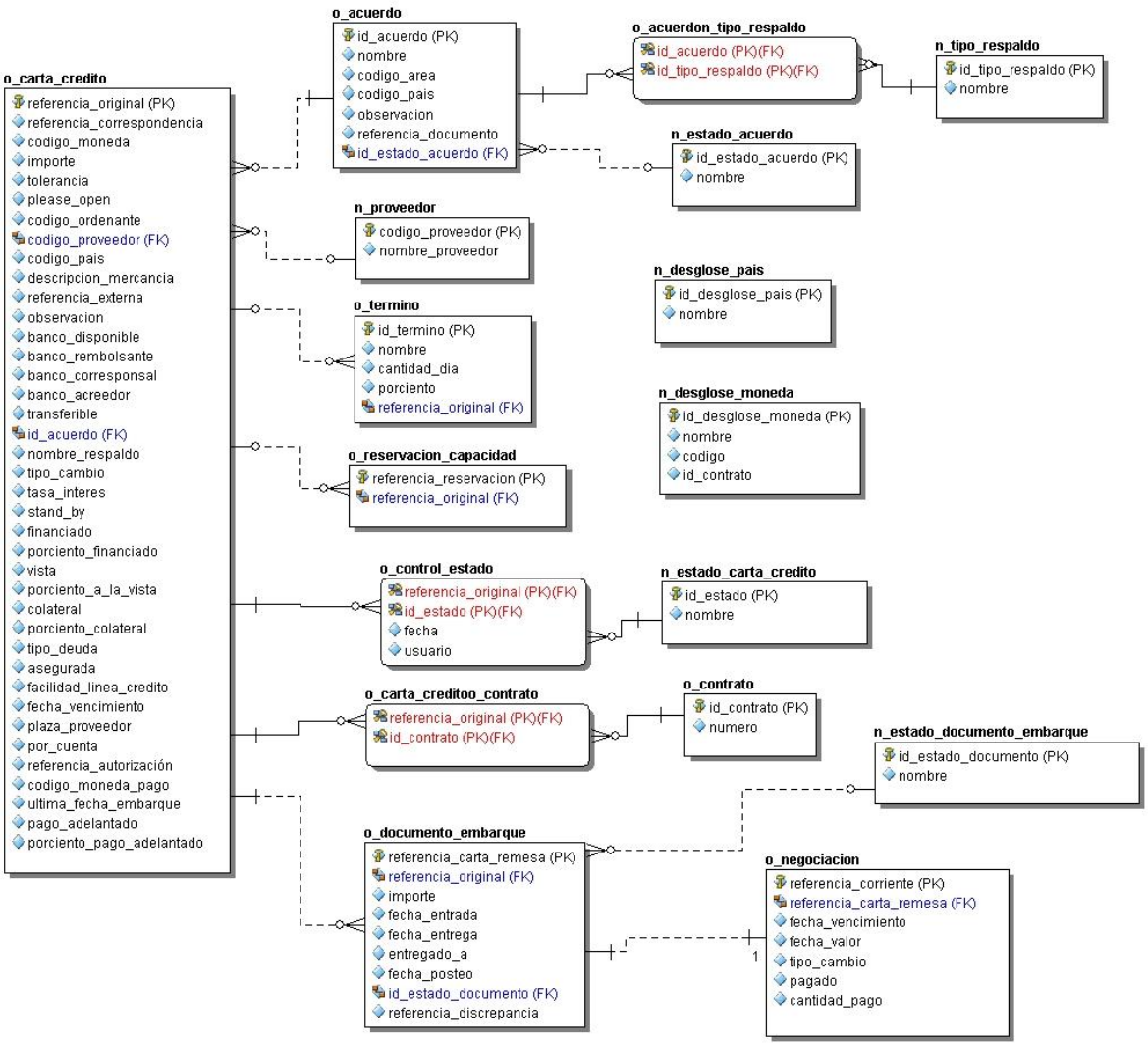

Ilustración 10: Modelo de datos del subsistema Cartas de Crédito.

La tabla principal operativa **o_carta_credito** es donde se almacenan los datos de la carta de crédito una vez que es registrada en el sistema. Esta a su vez posee una relación con la tabla **o_documento_embarque** encargada de almacenar la información de los documentos de embarques una vez llegados al puerto.

El nomenclador **n_estado_documento_embarque** indica el estado del mismo (Registrado, Discrepado y Negociado).
La tabla **o_negociacion** almacena los datos de las negociaciones de las cartas de crédito una vez que son contabilizadas.

2.7 Patrones de diseño empleados

Los patrones de diseño de software son una solución probada para un problema general de diseño, en un contexto determinado. Encierran la experiencia de los programadores e ingenieros que han ido adquiriendo en las soluciones de problemas comunes.

A continuación se describen un conjunto de patrones de diseño que son empleados en el subsistema Cartas de Crédito, contribuyendo a su implementación organizada y eficiente.

En este caso se emplearon los patrones GRASP, los cuales describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Los patrones GRASP que se utilizaron son los siguientes:

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

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

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

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

Durante el diseño del subsistema Cartas de Crédito se emplearon también los patrones GoF, específicamente:

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

- ✓ **Patrón de Acceso a Datos (DAO):** La utilización de este patrón se evidencia en la utilización de clases con la función de servir como intermediarias entre las clases del negocio y la fuente de datos, entre las que se encuentra NegociacionDAOImpl.

- ✓ **Composite View (Vistas Compuestas):** El uso de este patrón se evidencia en todos los ficheros JSP de los módulos con la utilización de la etiqueta <jsp: include> para incluir líneas de código separados en otros archivos ya que son comunes para varias páginas.

- ✓ **MVC (Modelo Vista Controlador):** Este patrón propone dividir la aplicación en tres capas el Modelo, la Vista y el Controlador, el modelo es la representación del dominio o datos del sistema, la vista se encarga de presentar la interfaz al usuario, en sistemas web, esto es típicamente HyperText Markup Language (HTML), aunque pueden existir otro tipo de formatos.
  En la vista sólo se deben de hacer operaciones simples y el controlador es el encargado de escuchar los cambios en la vista y enviarlos al modelo, el cual le regresa los datos a la vista como un ciclo. Cuando se aplica este patrón los accesos a la base de datos se hacen en el modelo, la vista y el controlador no deben de saber si se usa o no una base de datos. El controlador es el que decide que vista se debe de imprimir y que información es la que se envía. [17].

2.8 Validación del diseño

El diseño está enfocado a convertir los requisitos del cliente en un modelo que al ser implementado, se obtenga el producto deseado. Generalmente constituye el punto de partida para el desarrollo de software una vez especificados los requerimientos del sistema. Evidentemente, realizar una validación del mismo para verificar su calidad y flexibilidad, garantiza una buena base para la implementación.

Con este objetivo se utilizan un conjunto de métricas de software orientadas a determinar, entre otros aspectos, qué características del modelo de diseño se pueden estimar para comprobar que el sistema será fácil de implementar en cuanto a organización.

Para evaluar el diseño del subsistema Cartas de Crédito se utilizaron las métricas para sistemas orientados a objetos (OO). Estas métricas se han introducido para ayudar a un ingeniero del software a usar el análisis cuantitativo para evaluar la calidad en el diseño antes de que un sistema se construya. El enfoque de métricas OO está en la clase: la piedra fundamental de la arquitectura OO. [26].
Entre ellas se seleccionan las métricas Tamaño Operacional de Clase (TOC) y Relaciones entre Clases (RC) para validar el diseño del subsistema Cartas de Crédito. Sólo se emplearon estas métricas debido a que en la propuesta de diseño realizada no existen casos de herencia, lo que descarta la posibilidad de aplicar otras métricas.

2.8.1 Tamaño Operacional de Clase (TOC)

La métrica TOC se determina por el número total de operaciones que están encapsuladas dentro de la clase. Grandes valores de esta medida muestran que la clase puede tener demasiada responsabilidad, lo cual reducirá la reusabilidad de la misma y complicará su implementación.

Atributos de calidad que abarcan la métrica TOC:

- ✓ 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.

En la métrica TOC, los atributos de calidad Responsabilidad y Complejidad de implementación son inversamente proporcionales a la Reutilización, lo que puede ser traducido como, mientras mayor sea la Responsabilidad y Complejidad de implementación de una clase, menor será su nivel de Reutilización.

2.8.1.1 Resultados de la evaluación de la métrica TOC

Los valores umbrales para evaluar el diseño propuesto se muestran en la siguiente ilustración:

	Categoria	Criterio
Responsabilidad	Baja	< =Prom.
	Media	Entre Prom. y 2* PRom.
	Alta	> 2* Prom.
Complejidad implementación	Baja	< =Prom.
	Media	Entre Prom. y 2* Prom.
	Alta	> 2* Prom.
Reutilización	**Baja**	> 2*Prom.
	Media	Entre Prom. y 2* Prom.
	Alta	<= Prom.

Ilustración 11: Valores umbrales de la métrica TOC.

La aplicación de esta métrica demostró los siguientes resultados:

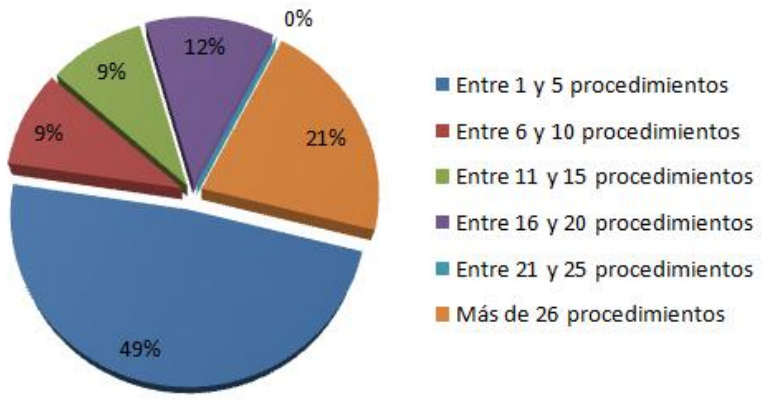

Ilustración 12: Gráfico que indica el % de clases según la cantidad de procedimientos.

Responsabilidad de las clases

Responsabilidad	Cantidad de clases
Baja	19
Media	5
Alta	9

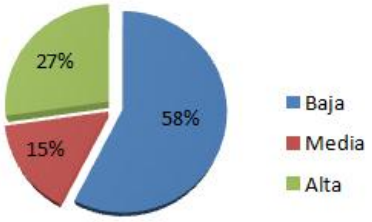

Ilustración 13: Resultados de la evaluación de la métrica TOC en el atributo Responsabilidad.

Complejidad de implementación de las clases

Complejidad	Cantidad de clases
Baja	19
Media	5
Alta	9

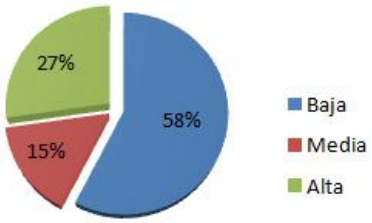

Ilustración 14: Resultados de la evaluación de la métrica TOC en el atributo Complejidad.

Reutilización de las clases

Reutilización	Cantidad de clases
Alta	19
Media	5
Baja	9

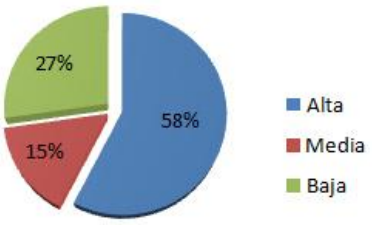

Ilustración 15: Resultados de la evaluación de la métrica TOC en el atributo Reutilización.

2.8.2 Relaciones entre Clases (RC)

La métrica RC se determina por la cantidad de relaciones existentes entre las clases contenidas en el diseño. El número de dependencias es directamente proporcional al nivel de acoplamiento, a la complejidad del mantenimiento y a la cantidad de pruebas a realizar sobre las clases, y es inversamente proporcional al grado de reutilización de las mismas.

Atributos de calidad que abarcan la métrica RC:

- ✓ Acoplamiento: Consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de reutilización.
- ✓ Complejidad de mantenimiento: Consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.
- ✓ Reutilización: Consiste en el grado de reutilización presente en una clase o estructura de clase dentro de un diseño de software.
- ✓ Cantidad de pruebas: Consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (unidad) del producto (componente, módulo, clase, conjunto de clases, etc.) diseñado.

2.8.2.1 Resultados de la evaluación de la métrica RC

La aplicación de esta métrica demostró los siguientes resultados:

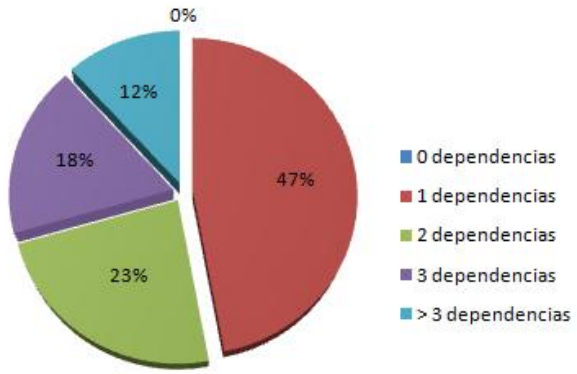

Ilustración 16: Gráfico que indica el % de clases según la cantidad de dependencias.

Acoplamiento

Acoplamiento	Cantidad de clases
Ninguno	0
Baja	8
Media	4
Alta	5

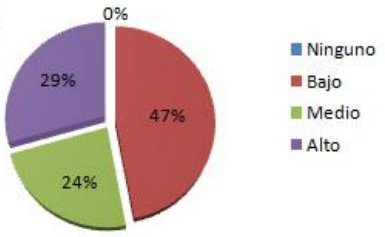

Ilustración 17: Resultados de la evaluación de la métrica RC en el atributo Acoplamiento.

Complejidad de Mantenimiento

Complejidad de Mantenimiento	Cantidad de clases
Baja	12
Media	5
Alta	0

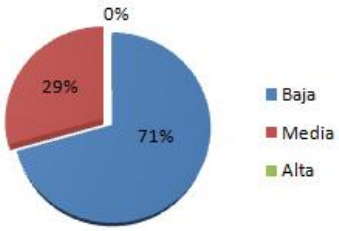

Ilustración 18: Resultados de la evaluación de la métrica RC en el atributo Complejidad.

Cantidad de Pruebas

Cantidad de Pruebas	Cantidad de clases
Baja	12
Media	5
Alta	0

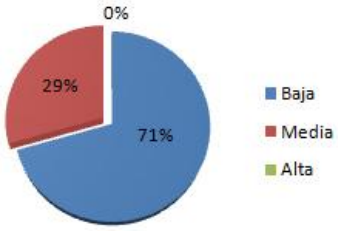

Ilustración 19: Resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.

Reutilización

Reutilización	Cantidad de clases
Baja	5
Media	4
Alta	8

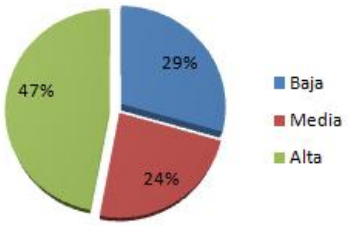

Ilustración 20: Resultados de la evaluación de la métrica RC en el atributo Reutilización.

2.8.3 Resultados de la evaluación de la métrica TOC y RC

La matriz de inferencia permite evaluar positiva o negativamente los resultados obtenidos de las relaciones atributo/métrica en una escala numérica, donde el valor 1 representa un resultado positivo y el valor 0 negativo.

Si la métrica no influye en el atributo de calidad la relación es considerada como nula y es representada con un guión simple (-). Al promediar por cada atributo las relaciones no nulas, se obtiene un resultado general que al ubicarse en el rango de evaluación (valor <=0.4: "Mala", valor >0.4 y valor<0.7: "Regular", valor>=0.7: "Buena") permite arribar a conclusiones sobre la calidad del diseño propuesto. Teniendo en cuenta que los resultados arrojados con la aplicación de las métricas fueron positivos para todos los atributos, la matriz de la inferencia queda elaborada de la siguiente manera:

Atributos\Métricas	TOC	RC	Promedio
Responsabilidad	1	(-)	1
Complejidad de implementación	1	(-)	1
Reutilización	1	1	1
Acoplamiento	(-)	1	1
Complejidad de Mantenimiento	(-)	1	1
Cantidad de pruebas	(-)	1	1

Tabla 1: Resultados de la evaluación de la relación atributo/métrica.

El promedio general obtenido es 1 luego de evaluar la relación atributo/métrica y teniendo en cuenta el rango de evaluación se concluye que el subsistema Cartas de Crédito presenta un diseño con buena calidad. Los atributos de calidad se encuentran en un nivel satisfactorio en las clases y se puede observar como existe una baja responsabilidad y complejidad de implementación en las clases, lo que promueve el bajo acoplamiento y facilitará la implementación y la comprobación al desarrollador. Esto permitirá promover la reutilización como elemento clave en el proceso de desarrollo de software.

2.9 Conclusiones del capítulo

En este capítulo se realizó el análisis de la arquitectura base definida por el sistema Quarxo y el diseño de los módulos del subsistema Cartas de Crédito mediante el correcto uso de patrones a partir de los cuales fue posible llevar a cabo un diseño flexible y escalable. Se diseñó un modelo de datos acorde con las tablas involucradas en la solución de los módulos. Finalmente el diseño fue comprobado a través de la aplicación de dos de las métricas de validación de diseño, sentando las bases para la implementación de las funcionalidades del subsistema.

CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN

3.1 Introducción

En el presente capítulo se explica y se presenta la implementación de los módulos del subsistema, modelándose de esta forma los artefactos pertenecientes al flujo de trabajo de Implementación 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 presentado en el capítulo anterior. Se muestran las clases que intervienen en los principales flujos de la implementación, así como los métodos más relevantes. Además se realizan las pruebas unitarias realizadas al sistema para dar cumplimiento a los requisitos especificados.

3.2 Modelo de implementación

La implementación es el principal flujo de trabajo en la fase de construcción. Describe cómo los elementos del modelo de diseño se implementan en términos de componentes. Está determinado por el lenguaje de programación y tiene como objetivo llevar a cabo la implementación de cada una de las clases significativas del diseño.

En este flujo de trabajo se define la organización del código en términos de los subsistemas de implementación organizados en capas. Se implementan los elementos del diseño en términos de implementación, obteniéndose los componentes necesarios para el funcionamiento de la aplicación así como el diagrama de componentes, que conforma lo que se conoce como un modelo de implementación al describir los componentes a construir, su organización y dependencia entre nodos físicos en los que funcionará la aplicación. [11].

3.3 Diagrama de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Permite modelar la vista estática del sistema mostrando las dependencias lógicas entre un conjunto de componentes de software.

Los componentes pueden ser de código fuente, librerías, binarios o ejecutables y tienen relaciones de traza con los elementos del modelo que implementan. [11].

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 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.

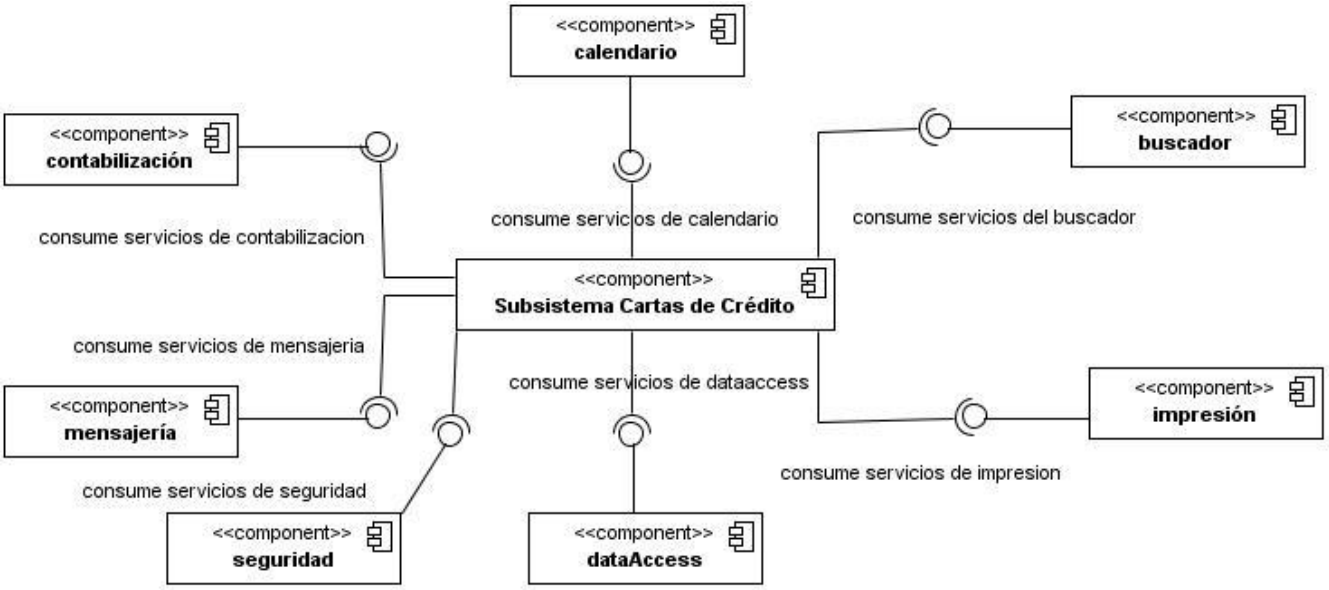

Ilustración 21: Modelo de componentes del subsistema Cartas de Créditos.

El componente **mensajería** contiene las funcionalidades que garantizan 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, se utiliza en la negociación y emisión de una carta de crédito. Por su parte, el componente **seguridad** gestiona la información de los usuarios, roles y permisos necesarios para lograr la confiabilidad requerida en el sistema. Es utilizado para controlar el acceso del usuario a las posibles operaciones a realizar sobre las cartas de crédito. El componente **contabilización** engloba el conjunto de funcionalidades que permiten la realización de dicho proceso; constituye el de mayor consumo por parte del subsistema.

En el caso del **dataAccess** su uso 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. La búsqueda fue posibilitada por la utilización del componente **buscador**. Para la impresión de información referente a la contabilidad se consumen los servicios de **impresión** y se hace uso del componente **calendario** para la gestión de las formas de pago que contienen los vencimientos.

3.4 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.

La generalización de aspectos tan simples como el trato de las mayúsculas, ayuda a eliminar conflictos de funcionalidades implementadas con nombres iguales y guían de forma clara el proceso de desarrollo.

Con el propósito de lograr una uniformidad en el desarrollo del sistema Quarxo, se definió una convención de código en cada una de las capas de la arquitectura definida.

3.4.1 Convenciones de nomenclatura

En el proyecto SAGEB se decidió definir para la nomenclatura de las clases la notación PascalCasing[5], la cual define que los nombres de las clases comienzan con la primera letra en mayúscula y el resto con minúscula, en caso de estar formado por palabras compuestas el inicio de cada palabra deberá tener mayúscula, se obvia el uso de artículos.

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

Los nombres de los métodos y los atributos de las clases, así como los nombres de ficheros de código javascript, sus funciones y variables internas comienzan con la primera letra en minúscula.

En caso de que sea un nombre compuesto se empleará notación CamelCasing[6].

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

[5] **PascalCasing:** Plantea que los identificadores y nombres de variables, métodos y funciones que están compuestos por múltiples palabras juntas, iniciarán cada palabra con letra mayúscula.

[6] **CamelCasing:** Plantea que los identificadores y nombres de variables, métodos y funciones que están compuestos por múltiples palabras juntas, iniciarán cada palabra con letra mayúscula excepto la primera palabra.

3.4.2 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: **CartaCreditoPropertyEditor**.

Paquete serviceFlow: [Nombre de la clase] + [Action o de MultiAction en dependencia de cuál de las dos clases herede]. Ejemplo: **ContabilizarNegociacionMultiAction**.

3.4.3 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:
Interfaces: [Nombre del módulo] + [Facade]. Ejemplo: **EmisionFacade**.
Y las clases que implementan las interfaces: [Nombre del módulo] + [FacadeImpl]. Ejemplo: **EmisionFacadeImpl**.

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

3.4.4 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:

Interfaces: [Nombre de la entidad] + [DAO]. Ejemplo: **NegociacionDAO**.

Y las clases que implementan las interfaces: [Nombre de la entidad] + [DAOImpl]. Ejemplo: **NegociacionDAOImpl**.

3.5 Descripción de las clases y las funcionalidades

Teniendo en cuenta el diseño de las clases correspondientes al subsistema Cartas de Crédito realizado en el capítulo anterior, se describirán las clases de mayor peso de los módulos para un mayor entendimiento de la implementación. De las clases que se encargan de manejar el flujo de cada módulo se describirá una, ya que tienen similar comportamiento, en este caso será del módulo Negociación la clase ContabilizarNegociacionMultiAction y la clase Negociacion que recoge la información necesaria de una negociación.

Nombre: ContabilizarNegociacionMultiActionController	
Tipo de clase: Controladora	
Atributo	**Tipo**
globalFacade	GlobalFacade
negociacionFacade	NegociacionFacade
calendarioFacade	CalendarioFacade
Para cada responsabilidad:	
Nombre	**Descripción**
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.
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.

segundaValidacion(RequestContext contex)	Invoca una segunda validación de los asientos e indica al flujo la presencia o no de mensajes de error o alerta.
construirAsientos(RequestContext contex)	Construye los asientos contables de la operación en curso.
esConMensajeria(RequestContext context)	Permite conocer si existe mensajería en la operación que se está realizando.
prepararDatosMensajeria(RequestContext context)	Permite conformar los datos que son necesarios para la mensajería.

Tabla 2: Descripción de la clase ContabilizarNegociacionMultiAction

Nombre: ContabilizarNegociacionMultiActionController	
Nombre	**Descripción**
checkError(RequestContext context, Object object)	Verifica si se produjo algún error en el proceso de mensajería.

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 3: Descripción de la clase Negociacion

Nombre: NegociacionDAOImpl	
Tipo de clase: Data Access Object	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**Descripción**:
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 4: Descripción de la clase NegociacionDAOImpl

3.6 Modelo de despliegue

El diagrama de despliegue describe la arquitectura física del sistema durante la ejecución en términos de procesadores, dispositivos y componentes de software. Describen la estructura de los elementos de hardware y el software que ejecuta cada uno de ellos. Es utilizado para capturar los elementos de configuración del procesamiento y las conexiones entre esos elementos. También se utiliza para visualizar la distribución de los componentes de software en los nodos físicos. La implementación produce un refinamiento de la vista de arquitectura del modelo de despliegue, donde los componentes ejecutables son asignados a nodos. [25].

El entorno de despliegue de la aplicación queda estructurado de la siguiente manera:

Se cuenta con aproximadamente 150 máquinas cliente con los programas instalados Mozilla Firefox 3.5 para acceder a la aplicación Quarxo, Adobe Reader, Microsoft Office y la Máquina virtual jdk 6.20; cada una está conectada a una impresora.

Los sistemas Quarxo, SLBTR y SISCOM se encuentran en el servidor de aplicaciones del BNC, que al igual que el servidor de Base de Datos posee como sistema operativo Windows Server 2003; requiere del contenedor web Apache Tomcat 6.0, la Máquina Virtual de Java y el Microsoft Office.

El servidor de base de datos donde se encuentran el núcleo del SABIC y el de SISCOM requiere del gestor de base de datos Microsoft SQL Server 2005.

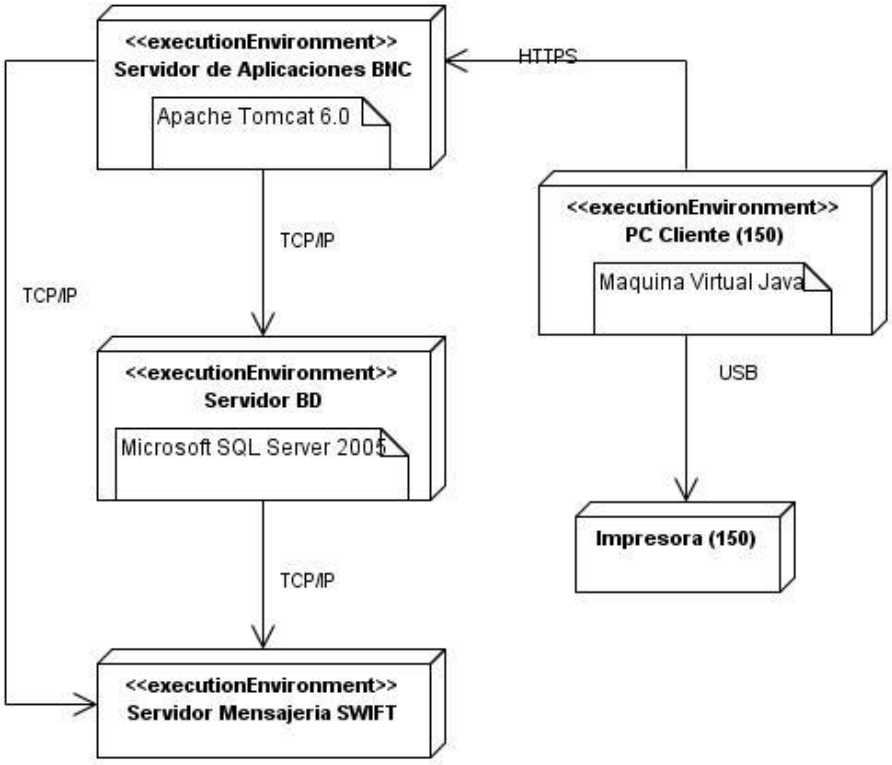

Ilustración 22: Modelo de Despliegue de Quarxo.

3.7 Validación de la solución

Las pruebas en un proceso de desarrollo de software son los diferentes procesos que se deben realizar con el objetivo de asegurar la terminación y calidad de la solución propuesta.

Existen diferentes métodos de pruebas, entre ellos se pueden encontrar:

- ✓ Las pruebas de caja negra: verifican las especificaciones funcionales sin tener en cuenta la estructura interna del programa y son realizadas sin el conocimiento interno del producto. El objetivo es demostrar que las funciones del software son operativas, que las entradas se aceptan de forma adecuada y se produce un resultado correcto, además que la integridad de la información externa se mantiene, saber qué es lo que hace el software pero sin entrar en detalles de código, es decir, que es lo que hace, y no cómo lo hace. Por ello se realizan sobre la interfaz del sistema controlando los datos de entrada y de salida.
- ✓ Las pruebas de caja blanca: denominadas pruebas de caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba.

En estas pruebas se examina el programa en varios puntos sobre el código para determinar si el estado real coincide con el esperado. Así como las pruebas de caja negra ejercitan los requisitos funcionales desde el exterior del módulo, las de caja blanca están dirigidas a las funciones internas. Estas pruebas se llevan a cabo en primer lugar sobre un módulo concreto, para luego realizar las de caja negra sobre varios subsistemas. El objetivo de las pruebas es la detección de defectos en el software (descubrir un error es el éxito de una prueba). Con el diseño de las pruebas se pretende encontrar y documentar los defectos que puedan afectar la calidad del software, asegurar que el software trabaje como fue diseñado, además de validar que los requisitos fueron implementados correctamente. [26].

3.7.1 Pruebas de caja blanca

El objetivo de realizar este tipo de prueba al sistema es que se garantice que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo o método, todos los bucles en sus límites operacionales así como las estructuras internas de datos para asegurar su validez. [26].

El proceso de pruebas de caja blanca se va a concentrar principalmente en validar a través del framework de software libre JUnit, que cada uno de los módulos o segmentos de códigos funcione apropiadamente. [31].

Para el desarrollo de estas pruebas unitarias se utilizó el framework JUnit, ya que ofrece las funcionalidades necesarias para implementar pruebas en un proyecto desarrollado en Java. Además cuenta con una interfaz simple que informa si cada una de las pruebas realizadas o conjunto de pruebas falló, pasó o fue ignorada. [31].

Inicialmente se definieron los casos de prueba, uno por cada clase implementada. Luego se definieron e implementaron los ficheros de configuración necesarios para establecer la comunicación entre las diferentes capas de la aplicación. Se identificaron los métodos a probar dentro de cada caso de prueba, así como el resultado esperado en cada uno. Y por último se procedió a realizar los casos de prueba diseñados. Con el objetivo de mostrar la realización de los casos de prueba, a continuación se muestran un grupo de imágenes con sus descripciones.

Se crea una clase en la que se van a definir los casos de prueba implementados para probar las funcionalidades que se encuentran en la clase. Primero se declara un atributo de dicha clase, luego dos mocks[7] para simular los objetos HttpServletRequest y HttpServletResponse que se le deben pasar a los métodos a probar, estos objetos son controlados por objetos de tipo MockControl, por último uno de tipo SessionFactory necesario para que Hibernate se comunique con la base de datos.

```
52 public class CargarDatosNegociacionTest extends
53         Test {
54
55     private CargarDatosNegociacionMultiActionController cargarDatosNegociacionMultiActionController;
56
57     private MockControl controlHttpServletRequest;
58     private HttpServletRequest mockHttpServletRequest;
59
60     private MockControl controlHttpServletResponse;
61     private HttpServletRequest mockHttpServletResponse;
62
63     private SessionFactory sf;
64
```

Ilustración 23: Declaración de los atributos en la clase de prueba.

El método setUp() es donde se inicializan todos los atributos declarados y contiene todos los beans de las clases necesarias para la ejecución de los procedimientos que se van a probar.

Para esconder y hacer más entendible el caso de prueba, se decidió poner en el fichero negociacion-context.xml todo el proceso de inicializar el objeto cargarDatosNegociacionMultiActionController. Debido a su complejidad los objetos mockHttpServletRequest, controlHttpServletRequest, mockHttpServletResponse, y controlHttpServletResponse son creados usando la librería EasyMock.

[7] **Mocks:** Son objetos simulados que imitan el comportamiento de los objetos reales en forma controlada. Normalmente se crea un objeto mock para probar el comportamiento de algún otro objeto.

```
67      protected void setUp()throws Exception{
68          AplicationContext context = new ClassPathXmlApplicationContext(
69                  "classpath:/cu/uci/finixubnc/cartacredito/negocicion/test/negociacion-context.xml");
70
71          cargarDatosNegociacionMultiActionController = (CargarDatosNegociacionMultiActionController)context.
72          getBean("cargarDatosNegociacion");
73
74          sf=(SessionFactory)context.getBean("finixuSessionFactory");
75
76          controlHttpServeltRequest = MockControl.createControl(HttpServletRequest.class);
77          mockHttpServeltRequest = (HttpServletRequest) controlHttpServeltRequest.getMock();
78
79          controlHttpServeltResponse = MockControl.createControl(HttpServletResponse.class);
80          mockHttpServeltResponse = (HttpServletResponse) controlHttpServeltResponse.getMock();
81
82          super.setUp();
83
84      }
```

Ilustración 24: Implementación del método setUp() en la clase de prueba.

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. Las verificaciones en este método se realizan de forma automática para todos los test definidos en la clase que se encuentra en este caso TestJUnit.

Se realizaron pruebas a los métodos carbarBancos(), cargarNegociacion(), cargarCC() y cargarReferenciaOriginal(). Primero se preparan los objetos mock para pasarle los datos necesarios al método que se desea probar. En la última línea del método de prueba se hace la llamada al método assertTrue() el cual recibe como parámetros el resultado esperado y la ejecución del método a probar para que este evalúe si coinciden ambos.

JUnit muestra una interfaz con los resultados de las pruebas realizadas, en este caso se muestran pruebas a los métodos de la clase CargarDatosNegociacionMultiActionController donde todas arrojaron resultados satisfactorios. A continuación se muestra el caso favorable para dicha prueba:

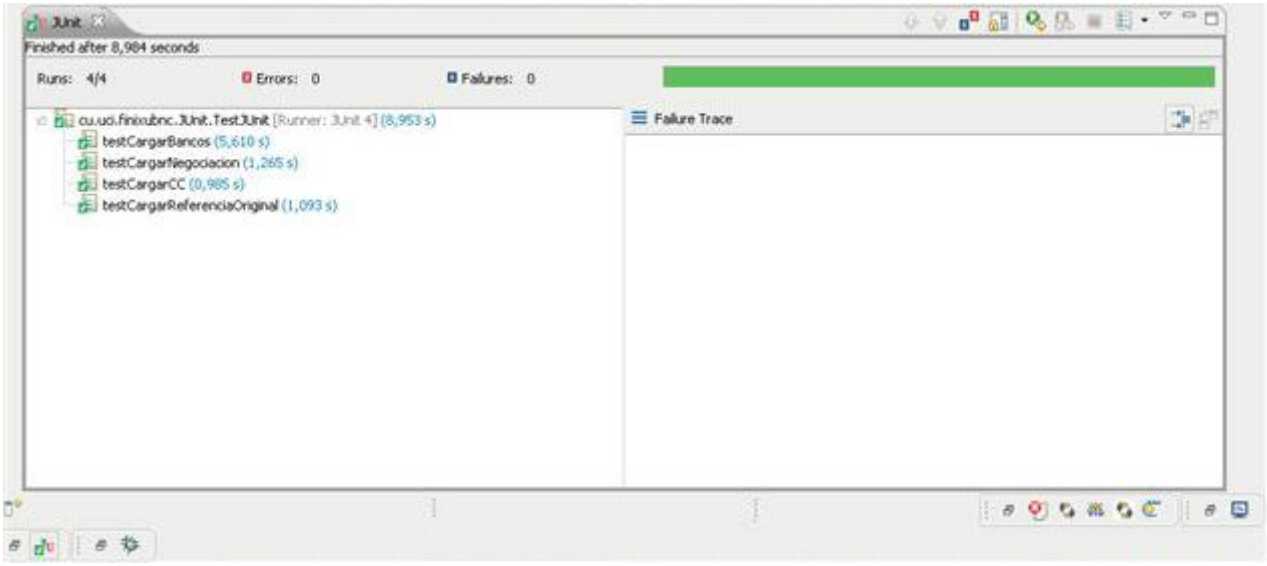

Ilustración 25: Resultado de la ejecución de las pruebas.

3.7.2 Pruebas de aceptación del cliente

Actualmente se firmó un documento de aceptación por parte del cliente en el BNC, donde avala que las nuevas funcionales a incorporar al subsistema Cartas de Crédito del sistema Quarxo presentan una correcta integración con los módulos del subsistema y del mismo con el resto de los componentes de la aplicación. Los módulos obtuvieron resultados satisfactorios desde el punto de vista de validación de datos y funcional. Los datos y los permisos de seguridad solamente son accedidos por los usuarios deseados. La impresora, hardware externo que debe integrarse para la impresión de la información referente a la contabilidad de las cartas de crédito procesadas también brinda resultados positivos.

En conclusión los módulos del subsistema Cartas de Crédito se encuentran completamente disponibles para su utilización en el BNC. El documento emitido por el BNC se puede consultar en el Anexo # 1 del documento.

3.8 Conclusiones del capítulo

Con la implementación del subsistema Cartas de Crédito se obtiene una aplicación funcional para el sistema Quarxo, que gestiona eficientemente el procesamiento de emisión y negociación de las cartas de crédito.

Esta etapa del desarrollo se rige por el diseño definido previamente, permitiendo que se realice de forma organizada y ágil. Las pruebas de aceptación del cliente validan la implementación realizada, demostrando que cumple con los requerimientos necesarios para satisfacer las necesidades del BNC. Al culminar la implementación de los módulos, se dio cumplimiento al objetivo general trazado inicialmente.

CONCLUSIONES

Una vez culminado el trabajo y como resultado del mismo se logró alcanzar el objetivo general propuesto, tras cumplir todas las tareas trazadas al inicio de la investigación. Con el desarrollo del presente trabajo de diploma, se cumplieron los objetivos específicos propuestos, permitiendo arribar a las siguientes conclusiones:

- ✓ La caracterización de las metodologías, lenguajes, tecnologías y herramientas definidas para el desarrollo del sistema Quarxo y la valoración de los sistemas informáticos contables nacionales e internacionales que involucran la gestión de los procesos de cartas de crédito, constituyeron los elementos necesarios para fundamentar las bases teóricas para darle solución al problema planteado.
- ✓ La generación de los artefactos durante el desarrollo de los flujos de Diseño e Implementación proporcionó la obtención del subsistema Cartas de Crédito Segunda Fase y la integración satisfactoria del mismo al sistema Quarxo.
- ✓ Se comprobó el correcto funcionamiento de la solución propuesta a través de pruebas de aceptación por parte del cliente que validaron las funcionalidades implementadas y la estabilidad necesaria para ser desplegado en el Banco Nacional de Cuba.
- ✓ Con el desarrollo del presente trabajo se consolidaron conocimientos adquiridos durante la carrera que permitieron la realización del mismo. Se logró una buena interrelación con los clientes del BNC, lo que permitió cumplir con el objetivo general a través del desarrollo claro y definido de las tareas que fueron propuestas.

RECOMENDACIONES

Tomando como base la investigación realizada y la experiencia acumulada durante el desarrollo del presente trabajo de diploma se recomienda:

- ✓ Agregar en el registro de negociación de cartas de crédito los mensajes 799, 754, 999.
- ✓ Permitir que el módulo Emisión de Cartas de Crédito pueda contabilizar las cartas de créditos de Gran Kaiman Teleco (GKT).
- ✓ Utilizar las tecnologías y herramientas presentadas en la investigación en el desarrollo de otros sistemas que pueda admitir cualquier tipo de carta de crédito.

BIBLIOGRAFÍA

1. ANEC. El economista de Cuba. [En línea] http://www.eleconomista.cubaweb.cu/.

2. CEPEC. Centro para la promoción del Comercio Exterior de Cuba. [En línea] http://www.cepec.cu/economia2.php

3. Ayaviri, D. G. Contabilidad Básica y Documentos Mercantiles. 1 ed. N-DAG, 1997.

4. Maldonador, R. Estudio de la Contabilidad General. La Habana: Félix Varela, 2006.

5. Barros, Juan Valenzuela. Estudio de la Contabilidad General. Antecedentes Históricos de la Contabilidad. 2000.

6. Tamargo, L. C. La contabilidad en una nueva tecnología [En línea] [Citado el: 6 de Diciembre de 2011]. Disponible en: http://www.betsime.disaic.cu/secciones/tec_feb_02.htm.

7. Lara, Rafael Bello, López, Tania Barroso. Diseño e Implementación del Subsistema Cartas de Créditos Del Proyecto SAGEB. 2010.

8. LA Sistemas - Er Desarrollo [En línea] [Citado el: 6 de Diciembre de 2011]. Disponible en: http://erdesarrollo.com/webs/lasistemas.com/producto-interna.php.

9. Global Trade Management (GTM) | Compliance Content Software. [En línea] [Citado el: 6 de Diciembre de 2011.] Disponible en: www.questaweb.com/sln-difference.aspx

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

11. Ivar Jacobson, Grady Booch, James Rumbaugh. El Proceso Unificado de Desarrollo de Software. s.l. : Pearson Educación, S.A, 2000.

12. Larman, Craig. UML y Patrones. S.l.: Prentice Hall, 1999.

13. Draper, Drave. Dojo Concepts For Java developers. Estados Unidos: IBM CORPORATION, 2008.

14. Sun 1999. Apache Tomcat. [En línea] 1999. [Citado el: 12 de Diciembre de 2011.] Disponible en: http://tomcat.apache.org/

15. Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato. Control de versiones con SubVersion. [En línea] 2004. [Citado el: 12 de Diciembre de 2011.] Disponible en: http://svnbook.red-bean.com/nightly/es/svn-ch-1-sect-1.html

16. Rodríguez, Roxana Bermúdez. Diseño e implementación de los módulos Documentos de embarque, Discrepancia y Negociación de Quarxo para el Banco Nacional de Cuba. 2011.

17. Patrones. [En línea] [Citado el: 12 de Diciembre de 2011.] Disponible en: http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf

18. Lou Torrijos, Ricard. Programación en castellano. [En línea] Diciembre 14, 2003. [Citado el: 8 de Diciembre de 2011.] Disponible en: http://www.programacion.com/java/tutorial/patrones2/8

19. Kaisler, Stephen H. Software Paradigms. 2005.

20. Craig Walls, Ryan Breidenbach. 2007. Spring in Action Second Edition. Estados Unidos: Manning Publications. 2007.

21. Peak, Patrick y Heudecker, Nick. Hibernate Quickly. S.l.: Manning Publications Co., 2006.

22. Iglesias, Adolfo Miguel. Documento de Arquitectura, Modernización Sistema del Banco Nacional de Cuba. 2009.

23. CONDUSEF. Carta de Crédito y su utilidad actual. [En línea]. [Citado el: 8 de Mayo de 2012.] Disponible en: http://www.liderempresarial.com/num102/14.php.

24. Latin Trade. [En línea]. [Citado el: 8 de Mayo de 2012.]. Disponible en: http://www.latintrade.com/newsite/esp/content/finance/types.cfm.

25. Ingenieriasoftwaredos. Diagrama de Despliegue. [En línea]. [Citado el: 20 de Mayo de 2012.]. Disponible en: http://www.ingenieriasoftwaredos.wikispaces.com/Diagramas+de+Artefactos+y+despliegue.

26. Pressman, Roger S. Ingeniería del Software. Un enfoque práctico. McGraw-Hil, 2002.

27. Juan Manuel Barrios N. 2003. Investigación de la plataforma JEE y su aplicación práctica. [En línea] 30 de Mayo de 2003. [Citado el: 15 de Junio de 2012.] Disponible en: http://www.dcc.uchile.cl/~jbarrios/J2EE/MemoriaJ2EEWeb.html.

28. Visual Paradigm for UML 8.2. [En línea]. [Citado el: 22 de Junio de 2012.] Disponible en: http://www.visual-paradigm.com/support/vpuml/releasenotes/820.jsp

29. Eclipse Galileo In Action. [En línea]. [Citado el: 22 de Junio de 2012.] Disponible en: http://www.eclipse.org/galileo/galileoinaction.php

30. CA ERwin Data Modeler Standard Edition. [En línea]. [Citado el: 22 de Junio de 2012.] Disponible en: http://erwin.com/mx/products/detail/ca_erwin_data_modeler_standard_edition/

31. Using JUnit With Eclipse IDE. [En línea] 2 de Abril de 2004. [Citado el: 22 de Junio de 2012.] Disponible en: http://www.junit.org/node/49http://www.junit.org/node/49

32. Instrumentos de Pago en el Comercio Exterior. [En línea]. [Citado el: 22 de Junio de 2012.] Disponible en: http://serviciosaduanerosycomerciales.wordpress.com/instrumetos-de-pago-en-el-comercio/

ANEXOS

Anexo 1: Certificado de aceptación por el BNC del subsistema Cartas de Crédito Segunda Fase del sistema Quarxo

BNC

12 de Junio de 2012
"Año 54 de la Revolución"

Asunto: Certificación del subsistema Cartas de Crédito Segunda Fase del sistema Quarxo desarrollado por la Universidad de las Ciencias Informáticas para el Banco Nacional de Cuba.

A favor de Rolando Choy Piñeiro, estudiante perteneciente al proyecto SAGEB, cuyo significado corresponde a: Sistema Automatizado para la Gestión Bancaria.

El subsistema Cartas de Crédito Segunda Fase comprueba la correcta integración con el resto de los componentes de la aplicación. Las validaciones de los datos, la aceptabilidad de las operaciones, los niveles de seguridad y la impresión de la información referente a la contabilidad de las cartas de crédito procesadas presentan buen funcionamiento y brinda resultados positivos.

Señalando como elementos positivos, la obtención de nuevas funcionalidades que permitirán agilizar la emisión y negociación de las cartas de crédito en el BNC.

Atendiendo a lo anterior y para que conste ante las instancias pertinentes de la UCI, firmo el documento.

Fraternalmente,

Bárbara Clavel Campos
Directora Operaciones 1

BANCO NACIONAL DE CUBA

Ilustración 26: Certificado de aceptación del subsistema Cartas de Crédito Segunda Fase de Quarxo.

Anexo 2: Interfaz Registrar Negociación

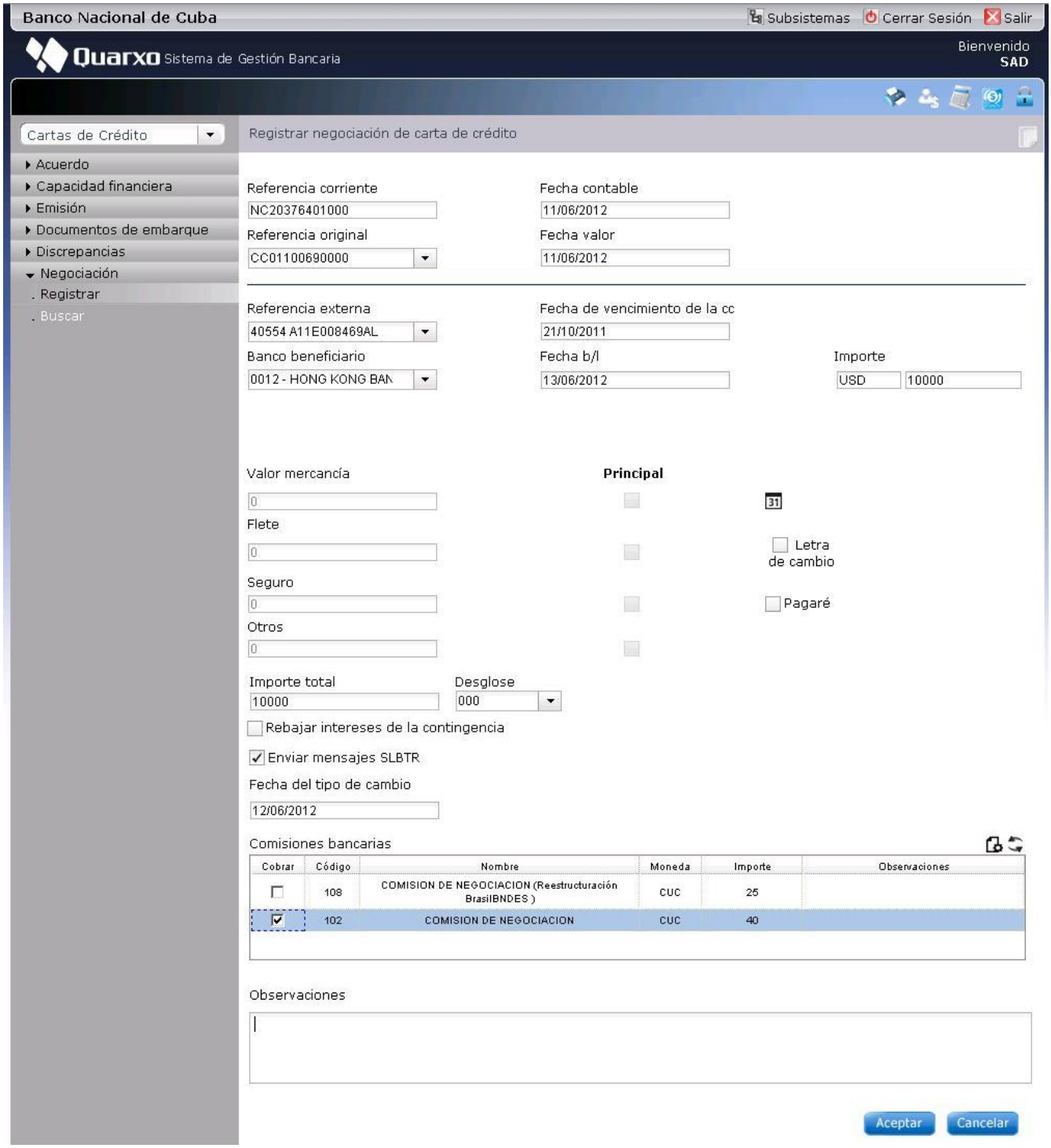

Ilustración 27: Interfaz del caso de uso registrar negociación.

Anexo 3: Interfaz de Asientos Contables del Cancelar Negociación

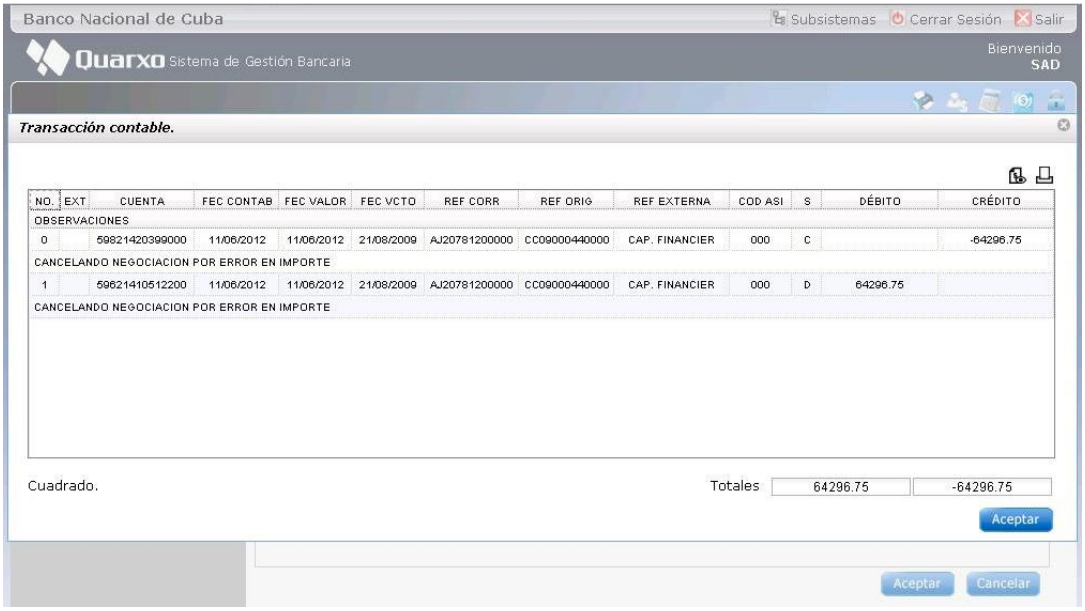

Ilustración 28: Interfaz de asientos contables del caso de uso cancelar negociación.

Anexo 4: Diseño de la capa de presentación del módulo Emisión (Spring MVC)

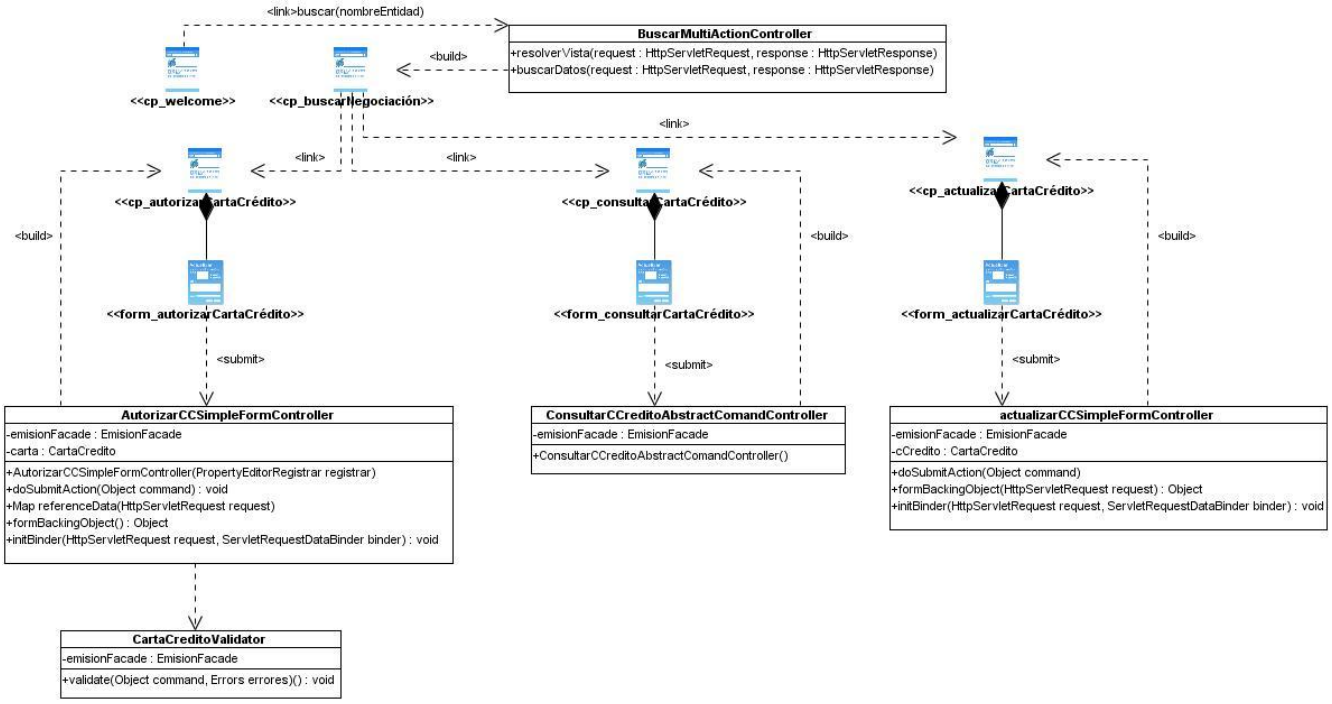

Ilustración 29: Diseño de la capa de presentación del módulo Emisión (SpringMVC).

GLOSARIO DE TÉRMINOS

API (Application Programming Interface): conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción. Usados generalmente en las bibliotecas.

Caso de Uso: fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido. Los casos de usos representan los requisitos funcionales del sistema.

HTML: siglas de HyperText Markup Language (Lenguaje de Marcado de Hipertexto), hace referencia al lenguaje de marcado predominante para la elaboración de páginas web que se utiliza para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. El HTML se escribe en forma de etiquetas, rodeadas por corchetes angulares (<,>).

IDE's (Integrated Development Environment): Entorno de Desarrollo Integrado, es un programa compuesto por un conjunto de herramientas para un programador.

Java Server Page (JSP): es una tecnología orientada a crear páginas web con programación en Java que nos permite mezclar HTML estático con HTML generado dinámicamente.

Módulo: cada módulo es una parte del sistema, que se instala y funciona por separado, entrelazándose con otros módulos con los que intercambia información.

MS-DOS: Microsoft Disk Operating System, Sistema operativo de disco de Microsoft.

SAGEB: Sistema Automatizado de la Gestión Bancaria.

Servlet: es una clase Java que ofrece funciones suplementarias al servidor.

SLBTR: Sistema de Liquidación Bruta en Tiempo Real.

SWIFT (System Worldwide Internacional Financial Transactions): es una sociedad creada por la comunidad internacional de entidades financieras para la transmisión rápida, segura y efectiva de documentos, dinero y mensajes.

Software Propietario: propietario significa que algún individuo o compañía retiene el derecho de autor exclusivo sobre una pieza de programación, al mismo tiempo que niega a otras personas el acceso al código fuente del programa y el derecho a copiarlo, modificarlo o estudiarlo. Sin embargo, el programa puede seguir siendo propietario aunque su código fuente se haya hecho público, si es que se mantienen restricciones sobre su uso, distribución o modificación.

TGC (Transacción General de Contabilidad): módulo del subsistema Contabilidad del sistema Quarxo que permite al usuario registrar transacciones contables insertando para ello los datos de los asientos que conforman la transacción. El objetivo principal de este módulo es poder registrar transacciones que no puedan ser llevadas a cabo por los demás módulos del sistema.

XML (Extensible Markup Language): lenguaje de marcas extensible, es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium.