UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FACULTAD 3

Título: Diseño e implementación de los módulos Documentos de embarque, Discrepancia y Negociación de Quarxo para el Banco Nacional de Cuba.

Trabajo de Diploma para optar por el título de

INGENIERO EN CIENCIAS INFORMÁTICAS

Autor(es): Roxana Bermúdez Rodríguez.

Tutor(es): Ing. Javier Martínez Muñoz

Ciudad de la Habana, junio 2010
"Año 53 de la Revolución"

"Los sueños se convierten en realidad si hay pasión, si hay tesón, si hay voluntad."

Fidel Castro Ruz.

DECLARACIÓN DE AUTORÍA

Declaro ser la única autora de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Roxana Bermúdez Rodríguez	Ing. Javier Martínez Muñoz
_____	_____
(Autor)	**(Tutor)**

DATOS DE CONTACTO

Ing. Javier Martínez Muñoz: Graduado en la Universidad de las Ciencias Informáticas. Instructor. Profesor de Matemática III y IV. 4 años de experiencia en el tema. 3 años de graduados. Analista principal de proyecto SAGEB.

Correo electrónico: jmmunoz@uci.cu

AGRADECIMIENTOS

A nuestra Revolución y al Comandante por esta oportunidad.

A mi familia y en especial a mis padres y a mi hermana, sin su apoyo esto no hubiera sido posible.

A mis fantásticas amigas Polly, Pity-Dori y Spike, por su tiempo, su increíble paciencia y por contagiarme la alegría de vivir.

A Anne, a Yass, a Day por los años vividos.

A Manu, por la ayuda de siempre y tu buen humor.

A Yaimi, Yamilka, Indiana, Mavis, Sandra, Yaslin, Ari, Yelena, Rene, Ariel, Edgar, Alain, Leo, Adrian, Yoan, Robert por dedicarme un pedacito de su tiempo.

A todos los profesores por su aporte conjunto a nuestra formación profesional y en especial a Javier y a Yoan Antonio, por la confianza y dedicación.

A los que me apoyaron de alguna forma, sería imposible mencionarlos a todos… (aunque no físico, sí tienen todos un espacio en cada momento difícil vivido, en cada alegría, en cada una de mis aspiraciones convertidas ahora en realidad). Siempre estarán en mí…esos buenos momentos que pasamos sin saber.

Gracias a todos…

DEDICATORIA

Para mi hermana, Susana. Nadie ha recibido más afecto ni apoyo tan incondicional como el que tú me has dado. Yo también te quiero.

A mis padres Fernán y Tere, la pareja feliz como dice Susi, por todo lo que me han dado en esta vida, especialmente por sus sabios consejos y por estar a mi lado en los momentos difíciles.

A mis abuelitos Fernando y Roberto, quienes desde el cielo me guían y estoy segura que en estos momentos están orgullosos de mí.

A mis abuelitas Raquel y María les quiero a montones.

A mi tío Robert quien me ha acompañado con una comprensión a prueba de todo.

A mis tíos Raque, Lorenzo y mis primos Emir e Indira por estar siempre dispuestos a ayudarme.

A mi futuro sobrinito, que ya vienes en camino, también te quiero Pelusín.

RESUMEN Y PALABRAS CLAVES

El Banco Nacional de Cuba (BNC) en conjunto con la Universidad de las Ciencias Informáticas (UCI) se proponen desarrollar el sistema Quarxo, utilizando el núcleo del Sistema Automatizado para la Banca Internacional de Comercio (SABIC) en busca de perfeccionamiento de servicios y seguridad para con los clientes del banco.

En la actualidad la gestión de los documentos de embarques, discrepancias y negociaciones, procesos claves dentro de las operaciones bancarias en el BNC no se encuentran informatizados por lo que se vuelven ineficientes teniendo en cuenta la complejidad y cantidad de información que hay que manejar provocando que el trabajo a realizar sea más engorroso y propenso a errores.

Con el propósito de incorporar dichas funcionalidades al sistema Quarxo, el presente trabajo de diploma comprende el Diseño y la Implementación de los módulos Documentos de Embarque, Discrepancia y Negociación, 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.

Como resultado se obtienen los módulos Documentos de Embarque, Discrepancia y Negociación del subsistema Cartas de Crédito en el sistema Quarxo.

Palabras claves: Documentos de Embarque, Discrepancia, Negociación, BNC, Quarxo, Diseño, Implementación.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Las Tecnologías de la Información y las Comunicaciones (TIC) son especialmente importantes para las empresas, atendiendo a que facilitan todo tipo de actividades que se realizan en ellas; generando una mayor productividad y eficiencia.

El gobierno cubano ha identificado desde muy temprano la conveniencia y necesidad de dominar e introducir en la práctica social las TIC, para lograr una cultura digital como una de las características que debe distinguir al hombre moderno. El país se ha centrado en un proceso de informatización de la sociedad, lo que trajo consigo la necesidad de crear sistemas eficientes y a la altura de las nuevas tecnologías, que como parte de los avances económicos, se van introduciendo.

Una de las instituciones estatales importante en el desarrollo y control económico del país, el BNC, ha utilizado durante muchos años el sistema contable SABIC que permite la gestión de las distintas operaciones financieras; cumpliendo con los requisitos necesarios para satisfacer las funciones que se llevan a cabo en el banco.

Con el transcurso del tiempo se introdujeron nuevos cambios en la actividad bancaria, así como se ha producido un gran aumento en el volumen de información que se procesa en el banco por lo que las funcionalidades del SABIC respecto al control de los servicios ofrecidos por las entidades financieras bancarias se han quedado limitadas. Además el sistema presenta otras desventajas como son: se encuentra desarrollado en FoxPro sobre MS-DOS que es una herramienta privativa, no tiene integración con los sistemas de mensajería, con el Sistema de Liquidación Bruta en Tiempo Real (SLBTR) ni con el Sistema de Comunicación (SISCOM) que existen actualmente para facilitar la comunicación entre los bancos nacionales e internacionales, no permite generar reportes personalizados, configurables por el usuario, que le permitan mostrar información específica sobre la actividad de la entidad y no almacena toda la información que se necesita para la confección de los informes, entre otras limitaciones.

El SABIC no responde eficientemente a la complejidad actual de los procesos que se llevan a cabo en el BNC, ejemplo de ello es que una gran parte de la gestión de los documentos de embarques, discrepancias y negociaciones se realizan de forma manual por los trabajadores del banco trayendo como consecuencia que los procesos se ejecuten de manera más lenta y engorrosa, dificultando el control e incrementando las probabilidades de cometer errores.

Debido a esta situación se determina la necesidad de realizar un nuevo sistema informático que satisfaga sus actuales requerimientos. La UCI como programa de la revolución que juega un papel importante en la informatización de la sociedad es la encargada de desarrollar el sistema Quarxo en el proyecto Sistema Automatizado de Gestión de Entidades Bancarias (SAGEB); implementado sobre el núcleo del SABIC. El sistema propuesto incluye un mayor nivel de gestión, control y seguimiento de los procesos que se ejecutan dentro de la entidad garantizando el almacenamiento de información de forma íntegra y segura. Entre las funcionalidades a las que debe dar respuesta se encuentran la gestión de los Documentos de Embarques, Discrepancias y Negociaciones.

Basándose en lo explicado, se puede plantear, que el **problema** a resolver en este trabajo es: ¿Cómo mejorar la gestión de los Documentos de Embarques, Discrepancias y Negociaciones en el Banco Nacional de Cuba?

El **objeto de estudio** queda enmarcado en la gestión de Documentos de Embarques, Discrepancias y Negociaciones en las entidades financieras bancarias, y delimitando el **campo de acción** a la gestión de Documentos de Embarques, Discrepancias y Negociaciones de Cartas de Créditos en el BNC.

Dada las condiciones descritas y en particular el estado en que se encuentra la obtención y las dificultades para el análisis de la información en el BNC, el **objetivo general** trazado para darle solución al problema establecido es: Desarrollar el diseño e implementación de los módulos Documentos de Embarque, Discrepancia y Negociación de Quarxo a partir del análisis de las funcionalidades establecidas en acuerdo con el cliente y haciendo uso de las tecnologías y herramientas definidas para el proyecto.

Para dar cumplimiento al objetivo planteado se definen las siguientes **tareas a realizar**:

- Caracterización de las tecnologías y herramientas a utilizar en el diseño e implementación de los módulos Documentos de Embarque, Discrepancia y Negociación de Quarxo.
- Caracterización de los sistemas automatizados nacionales e internacionales que soportan la realización de dichos procesos.
- Caracterización de los requisitos de software y el modelo de negocio correspondientes a los módulos Documentos de Embarque, Discrepancia y Negociación de Quarxo.
- Diseño de las clases de los módulos Documentos de Embarque, Discrepancia y Negociación de Quarxo
- Implementación de las funcionalidades de los módulos Documentos de Embarque, Discrepancia y Negociación de Quarxo.
- Construcción del modelo de componentes de los módulos Documentos de Embarque, Discrepancia y Negociación de Quarxo.

Posibles resultados: Obtener los Módulos de Documentos de Embarque, Discrepancia y Negociación del Quarxo.

El presente documento se divide en 3 capítulos, a continuación se presenta el nombre del capítulo y su objetivo en un contexto global:

Capítulo #1: "Fundamentación Teórica". El objetivo de este capítulo es abordar conceptos generales y básicos de las características y los procesos fundamentales involucrados en los módulos de Documentos de Embarque, Discrepancia y Negociación. Incluye un estado del arte referente al tema que se investiga. Se describen las metodologías, las tecnologías, el lenguaje de programación que se utilizara para resolver el problema y se realiza una breve descripción de las herramientas complementarias que se han seleccionado para el desarrollo.

Capítulo # 2: "Arquitectura y Diseño de los módulos Documentos de Embarque, Discrepancia y Negociación". Se presenta la descripción de la arquitectura propuesta por el proyecto SAGEB y se explica el modelo del diseño y el modelo de datos de los módulos Documentos de Embarque, Discrepancia y Negociación. Además se caracteriza los patrones de diseño para la búsqueda de soluciones a problemas comunes en el desarrollo.

Capítulo # 3: "Implementación de los módulos Documentos de Embarque, Discrepancia y Negociación". En este capítulo se realiza la construcción de la solución explicando los aspectos principales de la implementación como es la utilización de Spring WebFlow framework. Se realiza una descripción de las clases y funcionalidades, y del diagrama de interacción de los componentes de la aplicación con los módulos, así como la relación de los estándares de codificación utilizados.

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

Capítulo 1 - Fundamentación Teórica

1.1 Introducción

En este capítulo, como parte de la fundamentación teórica de la investigación, se exponen los conceptos fundamentales que incluye el problema a resolver, se analiza el estado del arte desde los sistemas que automatizan la gestión de los Documentos de Embarques, Discrepancias y Negociaciones a nivel mundial así como los que se utilizan a nivel nacional. Se hace necesario un estudio de herramientas de desarrollo para darle solución al problema de investigación planteado, siendo también en este capítulo donde se describen las tecnologías y metodologías que se utilizarán a la hora del desarrollo del producto.

1.2 Conceptos fundamentales

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

1.2.1 Banco

“Un banco es un intermediario financiero que se encarga de captar recursos en forma de depósitos, y prestar dinero, así como la prestación de servicios financieros.” [1] Para ello utilizan como principal instrumento las tasas de interés, las cuales son un incentivo tanto para ahorrar dinero como para pedir prestado.

El banco moderno tiene que cumplir tres grandes funciones: la Intermediación de Crédito, la Intermediación de los pagos y la Administración de los capitales. [2]

Banco Nacional de Cuba

El Banco Nacional de Cuba fue creado mediante la Ley 13, del 23 de diciembre de 1948, como Banco Central del Estado con autonomía orgánica, personalidad jurídica independiente y patrimonio propio.

"El Decreto Ley 181 del 23 de febrero de 1998, fija las nuevas funciones, atribuciones y deberes del Banco Nacional de Cuba, estableciendo entre las principales: actuar como agente corresponsal de bancos extranjeros; emitir garantías bancarias previa evaluación de la gestión económico-financiera de la entidad solicitante; obtener y otorgar créditos; mantener el registro, control, servicio y atención de la deuda externa de Cuba y realizar todo tipo de operaciones y negocios de intermediación financiera." [3]

1.2.2 Contabilidad

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." [4]

Los bancos como entidades encargadas de regular las relaciones financieras entre personas e instituciones necesitan de la utilización de la contabilidad para este fin.

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." [5]

Los bancos juegan un importante papel en el comercio tanto nacional como internacional. La contabilidad bancaria genera la necesidad de efectuar pagos y cobros a distancia entre partes ubicadas en distintos

municipios, provincias y países utilizando para ello los medios de pago.

1.2.3 Medios de Pago

"Los medios de pago son las herramientas utilizadas por los agentes de una economía para transferir dinero a cambio de bienes, servicios y activos financieros." [6]

Entre los documentos más utilizados por las instituciones bancarias para realizar el pago se encuentran: Orden de Pago simple, Orden de Pago Documentaria, Cartas de Créditos o Créditos Documentarios.

1.2.3.1 Carta de Crédito

Según la Cámara de Comercio Internacional, "una carta de crédito o crédito documentario es un acuerdo por el que un banco (banco emisor), obrando a petición y de conformidad con las instrucciones de un cliente (ordenante) se obliga a hacer un pago a un tercero (beneficiario) o, a su orden, aceptar y pagar letras de cambio u otros instrumentos de giro librados por el beneficiario contra la entrega de los documentos exigidos, siempre que se cumplan los términos y condiciones del crédito." [7]

Este documento evidencia el compromiso legal ineludible por el cual el banco emisor debe pagar o hacer pagar (generalmente a través de otro banco) al exportador, si éste cumplió con las obligaciones del crédito.

Los pagos, aceptaciones o negociaciones se efectúan sólo contra la presentación de aquellos documentos que se estipulan en el contrato de compra-venta, los que deberán guardar una apariencia formal con las condiciones del crédito.

Los documentos pueden ser financieros (letras de cambio, cheques, pagarés) o comerciales (facturas, documentos de transporte, de calidad, de seguro). [8]

Documentos Financieros

Título de valor en que se deja constancia que quien lo subscribe tiene la obligación de pagar a la fecha especificada en el documento y a la persona indicada en el mismo, una cierta suma de dinero.

- ✓ **El Cheque**

Es un documento financiero que permite al librador pagar sus deudas, sin necesidad de entregar dinero físico. El deudor entrega a su acreedor un cheque por el importe de su deuda, girado sobre el banco en el que posee fondos, de modo que el acreedor, al recibirlo, puede cobrarlo directamente o ingresar su importe en su propia cuenta bancaria.

- ✓ **Letra de Cambio**

"La Letra de Cambio se define como una orden incondicional, por escrito, dirigida por una persona (girador), a otra persona (girado), firmada por la persona que la da, requiriendo a la persona a la que está dirigida (girado) a pagar a la vista o a un tiempo fijo o futuro determinado, una suma definida de dinero a la orden de, a una persona específica, o al portador (beneficiario)." [9]

Cuando hablamos de persona se aplica tanto a personas naturales como a personas jurídicas, es decir, firmas comerciales, empresas, corporaciones, etc.

- ✓ **El Pagaré**

"Es un documento en virtud del cual una persona natural o jurídica (deudor) se compromete a pagar al acreedor una suma definida de dinero a su demanda o en fecha determinada en el futuro en un lugar también determinado." [8] Es un título valor muy similar a la Letra de Cambio, y su diferencia radica en que quien emite el pagaré es el propio deudor (y no el acreedor).

Documentos Comerciales o Documentos de Embarque

Como tal se designan a los documentos representativos de una transacción comercial internacional y del embarque efectuado en particular, que se exigen para demostrar que se ha cumplido con los requisitos del envío de las mercancías vendidas y para exigir el pago de las mismas por parte del comprador. [10]

Los contenidos de los distintos documentos deben concordar entre sí en su totalidad, referenciando los datos a la factura pro-forma que diera origen a la operación comercial.

De acuerdo a los requerimientos bancarios y de lo oportunamente acordado entre el exportador y el importador, los documentos de embarque originales pueden viajar con el medio de transporte o ser remitidos por separado.

Los documentos normalmente exigidos en las cartas de crédito comerciales son:

- ✓ **Factura comercial**

Documento emitido por el exportador en el que se recogen los datos fundamentales de la operación, señalándose como mínimo: Identificación del exportador y del importador, descripción de la mercancía, importe de la venta, con desglose entre los distintos componentes de la misma, forma de pago requerida y fecha de envío de la mercancía, medio de transporte y lugar de destino. El importe de la venta puede estar desglosado en:

- **Flete:** Tarifa básica pactada entre el transportador y el usuario del servicio, en el cual el primero se compromete a trasladar la mercancía desde un punto de origen hasta el de destino acordado previamente.
- **Seguro:** Contrato por el cual el asegurador se obliga, mediante el cobro de una prima a abonar, dentro de los límites pactados, un capital u otras prestaciones convenidas, en caso de que se produzca el evento cuyo riesgo es objeto de cobertura.
- **Mercancía:** Producto del trabajo destinado a satisfacer alguna necesidad del hombre y que se elabora para la venta, no para el propio consumo.

- **Otros:** Gasto adicional.

✓ **Conocimiento de embarque o Documento de transporte**

Justificante de que el exportador ha enviado la mercancía por el medio acordado (barco, tren, avión, etc.). Este documento es el que permitirá al importador retirar la mercancía en el punto de destino. En el mismo se detalla: punto de origen y de destino, medio de transporte utilizado, fecha de salida, fecha previsible de llegada, nombre del remitente, nombre del destinatario y mercancía transportada (descripción, peso, número de bultos, etc.)

✓ **Lista de empaque**

Documento donde el exportador detalla las características de embalaje de la mercadería de un embarque particular. En general incluye la siguiente información: datos del exportador, datos del importador, fecha de emisión, lugar de emisión, marcas de los bultos, números de los bultos, medio de transporte, datos del medio de transporte, detalle de la cantidad de bultos, peso de la mercancía, detalle de la mercadería contenida en cada uno de los bultos, total de los pesos bruto y neto, tipo de embalaje y firma del exportador.

✓ **Certificado de origen**

Documento emitido por las autoridades comerciales del país de origen, en el que se acredita que el producto ha sido elaborado en dicho país.

✓ **Certificado de calidad**

Documento emitido por una firma independiente de reconocido prestigio que declara que la mercancía cumple los estándares internacionales de calidad.

- ✓ **Certificados fitosanitarios**

Documento emitido por las autoridades sanitarias del país de origen, en el que se certifica que la mercancía exportada se encuentra en perfecto estado sanitario, cumpliendo la normativa vigente.

- ✓ **Documento del seguro**

Documento donde se describen los riesgos que cubre la póliza, determinando quién es el beneficiario en caso de indemnización. El costo del seguro puede ser por cuenta del exportador o del importador, dependiendo de lo acordado entre las partes.

En general, los documentos comerciales son todos los documentos que el importador necesite para asegurarse el cumplimiento del contrato por parte del vendedor.

Cualquier diferencia o anormalidad que tengan los documentos de embarque presentados por el exportador al banco, respecto de lo establecido en los términos del crédito, se denomina **discrepancia de los documentos de embarque.**

Si hay discrepancias, supuestamente el exportador no podrá hacer efectivo el crédito hasta tanto el banco emisor (y, obviamente, el importador) autoricen a pagarlo a pesar de ellas, para lo cual se pueden aplicar diferentes procedimientos. A esto se le denomina técnicamente “levantar las reservas”. El levantar o no las reservas es un derecho que le asiste al importador quien, por tanto, deberá hacerlo constar por escrito.

1.2.4 Negociación de las Cartas de Créditos

“Negociar significa hacer entrega del valor del/de los instrumento/s de giro y/o documento/s por parte del banco autorizado a negociar. El simple examen de los documentos sin hacer entrega de su valor no constituye una negociación.” [11]

Una vez revisada y encontrada toda la documentación ajustada estricta y literalmente a los términos de la carta de crédito, o con la autorización del levantamiento de las reservas, el banco negociador procede a realizar el pago o aceptación.

Las diferentes formas de pago de las cartas de créditos son:

- **Pago a la Vista**: Si el importador y el exportador no han convenido en un plazo para el pago de los montos acordados bajo la carta de crédito, una vez que el exportador haya presentado los documentos pagados al banco y esta obtenga los fondos correspondientes al banco confirmado, el pago de la carta de crédito se hará efectivo de inmediato si esta denominado en la moneda del importador, cuando lo está en otra moneda, el pago se hace en el plazo necesario para transferencia de fondos en la moneda correspondiente.
- **Con Aceptación**: Son aquellas pagaderas a los beneficiarios contra letras de cambio libradas a un plazo cierto, es decir, a días vista, a tantos días fecha, a fecha fija, etc.
- **Pago diferido**: A veces el importador aprovecha la carta de crédito para obtener términos de financiamientos que le permita recibir la mercancía y revenderla en su país. Para esto establece en la solicitud términos de pagos diferidos.
- **Mixta**: En este caso se pagaría la parte que es a la vista y se aceptaría la letra de cambio respectiva por el remanente.

Llevar la contabilidad de las Negociaciones de las Cartas de Créditos se vuelve una tarea compleja dado todas las variantes que son necesarias de utilizar para llevar a cabo estos procesos. En la actualidad los grandes avances en el área de la informática brindan facilidades a partir de la utilización de los sistemas contables.

1.3 Sistemas Informáticos Contables

“Un sistema de información contable comprende los métodos, procedimientos y recursos utilizados por una entidad para llevar un control de las actividades financieras y resumirlas en forma útil para la toma de decisiones.” [12] Es el conjunto de principios y reglas que facilitan el conocimiento y la representación adecuada de la empresa y de los hechos económicos que afectan a la misma.

Teniendo en cuenta el objetivo principal del presente trabajo, se hace importante realizar una investigación de los sistemas contables que existen en el mundo con el objetivo de enriquecer los conocimientos sobre las diferentes funcionalidades que debe ofrecer este tipo de sistema.

1.3.1 Sistemas Informáticos Contables Bancarios Internacionales

En el mundo existen diversos sistemas contables bancarios. Al realizar un estudio de ellos se reconoce que se utilizan como soporte para los procesos de Negociaciones de Carta de Créditos cuyo conocimiento pudiera resultar importante para el desarrollo de la futura aplicación. Dos de ellos se describen a continuación:

CARD CRED

Card Cred es un sistema desarrollado por la compañía venezolana “La Sistema” (Fábrica Venezolana de Tecnología Financiera) que permite llevar un control detallado de las acciones administrativas y operativas relacionadas con la Carta de Crédito tales como negociaciones y pago de las mismas. La aplicación se basa en una codificación de: Clientes; Corresponsales; Aduanas; Cartas de Crédito. Emite la relación contable diaria en reporte y archivo de textos, y genera las llamadas débitos/créditos de las operaciones efectuadas. Está dividido en 6 módulos. El módulo de Cartas de Crédito posibilita el registro de los instrumentos de pago utilizados en el comercio internacional. [13]

COBIS SCI

COBIS SCI (Sistema Integral de Comercio Internacional), desarrollado con tecnología de punta y arquitectura cooperativa Cliente-Servidor, brinda soporte a todas las actividades de negocio y operativas de comercio internacional que se llevan a cabo dentro de un Banco o Institución Financiera. Permite realizar de forma automática la contabilización de las transacciones financieras generadas. Este sistema maneja el proceso de negociación de una carta de crédito, con posibilidad de registrar discrepancias mediante la respectiva carta, indicándose también detalles de embarques y formas de pago, liquidación, abono y cancelación. [13]

Estos sistemas contables son propietarios y muy costosos, además de que la documentación de sus funcionalidades no se encuentra disponible y por dichas razones no se asegura que las mismas se asemejen a las necesidades que se requieren para el BNC.

1.3.2 Sistemas Informáticos Contables Bancarios Nacionales

El uso de los Sistemas Informáticos Contables resulta común en las entidades cubanas, orientados fundamentalmente a obtener una mayor eficiencia en la gestión contable empresarial. Centraremos nuestra atención en el que actualmente se utiliza en el BNC.

SABIC

El SABIC (Sistema Automatizado para la Banca Internacional de Comercio) es un sistema diseñado y desarrollado por la Dirección de Sistemas Automatizados del Banco Central de Cuba para satisfacer las necesidades de procesamiento de datos de bancos e instituciones no bancarias, utilizando los medios técnicos de computación disponibles en el mercado. Está desarrollado en el lenguaje de programación FoxPro para el sistema operativo MS-DOS. También existe una versión en Visual FoxPro para Windows con Servidor de Base de Datos: Microsoft SQLServer.

Entre las características fundamentales del SABIC se encuentran:

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

A pesar de las grandes ventajas que brinda el sistema SABIC, resulta incompatible con el sistema de mensajería que utiliza el BNC (SISCOM, que funciona sobre Windows). No ofrece la funcionalidad de obtener o crear reportes de salida dinámicos, ni ofrece tampoco la posibilidad de la gestión de las Negociaciones de Cartas de Crédito, que es un proceso que incluye las operaciones de contabilización, pero que contempla otras operaciones de similar importancia que el SABIC no gestiona; con lo que se evidencia la necesidad de obtener una aplicación informática que ofrezca una gestión integral del mencionado proceso.

1.4 Metodología, Tecnologías y Herramientas utilizadas en el desarrollo

Para el desarrollo de los módulos Documentos de Embarque, Discrepancia y Negociación se siguieron los lineamientos arquitectónicos establecidos por la dirección del proyecto SAGEB con el propósito garantizar la compatibilidad con el resto de los módulos que se realizan por parte de otros desarrolladores.

En este epígrafe se realiza una identificación de cada una de las metodologías y tecnologías así como la utilidad que proporcionan en el trabajo del diseño e implementación de los módulos implicados.

1.4.1 Metodología de Desarrollo de Software

Una metodología para el desarrollo de un proceso de software es un conjunto de filosofías, fases, procedimientos, reglas, técnicas, herramientas, documentación y aspectos de formación para los desarrolladores de Sistemas Informáticos. [14]

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 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.

1.4.1.1 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.” [14]

La metodología RUP, divide en 4 fases el desarrollo del software y sus objetivos son:

- Inicio: Determinar la visión del proyecto.
- Elaboración: Determinar la arquitectura óptima.
- Construcción: Llevar a obtener la capacidad operacional inicial.
- Transición: Llegar a obtener la liberación del proyecto.

RUP le proporciona a cada miembro de un equipo un fácil acceso a una base de conocimiento con guías, plantillas y herramientas para todas las actividades críticas del desarrollo. Su meta consiste en asegurar la producción de software de muy alta calidad que satisfaga la necesidad de los usuarios dentro de un calendario y presupuesto predecible.

Las disciplinas propuestas por RUP son: Modelado del negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas, entre otras. Cada iteración añade funcionalidades al producto de software o mejora las existentes.

A continuación se describen las disciplinas de mayor interés de acuerdo a los roles desempeñados para la realización del trabajo de diploma.

Disciplina 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. [14]

Disciplina 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. [14]

1.4.2 Lenguaje 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." [15]

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**.** [15]

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.

1.4.3 Herramientas de Modelado

1.4.3.1 Herramientas CASE

Las herramientas CASE (en inglés: Computer Aided Software Engineering) son un complemento de la caja de herramientas del ingeniero del software. CASE proporciona al ingeniero la posibilidad de automatizar actividades manuales y de mejorar su visión general de la ingeniería. Las herramientas CASE ayudan a asegurar la calidad de un producto desde su diseño antes de construirlo.

Entre las más conocidas se encuentra Rational Rose herramienta de modelado visual para el análisis y diseño de sistemas basados en objetos que se utiliza para modelar un sistema antes de proceder a construirlo y el Visual Paradigm que contiene características gráficas muy cómodas, que facilitan la realización de los diagramas de modelado que sigue el estándar de UML.

La herramienta CASE que propone utilizar el proyecto SAGEB para realizar el modelo del diseño es Visual Paradigm porque cumple perfectamente con todos los estándares UML además de ser la herramienta que tiene definida la universidad para su uso en los proyectos de desarrollo y para realizar el modelo de datos la herramienta Erwin por la rapidez que brinda para el desarrollo de bases de datos complejas.

Visual Paradigm

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

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 del proyecto.

1.5 Patrones de Diseño

En la tecnología de objetos, un patrón es una descripción del problema y su solución, a la que se da un nombre, y que se puede aplicar a nuevos contextos. Proporciona consejos sobre el modo de aplicarlo en circunstancias diversas, y considera los puntos fuertes y compromisos.

Los patrones del diseño tratan los problemas del diseño que se repiten y que se presentan en situaciones particulares del diseño, con el fin de proponer soluciones a ellas. Por lo tanto, los patrones de diseño son soluciones exitosas a problemas comunes.

A continuación se mencionan algunos de los patrones que son utilizados en el diseño de los módulos Documentos de Embarque, Discrepancia y Negociación.

1.5.1 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.

Patrón Experto

El patrón Experto ofrece una analogía con el mundo real. Asigna responsabilidades a individuos que disponen de la información necesaria para llevar a cabo la tarea, es decir, a la clase que cuenta con la información necesaria para cumplir con la responsabilidad. Se refuerza el encapsulamiento y se favorece el bajo acoplamiento. [15]

Patrón Creador

"El patrón creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos." [15] Su propósito principal es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Brinda soporte al bajo acoplamiento.

Patrón Controlador

Un controlador es un objeto que no pertenece a la interfaz de usuario, responsable de recibir o manejar un evento del sistema. El controlador es un intermediario entre la interfaz de usuario y el núcleo de las clases donde reside la lógica de la aplicación. Este patrón proporciona la llamada a los servicios más importantes de la capa de interfaz de usuario hacia las otras capas.

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. [15] Reduce el impacto de los cambios y permite crear clases más independientes, más reutilizables, lo que implica mayor productividad.

Patrón Alta Cohesión

La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Asignar una responsabilidad de modo que la cohesión siga siendo alta. [14] Brinda una mayor facilidad para darle mantenimiento, entender y reutilizar las clases.

1.5.2 Patrones Estructurales de 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.

Facade

El patrón de diseño Facade sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Reduce el número de objetos con los que tiene que interactuar un cliente proporcionando un menor acoplamiento y facilitando el cambio de componentes sin afectar a sus clientes.

Otros patrones que se utilizan en el desarrollo de este trabajo son:

1.5.3 Patrón de Acceso a Datos (DAO)

El objetivo fundamental es abstraer y encapsular todos los accesos a la fuente de datos aislando los objetos de negocio de una implementación particular de la persistencia. La ventaja de usar objetos de acceso a datos es que cualquier objeto de negocio no requiere conocimiento directo del destino final de la información que manipula y que se puede cambiar fácilmente de fuente de datos.

1.5.4 Patrón de Presentación 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*.

1.5.5 Patrón MVC (Modelo Vista Controlador)

El objetivo fundamental del uso del patrón MVC es separar el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

- **Modelo:** Administra el comportamiento y los datos del dominio de aplicación.
- **Vista:** Maneja la visualización de la información.
- **Controlador:** Controla el flujo entre la vista y el modelo (los datos).

Al aplicarse este modelo, la vista y el controlador se abstraen del acceso a los datos, del cual se encarga la capa de dominio.

1.6 Ambiente de desarrollo integrado

Un ambiente de desarrollo integrado IDE (en inglés: Integrated Development Environment) es un programa compuesto por un conjunto de herramientas para un programador. El mismo puede dedicarse a un solo lenguaje de programación o a varios.

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. J2EE entre sus principales ventajas 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.

1.6.2 Lenguajes de programación del lado del servidor

Los lenguajes del lado del servidor son aquellos lenguajes que son reconocidos, ejecutados e interpretados por el propio servidor y que se envían al cliente en un formato comprensible para él. En la actualidad existen un gran número de lenguajes, de los cuales se destacan atendiendo a la popularidad que presentan dentro de la comunidad de desarrollo de software, por la cantidad de sistemas que han sido desarrollados sobre ellos y de los resultados que se han obtenido mediante la explotación de estas aplicaciones, ellos son: C# desarrollado y estandarizado por Microsoft que soporta el paradigma orientado a objetos, PHP diseñado originalmente para la creación de páginas web dinámicas siendo además multiparadigma y finalmente Java que fue el seleccionado por el equipo de arquitectura del proyecto para su empleo en el desarrollo atendiendo a una serie de características.

Java

Java es un lenguaje multiplataforma de código abierto que proporciona un conjunto de clases para su uso en aplicaciones de red. Entre sus características se encuentran: es un lenguaje simple, seguro, orientado a objetos e independiente de la arquitectura. Java es una tecnología orientada al desarrollo de software con el cual podemos realizar cualquier tipo de programa. Es un lenguaje robusto que fue diseñado para crear software altamente fiable.

1.6.3 Lenguajes de programación del lado del cliente

En el caso de los lenguajes que sustentan la aplicación en el cliente, se definieron por parte del proyecto el uso de: HTML y JavaScript. HTML puede utilizar algunas funcionalidades de JavaScript, para la realización de una aplicación con la calidad requerida y para validar que la entrada de los datos sea correcta.

JavaScript

JavaScript un lenguaje interpretado, es decir que no requiere compilación y se utiliza comúnmente para la construcción de páginas web en combinación con el HTML, para el desarrollo de aplicaciones cliente-servidor dentro de Internet. Es un lenguaje muy utilizado que permite la validación de formularios dentro de una página, la personalización de la página por el usuario y la inclusión de datos del propio sistema (hora y fecha). Brinda respuesta a eventos locales dentro de la página y posibilita la realización de cálculos en tiempo real.

HTML

HTML (en inglés: Hypertext Markup Languaje) es un lenguaje simple utilizado para crear documentos de hipertexto para páginas web. HTML nos permite preparar documentos Web insertando en el texto de los mismos una serie de etiquetas que controlan los diferentes aspectos de la presentación y comportamiento de sus elementos. Se integra con lenguajes script como JavaScript, para lograr mayor interacción con los usuarios, esto se conoce como DHTML.

Después de un estudio de la metodología, los lenguajes y tecnologías que se utilizarán para la realización del diseño del sistema, se definen las herramientas con las que el sistema se debe desarrollar:

1.6.4 Entorno integrado de desarrollo

Eclipse 3.3 IDE

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

1.6.5 Contenedor Web

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.

1.6.6 Control de Versiones

SubVersion

SubVersion es un sistema de control de versiones de código abierto y gratuito. Es software libre bajo una licencia de tipo Apache/BSD (en inglés: Berkeley Software Distribution). Maneja ficheros y directorios a través del tiempo y la información radica en un árbol de ficheros en un repositorio central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Uno de sus principales objetivos es mantener íntegro el control de versiones. Se integra con el Eclipse mediante el plugin SubEclipse.

1.6.7 Base de Datos

Entre los gestores de base datos más utilizados están: Microsoft SQL Server que es una plataforma de base de datos y de análisis de datos que se utiliza en el procesamiento de transacciones en línea a gran escala, el almacenamiento de datos y las aplicaciones de comercio electrónico; Oracle que es considerado uno de los sistemas de bases de datos más completos, destacando su soporte de transacciones, estabilidad y escalabilidad; PostgreSQL sistema de gestión de base de datos relacional orientada a objetos, publicado bajo la licencia BSD de software libre y MySQL cuya arquitectura lo hace extremadamente rápido y fácil de personalizar.

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.

Microsoft SQL Server 2005

Microsoft SQL Server es un sistema para la gestión de bases de datos producido por Microsoft basado en el modelo relacional. Entre sus principales características se encuentra: soporte de transacciones, soporta procedimientos almacenados escalabilidad, estabilidad y seguridad. Sus lenguajes de consulta son el SQL y el T-SQL Apache (en inglés: Transact-SQL), siendo este último el principal medio de programación y administración del servidor. Además permite administrar información de otros servidores de datos.

1.7 Frameworks

Un framework es “una mini arquitectura reusable que provee una estructura y comportamiento genéricos para una familia de abstracciones de software [...]” [16]

Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo, como el uso de patrones.

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. El entorno de desarrollo J2EE cuenta con diferentes framework como Hibernate, Spring, Dojo, Tapestry, WebWork, entre otros. Específicamente para los lenguajes seleccionados con anterioridad existen un conjunto de ellos que se emplean con bastante frecuencia por los equipos de desarrollo.
Para lo referente al trabajo de acceso a datos se pueden mencionar a algunos de elevado prestigio como Athena Framework que es un marco de aplicaciones de código abierto para la plataforma Java que

soporta la interacción remota con Adobe Flex, Ebean ORM (en inglés: Object Relational Mapping) diseñado para simplificar el uso y entendimiento de JPA (en inglés: Java Persistence API) y JDO (en inglés: Java Data Objects) e Hibernate Framework que abstrae el trabajo con la base de datos, soportando la manipulación de los datos persistentes objetualmente, a través de ficheros que mapean clases Java contra tablas.

Por otro lado están un conjunto de frameworks que harán posible la aplicación del patrón MVC propuesto por la arquitectura base del proyecto SAGEB, y que a su vez poseen una elevada aceptación a nivel mundial, ellos son: Struts framework libre que permite reducir el tiempo de desarrollo y es compatible con todas las plataformas en las que Java Enterprise esté disponible, JBoss Seam framework desarrollado por JBoss donde se puede acceder a cualquier componente EJB (en inglés: Enterprise JavaBeans) desde la capa de presentación refiriéndote a él mediante su nombre de componente seam y Spring Framework que contiene en conjunto de librerías de clases que brindan una estructura conceptual y tecnológica para el desarrollo de aplicaciones de la plataforma J2EE, vale destacar que los dos últimos presentan un marcado uso en la universidad.

La lógica de presentación se ve favorecida al igual por la utilización de frameworks, potenciado principalmente por la importancia que posee esta en los sistemas informáticos debido a que con ellas es con la que interactúan los usuarios. Disímiles son las librerías desarrolladas que emplean JavaScript para enriquecer la interfaz de usuario y brindar un ambiente de trabajo más limpio en el uso de AJAX (en inglés: Asynchronous JavaScript And XML), entre ellas se destacan Qooxdoo de código abierto para el desarrollo de aplicaciones web interactivas que permite al desarrollador abstraerse del HTML, CSS (en inglés: Cascading Style Sheets) y DOM (en inglés: Document Object Model) de la aplicación, Ext JS desarrollada por Sencha con el fin de desarrollar aplicaciones web interactivas usando tecnologías AJAX y DOM que puede ejecutarse como una aplicación independiente; y Dojo Toolkit que contiene APIs (en inglés: Application Programming Interface) y widgets (controles) además de la integración con tecnologías AJAX permitiendo realizar peticiones asincrónicas al servidor.
En el desarrollo de la arquitectura base del SAGEB se decidió utilizar ateniendo a las características del proyecto y las bondades que brindan cada uno: el framework de aplicación Spring que al igual que el framework de soporte Hibernate son muy populares, por sus múltiples ventajas en el desarrollo de

aplicaciones Web dentro de la plataforma J2EE, y las librerías de Dojo Toolkit para facilitar el trabajo en la capa de presentación por la parte del cliente.

1.7.1 Spring

Spring es un framework de aplicaciones Java/J2EE desarrollado usando licencia de código abierto. Es potente en cuanto a la gestión del ciclo de vida de los componentes y fácilmente ampliable. El framework está dividido en módulos para su correcto funcionamiento, sean estos:

- **Spring Core:** Representa el núcleo de Spring, es donde se encuentran funcionalidades fundamentales que provee el framework.
- **Spring AOP:** Ofrece un extenso soporte para Programación Orientada a Aspectos, permitiendo definir entre otras cosas la política transaccional y de seguridad de una aplicación.
- **Spring ORM:** Brinda el soporte necesario para la integración con los frameworks Hibernate e iBates.
- **Spring DAO:** Provee un trabajo con JDBC (en inglés: Java Database Connectivity) en aras de hacer el código de acceso a datos más limpio y entendible, ofrece además una capa de excepciones para el manejo de los errores emitidos por los servidores de base de datos.
- **Spring Web:** Es el encargado de crear el contexto para aplicaciones web, incluye soporte para una variedad de tareas como la subida de archivos, la vinculación de parámetros de las peticiones a objetos del negocio, etc.
- **Spring Web MVC:** Ofrece gran soporte para el desarrollo de aplicaciones web utilizando la filosofía Modelo-Vista-Controlador, emplea la inversión de control para brindar una separación entre la lógica de los controladores y los objetos del negocio.
- **Spring Context:** Es el que consagra a Spring como un framework, ofrece soporte para la internacionalización, la aplicación de eventos de ciclo de vida. Brinda soporte a servicios como: correo electrónico, acceso JNDI (en inglés: Java Naming and Directory Interface), integración con EJB, etc.

1.7.2 Spring WebFlow

Spring WebFlow es un framework y a su vez un subproyecto de Spring dirigido a definir y gestionar los flujos de páginas dentro de una aplicación web. Los flujos Web en este framework son diseñados para ser auto-controlados, dando la posibilidad de definir reglas de navegación múltiples y complejas. [17]

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. [18]

1.7.4 Dojo Toolkit

Dojo Toolkit es una fuente abierta modular de librería JavaScript (o, más concretamente JavaScript Toolkit) diseñado para facilitar el rápido desarrollo de plataforma cruzada, JavaScript / Ajax basado en aplicaciones y sitios web. Provee un conjunto de componentes de interfaz gráfica de usuario, entre los que se encuentran los calendarios y menús, que se pueden insertar de manera sencilla en páginas HTML. [19]

1.8 Conclusiones Parciales

Como resultado del estudio de sistemas informáticos contables a nivel mundial que proporcionan funcionalidades que puede utilizar el BNC para la gestión de la Negociación de las Cartas de Crédito se concluyó que los existentes en el mundo constituyen sistemas propietarios extremadamente costosos, lo que implica una dependencia de sus fabricantes para la realización de los mantenimientos, modificaciones

y adecuaciones, y al mismo tiempo, al no tener acceso a sus fuentes, en las condiciones político-económicas en que se desarrolla Cuba, no permiten garantizar la seguridad informática que requiere la información y los procesos que el BNC ejecuta y que son vitales para la economía del país. Asimismo, el sistema informático nacional existente (SABIC), además de no poseer todas las funcionalidades que en la actualidad se requieren para el BNC y en particular en las relacionadas con la gestión de la Negociación de las Cartas de Crédito, se encuentra implementado en un sistema operativo que no resulta funcional con el desarrollo de las nuevas tecnologías informáticas. Estos elementos, conjugado con el alto potencial técnico informático alcanzado en nuestro país, nos permite justificar la necesidad del desarrollo de un nuevo sistema automatizado de gestión de entidades bancarias (Quarxo) que cumpla con los nuevos requerimientos, y sobre la base del cual se realizará el diseño e implementación de los módulos Documentos de Embarque, Discrepancia y Negociación, a partir de la definición de las funcionalidades.

Basado en el hecho de que Quarxo es una aplicación web de gestión empresarial, las principales herramientas y tecnologías propuestas permiten desarrollar un buen diseño. La aplicación en desarrollo está solicitada para que se ejecute sobre un ambiente libre. De esta manera las herramientas y tecnologías utilizadas en su mayoría son libres y multiplataforma, así como los frameworks Spring e Hibernate. Esto trae como ventaja una reducción notable del costo del sistema por concepto de licencias de software y soporte.

Capítulo 2 – Arquitectura y Diseño de los módulos Documentos de Embarque, Discrepancia y Negociación.

2.1 Introducción

En el siguiente capítulo se proporciona una breve descripción de los elementos fundamentales de la arquitectura utilizada para desarrollar los módulos Documentos de Embarque, Discrepancia y Negociación, la cual está regida por la arquitectura base del SAGEB, además se le da solución al diseño de los módulos antes mencionados y la descripción de los artefactos del diseño que se generan durante el flujo de trabajo como son: modelo de diseño, el modelo de datos, entre otros.

2.2 Estilo arquitectónico

Como se había mencionado en el capítulo anterior el sistema se desarrolla bajo la tecnología de Java Enterprise Edition donde esta plataforma define un modelo de programación encaminado a la creación de aplicaciones basadas en capas.

La arquitectura en capas proporciona grandes ventajas en cuanto al mantenimiento, extensibilidad y reutilización de componentes o artefactos ya que al tener divididas las capas reduce la dependencia entre las mismas lo que hace mucho más fácil realizar modificaciones sin afectar las demás.

La arquitectura definida para el desarrollo del sistema Quarxo está basada en una arquitectura de tres capas lógicas fundamentales: Capa de Presentación, Capa de Lógica de Negocio, y Capa de Acceso a Datos y una capa transversal a las otras con los objetos del dominio, como se muestra a continuación:

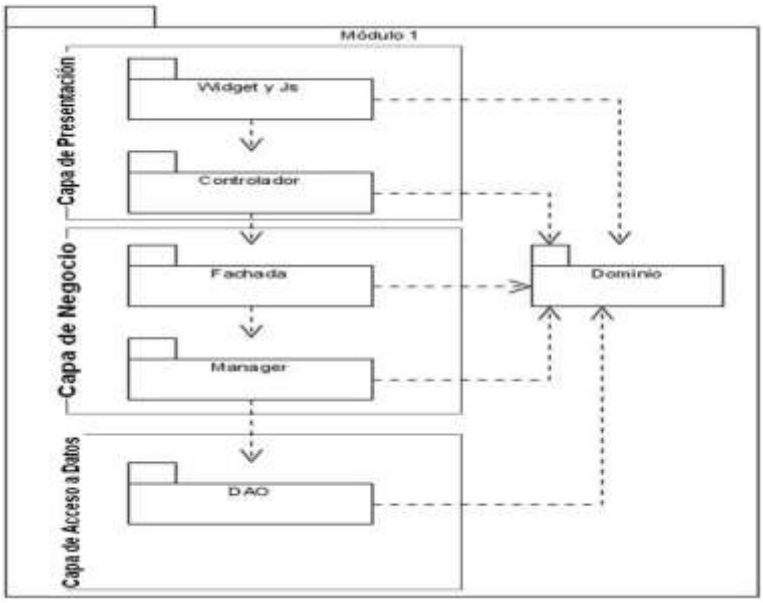

Figura 2.1 Capas lógicas de la Arquitectura Base del SAGEB

2.2.1 Capa de presentación

Esta capa es la encargada de interactuar con el usuario, obtiene todos los pedidos de la interfaz, valida y envía los datos al servidor y además, controla la apariencia visual que tendrá la aplicación. Contendrá los componentes que tengan la responsabilidad de manejar la lógica de presentación de la información que se gestiona. Para el trabajo en esta capa se utiliza del framework Spring, *Spring MVC* para atender las peticiones que ocurren desde el cliente y *Spring WebFlow* siempre y cuando el caso de uso que se desarrolle presente un flujo complejo y/o no lineal, además se utiliza la librería *Dojo* para generar las interfaces que interactuarán con el usuario. La capa de presentación se comunica con la capa de lógica de negocio a la cual envía todas las solicitudes de los usuarios y recibe las respuestas después de que hayan sido procesadas cada una de estas solicitudes y con la capa de dominio para generar las entidades del negocio.

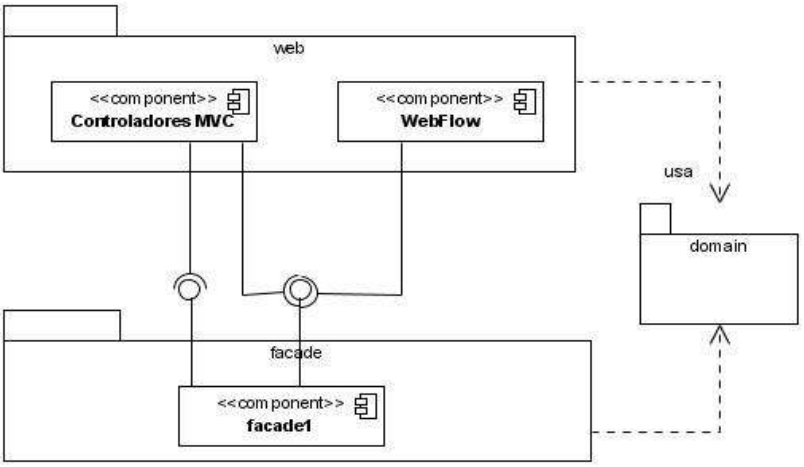

Figura 2.2 Arquitectura de la capa de presentación en el lado del servidor.

2.2.2 Capa de lógica de negocio

Esta capa es la encargada de modelar la lógica de negocio que dará solución a cada una de las funcionalidades de la aplicación, y en ella se establecen las reglas o restricciones que debe cumplir la misma. Contendrá todas las entidades de dominio y los componentes que tengan la responsabilidad de manejar la lógica de negocio de los mismos. Utiliza el framework Spring para representar las clases y sus relaciones, además de la política de transacciones que propone dicho contenedor para evitar inconsistencia en los datos, *Spring AOP* para ejecutar las transacciones y la auditoría de cada uno de los métodos y *Spring Security* para aplicar seguridad a nivel de métodos.

Se encuentra dividida en dos subcapas: capa Facade y capa Manager.
La subcapa Facade no contiene lógica de negocio sino que representa la interacción entre las capas de presentación y la de lógica de negocio. La subcapa Facade delega a la subcapa Manager la realización de la lógica del negocio.

La subcapa Manager contiene las clases de la aplicación que se especializan en un conjunto de funcionalidades que representan el negocio sobre una o varias entidades. Esta subcapa utiliza la capa de acceso a datos para obtener los datos persistidos; la capa de dominio para generar las entidades del negocio y la capa Facade de otros módulos para utilizar las funcionalidades que sean necesarias.

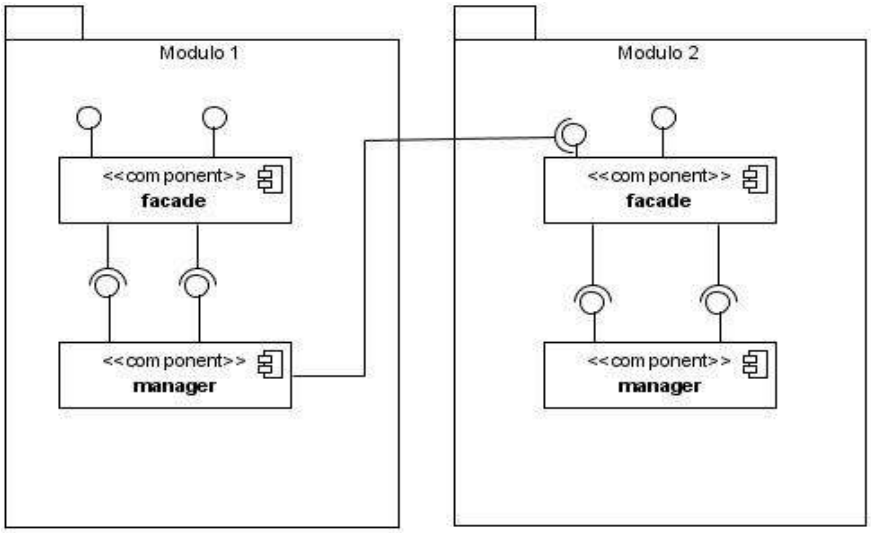

Figura 2.3 Arquitectura de la capa de negocio.

2.2.3 Capa de acceso a datos

Esta capa es la encargada de realizar las operaciones básicas con la base de datos tales como: persistir, consultar, actualizar y eliminar los objetos de dominio recibidos de la capa lógica y la invocación a funciones y procedimientos almacenados. Contendrá todos los objetos que encapsulan la lógica de acceso a datos. Para el trabajo en esta capa se utiliza del framework Spring, Spring con *Hibernate* para las operaciones con las tablas de la base de datos y Spring *JDBC* para las llamadas a funciones y procedimientos almacenados, además se utiliza Spring *ORM* y Spring *DAO* para soportar la integración con Hibernate y la utilización del patrón DAO para el desarrollo de la capa respectivamente.

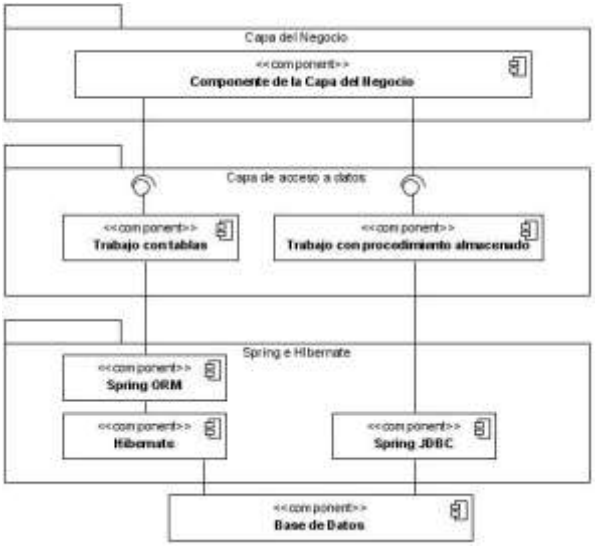

Figura 2.4 Arquitectura de la capa de acceso a datos.

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

2.3 Análisis de las funcionalidades

Las funcionalidades a diseñar e implementar descritas en la captura de requisitos por los analistas de SAGEB para los módulos Documentos de Embarque, Discrepancia y Negociación del subsistema de Cartas de Crédito son:

Caso de uso: Gestionar Documentos de Embarque:

- **Registrar documentos de embarque:** registra los datos de los documentos de embarque asociados a una carta de crédito.
- **Actualizar documentos de embarque:** modifica el documento de embarque registrado.
- **Consultar documentos de embarque:** consulta los detalles del documento de embarque asociados a una carta de crédito.
- **Eliminar documentos de embarque:** elimina el documento de embarque registrado a una carta de crédito.

Caso de uso: Gestionar Discrepancia:

- **Registrar discrepancia:** registra la contabilidad de la discrepancia asociada a un documento de embarque registrado.
- **Actualizar discrepancia:** modifica la contabilidad de la discrepancia de un documento de embarque registrado.
- **Consultar discrepancia:** consulta los detalles de la discrepancia del documento de embarque asociados a una carta de crédito.
- **Cancelar discrepancia:** cancela la contabilidad de la discrepancia asociada al documento de embarque registrado.

Caso de uso: Gestionar Negociación:

- **Registrar negociación:** registra los datos y la contabilidad de la negociación de un documento de embarque asociado una carta de crédito.
- **Actualizar negociación:** modifica los datos y la contabilidad de la negociación registrada.
- **Consultar negociación:** consulta los detalles de la negociación del documento de embarque registrado.
- **Unir negociaciones:** une los datos y la contabilidad de dos o más negociaciones registradas.

Las funcionalidades expuestas se encuentran especificadas por escrito de forma clara, precisa, completa, consistente, sin ambigüedad y contienen los prototipos de interfaz de usuario completos que han sido esenciales para el entendimiento del equipo de desarrollo. La especificación de requisitos realizado por los analistas de proyecto SAGEB cumple con la calidad requerida y han sido validados por el cliente, constituyendo el elemento fundamental para el diseño de los módulos Documentos de Embarque, Discrepancia y Negociación.

2.4 Diseño de la solución

“En el diseño se modela el sistema y se define su forma (incluida la arquitectura) para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se le suponen. Una entrada esencial

en el diseño es el resultado del análisis, o sea el modelo de análisis, que proporciona una comprensión detallada de los requisitos.” [13]

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, diagramas de clases y diagramas de interacción. Además sirve como representación para la implementación.

2.4.1.1 Estructura de paquetes de clases del diseño

Para ganar en organización en el desarrollo y en el despliegue del sistema, se agruparon los módulos y componentes por subsistema. Cada subsistema tendrá uno o más módulos y/o componentes estrechamente relacionados con las funcionalidades que ejecutan. Los módulos agrupan un conjunto de casos de uso relacionados con uno o más procesos bancarios; y los componentes representan un grupo de funcionalidades comunes que serán reutilizados por otros módulos del sistema. Los módulos y/o componentes estarán separados por las diferentes capas lógicas según la naturaleza de los mismos.

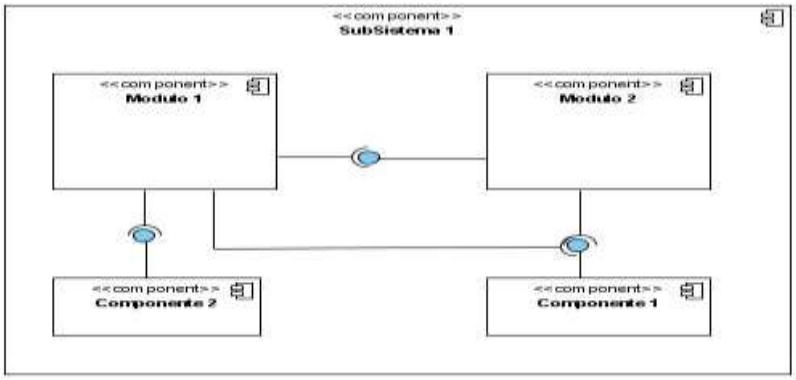

Figura 2.5 Arquitectura de un subsistema de SAGEB.

Un diagrama de paquetes es una colección de clases, relaciones, realizaciones de casos de uso, diagramas y otros paquetes que estén de alguna forma relacionados. Es usado para estructurar el modelo de diseño dividiéndolo en partes más pequeñas, como herramienta organizacional del modelo para agrupar elementos relacionados.

A continuación se muestra la estructura y dependencia de paquetes del módulo Negociación del subsistema Carta de Crédito y una explicación de la composición de cada uno.

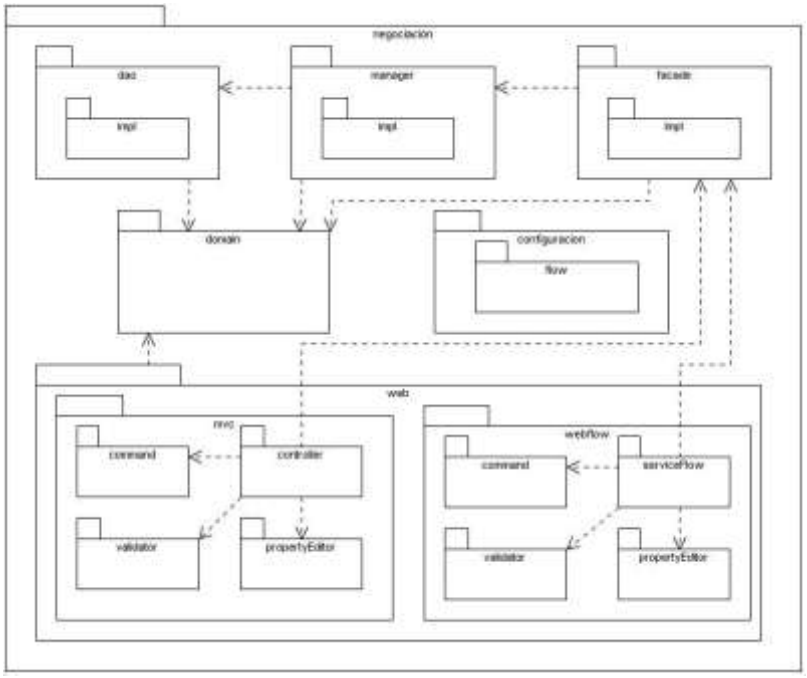

Figura 2.6 Modelo de Paquetes del módulo Negociación del subsistema Carta de Créditos

Descripción de cada uno de los paquetes utilizados

Paquete web: Contiene un grupo de clases y paquetes que son los encargados de realizar todo lo referente a la presentación en el lado del servidor. Este paquete se divide en dos sub-paquetes mvc y webflow.

Paquete mvc: Contiene los paquetes que manejan la lógica de presentación en el servidor para SpringMVC.

Paquete webflow: Contiene los paquetes que manejan la lógica de presentación en el servidor para SpringWebFlow.

Paquete command: Contiene las clases para mapear datos que se introducen desde el cliente a objetos del negocio.

Paquete controller: Contiene las clases que heredan de las clases "Controller" que brinda Spring MVC para responder a las peticiones realizadas por el cliente.

Paquete validator: Contiene las clases para desarrollar las validaciones en el servidor.

Paquete propertyEditor: Contiene las clases que convierten ciertos datos en objetos determinados.

Paquete serviceFlow: Contiene las diferentes clases para establecer la comunicación entre el flujo y la fachada del módulo.

Paquete manager: Contiene las interfaces e implementaciones que brindan las funcionalidades del negocio del módulo correspondiente.

Paquete facade: Contiene la interface y la implementación que brindan las funcionalidades que se consumirán por otros módulos o por la capa de presentación.

Paquete domain: Contiene las clases del dominio del módulo.

Paquete dao: Contiene las interfaces y las clases de implementación del acceso a datos del módulo, además los maps que son los ficheros de mapeos de hibernate.

Paquete configuration: Contiene los ficheros de configuración del módulo en formato XML de los diferentes contextos de Spring, en forma de mapeo de peticiones, controladores y vistas.

- *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 flujos.xml para Spring WebFlow del módulo correspondiente.

2.4.1.2 Diagrama de clases del diseño

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la

información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro.

A continuación se muestra el modelo de diseño de las diferentes capas para el módulo Negociación:

Diseño de la capa de presentación

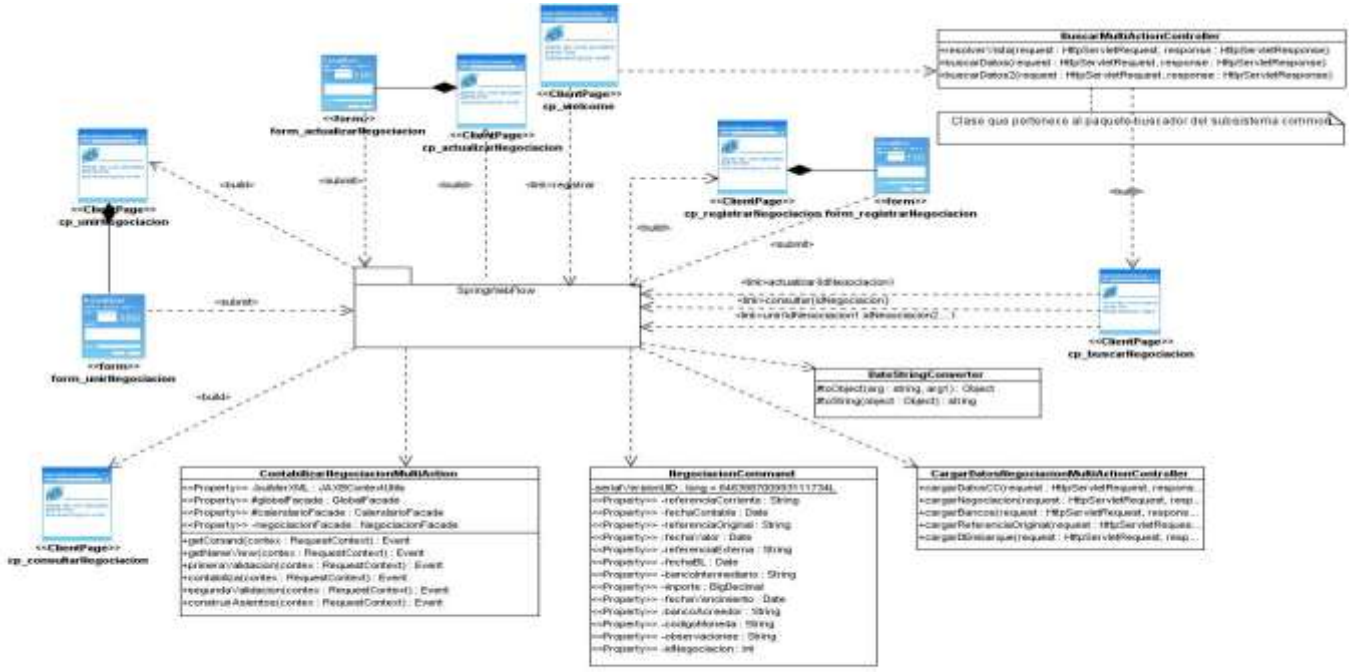

Figura 2.7 Diseño de la capa de presentación del módulo Negociación del subsistema Carta de Créditos.

Las clases **ClientPage** son las páginas web encargadas de mostrar los formularios de información al usuario. El paquete **SpringWebFlow** representa el mecanismo interno con el que SpringWebFlow 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, actualizar, consultar y unir, SpringWebFlow interactúa con ella para su funcionamiento.

Diseño de la capa de negocio

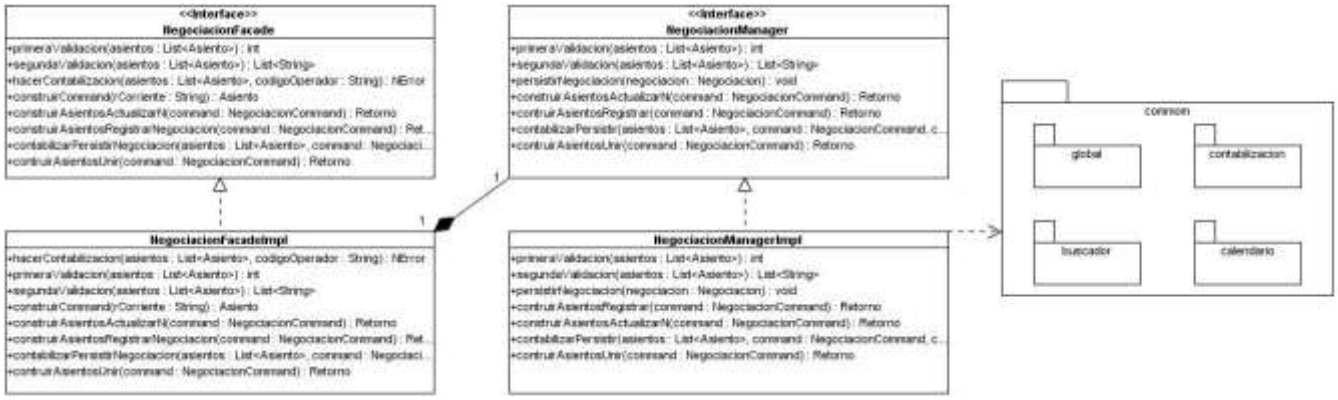

Figura 2.8 Diseño de la capa de negocio del módulo Negociación del subsistema Carta de Créditos.

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 diferentes subsistemas.

Diseño de la capa de acceso a datos

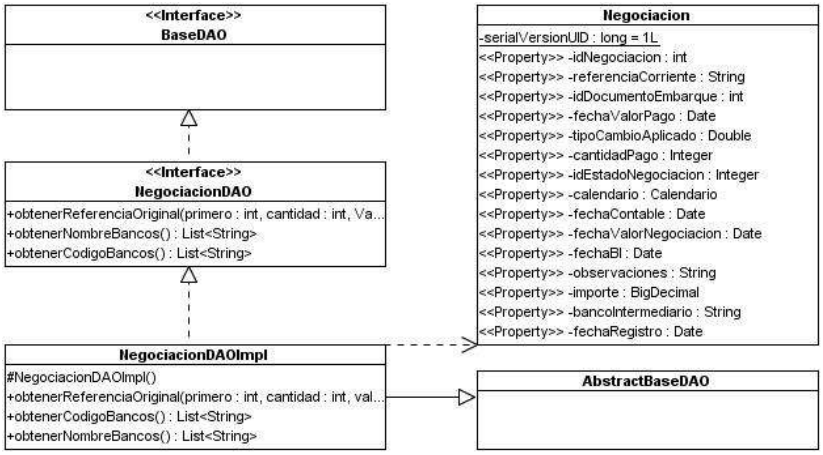

Figura 2.9 Diseño de la capa de acceso a datos del módulo Negociación del subsistema Carta de Créditos.

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 persiste en la base de datos asociada a las negociaciones, las clases **BaseDAO** y **AbstractBaseDAO** son respectivamente una interfaz y una clase abstracta brindadas por el DAO genérico utilizado en la aplicación. Ellas brindan un conjunto de funcionalidades básicas que se realizan con las clases persistentes, ya sea buscar por identificador, listar, persistir, actualizar y eliminar. Básicamente solo se necesita que las interfaces del acceso a datos implementen BaseDAO y las implementaciones hereden de AbstractBaseDAO.

2.4.1.3 Diagrama de interacción del diseño

Los diagramas de interacción capturan el comportamiento de los casos de uso mostrando la manera en que interactúan un grupo de objetos entre sí. Representa la forma en que un cliente u objetos se comunican entre ellos en petición a un evento. Existen dos tipos de diagramas de interacción: el Diagrama de Secuencia y el Diagrama de Colaboración.

En el flujo de diseño se utiliza fundamentalmente el diagrama de secuencia. Un diagrama de secuencia muestra las interacciones entre objetos, ordenadas en secuencia temporal durante un escenario concreto. Si los casos de uso tienen varios flujos o subflujos distintos, suele ser útil crear un diagrama de secuencia para cada uno de ellos, por esta razón se realizó un diagrama de secuencia para la petición de cargar datos y otra para la petición de persistirlos en cada módulo.

A continuación se encuentran los diagramas de clases del diseño para las funcionalidades del módulo Negociación.

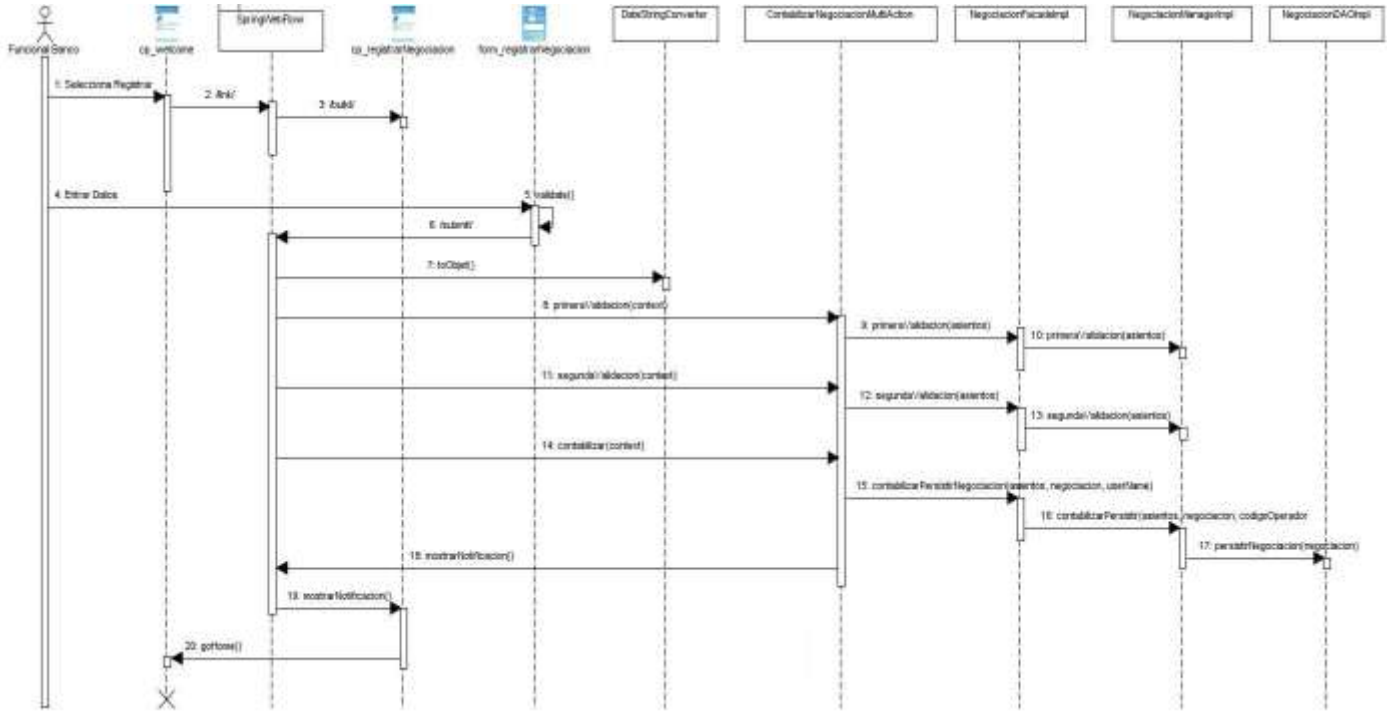

Figura 2.10 Diagrama de Secuencia: Registrar Negociación.

En el diagrama correspondiente al escenario Registrar Negociación, el actor que es el funcional del banco selecciona la opción registrar en la página cliente **welcome**, esta realiza una petición mediante un vínculo al motor de **SpringWebFlow** el cual se encarga de construir la página cliente **registrarNegociacion**. El actor mediante el formulario de registro contenido en la página registrarNegociacion introduce los datos necesarios y presiona aceptar. Estos datos son validados por el mismo formulario y enviados al motor de SpringWebFlow, quien a su vez hace uso del propertyEditor **DateStringConverter** para convertir las

fechas en objetos. Posteriormente debido a que este escenario contempla la contabilización, se le solicita a **ContabilizarNegociacionMultiaction** una primera validación, este a su vez la solicita **a NegociacionFacadeImpl** y este lo pide a **NegociacionManagerImpl**. De la misma forma se realiza una segunda validación pasando por los mismos objetos que la primera. Consecutivamente se pasa a contabilizar. SpringWebFlow hace la solicitud a ContabilizarNegociacionMultiaction, el cual la hace a NegociacionFacadeImpl y este a NegociacionManagerImpl el cual contabiliza y solicita a **NegociacionDAOImpl** que persista la negociación. Para finalizar se envía una notificación al actor y se redirecciona a la página cliente donde empezó el flujo.

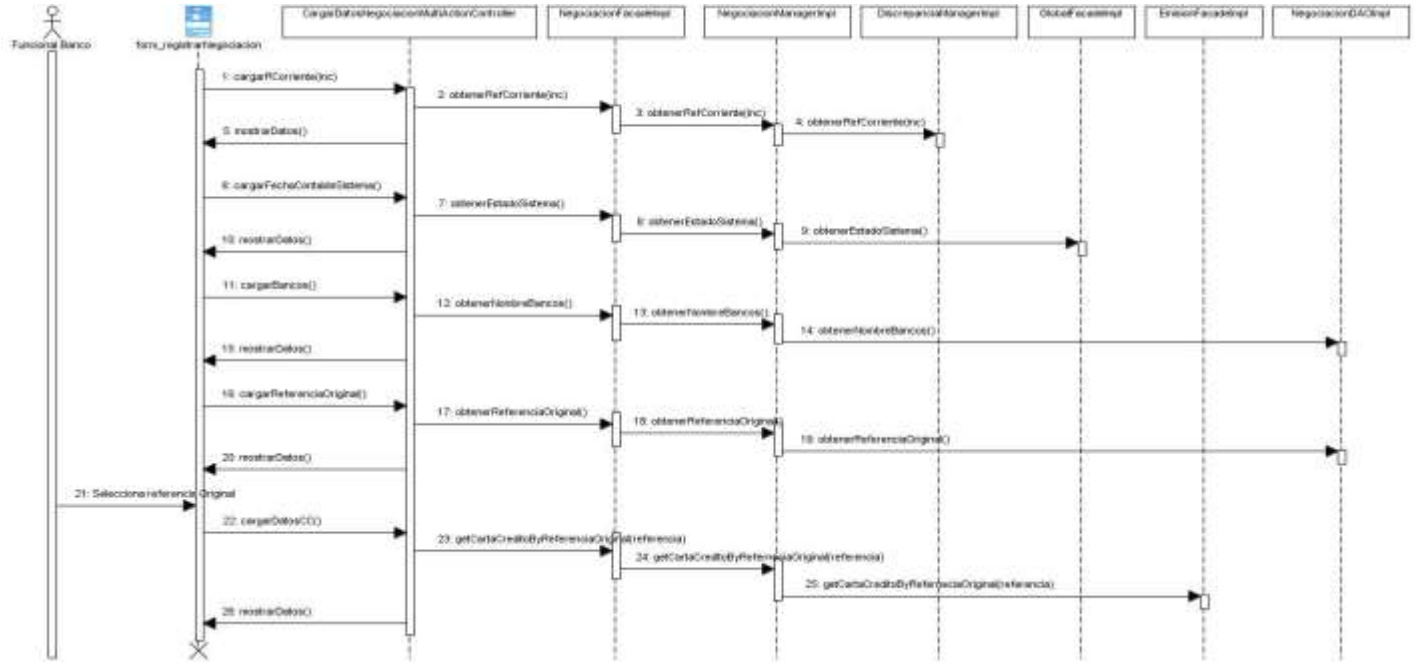

Figura 2.11 Diagrama de Secuencia: Registrar Negociación (Cargar Datos).

Este diagrama se realizó para restarle complejidad al primero y se encarga de modelar la carga de los datos del escenario Registrar Negociación. Comienza cuando SpringWebFlow construye la página cliente **registrarNegociacion**. Una vez cargada esta, solicita a **CargarDatosNegociacionMultiAction** cargar la referencia corriente, la cual utiliza a **NegociacionFacadeImpl**, ésta a su vez a **NegociacionManagerImpl** la que pide a **DiscrepanciaManagerImpl** y finalmente se muestra al actor. Conjuntamente se solicita a **CargarDatosNegociacionMultiAction** cargar la fecha contable, la cual se la pide al

NegociacionFacadeImpl ésta a su vez utiliza a **NegociacionManagerImpl** y este a **GlobalFacade** y finalmente se muestra al actor.

También se le solicita a **CargarDatosNegociacionMultiAction** cargar todos los bancos, la cual utiliza a **NegociacionFacadeImpl** ésta a su vez utiliza a **NegociacionManagerImpl** la que pide a **NegociacionDAOImpl** que se encarga de listar todos los bancos y finalmente se le muestra al actor. A la vez se le solicita a **CargarDatosNegociacionMultiAction** cargar las referencias originales la cual utiliza a **NegociacionFacadeImpl** ésta a su vez se la pide a **NegociacionManagerImpl** la que utiliza a **NegociacionDAOImpl** que se encarga de listar todas las referencias y finalmente se le muestra al actor. Luego el actor selecciona la referencia original, y automáticamente se invoca al **CargarDatosNegociacionMultiAction** para obtener, según la referencia seleccionada la carta de crédito a la que pertenece, el cual solicita a **NegociacionFacadeImpl** y este a **NegociacionManagerImpl,** que a su vez se lo hace de **EmisionFacadeImpl,** teniendo como resultado que se le muestre al actor la moneda, la fecha de vencimiento y las referencias externas de los documentos de embarques que la carta de crédito tiene asociados. Así concluye este escenario quedando listo la página para continuar con el flujo de sucesos de de Registrar Negociación.

2.4.2 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.

Teniendo en cuenta que el sistema Quarxo se desarrolló sobre el núcleo del SABIC se decidió por petición del cliente trabajar con dicha base de datos lo que provocó que fuera necesario agregar nuevas tablas

para garantizar la persistencia de las entidades implicadas en los módulos Documentos de Embarque, Discrepancia y Negociación. A continuación se muestra la estructura de las entidades relacionadas con Negociación y las relaciones existentes entre ellas.

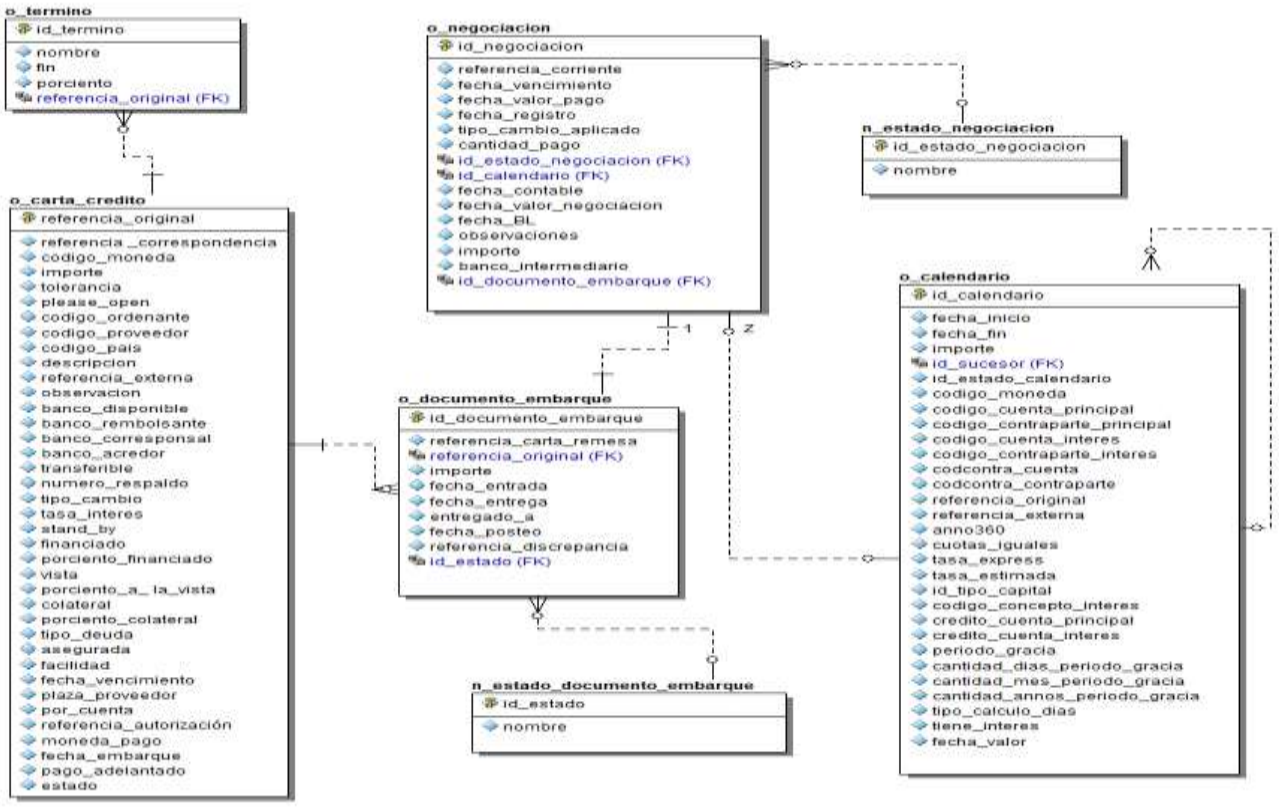

Figura 2.12 Modelo de Datos de los módulos Documentos de Embarque, Discrepancia y Negociación.

Teniendo en cuenta las funcionalidades de los módulos Documentos de Embarque y Negociación fue necesaria la creación de la tabla operativa **o_documento_embarque** donde se almacenan los documentos permitiendo las funcionalidades básicas necesarias para operar sobre los mismos (insertar, actualizar, eliminar) y presenta una relación de multiplicidad uno a muchos con el nomenclador **n_estado_documento_de_embarque** que indica el estado (Registrado, Discrepado y Negociado), así como una relación de multiplicidad uno a uno con el operativo **o_negociacion,** dicha tabla es para almacenar los datos de las negociaciones de las cartas de crédito unas vez que son contabilizadas, ésta a su vez posee relación de multiplicidad uno a mucho con el nomenclador **n_estado_negociacion** que

indica el estado(Registrada, Unida y Cancelada), y de uno a uno con la tabla **o_calendario** definida para el componente Calendario.

El módulo Discrepancia permite contabilizar las discrepancias asociadas a un documento de embarque, para ello se acreditan y debitan las cuentas obligaciones con banco y obligaciones con clientes respectivamente con el importe del documento. Es importante resaltar que por petición del cliente no es necesario un operativo para estos datos ya que pueden variar frecuentemente pues los documentos de embarque se pueden negociar estando discrepados o no. Para ello el banco emisor (y, obviamente, el importador) debe autorizar a pagar a pesar de las discrepancias.

2.4.3 Patrones de diseño empleados

Según el estudio realizado sobre los patrones de diseño para darle soluciones sencillas a los problemas comunes del diseño orientado a objetos, se emplearon en la solución del diseño los siguientes:

De los patrones **GRASP**, que son los que describen los principios fundamentales de la asignación de responsabilidades a objetos expresados en forma de patrones, la utilización del patrón **experto** se evidencia en la clase NegociacionDAO ya que esta clase es la única que dispone de la información necesaria para realizar todas las operaciones de las negociaciones ya sea persistir, consultar y actualizar. El patrón **controlador** se evidencia con la clase controladora CargarDatosNegociacionMultiAction, debido a que funciona como intermediaria entre la interfaz de usuario y el negocio; presenta la responsabilidad de recibir o manejar un evento del sistema que es el de cargar los datos del sistema que serán mostrados al usuario. El patrón de **bajo acoplamiento** se demuestra con la definición de interfaces e implementaciones, ejemplo de ello NegociacionFacade y NegociacionFacadeImpl permiten que CargarDatosNegociacionMultiAction y ContabilizarNegociacionMultiAction que son clases de la capa de presentación se relacionen solo con ellas para realizar sus operaciones, así presentan menos dependencias lo que resulta mejor para el mantenimiento y reutilización del código. El patrón de **alta cohesión** fue utilizado en el diseño de los módulos de manera general; donde se agruparon las clases en dependencia de los requerimientos a los que se les debía dar respuesta. Debido a que los casos de uso identificados (registrar negociación, actualizar negociación, etc.) guardaban una estrecha relación entre

ellos en cuanto a negocio, se agruparon en el módulo Negociación, por lo que sus implementaciones se encuentran en la misma área funcional.

De los patrones **estructurales de GoF** que son los que describen cómo las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades, el patrón **fachada** se evidencia en la utilización de la interfaz NegociacionFacade responsable de la comunicación entre la presentación y el grupo de clases e interfaces más complejas que se encargan de la lógica de negocio y el acceso a datos, reduciendo las dependencias en el sistema entre la parte del servidor y la parte del cliente.

El patrón **acceso a datos (DAO)** se evidencia con la definición de interfaz NegociacionDAO que maneja los datos de la entidad persistente Negociación en la base de datos, es responsable de separar la lógica de negocio de la de acceso a datos, permitiendo que los objetos del dominio del negocio no manejen la implementación de sus fuentes de datos, por lo que la complejidad de estos objetos es mínima.

El patrón **MVC (Modelo Vista Controlador)** donde su objetivo fundamental es separar el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario, se evidencia su aplicación en la arquitectura dividida en tres capas, utilizada en el proyecto SAGEB. La capa de negocio que contiene las interfaces e implementaciones de NegociacionFacade y NegociacionManager y la capa de acceso a datos que incluye la interfaz y la implementación de NegociacionDAO constituyen el Modelo. La capa de presentación en la parte del cliente que contiene las paginas JSP (en inglés: Java Server Pages) como registrarNegociacion.jsp que representan las interfaces de usuario, constituyen la Vista; y la capa de presentación en el lado del servidor, que engloba el conjunto de clases que hacen uso de los controladores que heredan de Spring MVC y de Spring WebFlow (CargarDatosNegociacionMultiAction y ContabilizarNegociacionMultiAction respectivamente), constituye el Controlador. La vista maneja la visualización de la información, el modelo administra el comportamiento y los datos del dominio de aplicación y el controlador actúa de intermediario entre la vista y el modelo.

El patrón de presentación **Composite View (Vistas Compuestas)** se evidencia en páginas JSP que

incluyen otras usando la directiva *include*. Como ejemplo se encuentra la página registrarNegociacion.jsp, la cual utiliza en su formulario el componente comisiones.jsp que representa otra página.

2.5 Conclusiones Parciales

En este capítulo se realizó el análisis de la arquitectura base definida por el proyecto SAGEB y el diseño de los módulos Documentos de Embarque, Discrepancia y Negociación 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 entidades involucradas en la solución de los módulos. Todo lo realizado incide de manera notable en el siguiente capítulo, ya que se logra una uniformidad y organización, además de proporcionar una entrada apropiada como punto de partida a la hora de implementar la solución propuesta.

Capítulo 3 – Implementación de los módulos Documentos de Embarque, Discrepancia y Negociación.

3.1 Introducción

En el último capítulo se da solución a los módulos Documentos de Embarque, Discrepancia y Negociación conforme a la implementación, flujo de trabajo importante propuesto por RUP. Se obtendrán un conjunto de artefactos que serán de gran valor para las posteriores etapas del desarrollo como: el modelo de componentes, los estándares de codificación, la descripción de clases, métodos y atributos implementados más relevantes del modelo de diseño presentado en el capítulo anterior. Su importancia se debe a que se obtiene como consecuencia un sistema ejecutable, siendo uno de los principales objetivos en el desarrollo de software.

Para finalizar el capítulo se realizará la validación de la solución mediante la aplicación de las Pruebas de Unidad en aras de localizar errores que imposibiliten el cumplimiento de los requisitos funcionales y las perspectivas del cliente, además de evaluar la calidad del producto que se desarrolla.

3.2 Modelo de Implementación

“El modelo de implementación describe como los elementos del modelo del diseño, como las clases se implementan en términos de componentes, como ficheros de código fuente, ejecutables, etc. El modelo de implementación describe también como se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y cómo dependen los componentes unos de otros.” [14]

3.2.1 Diagrama de Componentes

Los Diagramas de Componentes modelan la vista estática de un sistema. Se representa como un grafo de componentes de software unidos por medio de relaciones de dependencia (compilación, ejecución), pudiendo mostrarse las interfaces que estos soporten. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes, por lo cual cada diagrama describe un apartado del sistema.

El diagrama de componentes que se muestra a continuación se ha elaborado de forma tal que muestre las relaciones existentes entre los módulos Documentos de Embarque, Discrepancia y Negociación y el resto de componentes de empleo general realizados en el transcurso del proyecto.

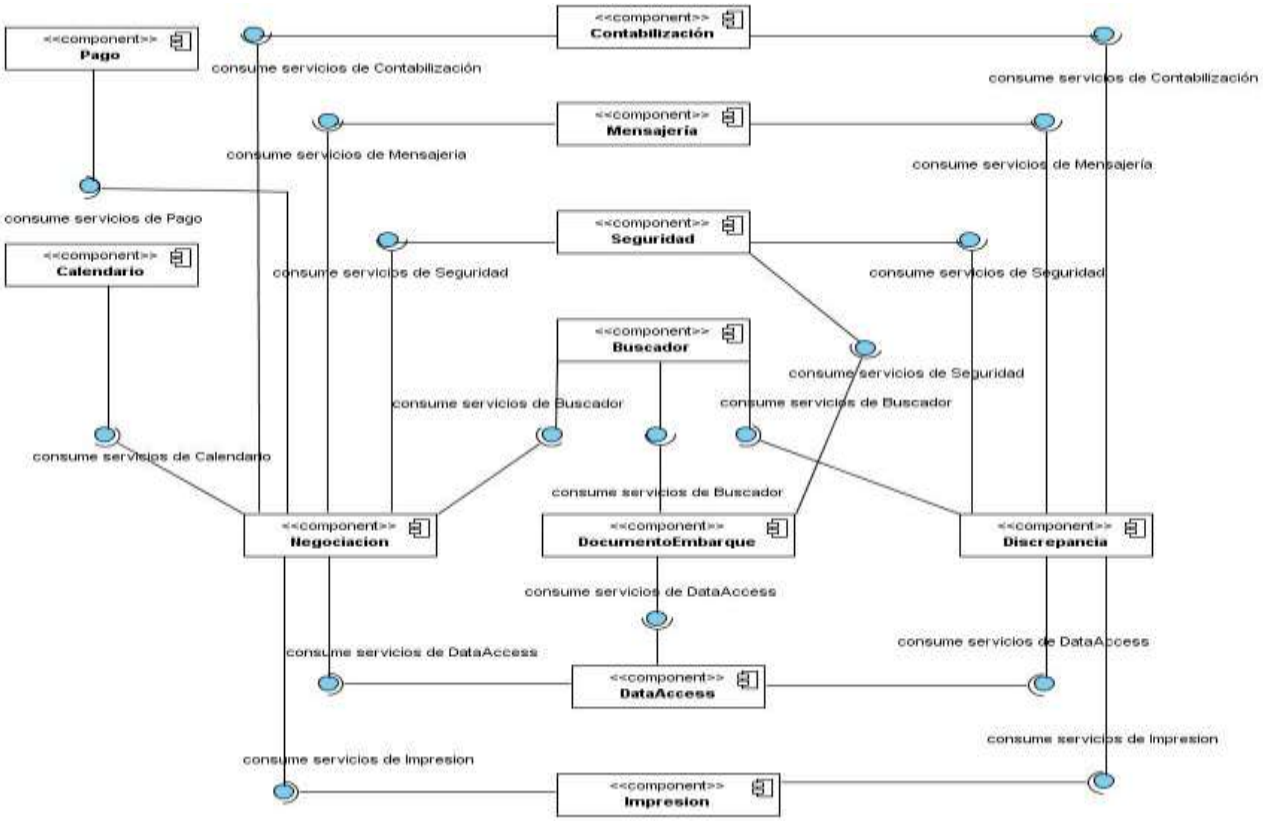

Figura 3.1 Modelo de Componentes de los módulos Documentos de Embarque, Discrepancia y Negociación.

A continuación se realiza una descripción más detallada de la funcionalidad de cada uno de estos componentes a partir de los servicios que brindan y reciben:

- **Buscador:** Brinda un conjunto de clases que constituyen el motor de búsqueda, así como una interfaz gráfica encargada de la búsqueda de diferentes conceptos en dependencia del lugar donde se emplee.
- **Seguridad:** Se encarga de mantener la seguridad en todo el sistema a través de la comprobación de las peticiones y sus permisos en dependencia del usuario que la realice.
- **Mensajería:** Brinda un conjunto de clases necesarias para el envió de mensajes SWIFT tras la realización de operaciones bancarias que requieran la notificación a los bancos implicados en ella.
- **Contabilización:** Brinda un conjunto de clases necesarias para la realizar la contabilización de determinadas operaciones que así lo requieran, así como otro grupo de clases e interfaz gráfica para el cobro de comisiones que se asocien a determinada operación.
- **Calendario:** Brinda un conjunto de clases además de una interfaz gráfica encargadas de gestionar los elementos necesarios para la creación de los cronogramas de compromisos de pago por cada funcionalidad.
- **Pago:** Brinda un conjunto de clases además de una interfaz gráfica necesaria para realizar el pago de una determinada operación bancaria que lo requiera.
- **Impresión:** Brinda un conjunto de clases que contienen las funcionalidades mediante las cuales es posible imprimir las operaciones que realicen contabilización de forma automática.
- **DataAccess:** Brinda todos los elementos necesarios para llevar a cabo la conexión a la base de datos.

3.3 Estándares de Codificación

Es un conjunto de directrices, normas y reglamentos que especifican la forma en que debe escribirse el código fuente. Por lo general, un estándar de codificación incluye pautas sobre la nomenclatura de las variables, clases y/o paquetes; la correcta indentación del código, cómo escribir estructuras de control o incluso qué información incluir en los comentarios.

Su importancia radica en que facilitan la compresión del código y, por tanto, permiten que los diferentes desarrolladores del equipo de proyecto puedan tratar cada porción del mismo como si se tratara de su propio código garantizando además que el trabajo se realice de forma coordinada.

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

3.3.1 Convenciones de nomenclatura

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

Para la nomenclatura de las variables y los métodos se definió la notación CamelCasing, que es muy similar a PascalCasing con la excepción de que la letra inicial siempre es minúscula, en caso de estar compuesta por más de una palabra, estas empezarán con mayúscula exceptuando la primera. Ejemplo: **idNegociacion**. Lo mismo se aplica a los nombres de ficheros de código javascript y sus funciones y variables internas. Ejemplo: **registrarNegociacion.**

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

3.3.2 Convenciones en la Capa de Presentación

Las clases pertenecientes a los sub-paquetes mvc y webflow del paquete web se nombrarán de la manera siguiente:
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: **DocumentoEmbarqueValidator.**
Paquete propertyEditor: [Nombre de la clase] + [PropertyEditor]. Ejemplo: **EstadoDocumentoPropertyEditor.**
Paquete flowHandler: [Nombre de la clase] + [FlowHandler]. Ejemplo: **NegociacionFlowHandler.**
Paquete serviceFlow: [Nombre de la clase] + [Action o de MultiAction en dependencia de cuál de las dos clases herede]. Ejemplo: **ContabilizarNegociacionMultiAction.**

3.3.3 Convenciones en la Capa de Negocio

Las clases pertenecientes a los paquetes facade y manager y sus sub-paquetes impl respectivamente se nombrarán de la manera siguiente:
Paquete facade: [Nombre del módulo] + [Facade]. Ejemplo: **NegociacionFacade.** Y las clases que implementan las interfaces [Nombre del módulo] + [FacadeImpl]. Ejemplo: **NegociacionFacadeImpl.**
Paquete manager: [Nombre del módulo] + [Manager] .Ejemplo: **NegociacionManager.** Y las clases que implementan las interfaces [Nombre del módulo] + [ManagerImpl]. Ejemplo: **NegociacionManagerImpl.**

3.3.4 Convenciones en la Capa de Acceso a Datos

Las clases pertenecientes al paquete dao y su sub-paquete impl se nombrarán de la manera siguiente:
Paquete dao: [Nombre de la entidad] + [DAO]. Ejemplo: **NegociacionDAO.** Y las clases que implementan las interfaces [Nombre de la entidad] + [DAOImpl]. Ejemplo: **NegociacionDAOImpl.**

3.4 Aspectos Principales de la Implementación

3.4.1 Utilización de Spring WebFlow Framework

En el capítulo anterior se especificaba el uso del framework Spring WebFlow por las facilidades que brinda en el desarrollo al definir y gestionar flujos complejos de vistas e información de páginas dentro de una aplicación web. Para el desarrollo de las funcionalidades de los módulos Discrepancia y Negociación se emplea este framework por la necesidad de interactuar con una serie de componentes como es el caso de

Calendario, Pago y Comisiones dentro de Contabilización. Además el framework Spring WebFlow es capaz de responder a vistas diferentes e iniciar con el mismo objeto (command) en distintos estados, simplificando la implementación de casos de uso agrupados por el patrón CRUD. De esta forma a partir de la declaración de un flujo común dará respuestas a un conjunto de funcionalidades de casos de usos.

A continuación se describe el funcionamiento del mismo desde el módulo Negociación:

```
<var name="negociacionCommand"
class="cu.uci.finixubnc.cartacredito.negociacion.web.webflow.command.NegociacionCommand" />

<on-start>
    <evaluate expression="contabilizarNegociacionMultiAction.getNameView" />
    <evaluate expression="contabilizarNegociacionMultiAction.getComand" />
</on-start>

<subflow-state id="contabilizar-NEGOCIACIONsubFlowPrincipal-flow"
               subflow="contabilizar-NEGOCIACIONsubFlowPrincipal-flow">
    <input name="negociacionCommand" />
    <input name="nameView" />
    <transition on="end" to="end"/>
</subflow-state>
```

El primer paso en los flujos de los módulos es la declaración de la variable que representará a la clase que se encuentra en el paquete command, clase que representa el objeto que se manipula en los formularios, en este caso **negociacionCommand**.

Luego desde la etiqueta **on-start** se invoca los métodos **getNameView (RequestContext context)** que representa el modelo correspondiente de la vista invocada y **getCommand (RequestContext context)** que es donde la vista recibe el valor inicial del objeto command. Todas las funcionalidades presentes en el flujo se encuentran implementadas en la clase ContabilizarNegociacionMultiAction que se utiliza como fachada para evitar la interacción directa desde el flujo con el negocio. Las funcionalidades que se exponen en el flujo no necesitan ser declaradas con los parámetros correspondientes, pues recibiendo el **RequestContext** como único parámetro, Spring WebFlow es capaz de reconocerlo.

Cada flujo presenta un subflujo encargado de realizar una llamada al flujo principal. Mediante la etiqueta **subflow** se hace entrada al flujo principal **contabilizar-NEGOCIACIONsubFlowPrincipal-flow** al cual es necesario transferirle las variables **negociacionCommand** y **nameView** (nombre de la vista) que se encuentran en el contexto.

```
<view-state id="negociaciones" model="negociacionCommand" view="${flowScope.nameView}">
    <transition on="aceptar" to="validarCalendario" />
    <transition on="cancel" to="end" />
    <transition on="aceptarM" to="contabilizar" />
    <transition on="cancelarM" to="negociaciones" />
    <transition on="aceptarF" to="end" />
    <transition on="verAsientos" to="verAsientosState" />
    <transition on="vercalendario" to="saveCalendario" />
    <transition on="actualizarDatosComisiones" validate="false" bind="false" to="negociaciones">
        <evaluate expression="comisionesMultiAction.actualizarDatosComisiones" />
    </transition>
    <transition on="gestionarComision" validate="false" to="gestionarComisionSubflow">
        <evaluate expression="comisionesMultiAction.registrarCallMetodo" />
    </transition>
    <transition on="pagar" to="pagarSubflow">
        <evaluate expression="contabilizarNegociacionMultiAction.armarPagarCommand" />
    </transition>
</view-state>
```

Una vez dentro del flujo principal encontramos la vista (**view-state)** que es el estado más importante del flujo; el flujo propuesto se detiene en **negociaciones** que se encarga de mostrar al usuario la página cliente (.jsp) donde se encuentre (Ejemplo: Si el nombre de la vista es registrarNegociacion se le mostrará al usuario la página cliente registrarNegociacion.jsp) y contiene diferentes transiciones (**transition)** que entrarán en acción a partir de los posibles eventos que se puedan generar por el usuario.

Como respuesta a cada evento se pasa a un nuevo estado en el flujo:

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

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

Luego de generarse el evento **verCalendario** indicando integración con el componente Calendario, se realiza la acción **saveCalendario** que se encarga de invocar al método **saveCalendarioInConversation ()** donde se guardan los datos del command que son necesarios para el calendario; y se redirecciona las acciones para el flujo calendario-flow brindado por el componente Calendario.

Todos los flujos que son externos al módulo se deben registrar en el contexto SpringWebFlow del módulo:

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

Luego de realizar cada uno de los eventos que se exigen en la vista correspondiente se reúne parte de la información que debe comprender la negociación que se debe registrar.

```
<transition on="aceptar" to="validarCalendario" />
```

Si desde la página que se muestra al usuario se ejecuta el evento ***aceptar*****,** se desencadenarán una serie de estados de acción y de decisión.

```
<action-state id="validarCalendario">
  <evaluate expression="contabilizarNegociacionMultiAction.compararCalendarioconNegociacion"/>
  <transition on="yes"  to="validarUno" />
   <transition on="no"  to="mostrarMensaje" />
</action-state>
```

La acción **validarCalendario** se encarga de invocar al método **compararCalendarioconNegociacion ()** donde se verifica que los datos del calendario corresponden con los de la vista de negociación y si todo está correcto se redirecciona la acción para el flujo **validarUno** en caso contrario para el flujo **mostrarMensaje**.

```
<action-state id="mostrarMensaje">
  <evaluate expression="contabilizarNegociacionMultiAction.mostrarMensajeCalendario"/>
  <transition on="success"  to="negociaciones" />
</action-state>

<action-state id="validarUno">
  <evaluate expression="contabilizarNegociacionMultiAction.primeraValidacion"/>
  <transition on="success"  to="probarprimera" />
</action-state>

<decision-state id="probarprimera">
  <if test="flowScope.mensajeError== 'm0' " then="segundavalidacion"
      else="negociaciones" />
</decision-state>
```

La acción **mostrarMensaje** se encarga de invocar al método **mostrarMensajeCalendario ()** donde se verifica el mensaje de error correspondiente al calendario y se le muestra al usuario en la vista. La acción **validarUno** se encarga de invocar al método **primeraValidacion ()** donde se realiza la primera validación de los asiento contables que se generaron y se guarda en el flujo la variable **mensajeError** que indicará la

respuesta. Si la variable indica **m0** significa que no ocurrió ningún error y se redirecciona a **segundavalidacion** sino se vuelve a la vista mostrando el error que se produjo.

```
<action-state id="segundavalidacion">
    <evaluate expression="contabilizarNegociacionMultiAction.segundaValidacion" />
    <transition on="success" to="testcontabilizar" />
</action-state>

<decision-state id="testcontabilizar">
    <if test="flowScope.mensajeError=='m2'" then="hayMensajeria" else="negociaciones" />
</decision-state>

<decision-state id="hayMensajeria">
    <if test="contabilizarNegociacionMultiAction.esConMensajeria(flowRequestContext)"
        then="mensajeria" else="contabilizar" />
</decision-state>
```

La acción **segundavalidacion** se encarga de invocar al método **segundaValidacion ()** donde se realiza la segunda validación de los asientos contables que se generaron y se guarda en el flujo la variable **mensajeError** que indicará la respuesta. Si la variable indica ***m2*** significa que no ocurrió ningún error y se redirecciona al estado decisión **hayMensajeria** sino se vuelve a la vista mostrando el error que se produjo. En el estado decisión **hayMensajeria** se invoca al método **esConMensajeria ()** donde se evalúa si la vista en que se encuentra contiene mensajería. En caso que lo requiera se redirecciona a **mensajería** de lo contrario a la acción **contabilizar.**

```
<action-state id="contabilizar">
    <evaluate expression="contabilizarNegociacionMultiAction.contabiliza" />
    <transition on="success" to="negociaciones" />
</action-state>
```

En el estado contabilizar se invoca al método **contabiliza ()** que se encarga de realizar la contabilización de los asientos contables generados. En el caso de que exista error se notifica al usuario, de lo contrario se persiste la entidad y se pasa al estado final del flujo:

```
<end-state id="end" />
```

En el caso que incluya mensajería, comisiones y pago se desencadenarán una serie de estados de acción y de decisión que manejarán las validaciones correspondientes de igual manera que el sub-flujo calendario hasta la contabilización de los asientos, que culminarán en un estado de finalización; **end-state**.

```
<end-state id="end"
view="externalRedirect:servletRelative:../../common/home.htm"/>
```

Cuando todas las operaciones del flujo principal han sido finalizadas se restablece el flujo que lo haya

invocado, que pasa al estado de finalización donde se redirecciona a la página cliente por donde se inicializó dando por terminada la funcionalidad.

3.5 Descripción de las clases y las funcionalidades.

Para un mejor entendimiento de la implementación de los módulos Documentos de Embarque, Discrepancia y Negociación se describirán las clases, los atributos y métodos de aquellas que son más importantes para el sistema desde el punto de vista funcional.

Se describirá la clase **ContabilizarNegociacionMultiAction** que es la encargada de manejar el flujo del módulo Negociación y la clase **Negociacion** que recoge la información necesaria de una negociación.

Nombre: ContabilizarNegociacionMultiAction	
Tipo de clase: Controladora	
Atributo	**Tipo**
negociacionFacade	NegociacionFacade
globalFacade	GlobalFacade
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.
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.
saveCalendarioInConversation(RequestContext contex)	Permite guardar el calendario de pagos que tiene el objeto Negociación de forma tal que el componente calendario pueda acceder a este.

saveCalendario(RequestContext context)	Permite actualizar el calendario de pagos a la Negociación una vez que fue modificado por el componente Calendario.
construirAsientos(RequestContext contex)	Construye los asientos en función del caso de uso.
primeraValidacion(RequestContext contex)	Realiza la primera validación de los asientos a contabilizar y en caso de obtener algún mensaje de error o alerta indicárselo al flujo.
segundaValidacion(RequestContext contex)	Realiza la segunda validación de los asientos contables y notifica al flujo la presencia o no de mensajes de error o alerta.
contabiliza(RequestContext contex)	Contabiliza la lista de asientos y notifica el mensaje de error en caso de existir.
esConMensajeria(RequestContext context)	Permite conocer si la vista lleva mensajería o no.
prepararDatosMensajeria(RequestContext context)	Permite conformar los datos que son necesarios para la mensajería.
checkError(RequestContext context, Object object)	Verifica si se produjo algún error en el proceso de mensajería.

Tabla 3.1: Descripción de la clase ContabilizarNegociacionMultiAction.

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

observaciones	String
importe	java.math.BigDecimal
clientePleaseOpen	String

Tabla 3.2: Descripción de la clase Negociacion.

3.6 Validación de la solución

Para verificar la calidad y evaluar la fiabilidad (resistente a fallos de implementación), la funcionalidad (hace lo que debe) y el rendimiento (lleva a cabo su trabajo de manera efectiva) de los módulos Documentos de Embarque, Discrepancia y Negociación se realizaron por parte del equipo de calidad del proyecto SAGEB las pruebas de Unidad, de Integración, de Sistema y de Aceptación.

Para probar las funcionalidades de los módulos se realizaron las pruebas que corresponden al nivel de Unidad, las cuales están enfocadas al código fuente de los componentes para verificar todos los flujos de control, probando de manera individual las partes del sistema que han sido desarrolladas. Adecuados a este nivel se realizaron los casos de pruebas aplicando el método de Caja Negra con el objetivo de verificar que las funciones son operativas a través de la interfaz del software, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, manteniendo la integridad de la información externa. Para la realización de las pruebas de caja negra existen diferentes técnicas: Particiones de Equivalencia, Análisis de Valores Límite y Grafos de Causa-Efecto**.** La técnica empleada fue la de Partición de Equivalencia, que representa una de las más efectivas pues permite examinar los valores válidos e inválidos de las entradas existentes en los módulos y descubre de forma inmediata una clase de errores, que de otro modo requerirían la ejecución de muchos casos antes de detectar el error genérico. El resultado de la realización de estas pruebas en varias iteraciones fue satisfactorio desde el punto de vista funcional, atendiendo al correcto comportamiento de las funcionalidades de cada módulo ante diferentes situaciones.

En el nivel de Integración se verifica el correcto ensamblaje entre los distintos componentes una vez que han sido probados unitariamente con el fin de comprobar que interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren la funcionalidad establecida y se ajustan a los requisitos no funcionales especificados en las verificaciones correspondientes. Se comprueba la correcta integración

de los módulos dentro del subsistema y con el resto de los componentes de la aplicación.

En el nivel de Sistema las pruebas se hacen para verificar que se han integrado adecuadamente todos los componentes del sistema desde el hardware a la documentación. Los tipos de pruebas realizadas pertenecientes a este nivel son: la de seguridad, que evalúa como el sistema se protege contra accesos, internos o externos y no autorizado; y la de rendimiento diseñada para probar los tiempos de respuesta y el espacio que ocupa el módulo en disco o en memoria. La impresora es el hardware externo que debe integrarse para la impresión de la información referente a la contabilidad de las distintas operaciones bancarias. La comprobación del buen funcionamiento de los módulos con dicha integración y de la seguridad también brinda resultados positivos.

Una vez probados los módulos Documentos de Embarque, Discrepancia y Negociación por el equipo de calidad del proyecto SAGEB, el departamento de calidad de la universidad (CALISOFT) procede a liberar los módulos de toda la aplicación a través de la realización de pruebas de función (nivel de Unidad), de carga: usada para validar y valorar la aceptabilidad de los límites operacionales de un sistema bajo carga de trabajo variable, mientras el sistema bajo prueba permanece constante, y de estrés: enfocada a evaluar cómo el sistema responde bajo condiciones anormales (extrema sobrecarga, insuficiente memoria, entre otros). Como resultado CALISOFT manifiesta su aprobación en el acta de liberación del sistema.

Actualmente se está sometiendo la aplicación a las pruebas de aceptación del cliente en el BNC, siendo la prueba final antes del despliegue del sistema, cuyo objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar las funciones para las cuales fue construido.

3.7 Conclusiones Parciales

Al concluir el capítulo se obtuvo la solución de los módulos Documentos de Embarque, Discrepancia y Negociación para el sistema Quarxo. Se abordó sobre los estándares de codificación que se emplearon en el desarrollo de los módulos para facilitar la comprensión del código. La solución exhibe valor técnico a partir de la utilización de Spring WebFlow como elemento fundamental, el cual brinda recursos potentes que permitieron entre otros aspectos la integración con componentes de carácter general dentro del

sistema. Se realizó la validación de los módulos desde el punto de vista funcional tanto por el grupo de calidad del proyecto como por el equipo de CALISOFT, demostrando que cumplen con los requerimientos necesarios para satisfacer las necesidades del cliente. Al culminar la implementación de los módulos, se dio cumplimiento al objetivo general trazado inicialmente.

CONCLUSIONES

El estudio de los sistemas informáticos contables a nivel mundial y nacional que proporcionan funcionalidades que puede utilizar el BNC para la gestión de los documentos de embarques, discrepancias y negociaciones evidenció la necesidad de desarrollar una solución informática capaz de ejecutar dichas funcionalidades, que cumpliera con los requisitos establecidos en el BNC. Las tecnologías y herramientas propuestas para el desarrollo del diseño e implementación de los módulos Documentos de Embarque, Discrepancia y Negociación fueron utilizadas de manera satisfactoria contribuyendo a las políticas trazadas por el país de sustitución de importaciones así como de independencia y seguridad tecnológica garantizando la disminución de los costos de implementación, mantenimiento del sistema y por concepto de licencias de software y soporte. Se realizó el diseño y la implementación de los módulos de acuerdo al análisis realizado de la arquitectura del proyecto SAGEB utilizando como base las funcionalidades definidas por los analistas del proyecto garantizando una gestión eficiente de dichos procesos en el BNC. Se comprobó el correcto funcionamiento de la solución propuesta a través de la realización de pruebas de calidad por parte de CALISOFT que validaron las funcionalidades implementadas.

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

Atendiendo a las conclusiones a las que se arriban con el desarrollo del trabajo, se recomienda:

- Adaptar los módulos Documentos de Embarque, Discrepancia y Negociación para que puedan ser utilizados en otras entidades bancarias que realicen la gestión de las Cartas de Crédito.
- Utilizar las tecnologías y herramientas presentadas en la investigación en el desarrollo de otros sistemas de gestión bancaria.

REFERENCIAS BIBLIOGRÁFICAS

1. **Peña, Dany Cruz.** Gestio Polis. *Fundamentos de gestión en Instituciones Financieras no Bancarias Cubanas.* [En línea] 2011. [Citado el: 6 de Enero de 2011.] http://www.gestiopolis.com/finanzas-contaduria-2/fundamentos-gestion-instituciones-financieras-no-bancarias-cubanas.htm.

2. Buenas Tareas. *Historia de la Banca*. [En línea] 2011. [Citado el: 6 de Enero de 2011.] http://www.buenastareas.com/ensayos/Historia-De-La-Banca/68182.html.

3. **BNC.2007.** *MANUAL DE INSTRUCCIONES Y PROCEDIMIENTOS.* 2007.

4. **Castillo, Miguel.** *LA CONTABILIDAD GUBERNAMENTAL EN CUBA.* 2008.

5. **Medina, Maria de los Angeles.** Scribd. *CAMPOS DE LA APLICACIÓN DE LA CONTABILIDAD.* [En línea] 2010. [Citado el: 20 de Diciembre de 2010.] http://es.scribd.com/doc/50595325/CAMPOS-DE-LA-APLICACION-DE-LA-CONTABILIDAD.

6. Buenas Tareas. *Medios De Pago.* [En línea] 2011. [Citado el: 7 de Enero de 2011.] http://www.buenastareas.com/ensayos/Medios-De-Pago/346835.html.

7. **Gavira, Luis Emilio Martínez.** *Medios de Pago,Cuba.* 2005.

8. **Giráldez, Gustavo.** Zona Bancos. *Modalidades e Instrumentos de Pagos Internacionales.* [En línea] 19 de Agosto de 2009. [Citado el: 7 de Enero de 2011.] http://www.zonabancos.com/ar/analisis/blogs/2-comercio-exterior-y-cambios-13473-modalidades-e-instrumentos-de-pagos-internacionales.aspx.

9. **Santiago, Ana Victoria Maura.** *DOCUMENTOS FINANCIEROS MÁS UTILIZADOS EN EL COMERCIO EXTERIOR COMO MEDIOS DE PAGO.* 2010. 2073-6061.

10. **Zapata, Cristina I.** De Gerencia. *Documentación de embarque internacional.* [En línea] 2003. [Citado el: 20 de Diciembre de 2010.] http://www.degerencia.com/articulo/documentacion_de_embarque_internacional_guia_practica/imp.

11. **UCP500.** *Reglas y Usos Uniformes de la CCI para Créditos Documentarios.*

12. **Josar, Cristina.** Gerencie. *La contabilidad y el sistema contable.* [En línea] 28 de Agosto de 2008. [Citado el: 2 de Febrero de 2011.] http://www.gerencie.com/sistema-contable.html.

13. **León, Yelani Reyes y Alonso, Yosbel Rivero.** *DEFINICIÓN DE LOS REQUERIMIENTOS FUNCIONALES DEL MÓDULO NEGOCIACIONES Y PAGO DEL PROYECTO BANCO NACIONAL.* 2008.

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

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

16. **Kaisler, Stephen H.** *Software Paradigms.* 2005.

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

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

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

BIBLIOGRAFÍA

1. Cortés, Lic. Danilo Hernández. ASPECTOS LEGALES DEL COMERCIO Y LA CONTRATACIÓN. 2004.

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

3. Scribd. *Conceptos de RUP.* [En línea] [Citado el: 5 de Febrero de 2011.] http://es.scribd.com/doc/7844685/CONCEPTOS-DE-RUP.

4. Desarrollo de Aplicaciones Informaticas. *Tipos de herramientas CASE.* [En línea] [Citado el: 3 de Marzo de 2011.] http://desarrollodeaplicacionesinformaticas.com/index.php/Analisis-y-diseno-detallado-de-aplic.-informaticas/Tema-12-Herramientas-CASE/3-tipos-de-herramientas-case.html.

5. **Arnol, Ken y Gosling, James.** *The Java Programming Language.* 1996.

6. **Pérez, Javier Eguíluz.** Introducción a JavaScript. [En línea] 2008.

7. **Gill, Rawld, Riecke, Craig y Russell, Alex.** *Mastering Dojo.* s.l. : Pragmatic Bookshelf, 2008. 1-934356-11-5.

8. Apache. *Documentación del Servidor HTTP Apache 2.0.* [En línea] [Citado el: 3 de Marzo de 2011.] http://httpd.apache.org/docs/2.0/es/.

9. **Galíndez, Rodrigo.** *Control de Versiones Usando Subversion.* [En línea] [Citado el: 5 de Marzo de 2011.] http://www.rodrigogalindez.com/files/14.pdf.

10. Desarrollo Web. *Arquitectura cliente-servidor.* [En línea] [Citado el: 11 de Marzo de 2011.] http://www.desarrolloweb.com/articulos/arquitectura-cliente-servidor.html.

11. **Rumbaugh, James, y otros.** *Object-Oriented Modeling and Desing.* Englewood Clifs, NJ : Prentice Hall, 1991.

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

13. **Hetzel, Bill.** *The Complete Guide to Software Testing.* Second Edition, Wellesley,MA : QED Information Sciences, Inc., 1988.

GLOSARIO

Ajax: (en inglés: Asynchronous JavaScript). Es una técnica de desarrollo web para crear aplicaciones interactivas.

AOP: (en inglés: Aspect Oriented Programming). Es un paradigma de programación que permite una adecuada modularización de las aplicaciones.

Caso de Prueba: Especifica una forma de probar el sistema, incluyendo la entrada o resultado con la que se ha de probar y las condiciones bajo las que hay que probarse.

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.

Catalizador: Para designar a aquel fenómeno o aparato que sirva para acelerar un proceso específico.

Cuenta Corriente: Contrato entre un banco y un cliente que establece que la entidad cumplirá las órdenes de pago de la persona de acuerdo a la cantidad de dinero que haya depositado o al crédito que haya acordado. Dicha cuenta puede ser abierta y administrada por una persona o por un grupo de personas.

Diario: Libro conocido como “Libro de Entrada Original”, donde se registran todas las transacciones que se dan lugar en una entidad.

Factura pro-forma: Documenta una oferta comercial, con indicación de la forma exacta que tendrá la factura tras el suministro. No tiene valor contable ni como justificante; se utiliza fundamentalmente en comercio internacional para obtener las licencias de importación, para la apertura de créditos documentarios o para el envío de muestras comerciales.

JDBC: (en inglés: Java Database Connectivity). Permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java.

Modelo de análisis: Es un modelo de objetos conceptual que analiza los requisitos mediante su refinamiento y estructuración. Incluye los siguientes elementos: Paquetes de análisis y sus dependencias, clases de análisis y sus responsabilidades, relaciones y requisitos especiales, realizaciones de casos de usos-análisis y la vista de la arquitectura del modelo de análisis. Se considera la entrada fundamental para las actividades del diseño.

ORM: (en inglés: Object Relacional Mapping). Persistencia automática y transparente de objetos de una aplicación en una base de datos relacional utilizando metadatos que describen la correspondencia entre el objeto y las tablas de la base de datos.

Plugins: Módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande.

Seam: Es una factoría que crea objetos según la definición del componente (clase). Seam promueve la interacción de estos objetos con estado a través de contextos, ensamblándolos acorde a los metadatos asociados a sus respectivas clases.

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

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

Remanente: Término contable con el que se designan los beneficios no repartidos pendientes de aplicación. Beneficios no repartidos ni aplicados específicamente a ninguna otra cuenta.