

**Universidad de las Ciencias Informáticas**

**“Facultad 3”**



**“Diseño e Implementación de los módulos Cheque y  
Carta de Remesa del sistema Quarxo.”**

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

**Autor:** Dagoberto Vladimir Gras Noa

**Tutor:** Ing. Yesenia Perdomo Bello.

**La Habana, Cuba**

31/05/2012

DECLARACIÓN DE AUTORÍA

Declaramos ser autores del presente trabajo de tesis y se le reconoce a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste se firma el presente a los \_\_\_\_ días del mes de \_\_\_\_\_ del año \_\_\_\_\_.

\_\_\_\_\_  
Dagoberto Vladimir Gras Noa

\_\_\_\_\_  
Ing. Yesenia Perdomo Bello

DEDICATORIA

A mis padres por ser tan dedicados durante toda mi formación estudiantil y en la vida.  
¡Excelentes amigos!

A mi hermanita Indira, que ha estado a mi lado desde que éramos de meses.

A mis abuelitas; a mis tías Jaqueline, Chichi, Yadia; a mis tíos Jose Miguel, Jose Alberto;  
por todo el cariño que me han dado.

A toda mi familia, que siempre están presente incondicionalmente en mi vida.

AGRADECIMIENTOS

A mi tutora Yesenia, excelente compañera de trabajo y buena amiga. Siempre atenta a cualquier ayuda que pueda brindar.

A mis compañeros de aula que son un grupo especial, siempre ayudándonos unos a otros.

A mis compañeros de proyecto, tan laboriosos y decididos.

A mis amigos Wilfredo, Bode, Nelson por compartir tantos ratos juntos en fiestas, estudio y trabajo. A Evelio, Hector, Annier, Dariel, Efraín, Emilio por ser tan atentos. A Liuba y sus amigas por ser tan cariñosas.

A todas aquellas hermosas personas que aparecen en la vida: Greter. Amiga que me ha brindado su apoyo y amor junto a su familia.

A la Universidad de las Ciencias Informáticas por la buena formación universitaria y profesional que me brindó.

## RESUMEN

Las entidades bancarias utilizan sistemas automatizados que permiten realizar su trabajo de manera óptima y eficiente; pero el avance de las tecnologías de la información, han acelerado el surgimiento de nuevos productos y servicios bancarios, siendo una necesidad disponer con la infraestructura tecnológica que se adapte a la estrategia de negocios de cada entidad bancaria.

EL Banco Nacional de Cuba decide aplicar las nuevas Tecnologías de la Información y las Comunicaciones en los procesos bancarios que se llevan a cabo. El Banco Nacional de Cuba le encomienda a la Universidad de las Ciencias Informáticas, desarrollar el sistema de gestión bancaria Quarxo que se adapte a las estrategias de negocio incluyendo la gestión de Cheques y Cartas de Remesa.

La Universidad de las Ciencias Informáticas se dio a la tarea de desarrollar un sistema de gestión bancaria que contribuya a informatizar los procesos que se realizan dentro de las gerencias del Banco Nacional de Cuba. El trabajo de diploma presenta el diseño e implementación de los módulos Cheque y Carta de Remesa del sistema Quarxo. El desarrollo de las funcionalidades para dichos módulos se basó en el uso de patrones, facilitando el diseño de una arquitectura en capas y modular, garantizando la integración con otros subsistemas y funcionalidades empleando herramientas y tecnologías Java con el fin de obtener una aplicación web.

**Palabras claves:** Entidades bancarias, Cheque, Carta de Remesa.

TABLA DE CONTENIDO

DECLARACIÓN DE AUTORÍA.....	II
DEDICATORIA .....	III
AGRADECIMIENTOS .....	IV
RESUMEN.....	V
INTRODUCCIÓN.....	1
CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA .....	3
1.1.    Introducción. ....	3
1.2.    Informatización del sistema bancario cubano. ....	3
1.3.    Conceptos Fundamentales.....	3
1.3.1    Cheque .....	3
1.3.1.1    Tipos de cheques .....	3
1.3.1.2    Operaciones sobre Cheque.....	4
1.3.2    Carta de Remesa.....	4
3.1.1.1    Operaciones sobre Carta de Remesa .....	5
1.4.    Sistemas Automatizados de Gestión Bancaria .....	5
1.5.    Metodología, Lenguaje y Herramientas.....	9
1.5.1.    Metodología RUP. ....	9
1.5.2.    Lenguaje de Modelado UML. ....	9
1.5.3.    Herramienta CASE. Visual paradigm. ....	9
1.5.4.    Eclipse 3.3 IDE. ....	10
1.5.5.    Contenedor Web (Apache Tomcat).....	10
1.5.6.    ER/Estudio.....	10
1.5.7.    Gestor de Bases de Datos. Microsoft SQL Server 2005.....	11
1.6.    Patrones de Diseño .....	11
1.6.1    Patrones de asignación de responsabilidades. (GRAPS).....	11
1.6.1.1    Experto.....	11
1.6.1.2    Creador.....	12
1.6.1.3    Bajo Acoplamiento.....	12
1.6.1.4    Controlador.....	12
1.6.1.5    Alta Cohesión .....	12
1.6.2    Patrones Estructurales. ....	12
1.6.2.1    Facade (Fachada). ....	12
1.6.3    Patrones de Comportamiento.....	12
1.6.2.2    DAO (Data Access Object).....	12
1.6.4    Patrones de Presentación. ....	13

1.6.3.1	Composite View (Vistas Compuestas) .....	13
1.6.3.2	MVC (Modelo Vista Controlador).....	13
1.7.	Ambiente de desarrollo.....	13
1.7.1	Lenguaje de programación. Java. ....	13
1.7.2	Entorno de desarrollo. J2EE.....	14
1.8.	Tecnologías y Frameworks. ....	14
1.8.1	Spring Framework .....	14
1.8.2	Spring Web Flow .....	14
1.8.3	Hibernate .....	15
1.8.4	Dojo toolkit .....	15
1.8.5	JUnit.....	15
1.9.	Conclusiones parciales.....	16
CAPÍTULO 2. ARQUITECTURA Y DISEÑO.....		17
2.1	Introducción. ....	17
2.2	Arquitectura de sistema Quarxo. ....	17
2.2.1	Capa de Presentación. ....	19
2.2.2	Capa de Negocio. ....	19
2.2.3	Capa de Acceso a Datos. ....	19
2.3	Diseño de los módulo Cheque y Carta de Remesa. ....	19
2.3.1	Diagramas de Paquete. ....	20
2.3.2	Diseño de la Capa de Presentación. ....	22
2.3.3	Diseño de la Capa de Negocio. ....	25
2.3.4	Diseño de la Capa de Acceso a Datos. ....	28
2.3.5	Diseño de la Capa de Dominio. ....	29
2.3.6	Patrones de diseño empleados. ....	30
2.3.7	Diagramas de Secuencia.....	31
2.4	Modelo de Datos.....	32
2.5	Conclusiones parciales.....	33
CAPÍTULO 3.IMPLEMENTACIÓN Y PRUEBA.....		34
3.1	Introducción. ....	34
3.2	Implementación de la solución para los módulos Cheque y Carta de Remesa.....	34
3.2.1	Diagrama de componentes. ....	34
3.2.2	Estándares de Codificación.....	35
3.2.2.1	Convenciones de Nomenclatura .....	35
3.2.2.2	Nomenclatura según el tipo de clases.....	36
3.2.3	Descripción de algunas clases principales utilizadas.....	36

3.2.4	Utilización del Framework Spring WebFlow .....	39
3.3	Pruebas .....	43
3.3.1	Aplicación de la Prueba de Caja Blanca. ....	43
3.3.1.1	Prueba del Camino Básico. ....	44
3.3.2	Aplicación de la Prueba de Caja Negra o Funcional. ....	48
3.3.3	Resultado de las pruebas. ....	49
3.4	Conclusiones parciales.....	50
	CONCLUSIONES .....	51
	RECOMENDACIONES .....	52
	BIBLIOGRAFÍA.....	53
	ANEXOS.....	56
	GLOSARIO DE TÉRMINOS.....	71



## INTRODUCCIÓN

El desarrollo acelerado del mundo actual hace imposible que cualquier entidad o institución pueda controlar de forma manual su actividad fundamental. La mayoría utilizan sistemas automatizados que permitan agilizar su trabajo de manera óptima, eficiente y con el personal necesario. Priorizar la modernización del sistema bancario cubano, con el fin de crear las bases indispensables, que permitan enfrentar con éxito las transformaciones en el orden económico; constituye un reto que permitirá asimilar las transacciones del comercio y los negocios electrónicos.

El sistema bancario cubano para controlar sus costos, su gestión de negocio y administrar sus operaciones, al mismo tiempo que se adaptan a las nuevas tecnologías; lleva a cabo la tarea de informatizar sus procesos de negocio como parte de la modernización de sus estrategias de negocio y sus actividades financieras y contables. Un proyecto productivo de La Universidad de las Ciencias Informáticas (UCI) se le asignó la tarea de desarrollar un sistema que automatice la gestión de los procesos de negocio que se realizan a diario en el BNC sobre la base de la utilización del núcleo central del SABIC.

El Banco Nacional de Cuba hace uso del Sistema Automatizado para la Banca Internacional de Comercio (SABIC) como sistema para la contabilización de sus operaciones. Dicho sistema presenta irregularidades en cuanto a la gestión de la información de los procesos relacionados con Cheques y Cartas de Remesa, afectando servicios tales como Certificación, Confirmación de Ingresos, Custodia y Venta de Títulos. Además, los procesos que se encuentran automatizado no cumplen en su totalidad con las exigencias de tener circulando en los bancos el Cheque como documento de pago inmediato; obstaculizando en tiempo la mayoría de las actividades que dependen de este Título Valor.

### **Problema a resolver**

¿Cómo mejorar la gestión de los procesos de los Cheques y las Cartas de Remesa en el Banco Nacional de Cuba, para disminuir el tiempo de los procesos de contabilización?

### **Objetivo General**

Desarrollar una solución general que mejore y automatice la gestión de Cheques y Cartas de Remesa en el Banco Nacional de Cuba.

### **Objeto de Estudio**

Gestión de los procesos de Cheques y Cartas de Remesa en entidades financieras bancarias.

### **El Campo de Acción**

Diseño e implementación de un sistema informático para la gestión de los procesos de Cheque y Cartas de Remesa en el Banco Nacional de Cuba.

### **Objetivos específicos**

- Obtener una fundamentación teórica de la solución.
- Obtener el modelo de diseño del módulo Cheque y el módulo Carta de Remesa.
- Obtener los componentes producto de la implementación del diseño realizado para el módulo Cheque y Carta de Remesa.
- Validar la solución implementada para la gestión de los procesos de Cheques y Cartas de Remesa.

### **Tareas a cumplir**

- Caracterización de los procesos relacionados con la gestión de Cheques y Cartas de Remesa en las entidades financieras bancarias.
- Caracterización de las herramientas, lenguajes y frameworks para el desarrollo de la solución que gestiona Cheques y Cartas de Remesa para el Banco Nacional de Cuba.
- Caracterización de los patrones de diseño para la búsqueda de soluciones a problemas comunes en el desarrollo.
- Realización del diagrama de clases del diseño de los módulos Cheque y Carta de Remesa.
- Realización del diagrama de secuencia del diseño de los módulos Cheque y Carta de Remesa.
- Realización del modelo de datos de los módulos Cheque y Carta de Remesa.
- Realización del modelo de componentes de los módulos Cheque y Carta de Remesa.
- Realización de pruebas unitarias de la solución para la gestión de los procesos de los Cheques y Cartas de Remesa en el Banco Nacional de Cuba.

### **Posible Resultado**

Producto del diseño e implementación, obtener los módulos Cheque y Carta de Remesa del sistema Quarxo.

## CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

### 1.1. Introducción.

En este capítulo se realiza un estudio de los sistemas que automatizan los procesos de gestión de Cheques y Cartas de Remesa a nivel nacional e internacional. Se especifican las características de la metodología, lenguajes, herramientas y frameworks que conforman el ambiente de desarrollo sobre el cual se realizó el sistema. Se realiza un análisis de los conceptos necesarios para la comprensión de la gestión bancaria asociada a los procesos de Cheques y Cartas de Remesa.

### 1.2. Informatización del sistema bancario cubano.

Como causa de las necesidades de información y de control sobre los recursos del país, se lleva a cabo a partir de 1995 el inmenso trabajo de automatización de los bancos cubanos. Se realizaron cambios en cuanto a la forma en que se lleva a cabo la planificación, los procesos afines que se ejecutan y la información que se brinda en cada proceso; esto a través de la adquisición de equipos computacionales, la implantación de redes locales de comunicación, cajeros automáticos y uso de tarjetas magnéticas. En 1996 es implantado en el Banco de Crédito y Comercio (BANDEC) el Sistema Automatizado para la Banca Internacional de Comercio (SABIC); actualmente el sistema es utilizado en el Banco Nacional de Cuba en la gestión de sus procesos bancarios financieros. (1)

### 1.3. Conceptos Fundamentales.

#### 1.3.1 Cheque

El cheque es una orden de pago dada sobre un banco en la cual tiene el librado fondos depositados a su orden: cuenta corriente con saldo a su favor o créditos en descubierto. (2)

##### 1.3.1.1 Tipos de cheques

###### Cheque nominativo

En este tipo de Cheque se consigna el beneficiario y no se permiten endosos.

###### Cheque a la orden

En este tipo de Cheque se consigna el beneficiario y se permiten endosos.

###### Cheque certificado

Este tipo de Cheque se garantiza por los bancos debitando previamente los fondos en la cuenta del girador, con lo que se convierte en una obligación del banco realizar el pago. Se consignan las firmas autorizadas del banco. Pueden ser nominativos o a la orden.

### Cheque de gerencia

Es el emitido por un banco contra sus fondos, puede ser nominativo o a la orden.

#### 1.3.1.2 Operaciones sobre Cheque

##### Certificar Cheque

El banco aprueba un Cheque de un cliente para ser pagado. Se comprueba la disponibilidad de fondos de la cuenta del cliente y se acredita una cuenta por cobrar para reservar dichos fondos. Esta operación requiere que el Cheque esté en estado circulando y pertenezca al Banco Nacional de Cuba.

##### Reintegrar Cheque

Requiere que el Cheque esté en estado Circulando, pertenezca al Banco Nacional de Cuba y esté Certificado. Esta operación acredita al cliente con el mismo monto del Cheque cuando fue Certificado, a la misma vez que hace un débito a la cuenta de Cheques Certificados del propio banco.

##### Suspender pago

Facilita el bloquear un Cheque por pérdida o deterioro, plasmando en las observaciones el motivo por el cual se bloquea. El Cheque pasaría a estado Bloqueado.

##### Caducar Cheque

El Banco Nacional de Cuba tiene un período razonable de 60 días para realizar el pago del Cheque; si en este período, después de haber sido emitido el Cheque, no es presentado, pasa a estado Caduco y el emisor quedará liberado de la responsabilidad por la pérdida causada de la demora.

#### 1.3.2 Carta de Remesa.

Es un documento elaborado por una entidad financiera que contiene un listado de Títulos valores, organizados por determinados criterios, el banco emisor del Título, tipo de Título Valor, la moneda y la cantidad de Títulos permitidos por Carta de Remesa. Las Cartas de Remesa son utilizadas para enviar hacia otros bancos o Entidades Financieras una recopilación de medios de pago con el total del importe, para que se le deposite en su cuenta el valor del importe total de los medios de pagos que se están cobrando. (3)

### 3.1.1.1 Operaciones sobre Carta de Remesa

#### Registrar Carta de Remesa

Esta operación se realiza para contabilizar los Cheques que son emitidos por el BNC y al estar circulando por el sistema bancario son recibidos por Carta de Remesa. Se completa el registro de la información del Cheque y se contabiliza el pago del mismo, quedando en estado Pagado.

#### Crear Carta de Remesa

En esta funcionalidad el sistema automáticamente detecta a partir del registro de Cheques, aquellos que no pueden ser enviados por Carta de Remesa; porque no están integrados al SLBTR o el importe para esa moneda supera los especificados para circular por el Sistema de Liquidación Bruta en Tiempo Real (SLBTR). Conformar las Cartas de Remesa por los diferentes criterios, permite añadirle la cuenta que se va acreditar por el total del importe e imprime la Carta de Remesa que será enviada al banco correspondiente vía correspondencia.

#### Confirmar Ingreso

La funcionalidad de Confirmar Ingreso constituye la respuesta contable del banco que recibió la Carta de Remesa. Se realiza un débito a la cuenta del banco y se acreditan las cuentas de los clientes para los que viene el total de los importes de los Cheques enviados previamente. Se actualiza la fecha valor de la confirmación y se registra en las observaciones los datos que se adhieren en la carta de remesa recibida.

## 1.4. Sistemas Automatizados de Gestión Bancaria

### *SABIC*

El SABIC es un sistema diseñado y desarrollado por la Dirección de Sistemas Automatizados del Banco Central de Cuba para satisfacer las necesidades de procesamiento de datos de bancos e instituciones no bancarias. Como herramienta para la implementación de la aplicación cliente se utilizó

el lenguaje de programación Visual FoxPro. Esta aplicación se ejecuta en cada máquina cliente de forma local para aprovechar la capacidad de procesamiento de cada una de ellas y no recargar al servidor innecesariamente, haciendo uso del modelo “cliente-servidor” y como servidor de base de datos a Microsoft SQL Server 2005. (4)

Entre sus principales características, está la contabilización en tiempo real (que permite mantener actualizados los ficheros contables); y contabilización multimoneda (que permite registrar los activos y pasivos en las monedas orígenes sin tener que realizar en el momento del registro las conversiones de monedas, lo cual aumenta la exactitud de la información sobre la posición financiera de la institución). Además, las operaciones contables se pueden realizar a través de transacciones tipificadas que generan los asientos contables de forma automática. (4)

### *Bantotal*

Es una solución que comprende el procesamiento integral de todas las operativas de la Entidad Financiera. Está especialmente diseñado para mantener las operaciones del Banco bajo control. Resuelve los problemas como Ordenes de pagos y transferencias (Remesas), Venta de cheques de viajeros, reconocimiento de los clientes, ya sea por su nombre o razón social, organización a la que pertenece, número de cuenta, grupo financiero o por algún parámetro de interés en especial. Permite a los usuarios saber el quién, qué, cuándo, dónde y cómo de cada una de las transacciones. Además permite obtener un análisis detallado de la utilidad que se genera a nivel de cada agencia o sucursal, por ejecutivo de cuenta, por cliente, por grupo financiero o por producto. (5)

La arquitectura del Sistema Bantotal está estructurada lógicamente en capas, orientada a componentes y explotando la re-usabilidad de los mismos. Permite ejecutarse en Java o .Net, utilizando una interfaz gráfica bajo Navegador Web (Microsoft Internet Explorer o Mozilla Firefox) y sobre las plataformas IBM i-Series, Windows Server / SQL Server, Sistemas Unix / con gestores de Base de Datos DB2 u Oracle. (6)

Utiliza la herramienta GeneXus, que permite el diseño, la generación y el mantenimiento, tanto de las aplicaciones como de la base de datos de forma totalmente automática. (7)

### *Sistema Bancario Financiero Byte*

El Sistema Bancario Financiero BYTE, es un conjunto de módulos independientes e integrables, que permiten la automatización de los departamentos de instituciones financieras, según sean las

necesidades. Cuenta con más de 80 módulos que cubren casi todas las necesidades de Sistemas de Información de la banca latinoamericana, a nivel Gerencial, Operativo y Administrativo, es multiempresa, multimonedada y multisucursal; de los cuales el módulo Captaciones gestiona Cheques a través de los servicios de Cheques Certificados y Compensación de Cheques. Es una aplicación multicapas, en el cual los datos son almacenados en la base de dato que sea de preferencia, ya que el acceso a la misma es de manera universal (JDBC). Hace uso de la tecnología JAVA por lo que es multiplataforma; y para el mejor aprovechamiento de esta tecnología utiliza el modelo WEB para su interfaz gráfica (8)

### *SAP for Banking*

Las innovadoras funcionalidades centrales de este software de gestión bancaria permiten una perfecta interconexión entre las actividades de front-office y los sistemas de back-office y un procesamiento en tiempo real de las transacciones financieras clave. (9)

Está diseñada con la arquitectura orientada al servicio (SOA), cuya funcionalidad está alineada con los procesos del negocio, en paquetes de servicios interoperables. (10)

Las soluciones SAP for Banking funcionan con la plataforma tecnológica SAP NetWeaver que es el cimiento tecnológico unificado del funcionamiento de SAP Business Suite; es decir es una plataforma de negocios con aplicaciones que se integran con la infraestructura informática existente para hacer posible el cambio y su administración, y además implementa la mayoría de estándares de conectividad para DataWarehouses. Permite administrar, integrar y ampliar los procesos y aplicaciones del negocio a lo largo de las redes del negocio de sus clientes y puede integrarse con estándares de Internet tales como HTTP, XML y servicios Web. Garantiza apertura e interoperabilidad con Microsoft .NET y Java 2 Platform Enterprise Edition (J2EE) y entornos como IBM WebSphere. (11)

Sistemas	<i>Bantotal</i>	<i>Sistema Bancario Financiero Byte</i>	<i>SAP for Banking</i>
Plataforma	-IBM i-Series. -Windows Server / SQL Server. -Sistemas Unix.	Multiplataforma	-SAP NetWeaver, para el negocio. -IBM WebSphere. -Microsoft .NET. -J2EE.

Gestor de Base de Datos	-DB2. -Oracle.	Gestor que sea de preferencia.	Gestor que sea de preferencia.
Ambiente de desarrollo	-Java. -.NET.	-Java.	-Java. -.NET.
Interfaz gráfica	Modelo Web	Modelo Web	Modelo Web
Licencia	Marca comercial de De Larrobla & Asociados Internacional.	Producto comercial registrado por la empresa BYTE.	Producto comercial registrado por la compañía SAP.
Funcionalidades	-Ordenes de pagos y transferencias (Remesas). -Venta de cheques de viajeros	-Cheques Certificados. -Compensación de Cheques.	-Gestión de clientes. -Contabilidad financiera. -El análisis de riesgos.
Porcentaje de utilización de los sistemas en función de las funcionalidades que necesita el BNC para trabajar con los Cheques	20%	13.3%	6.66%

**Tabla : Aspectos que se tuvieron en cuenta en los Sistemas Automatizados de Gestión Bancaria. .analizados.**

Los sistemas bancarios internacionales, en su mayoría son multiplataforma al utilizar la tecnología JAVA en su desarrollo y su interfaz gráfica hace uso del modelo WEB para mejorar el acceso y disponibilidad de los servicios ofrecidos, aprovechando esta tecnología. Estos sistemas son privativos, lo que desde el punto de vista económico no es rentable para el Banco Nacional de Cuba. La gestión de Cheques y Carta de Remesa que realizan no se adaptan al negocio del Banco Nacional de Cuba, ya que no cumplen con la mayoría de las operaciones bancarias llevadas a cabo en esta empresa relacionada con este Título Valor.

Con la necesidad de automatizar el Banco Nacional de Cuba con las nuevas tecnologías, se desarrolla el producto Quarxo, al cual se le añade los módulos Cheque y Carta de Remesa, en la búsqueda de un sistema flexible, robusto y de módulos independientes e integrables.



## 1.5. Metodología, Lenguaje y Herramientas.

### 1.5.1. Metodología RUP.

Una metodología se define como: *“Conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal.”* y un método como: *“Procedimiento que se sigue en las ciencias para hallar la verdad y enseñarla.”* (12)

Rational Unified Process (RUP) es una metodología de desarrollo de software que está basado en componentes e interfaces bien definidas; y junto con el Lenguaje Unificado de Modelado (UML), constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Es un proceso que puede especializarse para una gran variedad de sistemas de software, en diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto. (13)

RUP es forma disciplinada de asignar tareas y responsabilidades en una empresa de desarrollo (quién hace qué, cuándo y cómo). (14)

### 1.5.2. Lenguaje de Modelado UML.

Unified Modeling Language (UML) es ante todo un lenguaje. Un lenguaje proporciona un vocabulario y unas reglas para permitir una comunicación. En este caso, este lenguaje se centra en la representación gráfica de un sistema. Aunque está pensado para modelar sistemas complejos con gran cantidad de software, el lenguaje es lo suficientemente expresivo como para modelar sistemas que no son informáticos, como flujos de trabajo (workflow ) en una empresa, diseño de la estructura de una organización y por supuesto, en el diseño de hardware. (15)

Es recomendable usar un lenguaje de modelado estándar, como UML, para facilitar la comunicación del sistema entre los diferentes participantes del proyecto de desarrollo. Además tiene un lenguaje y una variedad de diagramas que arquitectos, diseñadores y demás participantes pueden fácilmente construir y entender.

### 1.5.3. Herramienta CASE. Visual paradigm.

CASE es un acrónimo para Computer-Aided Software Engineering, es una herramienta que ayuda al ingeniero de software a desarrollar y mantener software. (12)

Es una combinación de herramientas de software y metodologías de desarrollo. (16)

Es definido como herramientas individuales para ayudar al desarrollador de software o administrador de proyecto durante una o más fases del desarrollo de software (o mantenimiento). (17)

Visual paradigm es una herramienta de modelado visual para todo tipo de diagramas UML. Es compatible con la gestión extensiva de casos de uso, diagramas de SysML (Systems Modeling Language), requisitos y el diseño de bases de datos (con ERD, por su sigla en inglés, Entity-relationship diagrams) y entrega los esfuerzos más eficaces en el análisis y diseño de sistemas para UML. (18)

Durante el desarrollo de los artefactos que contiene el flujo de trabajo de Análisis y Diseño se utilizó la versión Visual paradigm 3.1 para UML.

#### 1.5.4. Eclipse 3.3 IDE.

El Eclipse es una plataforma que ha sido diseñado desde el principio en la creación web integrando herramientas para el desarrollo de aplicaciones. En el diseño, la plataforma no ofrece gran funcionalidad al usuario final por sí mismo. El valor de la plataforma es fomentado por el desarrollo rápido de las funciones integradas y basadas en un modelo de plugin.

Eclipse proporciona una interfaz común de usuario (UI) de modelo para trabajar con herramientas. Está diseñado para ejecutarse en múltiples sistemas operativos, al mismo tiempo que ofrece una integración sólida con cada sistema operativo subyacente. (19)

#### 1.5.5. Contenedor Web (Apache Tomcat).

Apache Tomcat es un contenedor web basado en el lenguaje Java que actúa como motor de Servlets y Java Server Pages (JSPs). Se ha convertido en la implementación de referencia para las especificaciones de Servlets y JSPs. Está desarrollado en un entorno abierto Open Source. (20)

Durante el desarrollo de Quarxo se utilizó la versión Tomcat 6.0.14

#### 1.5.6. ER/Estudio

ER/Studio es una solución intuitiva y visual de modelado de datos para el diseño y mantenimiento de bases de datos transaccionales, de ayuda a la toma de decisiones y para la Web. Permite a los profesionales de bases de datos controlar, documentar y desplegar rápidamente cambios en el diseño en las principales plataformas de bases de datos. (21)

En la realización del diseño de la base de datos se utiliza la versión Embarcadero ERStudio 8.0.0.5865.

A partir de la utilización del núcleo del SABIC se agregaron módulos al mismo, que permitieron garantizar la base para el desarrollo de módulos como Cheque y Carta de Remesa; utilizándose la versión 8.0.0.5865 de Embarcadero ERStudio.

#### 1.5.7. Gestor de Bases de Datos. Microsoft SQL Server 2005

SQL Server es una solución de datos globales, integrados y de extremo a extremo que habilita a los usuarios en toda su organización mediante una plataforma más segura, confiable y productiva para datos empresariales y aplicaciones de Business Intelligence (BI). Microsoft SQL Server 2005 provee herramientas sólidas y conocidas a los profesionales de las Tecnologías de la Información (IT, por su sigla en inglés), así como también a trabajadores de la información, reduciendo la complejidad de la creación, despliegue, administración y uso de aplicaciones analíticas y de datos empresariales en plataformas que van desde los dispositivos móviles hasta los sistemas de datos empresariales. A través de un conjunto global de características, la interoperabilidad con sistemas existentes y la automatización de tareas rutinarias, SQL Server 2005 ofrece una solución completa de datos para empresas de todos los tamaños. (22)

#### 1.6. Patrones de Diseño

Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular (Buschman, 1996). (23)

##### 1.6.1 Patrones de asignación de responsabilidades. (GRAPS).

Los patrones GRASP (General Responsibility Assignment Software Patterns) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. (24)

Es usado para la correcta asignación de responsabilidades en el diseño orientado a objetos.

##### 1.6.1.1 Experto.

Es un patrón que se usa para asignar la responsabilidad a objetos, de que cumplan con ellas con la información necesaria que poseen. Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento. El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases "sencillas" y más cohesivas que son más fáciles de comprender y de mantener. Así se brinda soporte a una alta cohesión. (24)

#### 1.6.1.2 Creador.

Guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. (24)

#### 1.6.1.3 Bajo Acoplamiento.

El Bajo Acoplamiento estimula asignar una responsabilidad de modo que su colocación no incremente tanto el acoplamiento para que no se produzcan los resultados negativos propios de un alto acoplamiento. Soporta el diseño de clases más independientes, que reducen el impacto de los cambios y también más reutilizables, que acrecientan la oportunidad de una mayor productividad. (24)

#### 1.6.1.4 Controlador.

Es el intermediario entre una interfaz y la implementación de la lógica de las operaciones del sistema. Se responsabiliza por controlar el flujo de eventos del sistema. (24)

#### 1.6.1.5 Alta Cohesión

Caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme, lo que no las hace susceptibles a constantes cambios. (24)

### 1.6.2 Patrones Estructurales.

#### 1.6.2.1 Facade (Fachada).

Es el único punto de entrada para los servicios de un subsistema; la implementación y otros componentes del subsistema son privados y no pueden verlos los componentes externos. Proporciona protección frente a los cambios en las implementaciones de un subsistema. (24)

Proporcionar una interfaz unificada para un conjunto de interfaces de un subsistema. Este patrón define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar. (25)

### 1.6.3 Patrones de Comportamiento

#### 1.6.2.2 DAO (Data Access Object).

Oculto la forma de acceder a los datos, es decir; trata de que el software cliente se centre en los datos que necesita y se olvide de cómo se realiza el acceso a los datos o de cuál es la fuente de almacenamiento. (26)

#### 1.6.4 Patrones de Presentación.

##### 1.6.3.1 Composite View (Vistas Compuestas)

Este patrón es con el propósito de crear Vistas Compuestas de varias sub-vistas de forma modular, flexible y extensible para construir vistas de páginas Java Server Page (JSP) para aplicaciones J2EE. Cualquier cambio realizado en una sub-vista es reflejado automáticamente en cada Vista Compuesta que la utilice. La Vista Compuesta también maneja la disposición de sus sub-vistas y proporciona una plantilla, dando una apariencia consistente y facilidades a la hora de modificarla y mantenerla a lo largo de toda la aplicación. (27)

##### 1.6.3.2 MVC (Modelo Vista Controlador)

El patrón Modelo-Vista-Controlador de arquitectura (MVC) divide una aplicación interactiva en tres componentes. El Modelo contiene la funcionalidad del núcleo y de datos. Las Vistas muestran información del usuario. Los Controladores se encargan de manejar la entrada del usuario, es decir; es el encargado de escuchar los cambios en la Vista y enviarlos al Modelo, el cual le regresa los datos a la Vista como un ciclo. (28)

#### 1.7. Ambiente de desarrollo

##### 1.7.1 Lenguaje de programación. Java.

Java es un lenguaje sencillo orientado a objetos; es independiente de plataforma, por lo que un programa hecho en Java se ejecutará igual en un PC con Windows que en una estación de trabajo basada en Unix. Su capacidad multihilo y su robusta integración que tiene con el protocolo TCP/IP, lo hace un lenguaje ideal para Internet. (29)

El JDK (Java Development Kit) es un producto que permite crear aplicaciones en Java. Este paquete incluye un conjunto de herramientas para compilar, depurar, generar documentación e interpretar código escrito en Java. (29)

Para poder ejecutar cualquier aplicación Java en cualquier sistema operativo es necesario tener instalado el JRE (Java RunTime Environment - Entorno de desarrollo de Java). El JRE se compone de herramientas necesarias como la máquina virtual de java (java.exe) y el conjunto de librerías estándar de Java. El JDK incluye a JRE. (29)

Para el desarrollo del producto Quarxo se utiliza la versión: Java (TM) 6 Update 20 versión: 6.0.200

### 1.7.2 Entorno de desarrollo. J2EE

La plataforma J2EE es una especificación para el desarrollo de aplicaciones empresariales basadas en web que corren en ambientes multicapa. J2EE proporciona un modelo de programación que consiste en un conjunto de Application Programming Interface (APIs) y dirige la manera de construir aplicaciones. Por otra parte, J2EE especifica cómo debe ser la infraestructura de la aplicación sobre la cual puedan correr las aplicaciones. Esta infraestructura de la aplicación es proporcionada por los contenedores de las implementaciones de J2EE. (30)

La plataforma J2EE es esencialmente un ambiente de servidor de aplicaciones distribuidas que proporciona:

- Un conjunto de API's de extensión Java para construir aplicaciones y que definen un modelo de programación para aplicaciones J2EE.
- Una infraestructura en tiempo de ejecución para hospedar y gestionar aplicaciones.

Las características y funcionalidad de la plataforma permiten la creación de aplicaciones basadas en componentes escalables, distribuidas y flexibles. (30)

## 1.8. Tecnologías y Frameworks.

### 1.8.1 Spring Framework

Spring Framework es libre y está compuesto por un conjunto de módulos, de los cuales se pueden tomar aquellos que faciliten la implementación del trabajo en cuestión. El centro de Spring está basado en el principio de Inyección de Dependencias. Esta técnica hace externa la creación y el manejo de las dependencias de las clases, con lo que se logra una mayor limpieza y claridad en el código. (31)

Se utiliza la versión 2.0.6 en el desarrollo de Quarxo.

### 1.8.2 Spring Web Flow

Es un framework que permite realizar la definición de flujos de interfaces de usuario en las aplicaciones web, además permite la definición de los flujos tanto en formato de clases Java, como utilizando un archivo de configuración Extensible Markup Language (XML), lo que facilita la tarea de la definición de transformaciones de modelo a texto y la abstracción de los conceptos fundamentales que se manejan. (32)

Spring Web Flow se centra en proporcionar la infraestructura para construir y ejecutar aplicaciones ricas de Internet que requieren de varios pasos interacciones. (33)

Spring Web Flow permite la implementación de los "flujos" de una aplicación web. Un flujo encapsula una secuencia de pasos que guían al usuario a través de la ejecución de una tarea de negocios, que abarca múltiples peticiones HTTP, es reutilizable y puede ser dinámico. (34)

Se utiliza para el desarrollo del producto Quarxo la versión 2.0.6.

### 1.8.3 Hibernate

Hibernate es framework para resolver el problema de persistencia de datos en Java, entre la aplicación y la base de datos relacional, dejando al desarrollador concentrarse en los problemas de la aplicación. La integración de Hibernate es sencilla con aplicaciones anteriores y nuevas. Resumiendo es un framework Java para el Mapeado Objeto/Relacional (Object/Relational Mapping, ORM por sus siglas en inglés). (35)

Se utiliza en el desarrollo de Quarxo la versión 3.5.

### 1.8.4 Dojo toolkit

DOJO es un framework de JavaScript que nos ofrece muchos instrumentos, como son controles del interfaz de usuario, efectos, utilidades comunes; es una API para gestionar el acceso asíncrono al servidor. (36)

Para el desarrollo del producto Quarxo se utiliza la versión 1.3.

### 1.8.5 JUnit

Framework desarrollado para realizar pruebas automatizadas a los sistemas informáticos durante la etapa de construcción. Su objetivo principal es evaluar el funcionamiento de cada uno de los métodos dentro de las clases que conforman el software. Facilita la aplicación de la práctica Test Driven Development (TDD). Incluye formas de ver los resultados que pueden ser en modo texto o gráfico. Es uno de los más utilizados de su tipo en el lenguaje Java, cuenta con una amplia comunidad de desarrollo y es bastante maduro. (31)

Para llevar a cabo las pruebas unitarias si uso la versión 4.10.

#### 1.9. Conclusiones parciales.

El análisis realizado en este capítulo permitió el desarrollo de los módulos Cheque y Carta de Remesa del producto Quarxo. Se realizó un estudio relacionado con la gestión de los procesos de Cheques y Cartas de Remesa en el Banco Nacional de Cuba, así como una breve caracterización de las herramientas, lenguajes, frameworks y patrones de diseño definidos en la arquitectura de Quarxo. El estudio y la caracterización realizada nos garantizarán la obtención de los módulos Cheque y Carta de Remesa con la calidad requerida para el BNC.



## CAPÍTULO 2. ARQUITECTURA Y DISEÑO.

## 2.1 Introducción.

Este capítulo hace uso de los requisitos funcionales para el correcto diseño de los módulos Cheques y Carta de Remesa, de manera tal que los artefactos generados sirvan de guía para la futura implementación del producto, haciendo uso de la arquitectura de desarrollo definida para el sistema Quarxo.

## 2.2 Arquitectura de sistema Quarxo.

El sistema Quarxo está implementado en una arquitectura por capas, de las cuales: *Capa de Presentación*, *Capa de Negocio*, *Capa de Acceso a Datos* son las principales; y además posee la *Capa de Dominio* con objetos del dominio que están presentes en el resto de la capas.

La organización de la arquitectura se realiza de forma jerárquica donde:

- *Subsistema*: Conjunto de módulos relacionados con los procesos que ejecutan.
- *Módulo*: Conjunto de Casos de Uso relacionados con uno o más procesos bancarios.
- *Componente*: Conjunto de funcionalidades comunes que son reutilizados por el resto de los módulos del sistema.

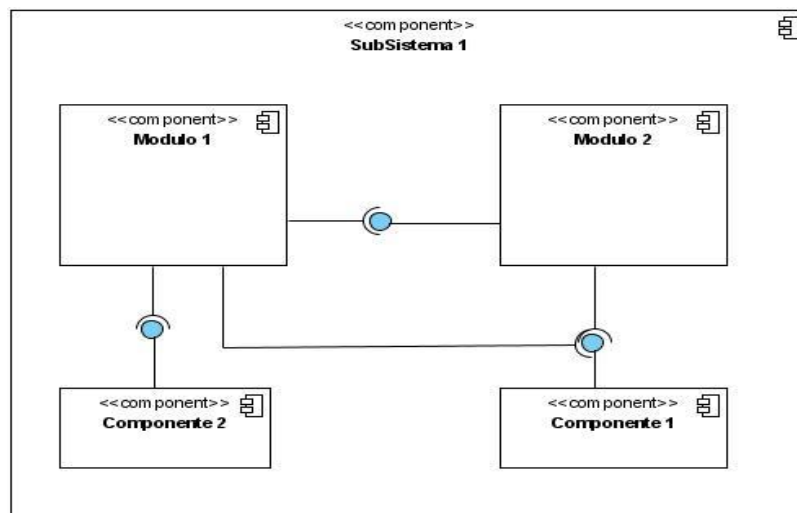


Figura : Arquitectura de un subsistema del sistema Quarxo. Reutilización de componentes.

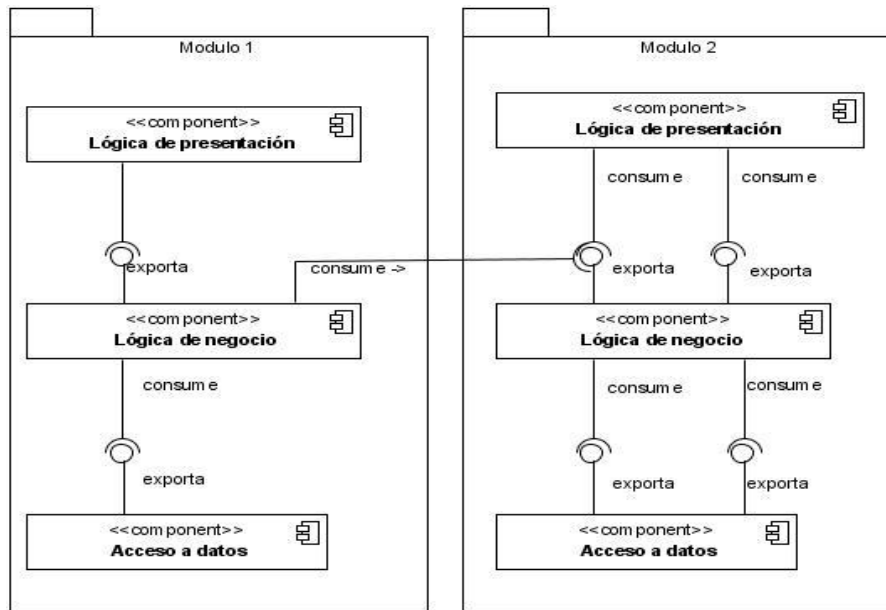


Figura : Arquitectura de un subsistema del sistema Quarxo. Relación entre módulos.

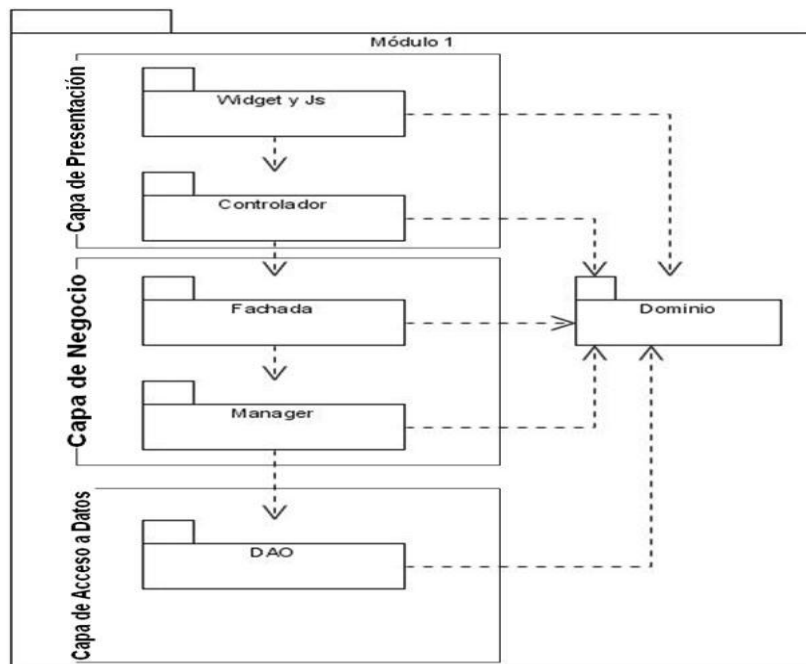


Figura : Arquitectura de un módulo del sistema Quarxo.

### 2.2.1 Capa de Presentación.

Esta capa es la encargada de llevar acabo toda la lógica de presentación, haciendo uso de frameworks como Spring WebFlow para la definición de flujos de interfaces de usuario y Dojo Toolkit para la decoración de interfaces que interactúan con el usuario. El framework Spring MVC para cargar datos necesarios para completar la funcionalidad que se lleve a cabo.

### 2.2.2 Capa de Negocio.

La Capa de Negocio está formada por dos paquetes: *Facade* y *Manager*. El paquete *Facade* agrupa las funcionalidades que puedan ser invocadas desde un módulo o desde la Capa de Presentación; hace función de puente. El paquete *Manager* contiene las clases necesarias para la implementar la lógica de negocio; hace uso de la Capa de Acceso a Datos para obtener y actualizar los datos persistidos y la Capa de Dominio para generar las entidades de negocio.

### 2.2.3 Capa de Acceso a Datos.

Esta capa es la encargada de la conexión con la base de datos y se realizan las operaciones que acceden a los datos del sistema. Se hace uso del framework Hibernate que tiene implícito el patrón DAO para su desarrollo y se accede desde la Capa de Negocio a través de sus interfaces.

## 2.3 Diseño de los módulo Cheque y Carta de Remesa.

Se hace uso de los requerimientos definidos por los analistas del sistema Quarxo, para diseñar las funcionalidades siguientes agrupadas por módulo:

#### Módulo Cheque:

- Registrar Cheque
- Actualizar Cheque
- Consultar Cheque
- Certificar Cheque

#### Módulo Carta de Remesa:

- Registrar Carta de Remesa
- Crear Carta de Remesa
- Consultar Carta de Remesa
- Confirmar Ingreso

- Reintegrar Cheque
- Suspender Pago
- Caducar Cheque

### 2.3.1 Diagramas de Paquete.

Se hace uso de este tipo de diagrama ya que nos ayuda a tener una visión clara de la organización y dependencias de los paquetes junto sus contenidos. Estos paquetes fueron creados según la funcionalidad e información que contengan.

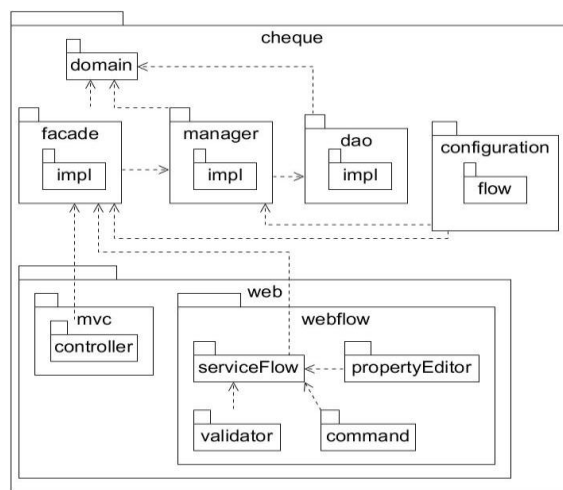


Figura : Diagrama de Paquete del módulo Cheque.

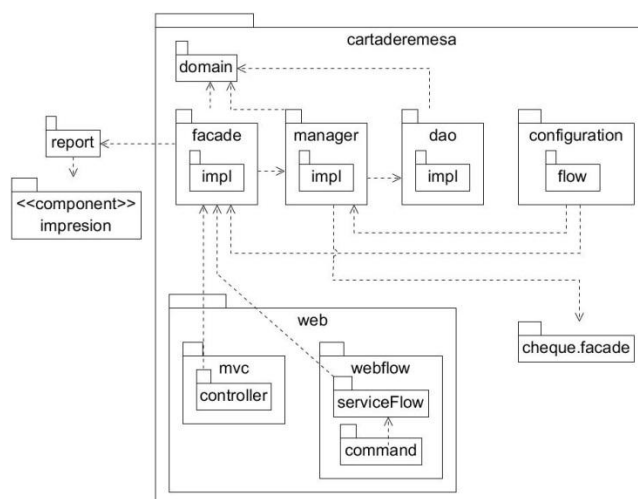


Figura : Diagrama de Paquete del módulo Carta de Remesa.

A partir de los diagramas presentados en la figura 4 y 5, se hará una breve descripción de los paquetes existentes en cada uno de ellos.

- *Paquete facade*: se encuentra la interfaz que brinda las funcionalidades que serán usadas por la presentación e invocadas desde otros módulos. La clase interfaz correspondiente a este paquete tiene como sufijo “*Facade*”.
- *Paquete manager*: se encuentra la interfaz que brinda las funcionalidades relacionadas con la lógica de negocio. La clase interfaz correspondiente a este paquete tiene como sufijo “*Manager*”
- *Paquete dao*: se encuentra la interfaz que brinda las funcionalidades encargadas de la comunicación con la base de datos y contiene además los ficheros de mapeo correspondientes al módulo que utilizados por el framework Hibernate. La clase interfaz correspondiente a este paquete tiene como sufijo “*DAO*”
- *Paquete impl*: se encuentra la clase que implementa la interfaz correspondiente al paquete que lo contiene. La clase que implementa la interfaz correspondiente tiene el nombre de la clase interfaz y se le añade sufijo “*Impl*”.
- *Paquete domain*: se encuentran las clases relacionadas con el dominio del módulo.
- *Paquete web*: se encuentran los paquetes y las clases encargadas de realizar la lógica de presentación en el lado del servidor.
- *Paquete mvc*: se encuentran todos los paquetes y clases encargadas de realizar la lógica de presentación cuando se hace uso del framework Spring MVC.
- *Paquete controller*: se encuentran las clases que heredan de los controladores que ofrece Spring MVC, encargadas de responder a las peticiones realizadas por el cliente.
- *Paquete webflow*: se encuentran todos los paquetes y clases encargadas de realizar la lógica de presentación cuando se hace uso del framework Spring WebFlow.
- *Paquete command*: se encuentran las clases que representan los datos que son introducidos y mostrados en los formularios pertenecientes a una interfaz usuario. Esta clase tiene como sufijo “*Command*”.
- *Paquete propertyEditor*: se encuentran las clases que son intermediarias para convertir los datos de la interfaz de usuario en objetos y viceversa. Esta clase tiene como sufijo “*PropertyEditor*”.

- *Paquete validator*: se encuentran las clases encargadas de validar los datos. Esta clase tiene como sufijo “*Validator*”.
- *Paquete serviceFlow*: se encuentran las clases encargadas de atender los eventos de Spring WebFlow, realizados en el lado del cliente a través de sus métodos. Esta clase tiene como sufijo “*MultiAction*”.
- *Paquete configuración*: contiene los ficheros XML que hace uso Spring Framework para su configuración.
  - -dataaccess.xml: contexto de acceso a datos.
  - -business.xml: contexto de negocio.
  - -webflow.xml: contexto de Spring WebFlow.
  - -servlet.xml: contexto de Spring.
- *Paquete flow*: contiene los ficheros XML que definen el flujo de las funcionalidades de un módulo.

### 2.3.2 Diseño de la Capa de Presentación.

En esta capa se hace uso del framework Spring WebFlow para dar solución a las funcionalidades:

#### Módulo Cheque:

- Registrar Cheque
- Actualizar Cheque
- Consultar Cheque
- Certificar Cheque
- Reintegrar Cheque
- Suspender Pago

#### Módulo Carta de Remesa:

- Registrar Carta de Remesa
- Crear Carta de Remesa
- Consultar Carta de Remesa
- Confirmar Ingreso

Y usa además el framework Spring MVC para la funcionalidad Caducar Cheque correspondiente al módulo Cheque.

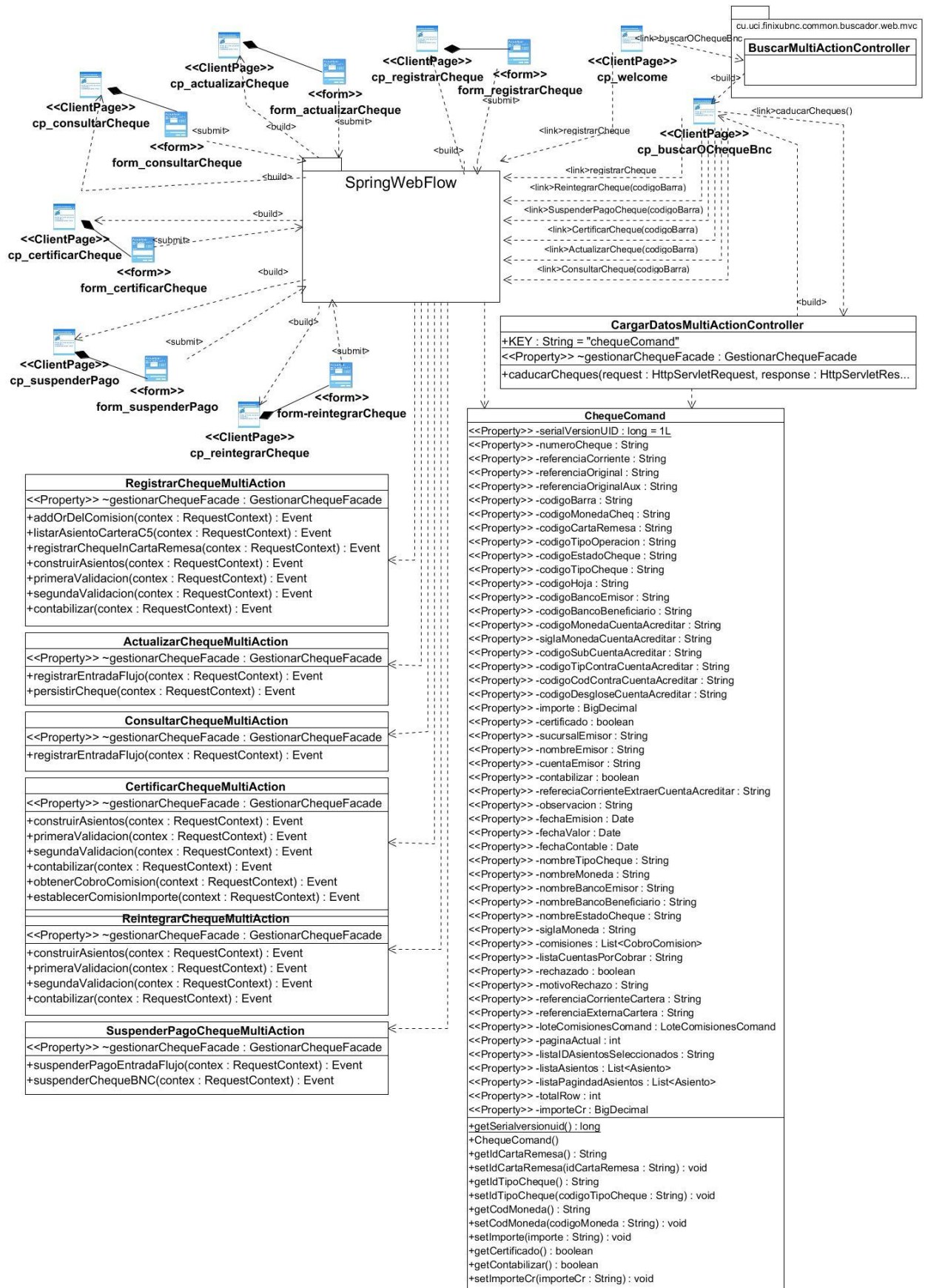


Figura : Diagrama de clases del diseño. Módulo Cheque.

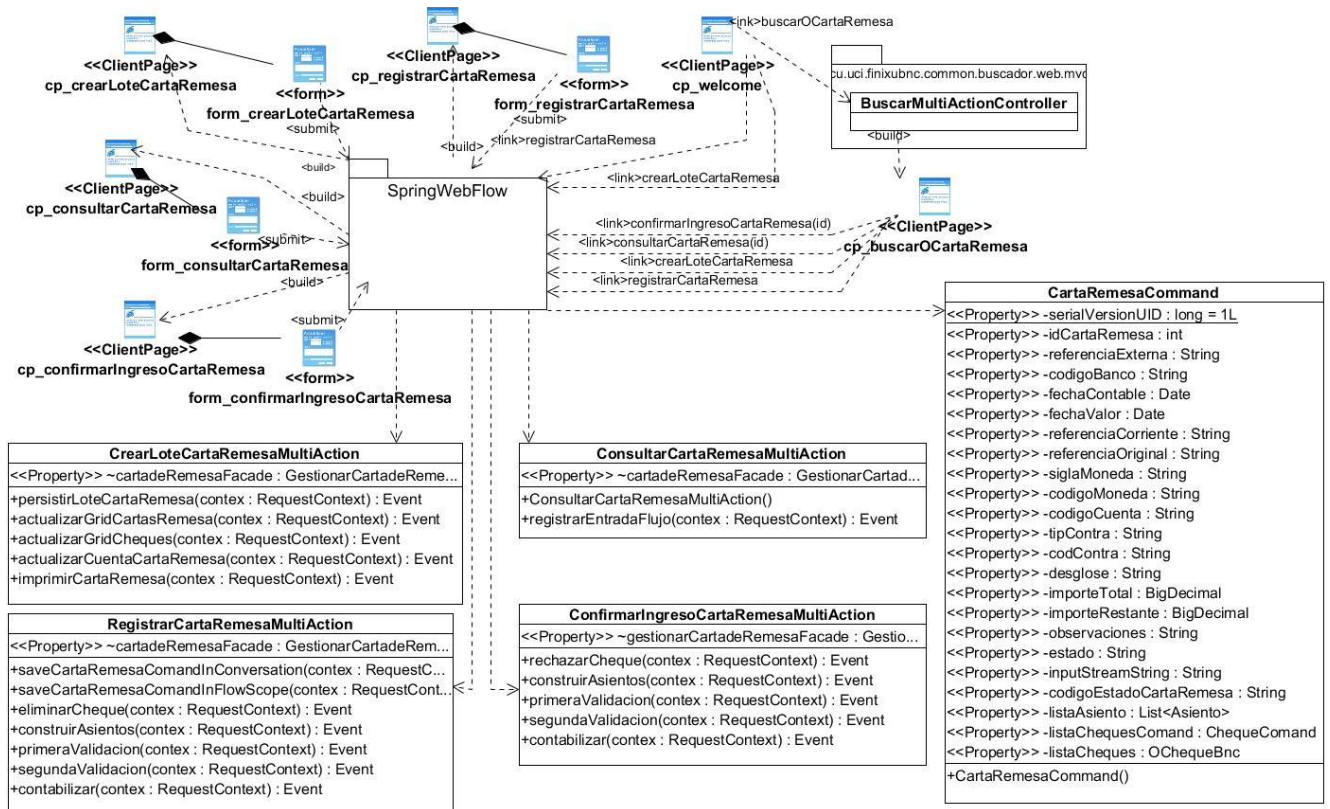


Figura : Diagrama de clases del diseño. Módulo Carta de Remesa.

Descripción de las clases presentes en la figura 6 y 7:

- *ClientPage*: son las páginas que están en el lado del cliente, normalmente páginas HTML y scripts (javascript).
- *form*: es la representación de un formulario. Es código HTML que contiene etiquetas de formulario como: <input>, <textarea>, <select>.
- *SpringWebFlow*: representa la función del framework Spring WebFlow, que es el encargado de controlar las peticiones realizadas desde el cliente en el flujo de trabajo.
- *Clases que tengan el sufijo "Command"*: son utilizadas para almacenar y enviar información presente en el formulario, y simula al objeto del dominio.



Ejemplo: Si la clase se llama *ChequeComand*, entonces esta clases almacena y envía información del formulario, y simula al objeto del dominio *OChequeBnc*.

- *Clases que tengan el sufijo "MultiAction"*: son las encargadas de tener los métodos asociados a las peticiones realizadas a través de Spring WebFlow. El prefijo hace referencia a la funcionalidad que atiende.

Ejemplo: Si la clase se llama *RegistrarChequeMultiAction*, entonces tiene los métodos asociados a las peticiones realizadas en el flujo de la funcionalidad "registrar Cheque".

- *CargarDatosMultiActionController*: se encarga de tener los métodos asociados a las peticiones realizadas a través del framework Spring MVC.

### 2.3.3 Diseño de la Capa de Negocio.

Las clases que tienen en el nombre el sufijo "*Manager*" son clases interfaces que contienen los métodos que podrán ser accedidos por una clase *Facade*; y las que contengan "*ManagerImpl*" implementan la lógica de negocio.

Las clases que contengan en el nombre el sufijo "*Facade*" son clases interfaces que brindan funcionalidades que podrán ser invocadas desde la Capa de Presentación o desde un módulo; y las que tengan "*FacadeImpl*" realizan la función de puente entre los paquetes *facade* y *manager*.



Figura : Diagrama de clases de la Capa de Negocio. Módulo Cheque.

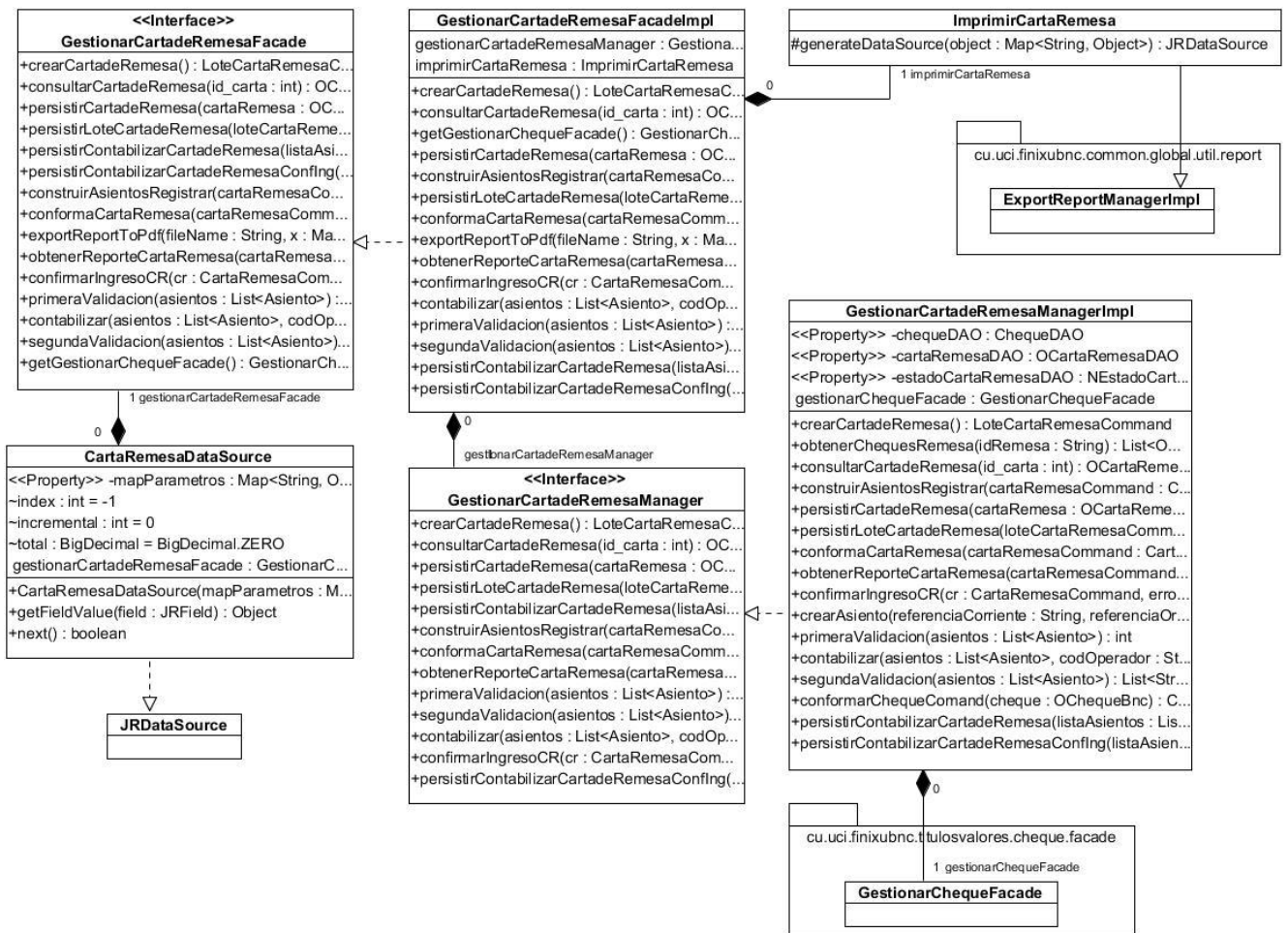


Figura : Diagrama de clases de la Capa de Negocio. Módulo Carta de Remesa.

En la figura 9 la clase *GestionarCartadeRemesaManagerImpl*, tiene un atributo que hace referencia a la clase *GestionarChequeFacade* para acceder a funcionalidades necesarias, para la gestión de Cheques contenidos en una Carta de Remesa; ya que una Carta de Remesa tienes un conjunto de Cheques del mismo tipo, moneda y banco emisor. La clase *JRDataSource* pertenece a la librería *jasperreport-4.1.1.jar* necesaria para la impresión de reportes; de esta clase hereda la clase *CartaRemesaDataSource*, la cual tiene un atributo que hace referencia a *GestionarCartaRemesaFacade* para que el método *getFieldValue (JRField field)*, haga uso de las funcionalidades necesarias para llenar los datos de la plantilla de impresión. La clase *ImprimirCartaRemesa* hereda de la clase *ExportReportManagerImpl*; el método *generateDataSource*

(Map<String, Object> object) crea un objeto *CartaRemesaDataSource* para la obtener un fichero con los datos para su futura impresión.

### 2.3.4 Diseño de la Capa de Acceso a Datos.

Las clases que tienen en el nombre el sufijo “DAO” son clases interfaces que brindan funcionalidades que podrán ser accedidas desde la Capa de Negocio y heredan de la clase *BaseDAO* del framework Hibernate.

Las clases que tienen en el nombre el sufijo “DAOImpl” implementan las funcionalidades de acceso a datos y heredan de la clase *AbstractBaseDAO* del framework Hibernate.

Las clases *BaseDAO* y *AbstractBaseDAO* brindan un conjunto de funcionalidades relacionadas con las clases de dominio como listar, persistir, actualizar, eliminar, buscar.

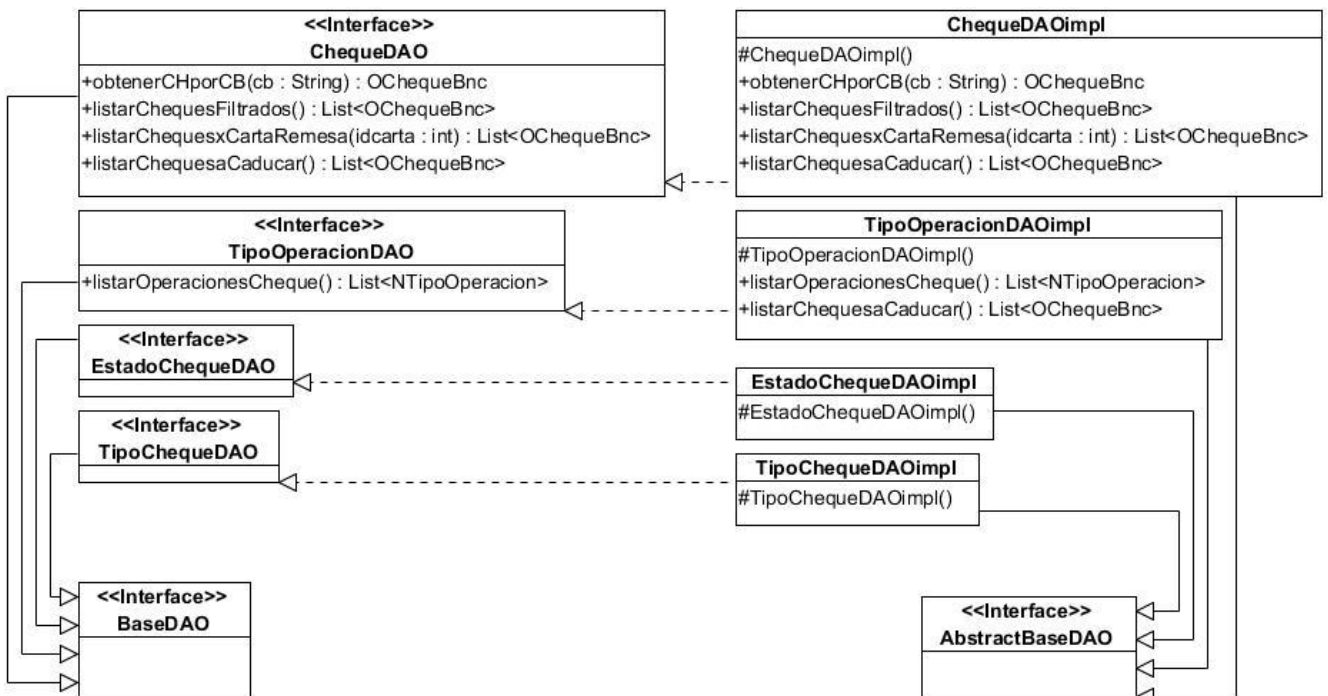


Figura : Diagrama de clases de la Capa de Acceso a Datos. Módulo Cheque

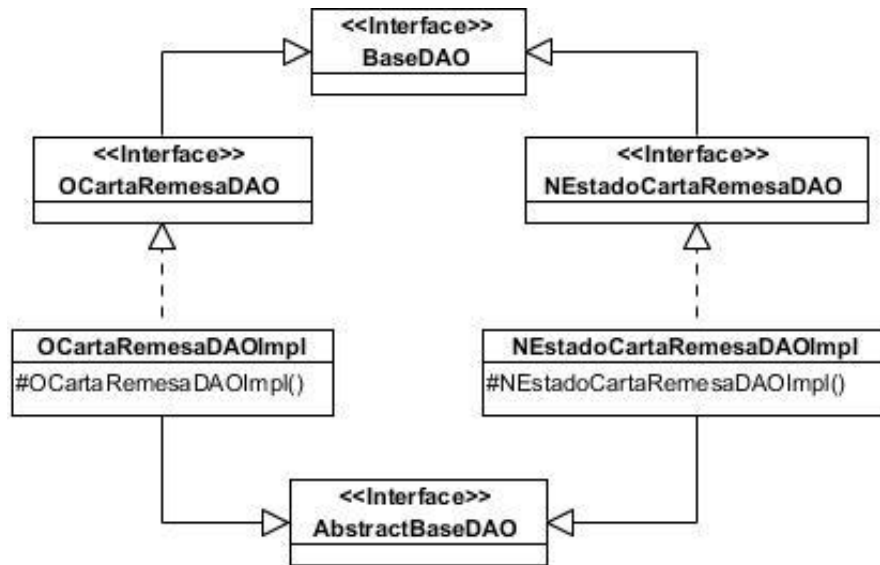


Figura : Diagrama de clases de la Capa de Acceso a Datos. Módulo Carta de Remesa

### 2.3.5 Diseño de la Capa de Dominio.

En esta capa se encuentran los objetos que existen en los módulos que se van a desarrollar y sus relaciones, para la gestión de los procesos de Cheques y Cartas de Remesa. Las reuniones hechas con los expertos de negocio hicieron posible la identificación de estas clases centradas en el parte del negocio.

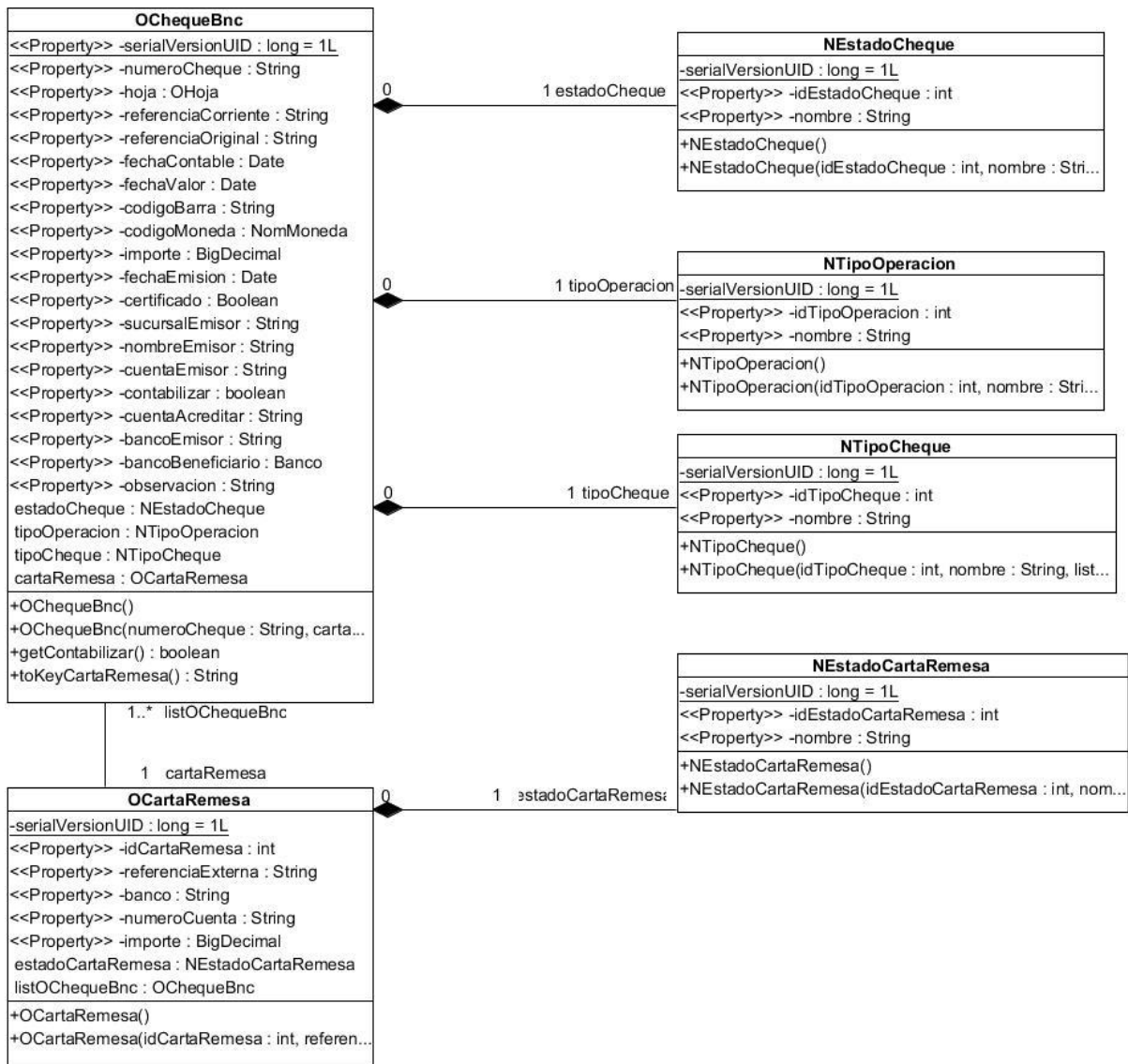


Figura : Diagrama de clases de la Capa Dominio.

### 2.3.6 Patrones de diseño empleados.

Durante el diseño de los módulos se utilizaron patrones para solucionar diversos problemas comunes en el desarrollo de software. A continuación se describe el uso de estos patrones:

*Experto:* Se evidencia la utilización del patrón en la clase *RegistrarChequeMultiAction*, responsable de manejar los eventos concernientes a la funcionalidad Registrar Cheque.

*Controlador:* La clase *CargarDatosMultiActionController*, responsable de contener los métodos que cargan los datos que serán visualizados en los componentes de la interfaz mostrada al usuario.

*Alta Cohesión:* Definición de dos módulos con la menor cantidad posible de clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.

*Bajo Acoplamiento:* Definición de clases interfaces accedidas por otras clases; clases controladoras que evidencian el desarrollo de sus responsabilidades, disminuyendo el impacto generados por los cambios en el negocio.

*Facade:* Figura : Diagrama de clases de la Capa de Negocio. Módulo Cheque., que muestra como fue utilizado dicho patrón.

*DAO (Data Access Object):* Figura : Diagrama de clases de la Capa de Acceso a Datos. Módulo Cheque muestra cómo fue llevado acabo su uso a través del framework Hibernate.

*MVC (Modelo Vista Controlador):* Figura : Diagrama de clases del diseño. Módulo Cheque. muestra cómo fue llevado acabo su uso, donde las clases con estereotipo ClientPage son la vista, la clase ChequeComand es el modelo; y las clases con sufijo MultiAction y la clase CargarDatosMultiActionController hacen función de controlador.

*Composite View (Vistas Compuestas):* Fichero *certificarCheque.jsp* que contiene código HTML para construir la estructura de la interfaz mostrada al usuario y para dar cumplimiento a la funcionalidad hace uso de la etiqueta *"include"* para contener otra vista en su estructura.

### 2.3.7 Diagramas de Secuencia.

Estos diagramas contienen las clases y objetos que se usan para implementar una funcionalidad y los mensajes intercambiados durante su ejecución para realizar las tareas necesarias.

Se muestran algunos diagramas de secuencia en el:

Anexo : Diagrama de secuencia. Registrar Cheque.

Anexo : Diagrama de secuencia. Certificar Cheque

Anexo : Diagrama de secuencia. Reintegrar Cheque

Anexo : Diagrama de secuencia. Registrar Carta de Remesa

## 2.4 Modelo de Datos.

El siguiente diagrama nos brinda una visión de las relaciones y restricciones de los datos del proceso de negocio, persistentes en los módulos a desarrollar.

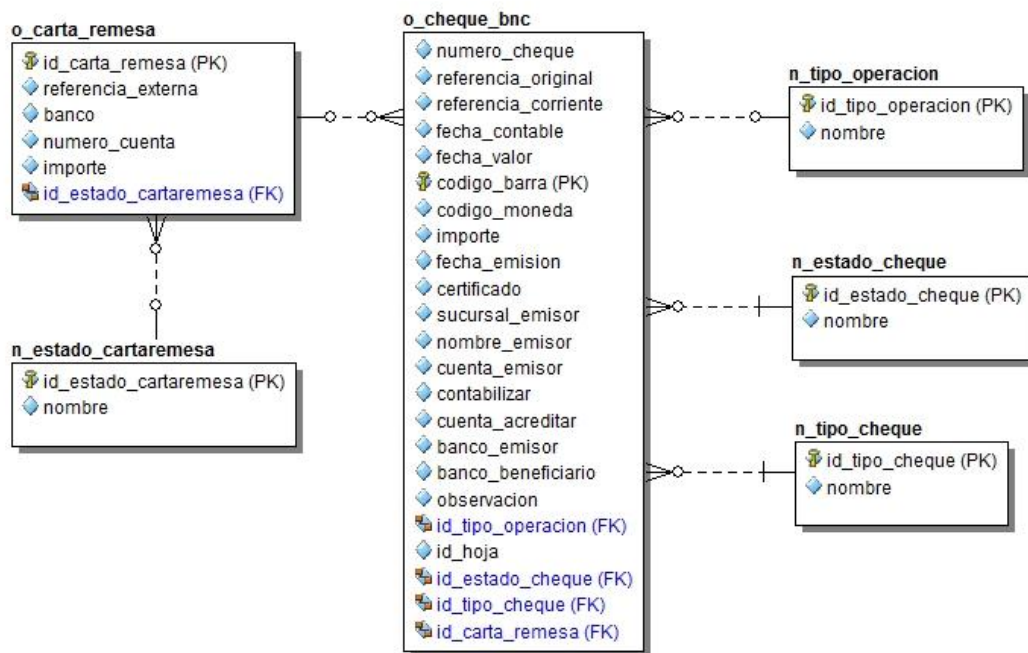


Figura : Modelo de Datos.

La tabla *o\_cheque\_bnc* almacena los datos de los Cheques que hayan sido gestionados. Las tablas *n\_tipo\_operacion*, *n\_estado\_cheque* y *n\_tipo\_cheque* almacenan los datos de los diferentes tipos de operación, estados y tipos de cheques respectivamente.

La tabla *o\_carta\_remesa* almacenadas los datos de las Cartas de Remesa que se hayan creado o registrado. La tabla *n\_estado\_cartaremesa* contiene los diferentes estados por los que puede transitar una Carta de Remesa.



### 2.5 Conclusiones parciales.

Este capítulo evidencia el uso de la arquitectura definida para el desarrollo del sistema Quarxo. Detallando los artefactos generados para un diseño flexible de los módulos Cheque y Carta de Remesa. Teniendo en cuenta la explicación de cada uno de los patrones utilizados, demostrándose así la efectividad que poseen al ser aplicables en diferentes problemas de diseño. El diseño realizado se corresponde con los requisitos funcionales definidos para los módulos Cheque y Carta de Remesa, dando paso a la futura implementación de los mismos.

## CAPÍTULO 3.IMPLEMENTACIÓN Y PRUEBA.

### 3.1 Introducción.

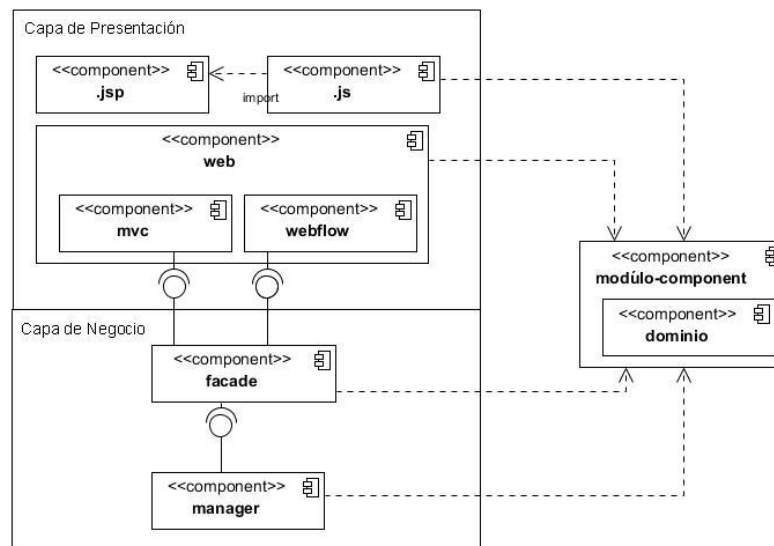
En el capítulo se detalla el desarrollo de la implementación de los módulos Cheque y Carta de Remesa. Se especifican los estándares de codificación usados en la generación del código. Se hace una breve descripción de los atributos y métodos de las clases relevantes desde el punto de vista funcional y se expone el diagrama de componentes realizado. Luego se lleva a cabo la validación de la implementación para la gestión de los procesos de Cheques y Cartas de Remesa, a través de las técnicas de diseño de casos de prueba de caja blanca y caja negra.

### 3.2 Implementación de la solución para los módulos Cheque y Carta de Remesa.

Los artefactos generados en la fase de diseño disponen de suficientes detalles para la implementación de los módulos Cheque y Carta de Remesa. Este flujo de trabajo propuesto por RUP, está determinado por el lenguaje de programación, se genera el diagrama de componentes para describir los componentes a construir, su organización y dependencia y se lleva a cabo la implementación de cada una de las clases significativas del diseño obteniendo como resultado un sistema ejecutable.

#### 3.2.1 Diagrama de componentes.

Se muestran las dependencias lógicas entre componentes software. Un diagrama de componentes permite visualizar con más facilidad la estructura general del sistema y el comportamiento del servicio que estos componentes proporcionan y utilizan a través de las interfaces.



**Figura : Diagrama de Componentes.**

### 3.2.2 Estándares de Codificación.

Los estándares de codificación son guías relacionadas con la generación del código, para facilitar la legibilidad, comprensión y mantenimiento del código entre distintos programadores. Define la nomenclatura de las variables, objetos, métodos y funciones.

#### 3.2.2.1 Convenciones de Nomenclatura

- El nombre de las clases tienen que ser sustantivos comenzando con la primera letra en mayúscula y el resto en minúscula, si es una palabra compuesta el inicio de cada palabra será en mayúscula.

Ejemplo: *GestionarChequeManager*.

- Las constantes se tienen que escribir en mayúscula.
- Los atributos, métodos y ficheros con extensión `.js` y `.jsp`, se nombrarán comenzando con la primera letra en minúscula, si es compuesta cada palabra comenzará en mayúscula excepto la primera palabra. Los métodos tienen que estar formado por palabras en infinitivo. Los atributos tienen el mismo nombre de la clases a la que instancian.

Ejemplos:

- El atributo *gestionarChequeManager* que instancia a la clase *GestionarChequeManager*.
- El método *validarCodigoBarra*.

### 3.2.2.2 Nomenclatura según el tipo de clases

- Las clases contenidas en el paquete *controller*, se nombran adicionándoles el nombre del controlador de Spring MVC de cual heredan.  
Ejemplo: *CargarDatosMultiActionController*
- Las clases contenidas en los paquetes *flowHandler*, *validator*, *command*, *propertyEditor*, se nombran adicionándoles el nombre del paquete al que pertenecen.  
Ejemplo: *OChequeBncPropertyEditor*
- Las clases contenidas en el paquete *serviceFlow*, se nombran adicionándoles el nombre *MultiAction* que es el controlador que heredan.  
Ejemplo: *RegistrarChequeMultiAction*
- Las clases contenidas en los paquetes *facade*, *manager*, *dao*, se nombran adicionándoles el nombre del paquete al que pertenecen y son clases interfaces. En el subpaquete *impl*, se nombran con el mismo nombre de las clases que implementan y se les adiciona el nombre del subpaquete.  
Ejemplo: La clase interfaz *ChequeDAO*, de la cual implementa la clase *ChequeDAOimpl*.

### 3.2.3 Descripción de algunas clases principales utilizadas.

<b>Nombre:</b> <i>CargarDatosMultiActionController</i>	
<b>Tipo de clase:</b> Controladora	
<b>Atributo</b>	<b>Tipo</b>
<i>gestionarChequeFacade</i>	<i>cheque.facade.GestionarChequeFacade</i>
Para cada responsabilidad:	
<b>Nombre:</b>	<b>Descripción:</b>

<i>caducarCheques</i> (HttpServletRequest request, HttpServletResponse response)	Caducan los cheques que hayan sido emitidos 60 días con anterioridad.
<i>obtenerChequePorCodigoBarra</i> (HttpServletRequest request, HttpServletResponse response)	Se obtiene la información del Cheque correspondiente al código de barra enviando en el request.
<i>validarCodigoBarra</i> (String codigoBarra)	Dado el código de barra este válida la estructura del código y verifica si la información que se pueda derivar existe en la base de datos.
<i>slbtrBanco</i> (String codigoBanco, String codigoMoneda)	Informa si el banco enviado por parámetro está integrado al SLBTR en una moneda dada.
<i>cargarBancosPaginado</i> (HttpServletRequest request, HttpServletResponse response)	Lista todos los bancos nacionales,
<i>listarCarteras</i> (HttpServletRequest request, HttpServletResponse response)	Lista todos los asientos referentes a una cuenta cartera.

**Tabla : Descripción de la clase CargarDatosMultiActionController**

<b>Nombre:</b> <i>GestionarChequeFacade</i>	
<b>Tipo de clase:</b> Interfaz	
<b>Atributo</b>	<b>Tipo</b>
-----	-----
Para cada responsabilidad:	
<b>Nombre:</b>	<b>Descripción:</b>

<i>certificarCheque</i> (ChequeComand cheque)	Se extrae el total del importe de la cuenta de emisor del Cheque y se acredita en una cuenta cartera
<i>reintegrarCheque</i> (ChequeComand cheque)	Se reversa la operación de certificar Cheque.
<i>suspenderPagoCheque</i> (ChequeComand cheque)	El Cheque pasa a estado Bloqueado, ya sea por pérdida, deterioro u otra razón.
<i>conformarCheque</i> (ChequeComand chequeComand)	Devuelve el objeto Cheque con los tipos de datos que corresponde a la tabla de la base de datos.
<i>validarContabilizacion</i> (ChequeComand chequeComand)	Valida los datos del Cheque para que se realice una correcta contabilización.
<i>persistirContabilizarCheque</i> (List<Asiento>listaAsientos, OChequeBnc chq)	En una misma ejecución garantiza que se contabilice el Cheque para luego ser registrado.
<i>crearTransaccionRegistrarCheque</i> (ChequeComand chq)	Conforma los asientos contables correspondientes al Cheque que se va a contabilizar.

**Tabla : Descripción de la clase GestionarChequeFacade**

<b>Nombre:</b> <i>RegistrarChequeMultiAction</i>	
<b>Tipo de clase:</b> Controladora	
<b>Atributo</b>	<b>Tipo</b>
<i>gestionarChequeFacade</i>	<i>cheque.facade.GestionarChequeFacade</i>
Para cada responsabilidad:	

Nombre:	Descripción:
registrarEntradaFlujo(RequestContext contex)	Se encarga de inicializar y actualizar las variables y los datos del ChequeComand cada vez que se inicie el flujo.
esOperacionContable(RequestContext contex)	Inicializa la variable contable en falso o verdadero en dependencia si el objeto ChequeComand se va a contabilizar o no.
validarCheque(RequestContext contex)	Verifica si el Cheque ya está contenido en una Carta de Remesa.
construirAsientos(RequestContext contex)	Crear los asientos que se corresponden con la contabilización del Cheque que valla a realizarse.
primeraValidacion(RequestContext contex)	Valida si los asientos están bien formados.
segundaValidacion(RequestContext contex)	Valida la disponibilidad de cuenta.
contabilizar(RequestContext contex)	Contabiliza y después persiste el Cheque en la base de datos.

**Tabla : Descripción de la clase RegistrarChequeMultiAction**

### 3.2.4 Utilización del Framework Spring WebFlow

El Framework Spring Web Flow al permitir llevar el control de las secuencias de pasos que se generan en la ejecución de una tarea del negocio, es utilizado en los módulos Cheque y Carta de Remesa para dar solución a las funcionalidades. A continuación se hará una descripción del flujo correspondiente a la funcionalidad Registrar Cheque del módulo Cheque; en la que interviene la clase *RegistrarChequeMultiAction* como controladora para dar respuestas a los eventos iniciados y el fichero XML *registrarCheque-flow.xml* responsable de definir el flujo:

Primeramente se crea la variable **chequeComand** que representa el objeto *ChequeComand* que se estará utilizando en el flujo; y después antes de entrar a la vista se ejecuta el método **registrarEntradaFlujo** (ver descripción del método en la Tabla : Descripción de la clase RegistrarChequeMultiAction).

```
<var name="chequeComand"
  class="cu.uci.finixubnc.titulosvalores.cheque.web.webflow.command.ChequeComand
" />

<on-start>
  <evaluate expression="registrarChequeMultiAction.registrarEntradaFlujo"/>
</on-start>
```

El flujo se inicia en la vista **registrarCheque** con mismo nombre del fichero con extensión .jsp que es la página que muestra. Tiene transiciones que serán ejecutadas en correspondencia de los eventos realizados desde el cliente y en su formulario va estar contenido el objeto *ChequeComand*.

```
<view-state id="registrarCheque" model="chequeComand" view="registrarCheque">
  <transition on="aceptar" validate="false" to="isCartaRemesa">
    <evaluate expression="registrarChequeMultiAction.esOperacionContable"/>
  </transition>
  <transition on="cancelar" validate="false" bind="false" to="end"/>
  <transition on="aceptarM" to="contabilizar"/>
  <transition on="cancelarM" to="registrarCheque"/>
  <transition on="verAsientos" to="verAsientosActionState"/>
  <transition on="seleccionarAsientoCartera" to="registrarCheque">
    <evaluate expression="registrarChequeMultiAction.seleccionarAsientoCartera"/>
  </transition>
  <transition on="eliminarAsientoCartera" to="registrarCheque">
    <evaluate expression="registrarChequeMultiAction.eliminarAsientoCartera"/>
  </transition>
  <transition on="addOrDelComision" to="registrarCheque">
    <evaluate expression="registrarChequeMultiAction.addOrDelComision"/>
  </transition>
  <transition on="vaciarListaAsientoChequeComand" to="registrarCheque">
    <evaluate
expression="registrarChequeMultiAction.vaciarListaAsientoChequeComand"/>
  </transition>
  <transition on="listarAsientoCarteraC5" to="registrarCheque">
    <evaluate expression="registrarChequeMultiAction.listarAsientoCarteraC5"/>
  </transition>
</view-state>
```

En caso de efectuarse el evento **aceptar**, se ejecuta el método **esOperacionContable**(ver descripción del método en la Tabla : Descripción de la clase RegistrarChequeMultiAction) donde se inicializa la variable **flowScope.contable**; y luego se va al estado de decisión **isCartaRemesa** en la cual si el valor



de la variable **flowScope.isCartaRemesa** es falso se va hacia el estado de decisión **esContable** y si es verdadero se va hacia el estado de acción **validarCheque**, esto ocurre si este flujo es accedido como un subflujo desde el flujo correspondiente a la funcionalidad Registrar Carta de Remesa.

```
<decision-stateid="isCartaRemesa">
  <iftest="flowScope.isCartaRemesa"then="validarCheque"else="esContable"/>
</decision-state>

<action-state id="validarCheque">
  <evaluate expression="registrarChequeMultiAction.validarCheque"/>
  <transition on="success" to="registrarChequeInCartaRemesa"/>
  <transition on="error" to="registrarCheque"/>
</action-state>

<action-stateid="registrarChequeInCartaRemesa">
  <evaluateexpression="registrarChequeMultiAction.registrarChequeInCartaRemesa"/>
  >
  <transition on="success" to="end"/>
  <transition on="error" to="registrarCheque"/>
</action-state>

<decision-stateid="esContable">
  <iftest="flowScope.contable"then="construirAsientos"else="actualizarCheque"/>
</decision-state>
```

En caso de que en el estado de decisión **isCartaRemesa** la variable tenga valor falso se va hacia el estado de decisión **esContable**, en la cual la variable **flowScope.contable** para este caso su valor será verdadero para iniciar el estado de acción **construirAsientos**; en este estado se ejecuta el método **construirAsientos**(ver descripción del método en la Tabla : Descripción de la clase RegistrarChequeMultiAction) y si es satisfactoria su ejecución se va hacia el estado de acción **primeraValidacion**, sino se va hacia la vista **registrarCheque**.

```
<action-stateid="construirAsientos">
  <evaluate expression="registrarChequeMultiAction.construirAsientos"/>
  <transition on="success" to="primeraValidacion"/>
  <transition on="error" to="registrarCheque"/>
</action-state>

<action-state id="actualizarCheque">
  <evaluate expression="registrarChequeMultiAction.persistirCheque"/>
  <transition on="success" to="registrarCheque"/>
</action-state>
```

La ejecución del método **construirAsientos** fue satisfactoria por lo que nos encontramos en el estado de acción **primeraValidacion**, en este estado se ejecuta el método **primeraValidacion** que si es

satisfactoria su ejecución se va hacia el estado de acción **segundavalidacion**, de lo contrario nos envía nuevamente al **<view-state>** con id=**registrarCheque** para informar del error ocurrido. La ejecución del método contenido en el estado de acción **segundavalidacion**, en caso de que sea satisfactoria va hacia el estado de acción **contabilizar** y si hubo error entonces hacia el estado **mostrarAdvertenciasContabilizar**. Ver descripción de los métodos mencionados en la Tabla : Descripción de la clase RegistrarChequeMultiAction

```
<action-state id="primeraValidacion">
  <evaluate expression="registrarChequeMultiAction.primeraValidacion"/>
  <transition on="success" to="segundavalidacion"/>
  <transition on="error" to="registrarCheque"/>
</action-state>

<action-state id="segundavalidacion">
  <evaluate expression="registrarChequeMultiAction.segundaValidacion"/>
  <transition on="success" to="contabilizar"/>
  <transition on="yes" to="mostrarAdvertenciasContabilizar"/>
</action-state>
```

La vista **mostrarAdvertenciasContabilizar** se muestra al usuario a través de un diálogo, el cual posee los eventos **contabilizar** para realizar la contabilización y el evento **terminar** para cerrar el diálogo para realizar algún cambio en el formulario o cancelar la operación. En el estado de acción **contabilizar** se ejecuta el método **contabilizar**, si es satisfactoria o no va hacia la vista **registrarCheque** para notificar al usuario; en caso de ser satisfactoria se le muestra al usuario un diálogo con mensaje "Operación satisfactoria".

```

<view-state id="mostrarAdvertenciasContabilizar"
view="mostrarAdvertenciasContabilizar">
  <transition on="contabilizar" to="contabilizar"/>
  <transition on="terminar" to="registrarCheque">
    <evaluate expression="registrarChequeMultiAction.anularMensajeError"/>
  </transition>
</view-state>

<action-state id="contabilizar">
  <evaluate expression="registrarChequeMultiAction.contabilizar"/>
  <transition on="success" to="registrarCheque"/>
  <transition on="error" to="registrarCheque"/>
</action-state>

<decision-state id="end">
  <if test="flowScope.isCartaRemesa"then="enReturnCartaRemesa"else="endHome"/>
</decision-state>

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

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

```

### 3.3 Pruebas

El desarrollo del software debe ir acompañado de una actividad que garantice la calidad; para la cual se crean pruebas minuciosas y bien planificadas, utilizando técnicas de diseño de casos de pruebas, de caja blanca y caja negra, para descubrir y corregir el máximo de errores posibles, antes de la entrega del producto al cliente. Estas técnicas facilitan una guía sistemática para diseñar pruebas que: comprueben la lógica interna de los componentes de software y verifiquen los dominios de entrada y salida del programa para descubrir errores en la funcionalidad, el comportamiento y rendimiento.

#### 3.3.1 Aplicación de la Prueba de Caja Blanca.

La prueba de caja blanca, denominada a veces prueba de caja de cristal es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba; es decir, se centra en la estructura de control. Mediante los métodos de prueba de caja blanca, el ingeniero del software puede obtener casos de prueba que: (37)

- Garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo.

- Ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa.
- Ejecuten todos los bucles en sus límites y con sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez.

Este método posee técnicas para la prueba de estructura de control, que mejoran la calidad de la prueba de caja blanca, estas variantes de pruebas de este método son: Camino Básico, Condición, Flujo de Datos y Bucles. De las técnicas mencionadas haremos uso de la Prueba del Camino Básico, que hace uso de grafos para obtener el conjunto de pruebas linealmente independientes que aseguren la total cobertura.

#### 3.3.1.1 Prueba del Camino Básico.

El método del camino básico permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa. (38)

Aplicaremos esta prueba al método *validarCodigoBarra* de la Clase *CargarDatosMultiActionController*, primero enumeramos las sentencias del código:

```

public Boolean validarCodigoBarra (String codigoBarra) {

    if (codigoBarra == null || codigoBarra.equals ("") //1
        || (codigoBarra.length () != 25 && codigoBarra.length () != 24)) //1
        return false; //2

    if (!validacionTipoDatosCodigoBarraEstandarizado (codigoBarra)) //3
        return false; //2

    if (codigoBarra.length () == 25) { //4
        if (!validarDigitoChequeo (codigoBarra)) //5
            return false; //2
    }

    NTipoCheque tipoCheque = obtenerTipoChequeCodigoBarra (codigoBarra); //6
    if (tipoCheque == null) //7
        return false; //2

    // Validar con los bancos nacionales si existe
    BanNac banco = obtenerBancoPorCodigoBarra (codigoBarra); //8
    if (banco == null) //9
        return false; //2

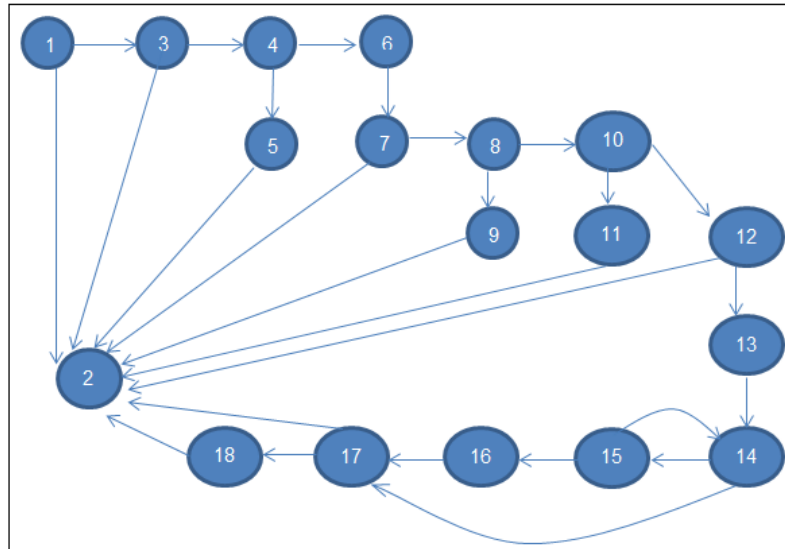
    NomMoneda moneda = obtenerMonedaCodigoBarra (codigoBarra); //10
    if (moneda == null) //11
        return false; //2
    ) // Cheques // de BNC
    if (obtenerCodigoBancoBase ().equals (banco.getCod_banco ())) //12
    {
        // Se valida el centro de costo
        String surcursal = obtenerSurcursalCodigoBarra (codigoBarra); //13
        List<CentroCosto> listaCentrosCostos = gestionarChequeFacade //13
            .getGlobalFacade ().listarCentroCosto (); //13
        Boolean encontrado = false; //13
        for (CentroCosto centroCosto : listaCentrosCostos) { //14
            if (centroCosto.getCodigo ().equals (surcursal)) { //15
                encontrado = true; //16
                break; //16
            }
        }
        if (! encontrado) //17
            return false; //2

        if (obtenerCuentaCodigoBarra (codigoBarra) == null) //18
            return false; //2
    }

    return true; //2
}

```

Haciendo uso del código del método *validarCodigoBarra*, generamos el grafo de flujo. Los números asignados representan los nodos del grafo:



Determinamos la complejidad ciclomática del grafo de flujo resultante, para definir el número máximo de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez. La complejidad ciclomática es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa.

La complejidad ciclomática tiene tres vías para ser calculada; en esta prueba se hará uso de todas las formas de cálculo para tener un resultado contundente:

1. El número de regiones del grafo de flujo coincide con la complejidad ciclomática; incluimos el área exterior del grafo, contando como otra región más. Entonces  $V(G)=R$ ; donde  $R$  es el número de regiones.  
 $R=11$   
 $V(G)=11$
2. La complejidad ciclomática,  $V(G)$ , de un grafo de flujo  $G$  se define como  $V(G)=A-N+2$ ; donde  $A$  es el número de aristas del grafo de flujo y  $N$  es el número de nodos del mismo.  
 $A=27$   
 $N=18$   
 $V(G)=26-18+2$   
 $V(G)=11$
3. La complejidad ciclomática,  $V(G)$ , de un grafo de flujo  $G$  también se define como  $V(G)=P+1$ ; donde  $P$  es el número de nodos predicado contenidos en el grafo de flujo  $G$ .  
 $P=10$   
 $V(G)=10+1$   
 $V(G)=11$

Las tres vías de cálculo dieron como resultado una complejidad ciclomática de 11, lo que indica que hay 11 caminos posibles por donde el flujo puede circular. A continuación presentamos los caminos básicos por lo que puede recorrer el flujo:

Camino básico #1: 1-2

Camino básico #2: 1-3-2

Camino básico #3: 1-3-4-5-2

Camino básico #4: 1-3-4-6-7-2

Camino básico #5: 1-3-4-6-7-8-9-2

Camino básico #6: 1-3-4-6-7-8-10-11-2

Camino básico #7: 1-3-4-6-7-8-10-12-2

Camino básico #8: 1-3-4-6-7-8-10-12-13-14-17-2

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

Camino básico #10: 1-3-4-6-7-8-10-12-13-14-15-16-17-18-2

Camino básico #11: 1-3-4-6-7-8-10-12-13-14-15-14-(...)

El diseño y uso de los casos de prueba, se realiza a través del framework JUnit: Este framework es una herramienta de Prueba unitaria para software desarrollados en Java. Para su uso se definen ficheros `dataaccess-context.xml` y `titulosValores-context.xml` para la conexión con la base de datos y cargar las clases necesarias para la realización de la prueba respectivamente; se define además, la clase `Test_JUnit.java` en la cual se crean métodos asociados a cada caso de prueba definido. Los métodos tienen que tener como prefijo `test`, el nombre del método al cual se le realiza la prueba seguido del sufijo `TEST_#` número de prueba. La clase `Test_JUnit.java` hereda de `TestCase`, clase perteneciente al framework, es necesaria esta herencia para el uso de las funcionalidades de prueba que ofrece JUnit.

Se muestra la estructura de un método correspondiente a un caso de prueba y una ilustración de cuando es ejecutado el framework:

```

public void testValidarCodigoBarraTEST_7 () {
    controlHttpServlet.replay();

    String codigoBarra="1010001000268002110000782";

    assertTrue(cargarDatosMultiActionControllerCheque.validarCodigoBarra(codigoBarra));
}

```

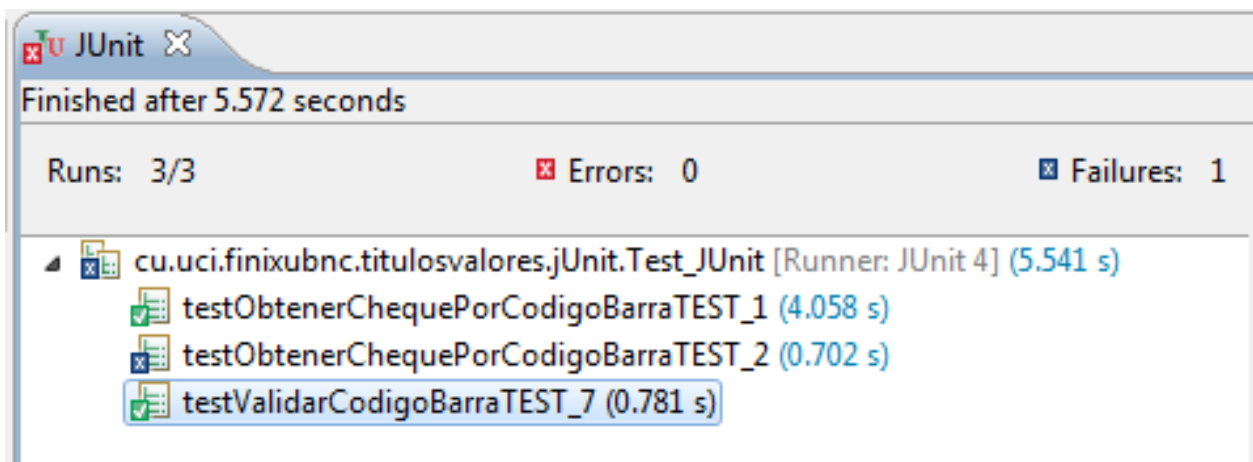


Figura : Ejecución del framework JUnit.

### 3.3.2 Aplicación de la Prueba de Caja Negra o Funcional.

Las pruebas de caja negra, también denominada prueba de comportamiento, se centran en los requisitos funcionales del software. Permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. Centra su atención en el campo de la información. (38)

La prueba de caja negra intenta encontrar errores de las siguientes categorías:

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



Para llevar a cabo esta prueba, se puede hacer uso de técnicas, como: los basados en Grafos, Partición Equivalente, Análisis de Valores Límite y Tabla Ortogonal. Para este método de prueba se hará uso de la técnica Partición Equivalente, que divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software. En el Anexo : Caso de prueba caja negra referente al requisito funcional Registrar Cheque se puede apreciar la aplicación de la técnica Partición Equivalente.

### 3.3.3 Resultado de las pruebas.

Durante la aplicación de estos métodos como pruebas internas, realizadas por el equipo de desarrollo de los módulos Cheque y Carta de Remesa, se obtuvieron excelentes resultados tanto funcionales como estructurales ante diferentes casos de pruebas.

El producto obtenido a partir de la implementación del diseño de los módulos Cheque y Carta de Remesa, fue liberado por el departamento de calidad de software perteneciente a la universidad (CALISOFT) y entraron a las pruebas de aceptación en el BNC. En el Anexo : Acta de Liberación del sistema de gestión bancaria Quarxo versión 1. Parte 1, se puede apreciar la cantidad de iteraciones y tipos de pruebas realizadas en los módulos Cheque y Carta de Remesa.

El producto obtenido permite tener en un mismo sistema servicios diferentes relacionados con la gestión de Cheques y Cartas de Remesa:

- Envío y recepción de Cartas de Remesa para los bancos que no estén integrado al SLBTR.
- Envío y recepción de pagos y cobros de Cheques mediante la integración con el sistema de mensajería SLBTR.
- Recepción de cobros y pagos de Cheques emitidos por el Banco Nacional de Cuba y por otros bancos.
- Impresión de Cartas de Remesa enviadas a otros bancos a partir de la creación de una plantilla con los datos de la carta.
- Permite la recepción de la información de Cheques mediante la utilización del dispositivo de lector de códigos de barra.

#### 3.4 Conclusiones parciales.

Este capítulo facilita la integración y utilización de los módulos correspondientes, a la gestión de los procesos relacionados con Cheques y Cartas de Remesa llevados a cabo en el Banco Nacional de Cuba, ya que define estándares de codificación para los nombres de métodos, clases, atributos, variables. Nos brinda una visión clara de las dependencias entre los componentes contenidos en los módulos. Garantiza que se hayan identificado el mayor número posible de errores, a través de pruebas internas realizadas, antes de su entrega a CALISOFT y posteriormente al BNC.

## CONCLUSIONES

Como resultado del desarrollo del trabajo presentado se concluye lo siguiente:

- Se obtuvo un conocimiento profundo acerca de la gestión de los procesos de Cheques y Cartas de Remesa en el Banco Nacional de Cuba.
- Se evidenció la necesidad de implementar los módulos Cheque y Carta de Remesa para el sistema de gestión bancaria Quarxo, ya que las funcionalidades que brindaban los sistemas informáticos analizados no se adaptaban a las necesidades del negocio del Banco Nacional de Cuba.
- Los artefactos generados en el diseño de los módulos, posibilitó la implementación de los componentes necesarios para cumplir con los requisitos funcionales de forma rápida y segura; ya sea por la integración con el sistema de mensajería SLBTR o el uso del escáner de códigos de barras manual para la lectura del código de barra de los Cheques.
- Se validó la implementación del diseño de los módulos Cheque y Carta de Remesa, obteniendo un producto con la calidad requerida para integrarse al sistema Quarxo, asesorado por las pruebas de calidad tanto internas como las realizadas por el grupo de CALISOFT.
- La utilización de estos módulos contribuye a mejorar la gestión de los procesos de Cheques y Cartas de Remesa en el Banco Nacional de Cuba, disminuyendo el tiempo de contabilización de los Cheques mediante la integración con el sistema de mensajería SLBTR, y el envío y recepción de Cartas de Remesa para los bancos que no estén integrados al SLBTR.

## RECOMENDACIONES

Haciendo uso de la investigación realizada se recomienda dar continuidad a este trabajo con el objetivo de agregar nuevas funcionalidades a la solución o desarrollar nuevas versiones haciendo uso de nuevas tecnologías y herramientas.

## BIBLIOGRAFÍA

1. **Blanco, Lázaro J. Encinosa.** infoMED. *RED DE SALUD EN CUBA*. [En línea] 2003. [Citado el: 5 de 11 de 2011.] [http://www.sld.cu/galerias/doc/sitios/infodir/apuntes\\_para\\_una\\_historia\\_de\\_la\\_informatica\\_en\\_cuba.doc](http://www.sld.cu/galerias/doc/sitios/infodir/apuntes_para_una_historia_de_la_informatica_en_cuba.doc).
2. **González, Juan José Bustamante.** El cheque, su aspecto mercantil y bancario y su tutela. [aut. libro] Juan José Bustamante González. 1.
3. **Perdomo, Yesenia Bello y Pérez, Leosvel Espinosa.** *Análisis y Diseño del Subsistema Títulos valores del Proyecto Modernización del Sistema Bancario Cubano*. La Habana : Universidad de las Ciencias Informáticas, 2009.
4. **Cerezal, Lourdes Tamargo.** La Revista del empresario cubano. *BETSIME*. [En línea] 2 de 2002. [Citado el: 5 de 12 de 2011.] [http://www.betsime.disaic.cu/secciones/tec\\_feb\\_02.htm](http://www.betsime.disaic.cu/secciones/tec_feb_02.htm).
5. **De Larrobla & Asociados (DL&A).** Bantotal. [En línea] [Citado el: 7 de 12 de 2012.] <http://www.bantotal.com/bantotal/acerca.asp>.
6. —. Bantotal. [En línea] [Citado el: 7 de 12 de 2011.] <http://www.bantotal.com/bantotal/arquitectura.asp>.
7. —. Bantotal. [En línea] [Citado el: 7 de 12 de 2011.] <http://www.bantotal.com/bantotal/herramientas.asp>.
8. **BYTE.** BYTE. [En línea] [Citado el: 8 de 12 de 2011.] [http://www.bytesw.com/new/sistema\\_bancario\\_financierobyte.asp](http://www.bytesw.com/new/sistema_bancario_financierobyte.asp).
9. **SAP.** SAP | Argentina. [En línea] [Citado el: 8 de 12 de 2011.] <http://www.sap.com/argentina/platform/index.epx>.
10. —. SAP | Argentina. [En línea] [Citado el: 8 de 12 de 2011.] <http://www.sap.com/argentina/industries/banking/index.epx>.
11. —. SAP | Argentina. [En línea] [Citado el: 8 de 12 de 2011.] <http://www.sap.com/argentina/solutions/netweaver/index.epx>.
12. **EUI - FI . Universidad Politécnica de Valencia (UPV).** <http://users.dsic.upv.es>. [En línea] [Citado el: 14 de 12 de 2011.] [http://users.dsic.upv.es/asignaturas/eui/mtp/doc-practicas/intro\\_case\\_SA.p](http://users.dsic.upv.es/asignaturas/eui/mtp/doc-practicas/intro_case_SA.p).
13. **Jacobson, I., Booch, G. y Rumbaugh, J.** *El proceso unificado de desarrollo de software*. [ed.] Addison Wesley. 2000. Vol. 7.
14. **DÍAZ, MIRIAN MILAGROS FLORES.** <http://www.usmp.edu.pe>. [En línea] [Citado el: 14 de 12 de 2011.] <http://www.usmp.edu.pe/publicaciones/boletin/fia/info49/articulos/RUP%20vs.%20XP.pdf>.
15. **Universidad Politécnica de Valencia (UPV).** <http://www.disca.upv.es/enheror/pdf/ActaUML.PDF>. [En línea] [Citado el: 14 de 12 de 2011.]
16. *The CASE experience.* **McClure, Carma** . 14, 1989, Byte, págs. 235-244.
17. *Terminology for Software Engineering Environment (SEE) and Computer-Aided Software Engineering (CASE).* **Terry, B. y Logee, D.** 1990, ACM SIGSOFT Software Engineering Notes, págs. 83-94.
18. **Visual Paradigm Company.** Visual Paradigm. [En línea] [Citado el: 14 de 12 de 2011.] <http://www.visual-paradigm.com/>.

19. **Sun Microsystems.** eclipse. [En línea] [Citado el: 13 de 12 de 2011.] [http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/int\\_eclipse.htm](http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/int_eclipse.htm).
20. **the apache software foudation.** apache tomcat. [En línea] [Citado el: 13 de 12 de 2011.] <http://tomcat.apache.org/>.
21. Guia de Soluciones TIC. *Punto de Encuentro para Proveedores y Compradores Tecnológicos*. [En línea] Dat@Market Solutions, 2011. [Citado el: 16 de 5 de 2012.] <http://www.guiadesolucionestic.com/software-del-sistema/herramientas-de-desarrollo/herramientas-de-desarrollo-de-software/972-embarcadero-erstudio>.
22. **Microsoft Corporation.** Microsoft. [En línea] 2011. [Citado el: 14 de 12 de 2011.] <http://www.microsoft.com/spain/sql/productinfo/overview/default.aspx>.
23. **CAMACHO , ERIKA , CARDESO, FABIO y NUÑEZ , GABRIEL .** *ARQUITECTURAS DE SOFTWARE, Guía de Estudio*. 2004.
24. **Larman, Craig.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos*.
25. **Wesley, Addison.** *Design Patterns Explained*.
26. **Segura , Aurelio Maroto .** Universitat Politècnica de València. [En línea] 2010 de 9 de 13. [Citado el: 2012 de 1 de 18.] <http://hdl.handle.net/10251/9132>.
27. **Recuenco , Javier Calvo y Rodríguez , Pablo Fernández.** <http://zarza.fis.usal.es/>. [En línea] 5 de 2010. [Citado el: 2012 de 1 de 18.] <http://zarza.fis.usal.es/~fgarcia/docencia/poo/04-05/Trabajos/Composite.pdf>.
28. **Buschmann , Frank , y otros, y otros.** *Pattern-Oriented Software Architecture (POSA). A System of Patterns*. 1996. Vol. 1.
29. **Ciberaula.** Ciberaula. [En línea] 26 de 10 de 2011. [Citado el: 13 de 12 de 2011.] [http://java.ciberaula.com/articulo/que\\_es\\_java](http://java.ciberaula.com/articulo/que_es_java).
30. **Hurtado, Julio Ariel Alegría y Castillo , Lina María Paredes.** <http://scienti.colciencias.gov.co>. [En línea] [Citado el: 12 de 12 de 2011.] <http://scienti.colciencias.gov.co:8084/publindex/docs/articulos/1692-374X/2/10.pdf>.
31. **Milián, V., y otros, y otros.** Universidad de San Buenaventura seccional cali. [En línea] 2010. [Citado el: 16 de 5 de 2012.] <http://usbvirtual.usbcali.edu.co/ijpm/images/stories/documentos/v1n2/003.pdf>.
32. **Reina , A. M. Quinter, Torres , J. Valderrama y Toro , M. Bonilla.** Sistedes. *Sociedad de Ingeniería del Software y Tecnologías de Desarrollo de Software*. [En línea] [Citado el: 13 de 12 de 2011.] <http://www.sistedes.es/TJISBD/Vol-1/No-6/articles/dsdm-07-reina-spring.pdf>.
33. **springsource community.** springsource. [En línea] [Citado el: 13 de 12 de 2011.] <http://www.springsource.org/documentation>.
34. —. springsource. [En línea] [Citado el: 13 de 12 de 2011.] <http://www.springsource.org/spring-web-flow#documentation>.
35. **BAUER, CHRISTIAN y KING, GAVIN.** *Hibernate in Action*.
36. **Rodríguez Alcalde, Á.L., Cala, C. y Barroso, J.Á.** DIGITAL.CSIC. *OPEN SCIENCE*. [En línea] 2007. [Citado el: 16 de 5 de 2012.] [http://digital.csic.es/bitstream/10261/5110/1/Comunicacion\\_TCO-280-2007TY.pdf](http://digital.csic.es/bitstream/10261/5110/1/Comunicacion_TCO-280-2007TY.pdf).
37. **Pressman, Roger S.** *Ingeniería del Software. Un enfoque práctico*. España : McGraw-Hil, 2002.

38. —. *Ingeniería del Software. Un enfoque práctico*. . España : McGraw-Hil, 2002.
39. **Caballero, Margarita Pulido y Simón, Elizabeth Melgarejo** . <http://cofinhabana.fcf.uh.cu>. [En línea] [Citado el: 5 de 12 de 2011.] [http://docs.google.com/viewer?a=v&q=cache:l7iQuVpuVvkJ:cofinhabana.fcf.uh.cu/index.php%3Foption%3Dcom\\_docman%26task%3Ddoc\\_download%26gid%3D144%26Itemid%3D+sabic%2Bcartas+de+remesas&hl=es&gl=cu&pid=bl&srcid=ADGEESgqx4Eph2QOzMMJ-a2ASKDUM--qLW9jbCcMbEwJewSv](http://docs.google.com/viewer?a=v&q=cache:l7iQuVpuVvkJ:cofinhabana.fcf.uh.cu/index.php%3Foption%3Dcom_docman%26task%3Ddoc_download%26gid%3D144%26Itemid%3D+sabic%2Bcartas+de+remesas&hl=es&gl=cu&pid=bl&srcid=ADGEESgqx4Eph2QOzMMJ-a2ASKDUM--qLW9jbCcMbEwJewSv).
40. **Cerezal , Lourdes Tamargo y Torres, Jorge Sanabria**. <http://www.bc.gov.cu>. [En línea] 2006. [Citado el: 5 de 12 de 2011.] [http://www.bc.gov.cu/antiores/RevistaBCC/2006/No1-2006/Documentos/Enriqueciendo\\_el\\_sistema\\_contable\\_en\\_el\\_banco\\_Central\\_de\\_Cuba-RevBCC-No1-2006.pdf](http://www.bc.gov.cu/antiores/RevistaBCC/2006/No1-2006/Documentos/Enriqueciendo_el_sistema_contable_en_el_banco_Central_de_Cuba-RevBCC-No1-2006.pdf).
41. **Llinares, Xabier Salas**. Auditoría Pública. [En línea] 9 de 2001. [Citado el: 6 de 11 de 2011.] [http://www.auditoriapublica.com/hemeroteca/200109\\_24\\_79.pdf](http://www.auditoriapublica.com/hemeroteca/200109_24_79.pdf).
42. **Banco Central de Cuba (BNC)**. Sitio del Gobierno de la República de Cuba. [En línea] [Citado el: 5 de 12 de 2011.] [http://www.cubagob.cu/des\\_eco/banco/espanol/automatizacion/automatizacion.htm](http://www.cubagob.cu/des_eco/banco/espanol/automatizacion/automatizacion.htm).
43. **Perdomo, Yesenia Bello**. Semana Tecnológica. [En línea] 9 de 2010. [Citado el: 5 de 12 de 2011.] <http://semanatecnologica.fordes.co.cu/ocs-2.3.2/public/site/313.pdf>.
44. **Machado, Sandy Scull y Bernal, Nesto Vidal**. Semana Tecnológica. [En línea] 2010. [Citado el: 6 de 12 de 2011.] <http://semanatecnologica.fordes.co.cu/ocs-2.3.2/public/site/175.pdf>.
45. **Gerencie.com**. Gerencie.com. [En línea] 12 de 6 de 2010. [Citado el: 7 de 12 de 2011.] <http://www.gerencie.com/cheque.html>.
46. **managinf**. Managinf. [En línea] 2010. [Citado el: 7 de 12 de 2011.] [www.managinf.com/sihrebizframe.pdf](http://www.managinf.com/sihrebizframe.pdf).
47. **Genexus**. GeneXus. [En línea] 2011. [Citado el: 8 de 12 de 2011.] <http://www.genexus.com/productos/genexus/genexus-home?es>.
48. **Soberón, Francisco Valdés, Ministro Presidente del Banco Central de Cuba**. BANDEC. [En línea] [Citado el: 14 de 12 de 2011.] <http://www.santiago.cu/hosting/bandec/Cobros%20y%20Pagos.htm>.
49. **Rumbaugh, James, Jacobson, Ivar and Booch, Grandy**. *El lenguaje unificado de modelado. Manual de referencia*. s.l. : Addison Wesley.
50. *Ingeniería de Software I Conferencia # 7 Flujo de Trabajo Prueba*. 2011.
51. **Massol, Vincent y Husted, Ted**. *JUnit in Action*.

## ANEXOS



## Acta de Liberación de Productos Software

Acta de Liberación de Productos Software

Fecha de liberación: 20 de diciembre de 2010.

Emitida a favor de: SAGEB.

### 1. Datos del producto

Artefacto	Versión	Estado final	Cantidad Iteraciones	Tipos de pruebas realizadas
Gestionar Acuerdo	V1.0		3	Funcionales, Carga y Estrés
Documentos de Embarque	V1.0		3	Funcionales, Carga y Estrés
Clientes Juridicos	V1.0		3	Funcionales, Carga y Estrés
Mensajería SLBTR	V1.0		3	Funcionales, Carga y Estrés
Préstamos	V1.0		3	Funcionales, Carga y Estrés
Gestionar Transferencias	V1.0		3	Funcionales, Carga y Estrés
Vencimientos	V1.0		3	Funcionales, Carga y Estrés
Ejercicio Contable.	V1.0		3	Funcionales, Carga y Estrés
Carta de Remesa	V1.0		3	Funcionales, Carga y Estrés
Emisión de CC	V1.0		3	Funcionales, Carga y Estrés
Negociación de CC	V1.0		3	Funcionales, Carga y Estrés
Seguridad	V1.0		3	Funcionales, Carga y Estrés
Gestionar Chequeras	V1.0		3	Funcionales, Carga y Estrés
Cheque	V1.0		3	Funcionales, Carga y Estrés

Anexo : Acta de Liberación del sistema de gestión bancaria Quarxo versión 1. Parte 1






## Acta de Liberación de Productos Software

### Acta de Liberación de Productos Software


Fecha de liberación: 20 de diciembre de 2010

Depósitos	V1.0		3	Funcionales, Carga y Estrés
Discrepancias de DE	V1.0		3	Funcionales, Carga y Estrés
Personas Autorizadas	V1.0		3	Funcionales, Carga y Estrés
Plan de Cuentas	V1.0		3	Funcionales, Carga y Estrés
Gestionar Banco	V1.0		3	Funcionales, Carga y Estrés
Medios de Comunicación	V1.0		3	Funcionales, Carga y Estrés
Permisos Sobre Cuentas	V1.0		3	Funcionales, Carga y Estrés
Libros Contables	V1.0		3	Funcionales, Carga y Estrés
Gestionar Cta. de Clientes	V1.0		3	Funcionales, Carga y Estrés
Tasa de Cambio	V1.0		3	Funcionales, Carga y Estrés
Cuentas de Banco	V1.0		3	Funcionales, Carga y Estrés
Gestionar Cta. Concepto	V1.0		3	Funcionales, Carga y Estrés
Transacciones Generales	V1.0		3	Funcionales, Carga y Estrés
Sobregiro Autorizado	V1.0		3	Funcionales, Carga y Estrés
Reservación de Fondos	V1.0		3	Funcionales, Carga y Estrés
Capacidad Financiera	V1.0		3	Funcionales, Carga y Estrés

  
Delvis Echeverría Perez

Nombre y Apellidos

Responsable Calisoft

  
Lissett Díaz Mesa

Nombre y Apellidos

Responsable Proyecto

**Anexo : Acta de Liberación del sistema de gestión bancaria Quarxo versión 1. Parte 2**

## Anexo : Caso de prueba caja negra referente al requisito funcional Registrar Cheque

### 1. Descripción general

El caso de uso se inicia cuando un usuario autenticado procede registrar un Cheque.

### 2. Condiciones de ejecución

- El usuario autenticado tiene que tener los permisos pertinentes a la funcionalidad a ejecutar.

### 3. Secciones

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC1 Registrar Cheque	EC1.1 Registrar Cheque parcialmente	Se introducen los datos necesarios en el formulario y se oprime el botón aceptar. El sistema valida los datos, llevar a cabo todas las operaciones para dar cumplimiento a la funcionalidad y muestra mensaje "Operación realizada satisfactoriamente".	<ol style="list-style-type: none"> <li>1. El usuario selecciona el subsistema Títulos Valores.</li> <li>2. El usuario selecciona el módulo Cheque.</li> <li>3. El usuario selecciona la funcionalidad Registrar Cheque. Esta funcionalidad puede ser accedida desde la interfaz buscar Cheque.</li> <li>4. El sistema muestra la interfaz con el formulario para efectuar el registro.</li> <li>5. El usuario llena el formulario con los datos necesarios. A la misma vez que se van entrando los datos el formulario valida el valor contenido en cada campo.</li> </ol>

			<ol style="list-style-type: none"> <li>6. El usuario oprime el botón Aceptar para completar el registro.</li> <li>7. El sistema valida los datos de entrada y muestra el mensaje “Operación realizada satisfactoriamente”.</li> <li>8. El usuario oprime el botón Aceptar.</li> <li>9. El sistema muestra la página del subsistema.</li> </ol>
	EC1.2 Registrar Cheque efectuando contabilización	Se llenan todos los campos del formulario y se oprime el botón aceptar. El sistema válida los datos, lleva a cabo todas las operaciones para dar cumplimiento a la funcionalidad y muestra mensaje “Operación realizada satisfactoriamente”	<ol style="list-style-type: none"> <li>1. El usuario selecciona el subsistema Títulos Valores.</li> <li>2. El usuario selecciona el módulo Cheque.</li> <li>3. El usuario selecciona la funcionalidad Registrar Cheque. Esta funcionalidad puede ser accedida desde la interfaz buscar Cheque.</li> <li>4. El sistema muestra la interfaz con el formulario para efectuar el registro.</li> <li>5. El usuario llena todos los campos del formulario y selecciona la opción “contabilizar”. A la misma vez que se van entrando los datos el formulario valida el valor contenido en cada campo.</li> <li>6. La interfaz muestra los campos ocultos necesarios para llevar a cabo la contabilización.</li> <li>7. El usuario llena los campos referentes a la contabilización.</li> </ol>

			<p>A la misma vez que se van entrando los datos el formulario valida el valor contenido en cada campo.</p> <p>8. El usuario oprime el botón Aceptar para completar el registro.</p> <p>9. El sistema valida los datos de entrada y muestra el mensaje “Operación realizada satisfactoriamente”.</p> <p>10. El usuario oprime el botón Aceptar.</p> <p>11. El sistema muestra la página del subsistema.</p>
	EC1.3 Cancelar Registrar Cheque	Se llena o no el formulario con los datos y si se oprime el botón cancelar, el sistema cancela la operación	<p>1. El usuario selecciona el subsistema Títulos Valores.</p> <p>2. El usuario selecciona el módulo Cheque.</p> <p>3. El usuario selecciona la funcionalidad Registrar Cheque. Esta funcionalidad puede ser accedida desde la interfaz buscar Cheque.</p> <p>4. El sistema muestra la interfaz con el formulario para efectuar el registro.</p> <p>5. El usuario puede llenar el formulario o no.</p> <p>6. El usuario oprime el botón Cancelar.</p> <p>7. El sistema muestra un mensaje de confirmación “¿Está seguro que desea cancelar esta operación?”.</p> <p>8. El usuario oprime el botón “Sí”.</p>

			9. El sistema muestra la página del subsistema.
--	--	--	---

#### 4. Descripción de variables

Nº	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Referencia corriente	TextBox	No	Se carga automáticamente.
2	Referencia original	TextBox/ ComboBox	No	Se carga automáticamente si no se contabiliza, de lo contrario se selecciona el dato desde la lista desplegable.
3	Fecha contable	CustomDateTextBox	No	Se carga automáticamente.
4	Fecha valor	CustomDateTextBox	No	Se carga automáticamente.
5	Código de barra	TextBox	No	Se entra el dato manualmente y es de 24 a 25 caracteres.
6	Número de cheque	TextBox	No	Se carga a partir de la variable 5.
7	Tipo de cheque	TextBox	No	Se carga a partir de la variable 5
8	Estado de cheque	ComboBox	No	Se carga a partir de la variable 5

9	Moneda	TextBox	No	Se carga a partir de la variable 5	
10	Importe	TextBox	No	Se entra el dato manualmente.	
11	Fecha de emisión	CustomDateTextBox	No	Se escoge la fecha de calendario que ofrece el campo.	
12	Certificado	CheckBox	No	Se selecciona o no.	
13	Banco emisor	TextBox	No	Se carga a partir de la variable 5	
14	Sucursal	TextBox	No	Se carga a partir de la variable 5	
15	Nombre emisor	TextBox	No	Se entra el dato manualmente.	
16	Cuenta emisor	TextBox	No	Se carga a partir de la variable 5	
17	Contabilizar	CheckBox	No	Se selecciona o no.	
18	Tipo de operación	ComboBox	No	Se selecciona el dato desde la lista desplegable.	
19	Cuenta	Moneda	ComboBox	No	Se selecciona el dato desde la lista desplegable.
20		Subcuenta	ComboBox	No	Carga los datos a partir de la variable 18 y se selecciona el dato desde la lista desplegable.

21	Tipo de contraparte	TextBox	No	Carga los datos a partir de la variable 20.
22	Contraparte	ComboBox	No	Carga los datos a partir de las variables 19, 20, 21 y se selecciona el dato desde la lista desplegable.
23	Desglose	ComboBox	No	Carga los datos a partir de las variables 19, 20, 21, 22 y se selecciona el dato desde la lista desplegable.
24	Banco beneficiario	ComboBox	No	Se carga y selecciona automáticamente el dato. Está deshabilitado.
25	Observaciones	TextArea	No	Se entra el dato manualmente.

## 5. Matriz de datos

### 5.1. SC1 Registrar Cheque

**N/A:** No accedido.

Id. del escenario	EC1.1	EC1.2	EC1.3
Escenario	Registrar Cheque parcialmente	Registrar Cheque efectuando contabilización	Cancelar Registrar Cheque

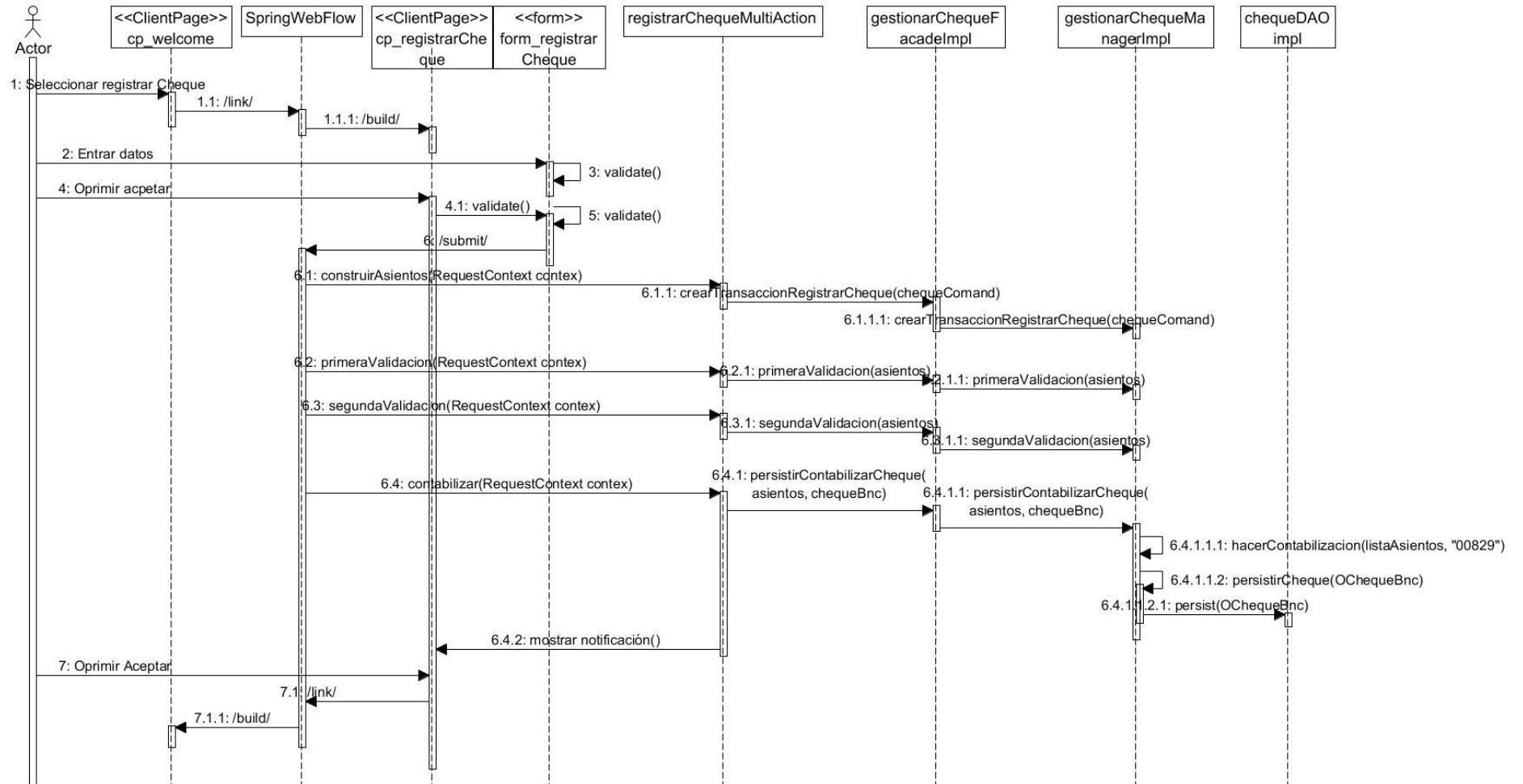
Referencia corriente	N/A	CH10105500000	N/A:
Referencia original	CC01101327000	CH10105500000	N/A:
Fecha contable	N/A	05/04/2012	N/A:
Fecha valor	N/A	05/04/2012	N/A:
Código de barra	1030690000287292283548280	1030690000287292283548219	N/A:
Número de cheque	28354828	28354821	N/A:
Tipo de cheque	Cheque Nominativo	Cheque Nominativo	N/A:
Estado de cheque	Pendiente	Al Cobro	N/A:
Moneda	CUC	CUC	N/A:
Importe	100	500	N/A:
Fecha de emisión	04/04/2012	01/04/2012	N/A:
Certificado	No seleccionado	No seleccionado	N/A:



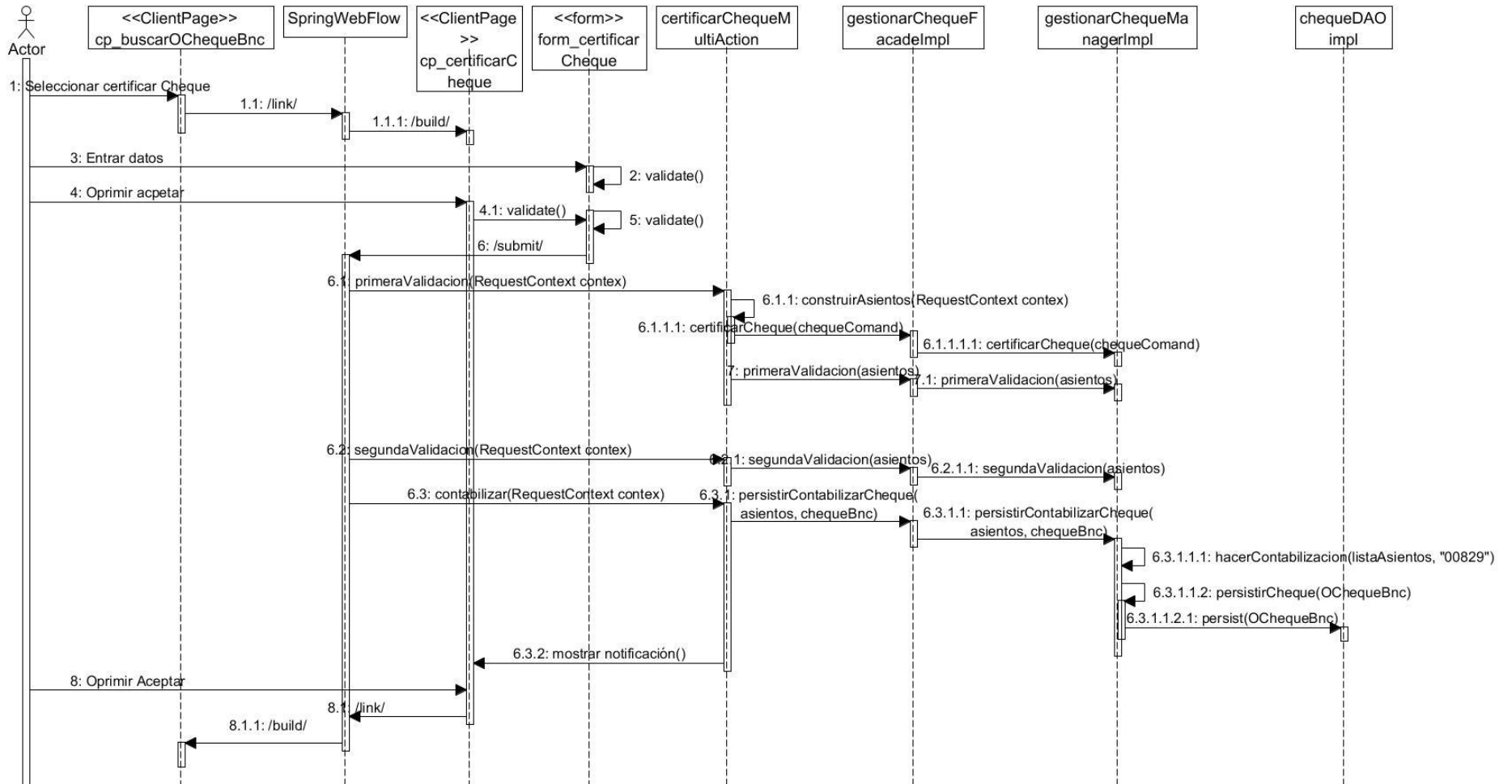
Banco emisor		Banco Financiero Internacional S.A	Banco Financiero Internacional S.A	N/A:
Sucursal		069	069	N/A:
Nombre emisor		MINISTERIO DE ECONOMIA Y PLANIFICACION	MINISTERIO DE ECONOMIA Y PLANIFICACION	N/A:
Cuenta emisor		0306900002872922	0306900002872922	N/A:
Contabilizar		No seleccionado	Seleccionado	N/A:
Tipo de operación		No accedido	Depósitos de Cheques (CuentaCorrientistas)	N/A:
Cuenta acreditar	Moneda	N/A	N/A:	N/A:
	Subcuenta	N/A	N/A:	N/A:
	Tipo de contraparte	N/A	N/A:	N/A:
	Contrapart e	N/A	N/A:	N/A:

	Desglose	N/A	N/A:	N/A:
Banco beneficiario		N/A	0136 - BNC	N/A:
Observaciones		Registrado 18/05/2012. emitido por MINISTERIO DE ECONOMIA Y PLANIFICACION	Registrado 18/05/2012. Emitido por MINISTERIO DE ECONOMIA Y PLANIFICACION. cobro de Comisiones	N/A:
Respuesta del sistema		El Sistema Registra la cuenta y se muestra un mensaje notificando el éxito de la operación.	El Sistema Registra la cuenta y se muestra un mensaje notificando el éxito de la operación.	El sistema cancela la operación.
Resultado de prueba		Satisfactoria	Satisfactoria	Satisfactoria

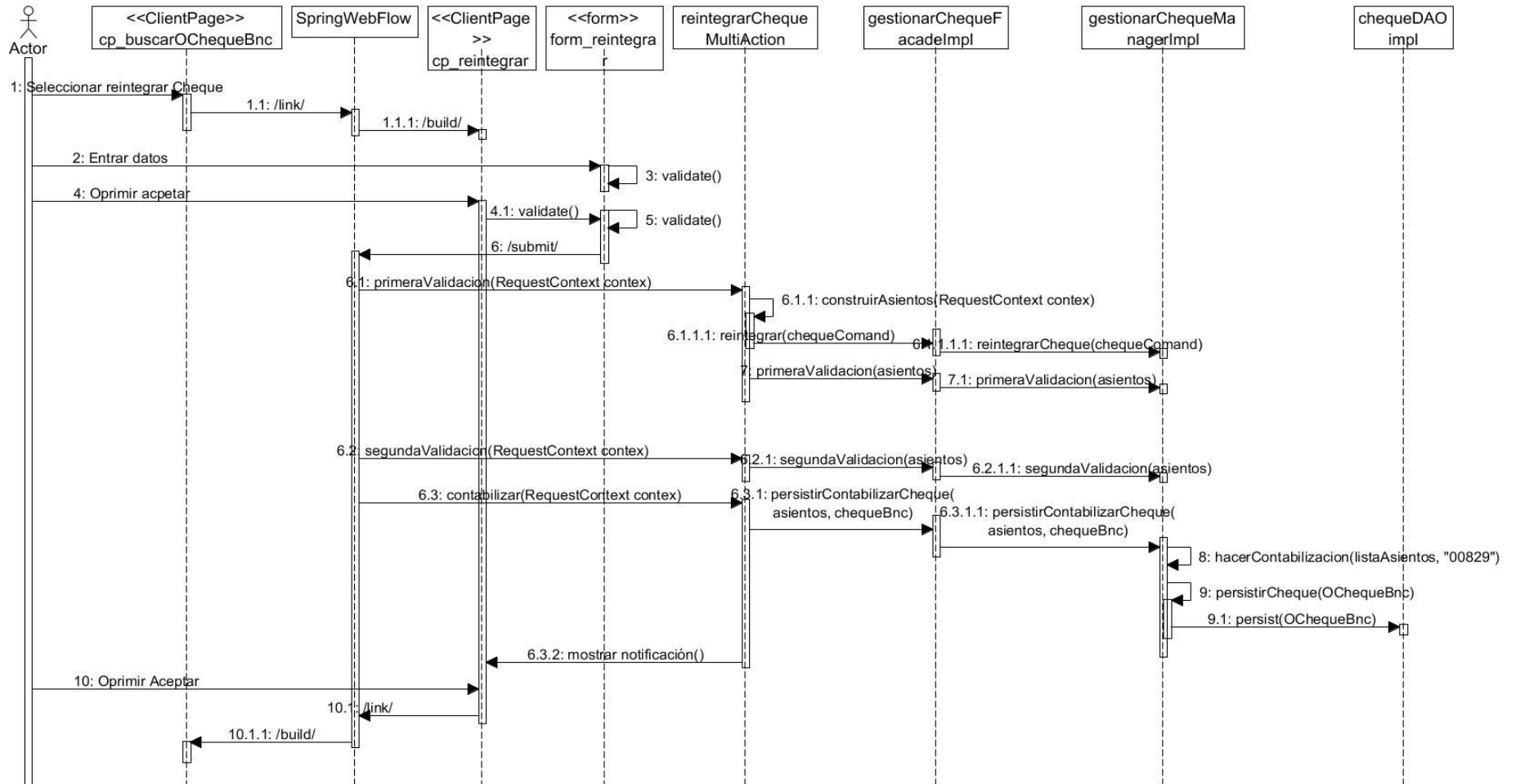
## Anexo : Diagrama de secuencia. Registrar Cheque



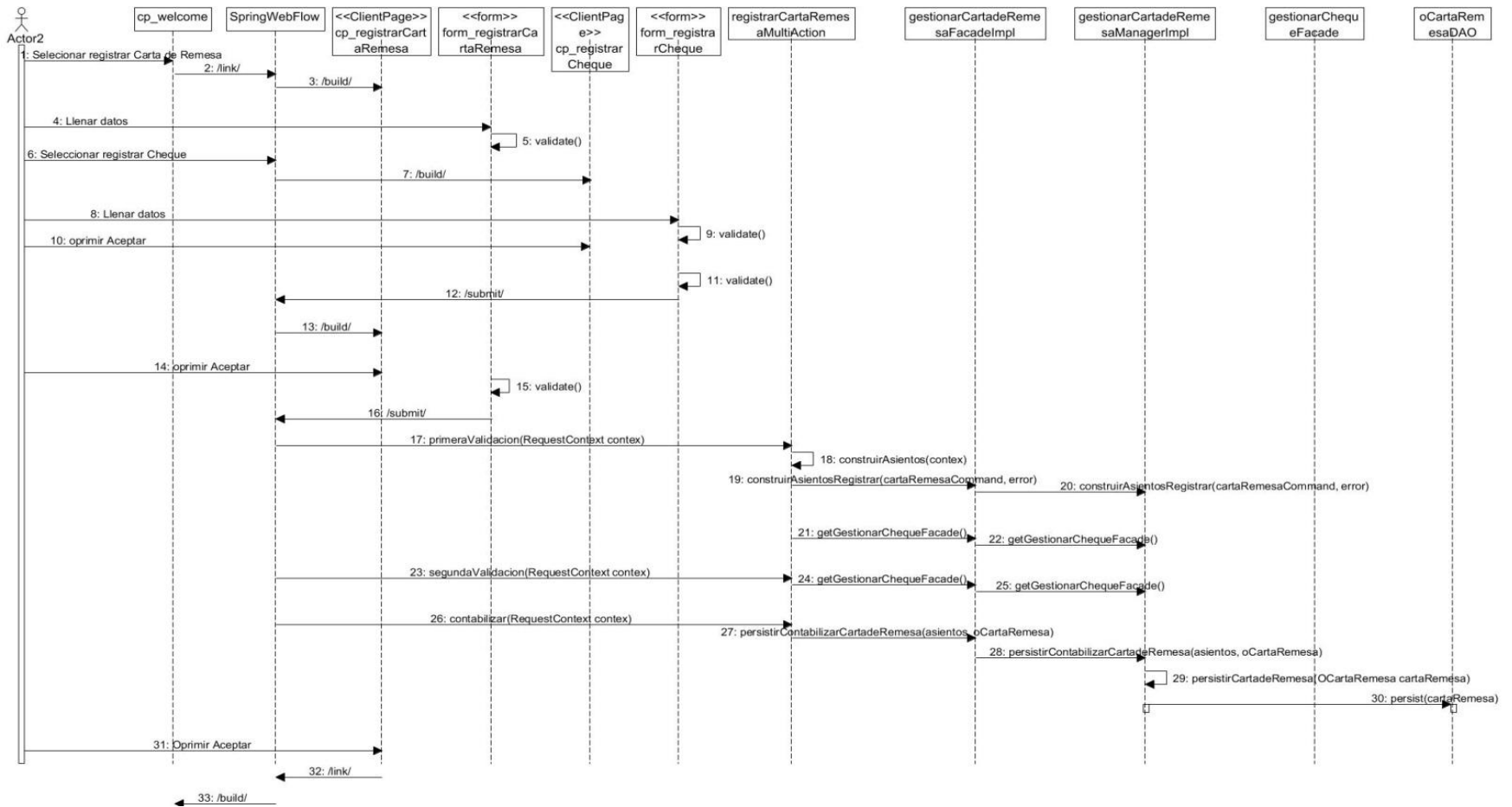
## Anexo : Diagrama de secuencia. Certificar Cheque



## Anexo : Diagrama de secuencia. Reintegrar Cheque



## Anexo : Diagrama de secuencia. Registrar Carta de Remesa



## GLOSARIO DE TÉRMINOS

*Herramienta GeneXus*: es una herramienta para el desarrollo de aplicaciones. Su objetivo es ayudar a los analistas de sistemas a implementar aplicaciones en el menor tiempo y con la mejor calidad posible. Es el producto principal de la compañía uruguaya Artech.

*Front-office*: es un lugar donde el cliente entra en contacto con la empresa; es el espacio en el cual el consumidor se vuelve protagonista absoluto y donde la empresa tiene que dar la mejor imagen de sí a los ojos de quien compra.

*Back office (trastienda de la oficina)*: es la parte de las empresas donde se realizan las tareas destinadas a gestionar la propia empresa y con las cuales el cliente no necesita contacto directo. Por ejemplo: el departamento de informática y comunicaciones que hace que funcionen los ordenadores, redes y teléfonos, el departamento de recursos humanos, el de contabilidad.

*Entidades bancarias (Bancos y Cajas de Ahorro)*: es una institución financiera que se encarga de administrar el dinero de unos para prestarlo a otros. La banca, o el sistema bancario, es el conjunto de entidades o instituciones que, dentro de una economía determinada, prestan el servicio de banco o banca.

*Endoso*: es lo escrito al dorso de un documento negociable o de otra naturaleza. El endoso es el medio, además de la entrega, por el cual los documentos a la orden pueden negociarse a otra persona.

*TDD (Test Driven Development)*: es una práctica de programación para el desarrollo guiado por pruebas con el objetivo de lograr un código limpio que funcione.

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

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

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

*Metalinguaje*: en informática es el lenguaje utilizado para describir un sistema de lenguaje de programación.

*IDEs (Integrated Development Environment)*: Entorno de desarrollo integrado, es un programa compuesto por un conjunto de herramientas para un programador.

*BI (Business Intelligence)*: son herramientas de soporte de decisiones que permiten en tiempo real, acceso interactivo, análisis y manipulación de información crítica para la empresa. Estas aplicaciones proporcionan a los usuarios un mayor entendimiento que les permite identificar las oportunidades y los problemas de los negocios.

*Complejidad Ciclomática (en inglés, Cyclomatic Complexity)*: es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Es una de las métricas de software de mayor aceptación, ya que ha sido concebida para ser independiente del lenguaje.