Universidad de las Ciencias Informáticas

Facultad 3

"Diseño e Implementación de los módulos Chequera y Transferencias del sistema Quarxo."

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

Autor(es): Yoan Asdrubal Quintana Ramírez.

Tutor: Ing. Yesenia Perdomo Bello.

La Habana, Cuba

Junio 2011

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor del trabajo titulado: *"Diseño e implementación de los Módulos Chequera y Transferencias del sistema Quarxo."* y autorizamos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Yoan Asdrubal Quintana Ramírez

_____

Firma del Autor

Ing. Yesenia Perdomo Bello.

_____

Firma del Tutor

"…El hombre debe transformarse al mismo tiempo que la producción progresa; no realizaríamos una tarea adecuada si fuéramos tan sólo productores de artículos, de materias primas y no fuéramos al mismo tiempo productores de hombres…."

RESUMEN

El BNC[1]en conjunto con la Universidad de las Ciencias Informáticas se encuentran inmersos en la tarea de desarrollar el sistema Quarxo, realizado con el objetivo de automatizar un gran número de procesos que se llevan a cabo actualmente en esta entidad bancaria de forma manual a partir de que utiliza el SABIC[2], el cual se encuentra desarrollado sobre una tecnología que ha quedado atrás con la aparición de herramientas computacionales en la actualidad, por lo que dificulta la gestión de las Chequeras de forma automatizada y no permite la gestión de Transferencias de manera íntegra con el sistema de mensajera usado en la entidad.

El presente trabajo de diploma se basa en el Diseño y la Implementación de los módulos Chequera y Transferencia del sistema Quarxo. El mismo fue realizado sobre la base de patrones que facilitan y garantizan el desarrollo ágil y con calidad del sistema. Para la implementación de dichos módulos son utilizadas herramientas y tecnologías de la plataforma de Java, con el objetivo de aprovechar su alto potencial para el desarrollo de aplicaciones web, además de ser software libre. Se hará referencia al uso de algunos elementos fundamentales tales como la utilización de SpringWebFlow en la parte de implementación para facilitar el control de los flujos web; y la realización de pruebas unitarias y pruebas de caja negra con el objetivo de validar la solución propuesta.

PALABRAS CLAVE

Análisis, diseño, Quarxo, Chequeras, Transferencias.

[1] Banco Nacional de Cuba
[2] Sistema Automatizado para la Banca Internacional de Comercio

INDICE DE CONTENIDO

INTRODUCCIÓN

La economía cubana está inmersa en el creciente cambio que trae consigo la incorporación de sus sectores a la utilización de las tecnologías de la información. El sistema bancario cubano como eslabón fundamental que controla los flujos monetarios y de negocios de nuestro país necesita informatizar sus procesos y lograr el avance tecnológico requerido para brindar los mejores y más eficientes servicios a sus clientes y a otros bancos. La utilización de herramientas computacionales permite obtener sistemas informáticos que se adapten a las estrategias de negocio de cada entidad, facilitando el perfeccionamiento de sus procesos y la utilización de las nuevas tecnologías para agilizar sus servicios. El BNC como parte de la modernización del sistema bancario se encuentra estructurando sus procesos y definiendo los procesos sin automatización con el principio de tener un sistema que le permita obtener agilidad y seguridad en sus obligaciones bancarias.

En la actualidad el BNC utiliza el SABIC en su versión MS-DOS. Este sistema partiendo de su diseño garantiza los aspectos relacionados con la integridad de la información que maneja. El SABIC es un sistema desarrollado sobre tecnología decadente, imposibilitando la ejecución y el control de la mayoría de los procesos que se ejecutan en las gerencias del propio banco. Los procesos asociados a la venta de Talonarios de Cheques o Chequeras no se encuentran automatizados en su totalidad. Permitiendo el sistema sólo el registro de las formas numeradas u hojas de Cheques. Este sistema permite la gestión de transferencias bancarias pero no permite su integración en un único sistema con la mensajería SWIFT[3] asociada a las transferencias enviadas al usar para esto el sistema de mensajería SISCOM[4], y estar el mismo desarrollado sobre Windows, por lo que los trabajadores de esta área realizan las operaciones contables usando sistema operativo MS-DOS pero tienen que reiniciar y cambiar a Windows para realizar las operaciones de mensajería necesarias.

Por lo antes expuesto el Banco Nacional de Cuba le solicitó a la Universidad de las Ciencias Informáticas (UCI) el desarrollo de un software capaz de gestionar los diferentes procesos existentes en esa entidad;

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

[4] Sistema de Comunicación mensajería SWIFT usado entre los bancos de a nivel mundial.

entre los que se encuentran los relacionados con la Venta de Talonarios de Cheques a los clientes y el envío, recepción o traspaso de transferencias bancarias, integrando dichos procesos con el envío de mensajes relacionados a las transferencias enviadas.

Apoyándose en el estudio realizado se define el siguiente **problema a resolver**: ¿Cómo mejorar la gestión de Chequeras y Trasferencias en el BNC? Se define como **objetivo general**: Obtener una solución general que permita gestionar Chequeras y Transferencias en el BNC. Teniendo como **objeto de estudio:** La gestión de los procesos de Chequeras y Trasferencias en entidades financieras bancarias centrando como **campo de acción:** La gestión de Chequeras y Trasferencias en el BNC.

Partiendo de esto se identifican los siguientes **objetivos específicos**:

- Realizar una fundamentación teórica de la solución.
- Diseñar e implementar los módulos Chequeras y Transferencias.
- Validar la solución implementada para la gestión de Chequeras y Transferencias.

Se determinaron las siguientes **tareas a cumplir**:

- Caracterización de las herramientas, lenguajes y frameworks para el desarrollo de la solución que gestiona Chequeras y Transferencias para el BNC.
- Caracterización de los procesos relacionados con la gestión de las Chequeras y las Transferencias en las entidades financieras bancarias en el BNC.
- 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 Chequeras y Transferencias.
- Realización del diagrama de secuencia del diseño de los módulos Chequera y Transferencias.
- Realización del modelo de datos de los módulos Chequera y Transferencias.

- Realización del modelo de componentes de los módulos Chequeras y Transferencias.

- Realización de pruebas unitarias de la solución para la gestión de las Chequeras y las Transferencias en el BNC.

Teniendo como **posible resultado** la obtención a partir del diseño y la implementación los módulos Chequera y Transferencia del sistema Quarxo.

El presente trabajo fue estructura en tres capítulos:

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

En este capítulo se hace un estudio del estado del arte de la gestión de Chequera y las Transferencias bancarias en el mundo. Se analizan algunas ventajas y desventajas de algunos sistemas bancarios mundiales. Son especificadas las características de la metodología utilizada así como lenguajes, herramientas y frameworks que conforman el ambiente de desarrollo sobre el cual se realizo el sistema.

CAPÍTULO 2. ARQUITECTURA Y DISEÑO DEL SISTEMA.

En este capítulo se realiza un estudio de la arquitectura definida por la dirección del proyecto, además fueron modelados los diagramas correspondientes al modelo de diseño de los módulos Chequeras y Transferencias del proyecto Quarxo. También se caracterizan los patrones usados durante el desarrollo del sistema.

CAPÍTULO 3. IMPLEMENTACION Y PRUEBA.

Esta centrado en la implementación de la solución propuesta para los módulos Chequera y Transferencia del sistema Quarxo, explicando en cada caso aspectos como los estándares de codificación, la interacción de los componentes a través de los diagramas correspondientes, descripción de un conjunto de clases y funcionalidades, así como una breve descripción de la utilización de Spring WebFlow. Además es realizada la validación de la solución a partir de la aplicación de las pruebas unitarias, mostrando una explicación de las mismas.

Finalmente se muestran las conclusiones del trabajo, las recomendaciones propuestas, la bibliografía, los anexos y el glosario de términos.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

El presente capítulo pretende realizar un estudio del estado del arte abordando en primer lugar los conceptos y definiciones relacionados con la contabilidad bancaria que permita una comprensión de los procesos asociados a la gestión de Chequeras y gestión de Transferencias. En segundo lugar se definen las metodologías, herramientas y lenguajes de modelado a utilizar a lo largo de la investigación así como el ambiente de desarrollo sobre el cual se desarrolló el sistema.

1.2. Conceptos Fundamentales.

1.2.1. Sistema Bancario Cubano. BNC.

Según Muñoz un Sistema Bancario es el conjunto de entidades o instituciones que dentro de una economía, prestan los servicios de banco, es decir almacenan los fondos, y garantizan que estén disponibles para cuando se solicite (Muñoz, 2008)

Los bancos por su parte son instituciones que realizan operaciones con créditos, recibiendo y concentrando en forma de depósitos los capitales captados para ponerlos a disposición de quienes puedan hacerlos fructificar. Su función principal es administrar y negociar el dinero depositado por las personas jurídicas o naturales que confían estas instituciones. También actúan como intermediarios pues son los encargados de tramitar todos los pagos e ingresos resultados de los negocios de sus clientes, al ser las entidades que almacenan y manipulan sus fondos.

La política económica cubana se encuentra evolucionando sustancialmente hacia un proceso de reformas y ajustes económicos que se adapten a las nuevas circunstancias en que se desarrolla la sociedad, en este contexto la banca juega un papel muy importante.

El actual sistema financiero cuenta con 8 bancos comerciales, 17 instituciones financieras no bancarias y 11 oficinas de representación de instituciones financieras extranjeras. El BNC fue creado en 1948, como banco central del Estado, con autonomía orgánica, personalidad jurídica independiente y patrimonio

propio, funcionando así hasta el 28 de mayo de 1997. Como consecuencia del proceso de organización de sistema bancario cubano, el 23 de febrero de 1998 se le asignan, por decreto, estructura, funciones y actividades asignadas como organización bancaria internacional. (LLORENTE, 2002)

Entre sus funciones se encuentran:

- Obtener y otorgar créditos en cualquier moneda. (Muñoz, 2008)
- Centralizar las relaciones oficiales con las entidades extranjeras de seguro de crédito oficial a la exportación, según decida el BCC. (Muñoz, 2008)
- Constituir entidades de seguro de crédito oficial a la exportación con arreglo a la legislación vigente en materia de seguros. (Muñoz, 2008)
- Mantener el registro, control, servicio y atención de la deuda externa que el Estado cubano y el BNC tiene contraída con acreedores extranjeros hasta la fecha de entrada en vigor del Decreto Ley No. 172, de 1997, del Banco Central de Cuba. (Muñoz, 2008)
- Se centra exclusivamente en la financiación y garantía de aquellas operaciones comerciales que el gobierno califica de prioritarias. (Muñoz, 2008)

1.2.2. Chequeras y Transferencias.

El Cheque es un titulo valor por medio del cual una persona llamada girador (quien posee una cuenta corriente bancaria) ordena a un banco llamado girado, que pague una determinada suma de dinero a la orden de un tercero llamado beneficiario.

Para efectuar un pago mediante cheque, primero se llena el talonario para así tener control de a quien se le emitió el cheque y por cuanto se lo emitió. El talonario contiene los siguientes datos:

1. Número de Cheque
2. Cantidad del Cheque
3. Fecha
4. A quien se le emitió
5. Propósito o motivo por el cual se extendió
6. Saldo anterior

7. Cantidad depositada
8. Total
9. Menos la cantidad del cheque
10. Saldo disponible

Las Chequeras son Talonarios de Cheques que se confeccionan y se les vende a los clientes de BNC para que realicen el pago de sus obligaciones u otras negociaciones.

La transferencia bancaria es una operación mediante la cual el titular de una cuenta disponible a la vista en una entidad de crédito, ordena a esta que transfiera determinados fondos de la misma, a una cuenta también disponible a la vista a nombre de un tercero, abierta en la misma u otra entidad de depósito. La transferencia es iniciada por una orden del titular (ordenante) de una cuenta, que solicita en la misma un traspaso de fondos entre dos cuentas, donde la cuenta destino puede ser de él o de otro titular. El banco emisor lo puede hacer de forma directa o utilizando los servicios de otra entidad. (Muñoz, 2008)

En el BNC se llevan a cabo tres tipos de transferencias, transferencias enviadas, recibidas y de traspaso.Las trasnferencias enviadas son aquellas que se llevan a cabo desde un cliente hacia una institución bancaria, las mismas pueden tener asociadas opcionalmente el uso de la mensajeria SWIFT. Las transferencias recibidas se llevan a cabo desde una cuenta bancaria hacia una cuenta de un cliente. Las transferencias de traspaso son aquellas en las que la cuenta origen y la cuenta destino son ambas del mismo tipo, o sea, son cuentas asociadas a clientes o cuentas asociadas a entidades bancarias.

1.3. Sistemas Automatizados de Gestión Bancaria.

1.3.1. SABIC

La Dirección de Sistemas Automatizados del BCC para satisfacer las necesidades de procesamiento de datos de bancos e instituciones financieras desarrolló el Sistema Automatizado para la Banca Internacional de Comercio (SABIC). Dentro de las principales características funcionales del SABIC se encuentran la contabilización en tiempo real que permite la extracción de dinero de una cuenta o el sobregiro de cualquier otra cuenta se realice de manera segura, la contabilización multimoneda que permite poder registrar los activos y pasivos sin tener que hacer conversiones de moneda lo cual garantiza una mayor exactitud de la información. La característica transaccional del sistema se basa en la contabilización de operaciones mediante transacciones las cuales son el conjunto de asientos requeridos para registrar una operación. Es un sistema modular que facilita la adaptabilidad, flexibilidad y evolución del sistema sin tener que efectuar cambios en sus programas generales. (SARDIÑA, 1998). A pesar de estas facilidades no facilita la realización de los procesos asociados a las Chequeras y las Transferencias. No tiene

automatizado un sistema de gestión de la información y los estados de las hojas y Cheques que se generan del Talonario de Cheques. No posee de forma integrada un sistema de mensajería necesario para la realización de transferencias bancarias de tipo *enviadas.*

1.3.2. Banco Bolivariano

Este sistema permite solicitar la creación de Chequeras de manera online en internet, posibilitando escoger la cuenta corriente de la cual va a ordenar la chequera, los números de cheques que desea ordenar así como la oficina de entrega donde puede retirar la Chequera. Una vez realizada esta operación es generado un comprobante de transacción el cual indica el tiempo que debes esperar para ir a retirar tu chequera. Para retirar la chequera la persona que se presenta, la cual puede ser el propio emisor o un sustituto con previa autorización, debe llenar una solicitud impresa que trae consigo la Chequera. Este sistema ofrece gran agilidad en sus servicios y disponibilidad constante durante el año completo pero no se adapta a las estrategias de negocio del BNC, al utilizar recursos que se hace difícil adquirir por su alto coste y la situación económica existente en el país.

1.3.3. ABANFIN

Este sistema garantiza la gestión de transferencias bancarias utilizando dos vías diferentes:

- Mediante la conexión a sistema de banca electrónica: nos permite ordenar la transferencia en cualquier horario del día, siendo ordenada por el propio usuario a partir de la utilización de una firma electrónica que lo identifica de manera única.
- Mediante la transmisión de un fichero normalizado a la entidad financiera: cuando se ha de realizar un número elevado de transferencia son usados los cuadernos[5] de la AEB[6]. Esto permite que los programas modernos confeccionen de manera automática los ficheros con los datos necesarios y sean transmitidos posteriormente vía internet a la entidad donde deseamos realizar las transferencias.

[5] Serie de normas usada para la codificación de un fichero electrónico en el cual se detallan los datos tanto del emisor como el beneficiario de las transferencias bancarias.

[6] Asociacion Española de la Banca

Este sistema por sus características no se adapta a las estrategias de negocio del BNC ya que no permite definir los tipos de transferencias que son manejadas en el mismo, además no ofrece la posibilidad del uso de mensajería SWIFT.

Realizado el análisis de algunos sistemas informáticos a nivel mundial y como manejan la gestión de Chequera y Transferencias se llego a la conclusión de que se hace poco factible adquirir alguno de ellos ya que se encuentran desarrollados sobre software privativos, además no se adaptan a las estrategias de negocio del BNC al poseer estas características particulares, por lo que se hace necesario la implementación de un sistema contable que se adapte a las necesidades de dicha entidad.

1.4. Tecnologías y herramientas utilizadas en el modelado.

1.4.1. Metodología. (RUP[7])

Una metodología no es más que el estudio de los métodos más apropiados que se emplean para desarrollar software de manera eficiente; o como precisan otros autores (Jacobson, 2000), define Quién debe hacer Qué, Cuándo, y Cómo debe hacerlo.

RUP es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyectos. Utiliza el Lenguaje Unificado de Modelado para preparar todos los esquemas de un sistema software, siendo UML una parte esencial del Proceso Unificado. (Jacobson, 2000)

1.4.2.Lenguaje de Modelado. (UML[8])

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimientos sobre los sistemas que se deben construir. Se usa para entender, diseñar, configurar y

[7]Proceso Unificado de Desarrollo por sus siglas en español.

[8] Lenguaje Unificado de Modelado por sus siglas en español.

controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. UML incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. Puede ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código así como generadores de informes. La especificación de UML no define un proceso estándar pero es útil en un proceso de desarrollo iterativo. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientado a objetos. (IvAr Jacobson, 2001)

Es un lenguaje gráfico para visualizar, especificar y documentar un sistema de software. Ofrece un estándar para describir un panorama del sistema (modelo), incluyendo aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos, componentes de software reutilizables y aspectos conceptuales como los procesos de negocios y funciones del sistema (Jacobson, 2000).

1.4.3. Herramienta de modelado Visual Paradigm para UML.

Visual Paradigm para UML es una Herramienta CASE[9] que soporta las últimas versiones del Lenguaje Unificado de Modelado y la Notación de Modelado de Procesos de Negocios.

Visual Paradigm es una herramienta que ofrece un entorno de creación de diagramas usando las notaciones UML 2.0 y BPMN[10], con un diseño centrado en casos de uso y enfocado además al negocio, lo cual genera un software de alta calidad. Esta herramienta fue creada para el ciclo completo de desarrollo del software que lo automatiza y acelera, permitiendo la captura de requisitos, análisis, diseño e implementación, también proporciona características tales como generación del código, ingeniería inversa y generación de informes. Permite escribir toda la especificación de un caso de uso sin necesidad de utilizar una herramienta externa como editor de texto, utilizando plantillas que se encuentran definidas, o que pueden ser creadas por los usuarios. (RODRÍGUEZ, 2008)

[9] Computer Aided Software Engineering / Ingeniería de Software Asistida por Ordenador

[10] Notación de Modelado de Procesos de Negocios por sus siglas en español.

Siendo Visual Paradigm una herramienta con disímiles ventajas para el modelado de cualquiera de los diagramas para UML, se decidió utilizar para el desarrollo de los diagramas del proyecto Quarxo.

1.5. Patrones de diseño.

1.5.1. Patrones de asignación de responsabilidades. (GRAPS[11])

Los patrones GRASP, describen los principios fundamentales de la asignación de responsabilidades a objetos, expresadas en forma de patrones. Craig Larman en su libro "UML y Patrones" define: "Los patrones GRASP constituyen un apoyo para la enseñanza que ayuda a uno a entender el diseño de objetos esencial, y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable." (Lago, 2007)

1.5.1.1. Experto.

Este patrón permite asignar una responsabilidad al experto en información, la clase que cuenta con la información necesaria para cumplir la responsabilidad. Durante el diseño orientado a objetos, cuando se definen las interacciones entre los objetos, tomamos decisiones sobre la asignación de responsabilidades a las clases. Si se hacen en forma adecuada, los sistemas tienden a ser más fáciles de entender, mantener y ampliar, y se nos presenta la oportunidad de reutilizar los componentes en futuras aplicaciones. Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto provee un bajo nivel de acoplamiento. (Lara, 2010)

1.5.1.2. Creador.

Este patrón permite definir quién debería ser responsable de crear una nueva instancia de alguna clase. La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. En consecuencia, conviene contar con un principio general para asignar las responsabilidades concernientes a ella. El diseño, bien asignado, puede soportar un bajo acoplamiento, una mayor claridad, el encapsulamiento y la reutilizabilidad. (Lago, 2007)

[11] Patrones generales de software para asignar responsabilidades por sus siglas en español.

1.5.1.3. Bajo acoplamiento.

Una clase con bajo acoplamiento no depende de “muchas otras” clases. Las clases con alto acoplamiento recurren a muchas clases y no es conveniente. Son más difíciles de mantener, entender y reutilizar. (Lara, 2010)

El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo (o débil) acoplamiento no depende de muchas otras. Una clase con alto (o fuerte) acoplamiento recurre a muchas otras. (LARMAN, 1999)

1.5.1.4. Alta cohesión.

En la perspectiva del diseño orientado a objetos, la cohesión (o, más exactamente, la cohesión funcional) es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme y evita problemas tales como: difíciles de comprender, difíciles de reutilizar, difíciles de conservar y ser delicadas al ser afectadas constantemente por los cambios.

1.5.1.5. Controlador.

El controlador es un intermediario entre la interfaz de usuario y el núcleo de las clases donde reside la lógica de la aplicación. El controlador no realiza mucho trabajo por sí mismo; más bien coordina la actividad de otros objetos. (Lara, 2010).

Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones (LARMAN, 1999):

- El sistema global (controlador de fachada).
- La empresa u organización global (controlador de fachada).
- Algo en el mundo real que es activo (por ejemplo, el papel de una persona) y que pueda participar en la tarea (controlador de tareas).
- Un manejador artificial de todos los eventos del sistema de un caso de uso, generalmente denominados controlador de casos de uso).

1.5.2. Patrones Estructurales.

1.5.2.1. Fachada.

Este patrón permite simplificar el acceso a un conjunto de clases o interfaces. Proporcionar una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. La fachada satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas. (Prieto, 2009)

Según otros autores el objetivo de la utilización de este patrón es proveer un intermediario reduciendo el número de objetos con los que interactúa un cliente y una interfaz, proporcionando un bajo acoplamiento, además de emitir cambios en el la lógica implementada sin afectar al cliente que la utiliza ya que la misma esta oculta tras la Facade. (Lara, 2010)

1.5.2.2. Patrón de Acceso a Datos. (DAO)

Este patrón centraliza todo el acceso a datos en una capa independiente, aislándolo del resto de la aplicación. Desacoplando la lógica de negocio de la lógica de acceso a datos, de manera que se pueda cambiar la fuente de datos fácilmente Sus principales beneficios son que reduce la complejidad de los objetos de negocio al abstraerlos de la implementación real de la comunicación con la fuente de datos. Cada DAO tiene que implementar los métodos declarados en la interfaz DAO correspondiente. (Lara, 2010)

1.5.3. Patrones de presentación.

1.5.3.1. Vistas Compuestas.

Un objeto vista que está compuesto de otros objetos vista. Por ejemplo, una página JSP que incluye otras páginas JSP y HTML usando la directiva *include.* (Lara, 2010)

1.5.3.2. Modelo Vista Controlador. (MVC)

Este patrón propone dividir la aplicación en tres capas el Modelo, la Vista y el Controlador, el modelo es la representación del dominio o datos del sistema, la vista se encarga de presentar la interfaz al usuario, en sistemas web, esto es típicamente HTML, aunque pueden existir otro tipo de formatos. En la vista sólo se deben de hacer operaciones simples y el controlador es el encargado de escuchar los cambios en la vista y enviarlos al modelo, el cual le regresa los datos a la vista como un ciclo. Cuando se aplica este patrón los accesos a la base de datos se hacen en el modelo, la vista y el controlador no deben de saber si se usa o no una base de datos. El controlador es el que decide que vista se debe de imprimir y que información es la que se envía. (Lara, 2010)

1.6. Ambiente de Desarrollo.

1.6.1. Plataforma JEE.

Java 2 Enterprise Edition define una plataforma para crear aplicaciones empresariales usando un modelo multicapas, dividiendo la aplicación en diferentes niveles, especializándose cada uno de en una tarea en particular.

Según Núñez su estructura está basada en Java 2, Standard Edition (J2SE) y un conjunto de sus APIs, a la cual J2EE aporta la especificación de componentes, containers y las APIs para los servicios de transacciones, mensajería, envío de correos y conectores a recursos externos. (NÚÑEZ, 2003)

1.6.2. Lenguaje Java.

El lenguaje Java es un lenguaje de propósito general, de alto nivel, que utiliza el paradigma de orientación a objetos. Algunas ventajas están basadas en la administración de memoria, siendo ejecutada por la máquina virtual automáticamente y no por el código de cada programa, y el soporte de procesos livianos o threads a nivel del lenguaje, que ayuda a controlar la sincronización de procesos paralelos. Estas características dan al lenguaje Java las propiedades de robustez y seguridad, evitando por ejemplo problemas de buffer overflow utilizados en ataques a sistemas. (NÚÑEZ, 2003)

Según otros autores es un lenguaje de programación orientado a objetos, robusto, simple, poderoso y fácil de aprender. Es un lenguaje multiplataforma de código abierto, distribuido, que proporciona un conjunto de clases para su uso en aplicaciones de red, permitiendo abrir sockets y establecer conexiones con servidores o clientes remotos. Además de ser poderoso manejando threads y excepciones. Java fue diseñado para crear software altamente fiable. (Lara, 2010)

1.6.3. Contenedor Web (Tomcat).

Tomcat es un contenedor web basado en el lenguaje Java que actúa como motor de Servlets y JSPs. Se ha convertido en la implementación de referencia para las especificaciones de Servlets y JSPs. Fue seleccionado como la implementación de referencia de contenedores de componentes web Sun (JSPs/Servlets) y está desarrollado en un entorno abierto Open Source. (Sun , 2011)

1.6.4. Gestor de Base de Datos. (SQL[12])

SQL Server es una solución de datos global, e integrada que habilita a los usuarios en toda su organización mediante una plataforma más segura, confiable y productiva para datos empresariales y aplicaciones de Inteligencia Empresarial (BI). SQL Server 2005 provee herramientas sólidas y conocidas a los profesionales , 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. (Microsoft, 2011)

1.6.5. Eclipse 3.3 IDE.[13]

Eclipse es un IDE para Java muy potente. Fue creado por la IBM bajo la filosofía de software libre. Se está convirtiendo en el estándar de puntera de los entornos de desarrollo para Java. Es Eclipse un marco de trabajo que está compuestos por componentes que se pueden o no incluir en dependencia de las necesidades del desarrollador, a estos complementos se les llama (plugins). De hecho, existen

[12] Structure Query Lenguage (Lenguaje Estructurado de Consultas por sus siglas en español).

[13] Integrated Development Environment (Entorno de Desarrollo Integrado por sus siglas en español.)

complementos que permiten usar Eclipse para programar en otros lenguajes aparte del Java como son PHP, Perl, Python, C/C++. (Lara, 2010)

1.7. Tecnologías y Frameworks.

Los Frameworks son librerías de clases que dan solución a un problema determinado o brindan utilidades sobre áreas definidas. Resulta normal que un framework no se ajuste totalmente a las necesidades de un proyecto en específico, por lo que es razonable que se necesite variar el funcionamiento que propone por lo que posee una estructura de Software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación.

1.7.1. Spring Frameworks

Es un framework de Java que facilitará la creación de aplicaciones para la empresa. Diseñado en módulos, con funcionalidades específicas y consistentes con otros módulos, facilita el desarrollo y hace que la curva de aprendizaje sea favorable para el desarrollador. (Lara, 2010)

Dentro de las ventajas que ofrece Spring, se puede decir que facilita la manipulación de los objetos se usen Enterprise JavaBeans (EJBs) o no, elimina la necesidad de usar distintos y variados tipos de ficheros de configuración, mejora la práctica de programación, realizando el mismo tipo de funciones sin ellos. Ofrece mucha libertad a los desarrolladores en Java, brindando soluciones bien documentadas, seguras y robustas al mismo tiempo que está compuesto por 11 grandes módulos.

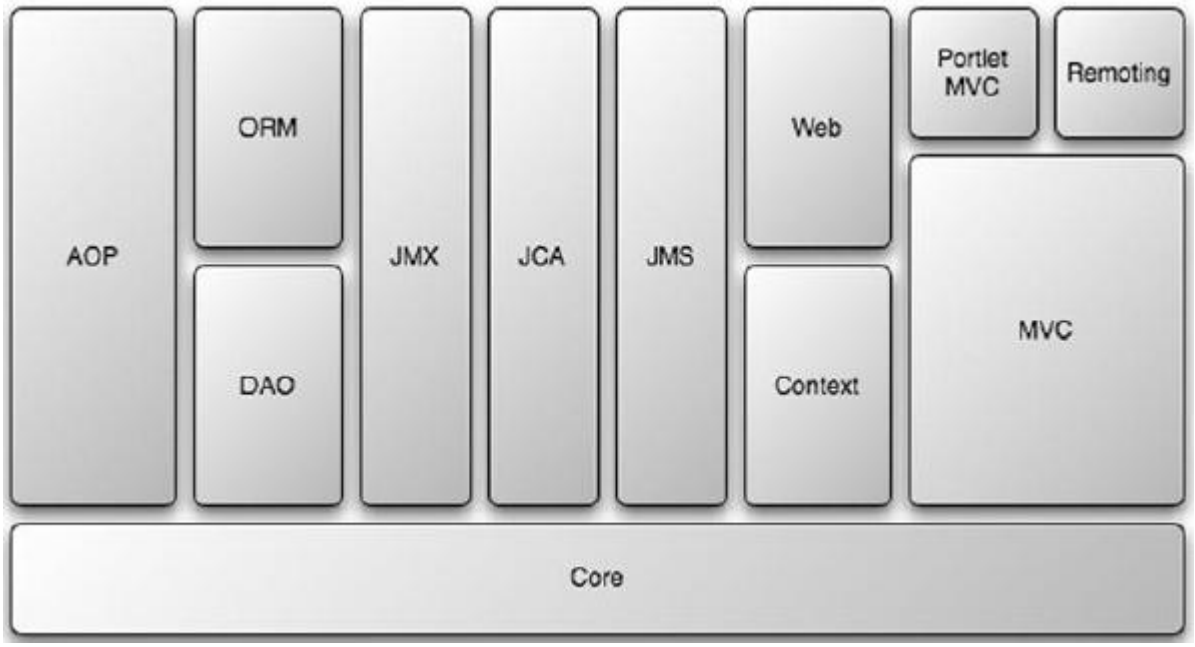

Figura 1. Módulos de Spring Framework.

1.7.2. Spring WebFlow.

Spring WebFlow es un framework que permite operar la navegación de la aplicación Web. Al ser limitado el flujo de páginas brindado por los Frameworks MVC clásico, surge el framework Spring WebFlow. Los flujos web en este framework son diseñados para ser auto-controlados, dando la posibilidad de definir reglas de navegación múltiples y complejas.

En Spring WebFlow un flujo controla la conversación (entorno nuevo que define el vacío entre Sesión y Petición) completa, desde que inicia hasta que termina, limpiando automáticamente la memoria siempre y cuando se termine el flujo. Permite la creación de flujos reutilizables en toda la aplicación. (Lara, 2010)

1.7.3. Hibernate.

Es un framework para persistir objetos en una base de datos relacional, disponible para Java con el nombre de Hibernate y para tecnología .NET con el nombre de NHibernate. Tiene grandes ventajas en cuanto a productividad, mantenibilidad, rendimiento e independencia del proveedor. Entre sus funciones

está permitir transacciones, asociaciones, polimorfismo, herencia, carga mediante referencia, la cual consiste en cargar sólo la referencia de donde se encuentran los datos, es llamada (carga lazy), persistencia transitiva, estrategias de fetching además de permitir distintas formas de realizar las consultas como son Hibernate Query Language (HQL), Java Persistence Query Language (JPQL), consultas por criterios y SQL nativo. (COMUNITY, 2001)

1.7.4. Dojo Toolkit.

Esta herramienta es un framework que contiene API[14] y controles para ayudar al desarrollo de aplicaciones web. Incluye un sistema de empaquetado, los efectos visuales de la interfaz de usuario, abstracción de eventos, almacenamiento de APIs en el cliente, e interacción de APIs con AJAX[15]. Dojo proporciona variadas opciones en una sola biblioteca haciendo un mejor trabajo que sustenta los nuevos y viejos Browsers, resolviendo los problemas de compatibilidad entre los navegadores. Tiene múltiples puntos de entrada, es independiente del intérprete y unifica estándares de codificación. (Lara, 2010)

1.8. Conclusiones Parciales.

Actualmente en el BNC se ven afectados los procesos asociados a la gestión de Chequeras y a la integración del sistema de mensajería con las Transferencias bancarias, haciéndose necesario un sistema que gestione de manera eficiente la que garantice estos aspectos al dificultarse la adquisición de alguno de los existentes actualmente por su alto coste y no adaptarse a las estrategias de negocio de la entidad.

La utilización de los lenguajes, frameworks y herramientas caracterizados facilitara el desarrollo del sistema de manera que se garantice la obtención final de un producto con la calidad requerida al menor coste posible y apto para funcionar en el BNC.

[14] Application Program Interface (Interfaz de Programación de Aplicaciones por sus siglas en Español)

[15] Asynchronous JavaScript And XML (JavaScript asíncrono y XML por sus siglas en Español)

CAPÍTULO 2 ARQUITECTURA Y DISEÑO DEL SISTEMA.

2.1. Introducción.

El presente capítulo tiene como objetivo la transformación de los requisitos funcionales en el diseño del futuro sistema, asegurando un mejor entendimiento de los artefactos necesarios para su posterior implementación siguiendo los principios definidos por la arquitectura de desarrollo propuesta por el proyecto. Se obtendrá como resultado un conjunto de componentes y artefactos que responden al flujo de trabajo de Diseño de la metodología utilizada. Además se caracterizará la arquitectura del sistema Quarxo, la cual responde a las necesidades planteadas por el proyecto.

2.2. Arquitectura del sistema Quarxo.

La arquitectura definida por el proyecto y sobre la cual se desarrolla el sistema es conocida como arquitectura por capas. Está compuesta por una Capa de Presentación, una capa de Negocio, una Capa de Acceso a Datos y además una capa con los objetos del dominio o Capa de Dominio transversal a las capas antes mencionadas. Esta arquitectura fue organizada de forma jerárquica por subsistemas, módulos y componentes, quedando estructurado de la forma siguiente:

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

Módulo: Conjunto de Casos de Uso relacionados con uno o más procesos bancarios.

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

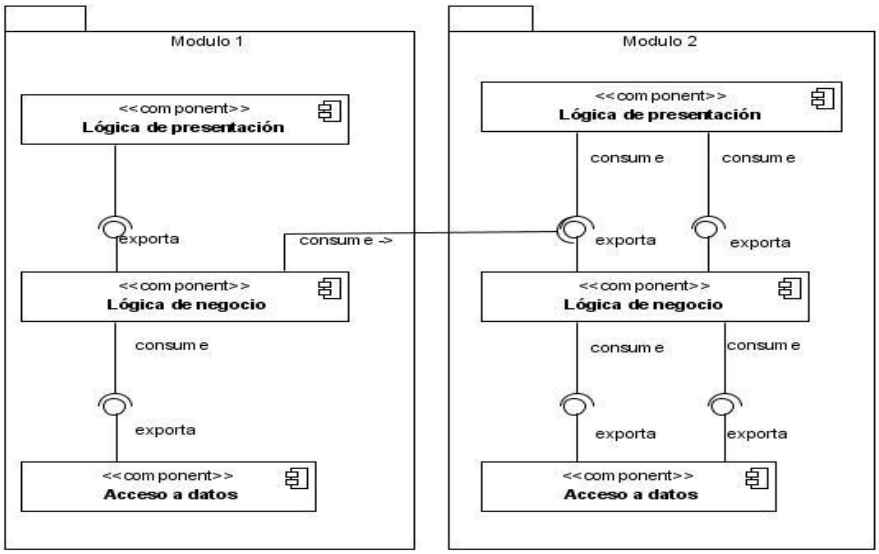

Figura 2: Arquitectura de un Subsistema de Quarxo.

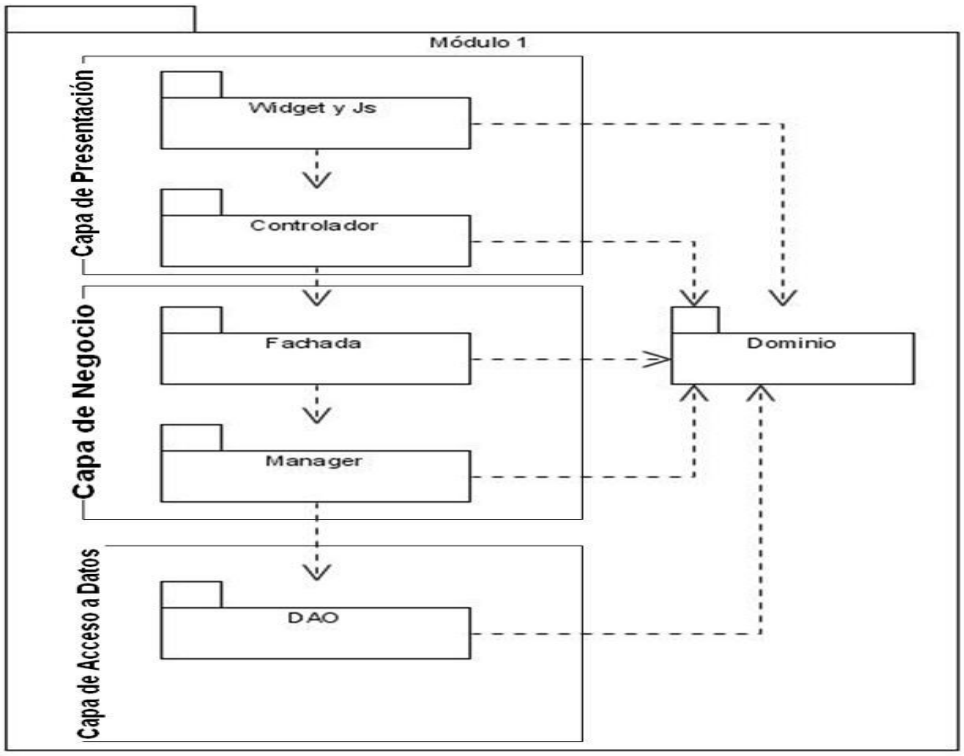

Figura 3: Arquitectura de un módulo de Quarxo.

2.2.1. Arquitectura de la Capa de Presentación.

En esta capa se desarrolla la lógica de presentación, usándose del lado del servidor Spring MVC (modelo Vista Controlador), con el objetivo de controlar, enviar y recibir peticiones realizadas desde el cliente. Por parte del cliente será usado el frameworks de presentación Dojo Toolkit, el cual permite la generación dinámica de interfaces agradables al usuario y Spring WebFlow para controlar los flujos de la aplicación.

2.2.2. Arquitectura de la Capa de Negocio.

La capa de negocio se encuentra dividida en dos secciones, Fachada y Administrador. Fachada es la encargada de facilitar las diferentes funcionalidades a la vista de la capa de presentación. Por su parte Administrador al interactuar con la capa de acceso a datos tendrá acceso a la información almacenada en la base de datos, por lo que será la parte del negocio encargada de implementar toda la lógica necesaria que responda a los requerimientos del módulo.

2.2.3. Arquitectura de la Capa de Acceso a Datos.

Esta capa es la encargada del acceso al gestor de base de datos, en la misma se realizaran todas las operaciones que permitan el acceso a los datos del sistema. Para la realización de esta capa será usado el patrón DAO[16] se relaciona con la capa de negocio a partir de las interfaces.

2.3. Diseño de la solución.

Uno de los artefactos más importantes que se generan a partir del flujo de trabajo de Requisitos lo constituye la Especificación de Requisitos, este elemento constituye la base para el diseño y la implementación de la solución planteada. En este caso se tomaran como entrada los requisitos funcionales relacionados con los módulos Chequera y Transferencia definidos anteriormente por los analistas del proyecto.

Cada requerimiento funcional se ve representado por los casos de uso que se mencionan a continuación:

[16]Objetos de Acceso a Datos (Data Access Object por sus siglas en ingles)

Subsistema Chequera:

- Registrar Chequera
- Imprimir Conformidad
- Consultar Chequera
- Verificar Hojas de Cheques

Subsistema Transferencia:

- Registrar Transferencia

2.3.1. Modelo del Diseño.

El modelo de diseño define la estructura estática del sistema en forma de subsistemas, clases o interfaces y los casos de usos reflejados como colaboraciones entre subsistemas, clases o interfaces. Describe la realización de los casos de usos del sistema basado en como los requisitos funcionales del sistema unido a otras restricciones asociadas al entorno de implementación tienen impacto en el sistema. Está compuesto por artefactos como los diagramas de paquetes, diagramas de clases, diagramas de interacción, además constituye la base para la entrada al flujo de trabajo de implementación empleándose como la entrada principal para esta etapa de desarrollo.

2.3.1.1. Diagrama de paquetes.

Con el objetivo de facilitar el entendimiento de la estructura del sistema resulta muy conveniente el desarrollo de los diagramas de paquetes, los cuales muestran las dependencias entre las agrupaciones lógicas en las que se encuentra dividido. Estas agrupaciones se pueden realizar atendiendo a diferentes criterios, en el caso de los módulos Chequera y Transferencia la agrupación se realizó en dependencia del rol que desempeñan estos ficheros en la aplicación.

A continuación se representa el diagrama de paquetes correspondiente al módulo Chequera, y una explicación de los componentes de cada uno de ellos.

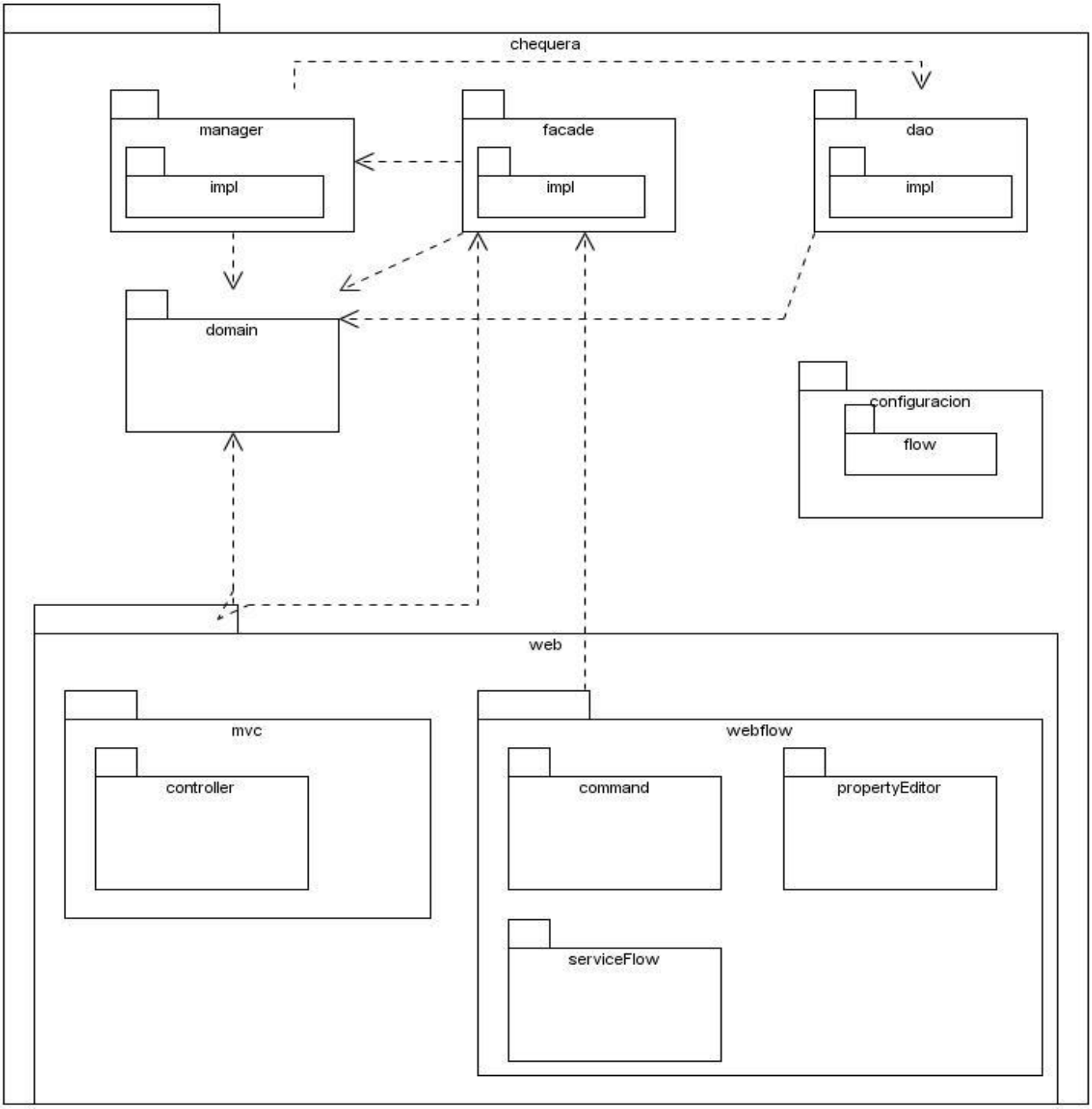

Figura 4: Diagrama de Paquetes del módulo Chequera.

Paquete configuración: En este paquete se encuentran los ficheros de configuración en formato XML de los diferentes contextos de Spring para el módulo, los cuales se mencionan a continuación:

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

Paquete Flow: En este paquete están presentes los archivos XML que definen los flujos para SpringWebFlow.

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

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

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

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

Paquete web: El paquete web agrupa un grupo de clases y paquetes que son los encargados de realizar todo lo referente a la presentación en el lado del servidor.
Paquete MVC: En este subpaquete se agrupa todo lo referente a la lógica de presentación cuando se hace uso de SpringMVC.
Paquete WebFlow: En este subpaquete se agrupa todo lo referente a la lógica de presentación cuando se hace uso de Spring WebFlow.

Paquete Controller: En este paquete se hallan las clases que extienden de los controladores propuestos por Spring MVC, encargadas de responder a las peticiones realizadas por el cliente.
Paquete ServiceFlow: Contiene las clases encargadas de establecer la comunicación entre el flujo y la fachada del módulo.
Paquete Command: Se encuentran las clases que representan los datos que son introducidos y mostrados en los formularios de la interfaz de usuario.
Paquete PropertyEditor: Agrupa las clases que actúan como intermediarias para convertir datos de la interfaz de usuario en objetos y viceversa.

2.3.1.2. Diagrama de Clases.

Un diagrama de clases es una representación gráfica que muestra una colección de elementos declarativos del modelo como clases, tipos y sus contenidos y relaciones. La realización de casos de uso del diseño describe como se realiza un caso de uso especifico y como se ejecuta en términos de clases de diseño y sus objetos. Una realización de caso de uso contiene diagramas de clases que muestran sus clases de diseño participantes. Específicamente los diagramas de clases de diseño son muy útiles porque muestran a través de atributos y métodos la estructura de las clases que después serán escritas en algún lenguaje de programación.

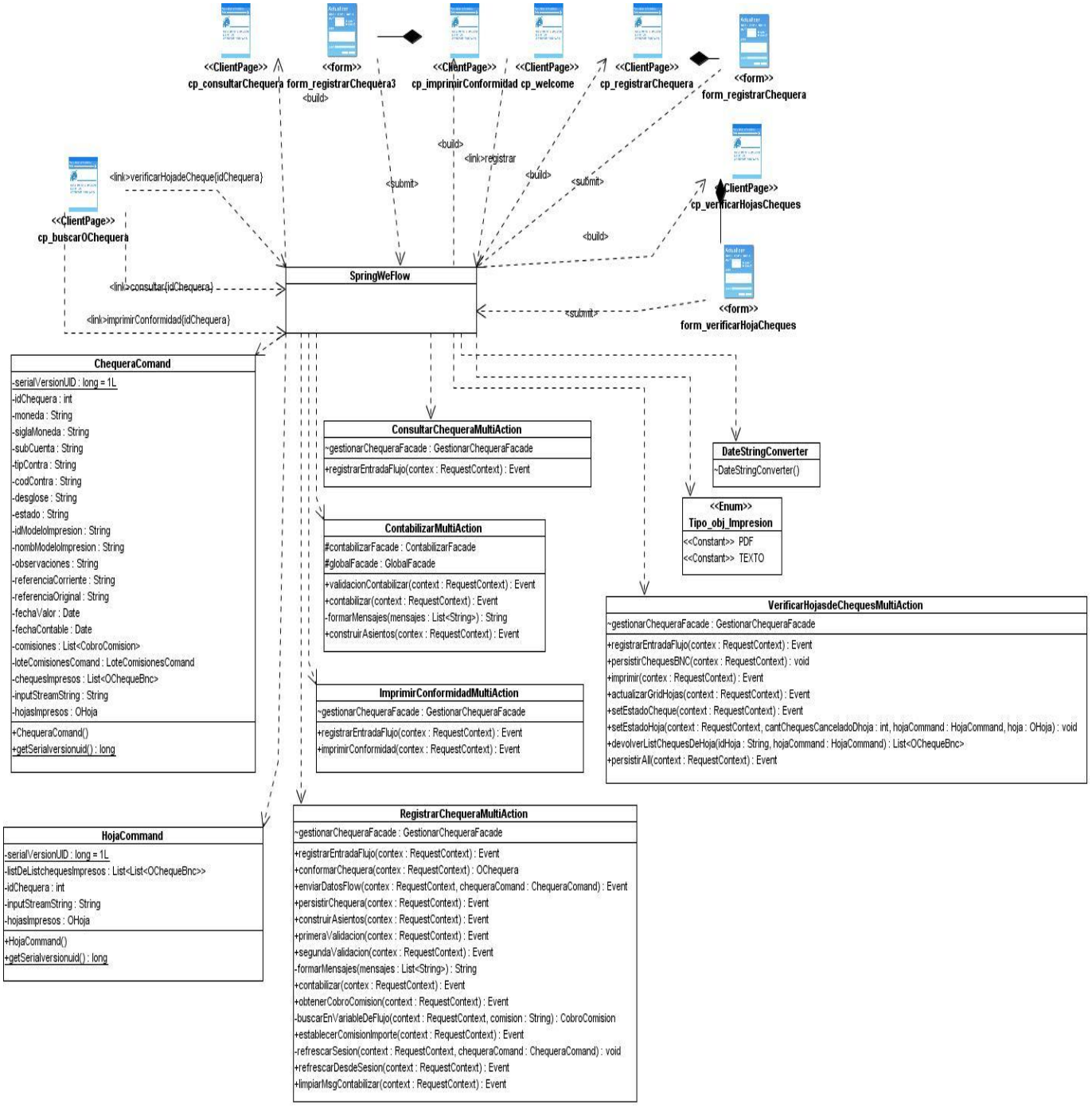

Figura 5: Diagrama de Clases de la capa de presentación del módulo Chequera.

Clases ClientPage: Páginas webs encargadas de mostrar los formularios de la información a los usuarios.

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

ConsultarChequeraMultiAction: Es la clase encargada de gestionar las acciones asociadas al flujo *consultar Chequera*, Spring WebFlow interactúa con ella para su funcionamiento.

ContabilizarMultiAction: Es la clase encargada de gestionar las acciones asociadas al flujo en el que se *contabiliza la Chequera*, Spring WebFlow interactúa con ella para su funcionamiento.

*ImprimirConformidadMultiAction***:** Es la clase encargada de gestionar las acciones asociadas al flujo *imprimir Conformidad*, Spring WebFlow interactúa con ella para su funcionamiento.

*RegistrarChequeraMultiAction***:** Es la clase encargada de gestionar las acciones asociadas al flujo *registrar Chequera*, Spring WebFlow interactúa con ella para su funcionamiento.

*VerificarHojasDeChequesMultiAction***:** Es la clase encargada de gestionar las acciones asociadas al flujo *verificar Hojas de Cheques*, Spring WebFlow interactúa con ella para su funcionamiento.

DateStringConverter: Property Editor usado por Spring que permite la transformación de un tipo de dato String en una fecha.

Tipo_obj_impresion: Enumerativo para manejar el tipo de impresión que se realizara en los casos de uso en los que se imprime.

ChequeraCommand: Clase usada por Spring FrameWorks que representa al objeto del dominio OChequera, empleada para enviar o recibir información de una página que represente a una Chequera.

HojaCommand: Clase usada por Spring FrameWorks que representa al objeto del dominio OHoja, empleada para enviar o recibir información de una página que represente a una Hoja de una Chequera.

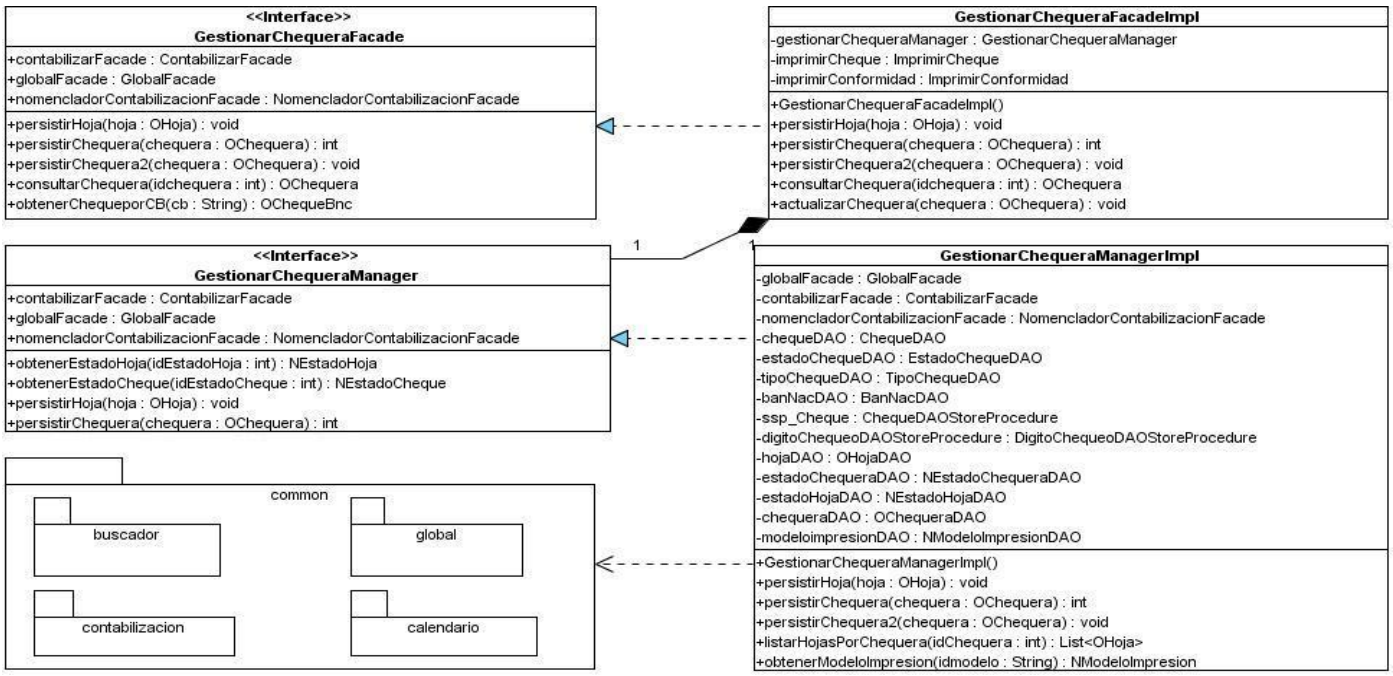

Figura 6: Diagrama de clases de Negocio del módulo Chequera.

En este diagrama solo se muestran algunas funcionalidades de las diferentes clases representadas con el objetivo de lograr un entendimiento de la estructura de clases de esta capa, dichas clases son mostradas detalladamente en los anexos con todas sus funcionalidades.

GestionarChequeraFacade: Interfaz encargada de brindar las funcionalidades del módulo a la capa de presentación o a otros módulos que lo requieran.

GestionarChequeraFacadeImpl: Clase encargada de implementar la lógica de programación de la interfaz GestionarChequeraFacade.

GestionarChequeraManager: Interfaz que contiene las funcionalidades que deben implementarse para dar respuesta a las funcionalidades requeridas desde GestionarChequeraFacadeImpl.

GestionarChequeraManagerImpl: Clase encargada de implementar la lógica de programación definida en GestionarChequeraManager.

Common: Es un paquete que representa las funcionalidades que son comunes para toda la aplicación, el mismo es usado desde la implementación GestionarChequeraManagerImpl.

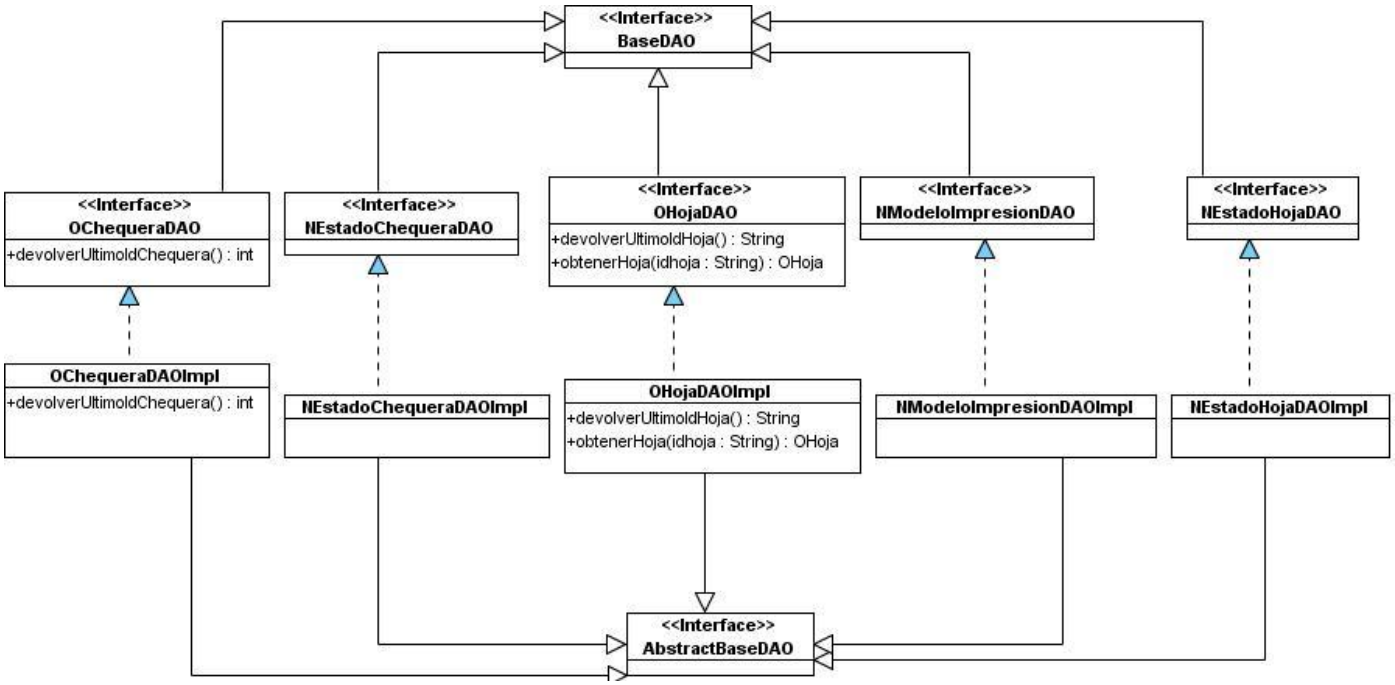

Figura 7: Diagrama de clases de Acceso a Datos del módulo Chequera.

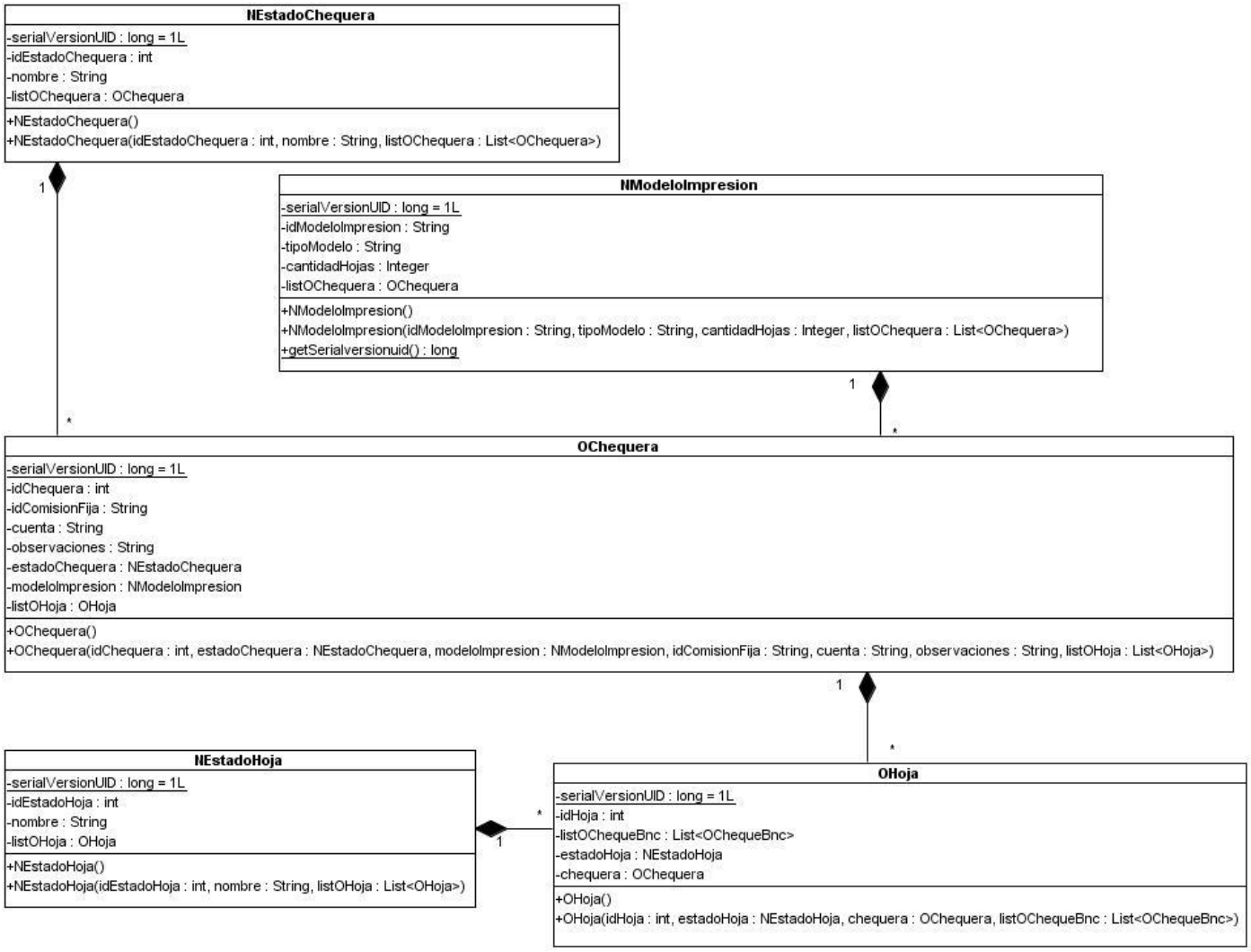

Figura 8: Diagrama de clases del Dominio del módulo Chequera.

- *BaseDAO y AbstractBaseDAO:* Interfaz y clase abstracta respectivamente, brindadas por el DAO genérico utilizado en la aplicación, ellas bridan un conjunto de funcionalidades básicas que se realizan con las clases persistentes, ya sea de buscar por un identificador, listar, persistir, actualizar y eliminar. Básicamente solo se necesita que las interfaces del acceso a datos implementen BaseDAO y las implementaciones hereden de AbstractBaseDAO.

OChequeraDAO: Interfaz encargada de brindar las funcionalidades de acceso a datos del objeto del dominio OChequera.

OChequeraDAOImpl: Clase que implementa las funcionalidades de la capa de acceso a datos definidas en OChequeraDAO.

NEstadoChequeraDAO: Interfaz encargada de brindar las funcionalidades de acceso a datos del objeto del dominio OChequera.

NEstadoChequeraDAOImpl: Clase que implementa las funcionalidades de la capa de acceso a datos definidas en OChequeraDAO.

OHojaDAO: Interfaz encargada de brindar las funcionalidades de acceso a datos del objeto del dominio OChequera.

OHojaDAOImpl: Clase que implementa las funcionalidades de la capa de acceso a datos definidas en OChequeraDAO.

NModeloImpresionDAO: Interfaz encargada de brindar las funcionalidades de acceso a datos del objeto del dominio OChequera.

NModeloImpresionDAOImpl: Clase que implementa las funcionalidades de la capa de acceso a datos definidas en OChequeraDAO.

NEstadoHojaDAO: Interfaz encargada de brindar las funcionalidades de acceso a datos del objeto del dominio OChequera.

NEstadoHojaDAOImpl: Clase que implementa las funcionalidades de la capa de acceso a datos definidas en OChequeraDAO.

2.3.1.3. Diagramas de interacción

Los diagramas de interacción son usados para representar los aspectos dinámicos del sistema, mostrando la realización de un flujo o escenario concreto de un caso de uso basado en la interacción de los objetos del diseño. Se le confiere gran importancia porque a través de los mismos se puede llegar a una gran comprensión del sistema. En este caso se realizaron los diagramas de secuencia, los cuales destacan principalmente el orden temporal de los mensajes. A continuación se presenta el diagrama de interacción para el escenario Registrar Chequera.

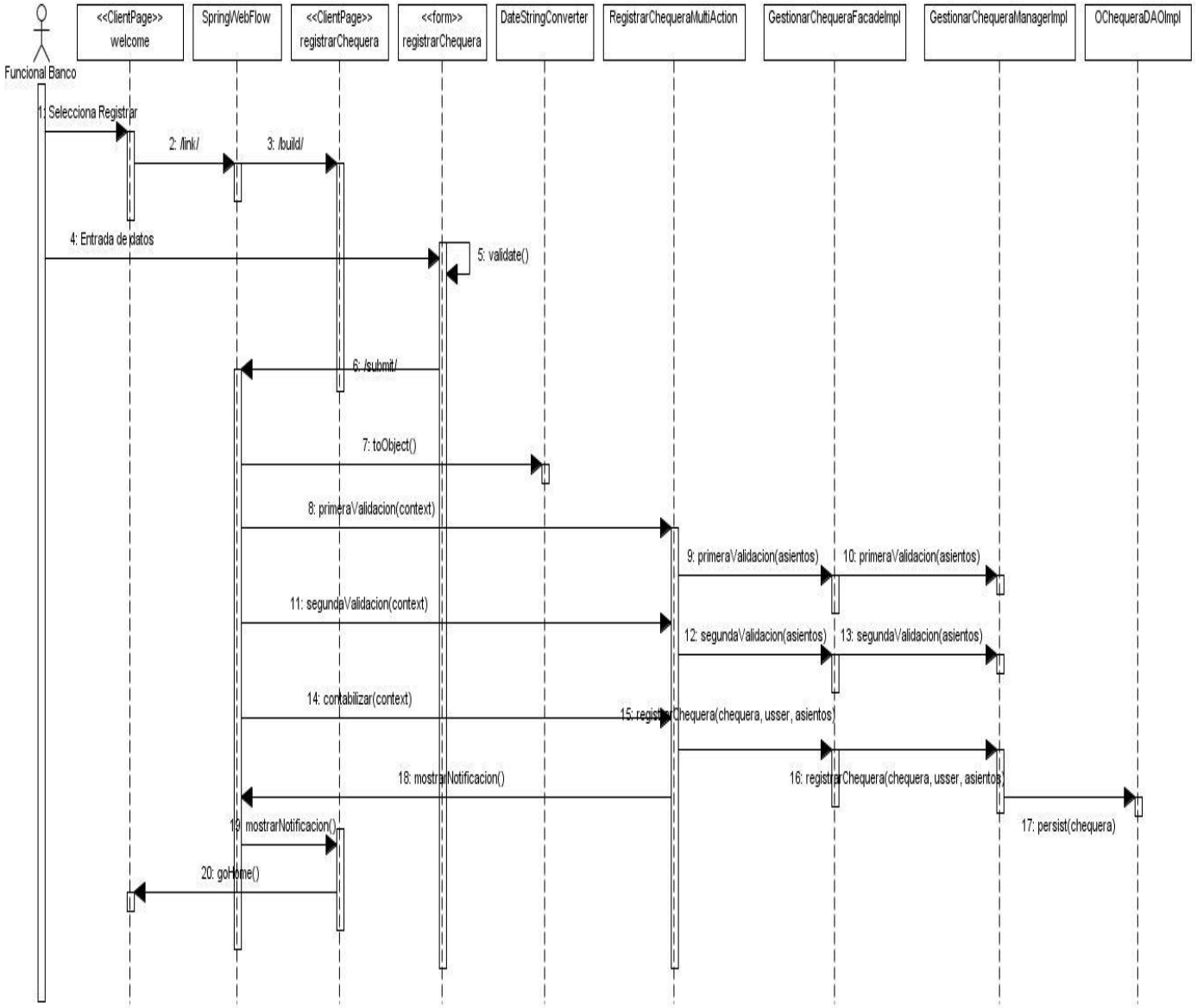

Figura 9: Diagrama de Secuencia para el esceneario Registrar Chequera.

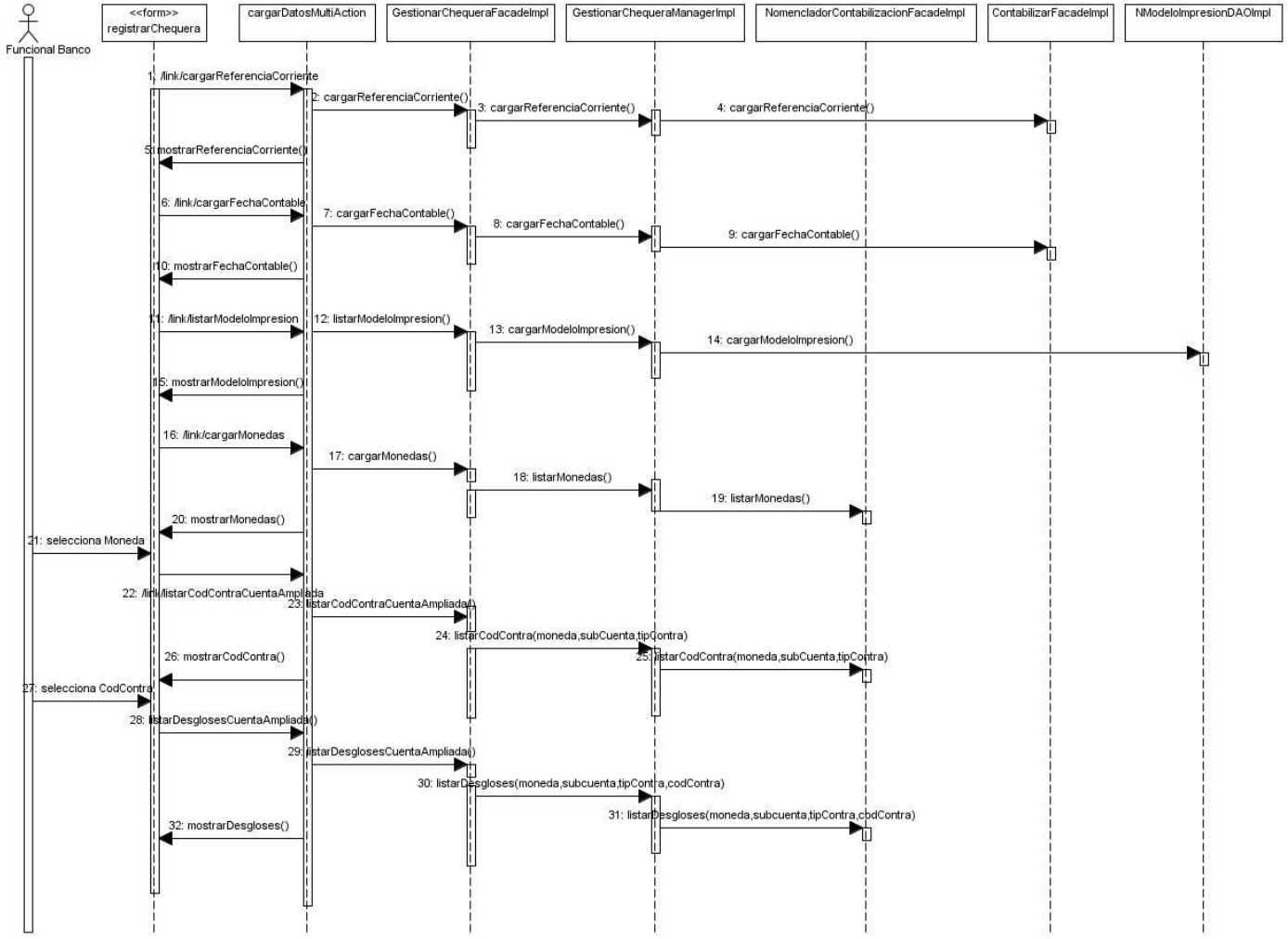

Figura 10: Diagrama de Secuencia del escenario de cargar datos en Registrar Chequera.

Descripción del flujo de sucesos.

En el diagrama correspondiente al escenario Registrar Chequera, el funcionario del banco selecciona la opción registrar en la página cliente Welcome, esta realiza una petición mediante un vínculo al motor de Spring WebFlow el cual se encarga de construir la página cliente RegistrarChequera.

El actor mediante un formulario contenido en la página introduce los datos necesarios y presiona aceptar, estos datos son validados por el mismo formulario y enviados al motor de Spring WebFlow, este a su vez hace uso del PropertyEditor DateStringConverter para convertir las fechas en objetos. Posteriormente,

debido a que este escenario contempla la contabilización se le solicita a RegistrarChequeraMultiAction una primera validación, este a su vez la solicita a GestionarChequeraFacadeImpl y pide a GestionarChequeraManagerImpl. De la misma forma se realiza una segunda validación pasando por los mismos objetos que la primera. Consecutivamente se pasa a contabilizar, SpringWebFlow hace la solicitud a RegistrarChequeraMultiAction, el cual la hace a GestionarChequeraFacadeImpl y este a GestionarChequeraManagerImpl el cual contabiliza y solicita a OChequeraDAOImpl que persista la chequera. Para finalizar se envía una notificación al actor y se redirecciona a la página cliente donde empezó el flujo.

Fue modelado un segundo diagrama para disminuir la complejidad de la comprensión del primero en el cual se representa la carga de datos del escenario Registrar Chequera. Comienza cuando Spring WebFlow construye la página cliente registrarChequera, una vez cargada esta se solicita a CargarDatosMultiAction la carga de la referencia corriente, este a su vez lo solicita a GestionarChequeraFacadeImpl, el cual lo pide a GestionarChequeraManagerImpl y este último lo solicita a ContabilizarFacadeImpl, esta clase se encuentra en el paquete Common de la aplicación, finalmente se muestran los datos. Seguidamente se pide la carga de la fecha contable del sistema, esta solicitud es atendida directamente por CargarDatosMultiAction, el cual lo solicita a GestionarChequeraFacadeImpl, esta clase lo pide a GestionarChequeraManagerImpl y este último lo solicita a ContabilizarFacadeImpl, el dato es mostrado al ser devuelto a la página RegistrarChequera.

Luego de esto se solicita la carga de los modelos de impresión, atendida directamente por CargarDatosMultiAction, este solicita dicha información a GestionarChequeraFacadeImpl, dicha clase lo pide a GestionarChequeraManagerImpl el cual obtiene los datos a partir de NModeloImpresionDAOImpl y son mostrados finalmente los datos. Luego se solicita a CargarDatosMultiAction la carga de las monedas del sistema, este interactúa con GestionarChequeraFacadeImpl para obtener los datos solicitados, para lo cual GestionarChequeraFacadeImpl interactúa con GestionarChequeraManagerImpl, este último lo hace con el nomenclador del paquete Common NomencladorFacadeImpl para obtener la información correspondiente y finalmente son mostrados los datos. En este estado el Funcionario del banco selecciona la moneda deseada y son cargados los datos asociados a los codContras[17] y a los desgloses, estas

[17] Códigos de contraparte asociados a las cuentas bancarias

peticiones son atendidas por CargarDatosMultiAction, para lo que este le solicita a GestionarChequeraFacadeImpl la información correspondiente, dicha clase lo solicita a GestionarChequeraManagerImpl y finalmente son mostrados los datos deseados.

Debido a la complejidad de los diagramas elaborados durante la etapa de diseño para responder a los requisitos funcionales, se podrá hacer consulta del resto de estos en la sección de anexos del presente trabajo.

2.3.2. Modelo de Datos.

Un modelo de datos es la representación abstracta que describe como deben ser representados los datos en el sistema. Es usado para describir la estructura de la base de datos así como los datos, las relaciones y restricciones que deben cumplirse entre ellos aportando la base conceptual para diseñar aplicaciones que hacen uso intensivo de datos.

Con el objetivo de garantizar el almacenamiento de la información persistente usada por el sistema en el módulo Chequera fue realizado el modelo de datos que cumple con las siguientes restricciones.

Las tablas *o_chequera, n_estado_chequera, o_hoja, n_estado_hoja, n_modelo_impresion* son las encargadas de responder al almacenamiento de la información relacionada con el módulo Chequera del sistema Quarxo.

Las tablas *o_chequera y n_estado_chequera* son las encargadas de almacenar la información referente a las chequeras y a los estados de las mismas.

Las tablas *o_hoja y n_estado_hoja* son las encargadas de almacenar la información referente a las hojas de impresión y a los estados de las mismas.

La tabla *n_modelo_impresion* es la encargada de almacenar la información referente a los diferentes modelos de impresión que se manejan en el módulo.

A continuación se muestra el modelo de datos correspondiente al módulo Chequera.

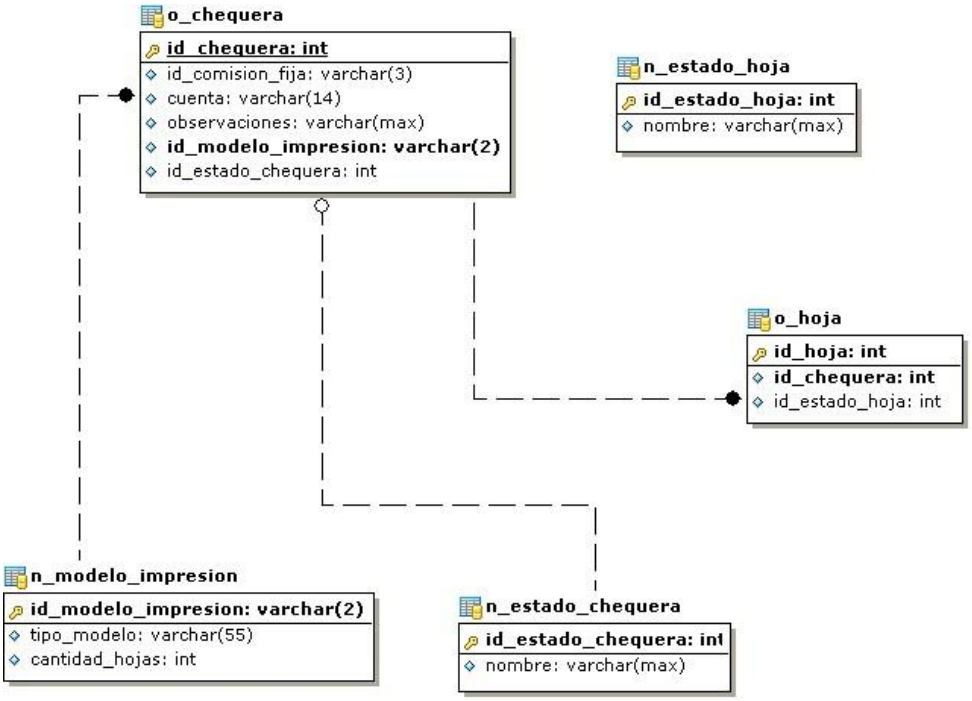

Figura 11: Modelo de datos del módulo Chequera.

En este modelo no está representada la relación entre *la tabla o_hoja* y *n_estado_hoja* físicamente, pero si a través de restricciones pues se maneja que el atributo *id_estado_hoja* en la tabla *o_hoja* representa el *id estado _hoja* de la *tabla n_estado_hoja*, esto es debido a que se está trabajando sobre la base de datos existente en el BNC, la cual no está normalizada.

2.3.3. Patrones de Diseño empleados.

El diseño fue elaborado siguiendo patrones basados en la experiencia, que de manera general constituyen soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. En este caso se emplearon los patrones GRASP, los que describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Los patrones GRASP que se utilizaron son los siguientes:

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

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

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

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

Durante el diseño del componente se emplearon patrones GOF[18], específicamente:

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

MVC: Cuando desde la vista se solicita información que se encuentra en la base de datos, esta es atendida primeramente por los Controller, luego por el Fachada y Administrador y finalmente el Objeto de Acceso a Datos, evidenciándose de esta manera la utilización de dicho patrón.

2.4. Conclusiones Parciales.

El presente capítulo fue desarrollado sobre la base de la arquitectura definida por el proyecto y los artefactos generados como resultado de la especificación de requisitos realizada correspondiente a los procesos de Chequera y Transferencia, posibilitando un diseño flexible mediante el correcto uso de

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

patrones. Después de analizada la arquitectura del proyecto Quarxo y realizado el diseño correspondiente al módulo Chequera y Transferencia, se lograron los siguientes aspectos:

- La comprensión de la estructura de paquetes que se encuentra en el mismo.
- Entendimiento del diseño realizado.

El diseño obtenido responde a los requisitos funcionales pertenecientes a dichos módulos, proporcionando una entrada apropiada como punto de partida para la implementación.

CAPÍTULO 3: IMPLEMENTACION Y PRUEBA

3.1. Introducción.

El presente capítulo está orientado a darle cumplimiento a dos de las principales tareas en el proceso de desarrollo de software. La implementación de los subsistemas Chequera y Transferencias, en la cual será representado la interacción entre los diferentes componentes mediante el diagrama de componentes, además serán especificados los estándares de codificación usados para la escritura del código y serán detallados los atributos y métodos de las clases más importantes del modelo de diseño obtenido en el capítulo anterior, seguido de esto será descrito el funcionamiento de Spring WebFlow a partir de uno de los escenarios del sistema. Luego se detallará la validación de la solución a partir de la aplicación de las pruebas de unidad con el objetivo de localizar errores en el funcionamiento del sistema que imposibiliten el cumplimiento de los requisitos funcionales definidos, siendo aplicadas además las pruebas de caja negra con el objetivo de validar la solución propuesta desde el punto de vista funcional.

3.2. Implementación.

La implementación es uno de los flujos de trabajos propuestos por RUP, en el cual se toma como punto de partida el resultado obtenido en el diseño y es implementado el sistema en términos de componentes tales como ficheros de código binario, código fuente, scripts y ejecutables, entre otros. Su importancia radica en que es obtenido como resultado un sistema ejecutable, siendo esto uno de los principales objetivos en el proceso de desarrollo del software.

3.2.1. Estándares de Codificación.

Los estándares de codificación son definidos por el equipo de desarrollo con el objetivo de lograr generalidad en la programación del sistema. Estos estándares facilitan la lectura, comprensión, mantenimiento del código y la reutilización a lo largo del proceso de desarrollo del software. Se basan en la estructura y apariencia física de un programa. La generalización de aspectos sencillos tales como el trato de mayúsculas y minúsculas ayuda a eliminar conflictos de nombres de funcionalidades implementadas con el mismo nombre guiando de forma clara el proceso de desarrollo.

3.2.1.1. Convenciones de nomenclatura.

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

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

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

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

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

Ejemplo: ChequeraCommand.

FlowHandler: En este paquete a las clases después de nombre se les coloca la palabra FlowHandler. El nombre responde a los que realiza el flujo que se quiere personalizar.

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

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

Ejemplo: GestionarChequeraFacade, GestionarChequeraFacadeImpl.

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

Ejemplo: GestionarChequeraManager y GestionarChequeraManagerImpl.

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

Ejemplo: OChequeraDAO, OChequeraDAOImpl.

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

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

3.2.2. Modelo de componentes.

Un diagrama de componentes describe los elementos físicos del sistema así como las relaciones entre ellos. Muestran además las opciones de realización que incluyen código fuente, binario y ejecutable. Los componentes representan los elementos del software que forman parte de la fabricación de aplicaciones informáticas.

El siguiente diagrama de componentes muestra las relaciones existentes entre los módulos Chequera y Transferencia con el resto de los componentes del sistema.

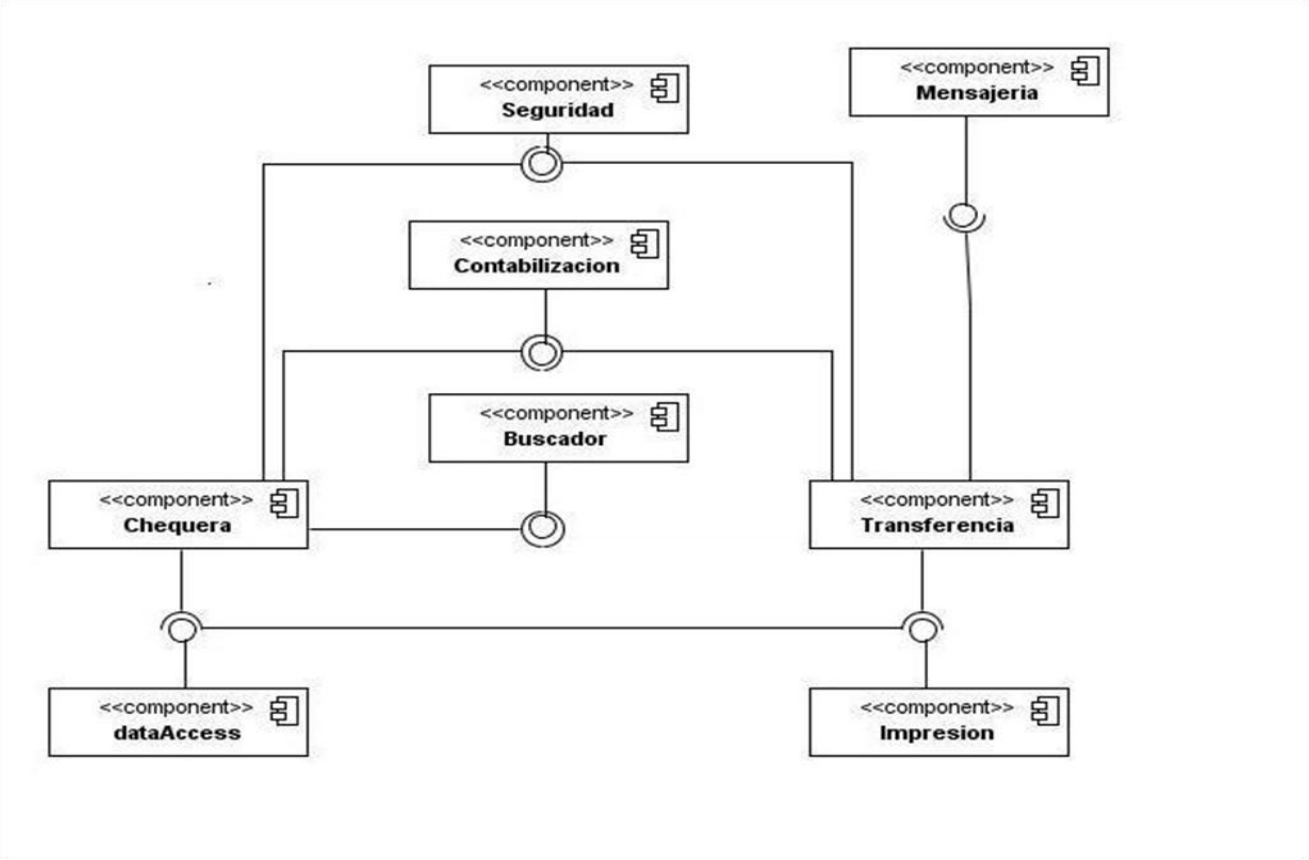

Figura 12: Diagrama de componentes del módulo Chequera.

Seguidamente se explica de manera general cada uno de los componentes que se relacionan con los módulos Chequera y Transferencia.

El componente *Buscador* ofrece un conjunto de clases que constituyen el motor de búsqueda, presentando una interfaz gráfica mediante la cual se solicitan los diferentes conceptos en dependencia del módulo donde se emplee. Para la correcta gestión de la información en el módulo Chequera se hace necesario el uso de este componente ya que varias funcionalidades dependen de obtener una chequera primeramente.

El componente *Contabilización* se propone un grupo de clases necesarias para realizar la contabilización de determinadas operaciones que así lo requieran, además posee otro grupo de clases y una interfaz gráfica para el cobro de comisiones que se asocien a determinada operación. La realización

de los módulos Chequera y Transferencia necesitan interactuar con este componente ya que en ambos casos se llevan a cabo acciones que traen consigo el cobro de comisiones.

El componente *Mensajería* es el encargado de conceder las clases necesarias para el envió de mensajes SWIFT tras la realización de operaciones bancarias que requieran la notificación a los bancos implicados en ella. El módulo Transferencia interactúa con este componente al llevar a cabo el envío de mensajes en el caso de realizar una transferencia bancaria de tipo *enviada*.

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

El componente *Impresión* tiene como función brindar las clases que contienen las funcionalidades mediante las cuales es posible imprimir determinada información y de forma automática las operaciones que realicen contabilización. Específicamente el módulo Chequera interactúa directamente con este componente al llevar a cabo la impresión de las hojas de cheque.

El componente *DataAccess* tiene la función de brindar todos los elementos necesarios para llevar a cabo la conexión a la base de datos y por ende todos los subsistemas dependen de él para poder realizar las funcionalidades que engloban.

3.2.3. Descripción de clases y funcionalidades.

Basado en el diseño de las clases perteneciente al módulo Chequera realizado en el capitulo anterior, a continuación se describirán los atributos y métodos de las que se consideran más importantes desde el punto de vista funcional asociadas a este módulo.

Tabla 1: Descripción de la clase ContabilizarMultiAction

Nombre: ContabilizarMultiAction	
Tipo de clase: Controller	
Atributo	**Tipo**

contabilizarFacade	ContabilizarFacade
globalFacade	GlobalFacade
Para cada responsabilidad:	
Nombre:	**Descripción**:
validacionContabilizar(context: RequestContext)	Permite usar la primera y segunda validación de la contabilización indicándole al flujo la existencia de algún error o alerta en caso de que exista.
contabilizar(context : RequestContext) :	Permite la contabilización de los asientos, en caso de existir algún error o alerta en este proceso se le informa al flujo.
construirAsientos(context : RequestContext)	Construye los asientos contables de la operación que se está realizando.

Tabla 2: Descripción de la clase ImprimirConformidadMultiAction

Nombre: ImprimirConformidadMultiAction	
Tipo de clase: Controller	
Atributo	**Tipo**
gestionarChequeraFacade	GestionarChequeraFacade
Para cada responsabilidad:	
Nombre:	**Descripción**:

imprimirConformidad(context : RequestContext)	Permite imprimir los datos asociados a una chequera.

Tabla 3: Descripción de la clase VerificarHojasDeChequesMultiAction.

Nombre: VerificarHojasDeChequesMultiAction	
Tipo de clase: Controller	
Atributo	**Tipo**
gestionarChequeraFacade	GestionarChequeraFacade
Nombre:	**Descripción**:
Para cada responsabilidad:	
persistirChequesBNC(contex: RequestContext)	Persisten los cheques asociados a la chequera y la hoja que se maneja en el contexto.
imprimir(context : RequestContext)	Imprime los datos de la Hoja que se maneja en el contexto.
actualizarGridHojas(context : RequestContext)	Permite obtener los datos de las hojas para actualizar la vista.
devolverListChequesDeHoja(idHoja : String, hojaCommand : HojaCommand)	Devuelve el listado de los cheques correspondientes a la hoja especificada.
persistirAll(context : RequestContext)	Permite persistir todos los datos que se manejan en el flujo asociados a cheques y hojas de la chequera.

Tabla 4: Descripción de la clase RegistrarChequeraMultiAction.

Nombre: RegistrarChequeraMultiAction	
Tipo de clase: Controller	
Atributo	**Tipo**
gestionarChequeraFacade	GestionarChequeraFacade
Para cada responsabilidad:	
Nombre:	**Descripción**:
conformarChequera(contex : RequestContext)	Permite conformar los datos de la chequera a partir de los datos que se manejan en el flujo.
enviarDatosFlow(contex : RequestContext, chequeraComand : ChequeraComand)	Envía los datos de la chequera al context del flujo.
persistirChequera(contex : RequestContext)	Persisten los datos de la chequera.
construirAsientos(contex : RequestContext)	Construye los asientos correspondientes a la operación de crear la chequera.
primeraValidacion(contex : RequestContext)	Permite invocar la primera validación de los asientos, en caso de existir algún error o alerta indicárselo al flujo.
segundaValidacion(contex : RequestContext)	Es invocada una segunda validación de los asientos y en caso de algún error se le informa al flujo.
contabilizar(contex : RequestContext)	Son contabilizados los asientos asociados a

	la operación en curso y en caso de existir algún error es informado al flujo.

Tabla 5: Descripción de la clase GestionarChequeraManagerImpl.

Nombre: GestionarChequeraManagerImpl	
Tipo de clase: Manager	
Atributo	**Tipo**
globalFacade	GlobalFacade
contabilizarFacade	ContabilizarFacade
nomencladorContabilizacionFacade	NomencladorContabilizacionFacade
chequeDAO	ChequeDAO
estadoChequeDAO	EstadoChequeDAO
tipoChequeDAO	TipoChequeDAO
banNacDAO	BanNacDAO
ssp_Cheque	ChequeDAOStoreProcedure
digitoChequeoDAOStoreProcedure	DigitoChequeoDAOStoreProcedure
hojaDAO	OHojaDAO
estadoChequeraDAO	NEstadoChequeraDAO
estadoHojaDAO	NEstadoHojaDAO
chequeraDAO	OChequeraDAO
modeloimpresionDAO	NModeloImpresionDAO
Para cada responsabilidad:	

Nombre:	**Descripción**:
persistirHoja(hoja : OHoja)	Persiste la información de una Hoja en la BD.
persistirChequera2(chequera : OChequera)	Persiste la información de una Chequera en la BD.
obtenerChequeporCB(cb : String)	Obtiene un cheque a partir de un código de barra.
consultarChequera(idchequera : int)	Devuelve los datos de la chequera con el ID especificado.
actualizarChequera(chequera : OChequera)	Actualiza los datos de una Chequera.
listarHojasPorChequera(idChequera : int)	Devuelve un listado con las Hojas correspondientes a la Chequera con el ID especificado.
listarModeloImpresion()	Devuelve un listado con los modelos de impresión que existen en la BD.
listarChequesPorHojas(idhoja : String)	Devuelve un listado con los Cheques correspondientes a una Hoja con el ID especificado.
obtenerCodigoBarra(cuentaemisor : String, tipoCheque : int, codBanco : String)	Devuelve el código de barra correspondiente a los datos especificados.
obtenerNroCheque() : String	Obtiene el último número de cheque de la base de datos.
obtenerMdaChequera(código : String)	Devuelve la moneda con el código de moneda especificado.
obtenerReporteConformidad(chequera: ChequeraComand) :	Son transformados los datos de la chequera especificada para ser usados en la impresión

	de la misma.
obtenerSigla(codMon : String)	Devuelve las siglas de la moneda con el código especificado.
obtenerFecha() : String	Obtiene la fecha contable del sistema.
obtenerModeloImpresion(idmodelo : String)	Devuelve los datos correspondientes al modelo de impresión con el Id especificado.
obtenerEstadoChequera(idestado : String)	Devuelve los datos de un Estado de Chequera con el ID especificado.
obtenerTipoCheque(modeloimpresión : String)	Obtiene el tipo de cheque según el modelo de impresión usado.
crearChequeBNC(chequera : OChequera, idhoja : String)	Crea un cheque a partir de una chequera y el id de la hoja en la que se encontrara.
persistirChequesBNC2(cheque : OChequeBnc)	Persiste los datos correspondientes a un Cheque del BNC.
obtenerHojabyID(idhoja : String)	Se obtienen los datos asociados a una Hoja con el ID especificado.
obtenerNombreCliente(id : String)	Devuelve el nombre del cliente dado un ID.
contabilizarChequera(chequera:ChequeraComand)	Devuelve los asientos de la contabilización correspondiente a la chequera.

La clase que se explica a continuación posee los métodos elementales de acceso a datos como adicionar, eliminar, buscar, modificar, listar y buscar por el ID de la entidad Chequera. Debido a esto solo se representaran las clases de acceso a datos en las cuales se implementen otros métodos que no sean los mencionados.

Tabla 6: Descripción de la clase OChequeraDAO

Nombre: OChequeraDAO

Tipo de clase: Acceso a Datos	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**Descripción**:
devolverUltimoIdChequera()	Devuelve el id de la ultima chequera persistida en la base de datos.
devolverUltimoIdHoja() : String	Devuelve el id de la ultima hoja persistida en la base de datos.
obtenerHoja(idhoja : String)	Devuelve una Hoja de la base de datos con el identificador definido por parámetros.

Tabla 7: Descripción de la clase OHojaDAO.

Nombre: OHojaDAO	
Tipo de clase: Acceso a Datos	
Atributo	**Tipo**
Para cada responsabilidad:	
Nombre:	**Descripción**:
devolverUltimoIdHoja() : String	Devuelve el id de la ultima hoja persistida en la base de datos.
obtenerHoja(idhoja : String)	Devuelve una Hoja de la base de datos con el identificador definido por parámetros.

3.2.4. Aspectos principales de la implementación.

A lo largo de este trabajo está presente la utilización de Spring WebFlow debido a las facilidades que brinda para el desarrollo, fundamentalmente en escenarios donde existen flujos complejos de interacción entre sus vistas o la información que manejen. Para el desarrollo de los módulos Chequera y Transferencia es usado este framework en todas sus funcionalidades debido a la necesidad de interactuar con otros componentes como Contabilización.

A continuación es descrito el funcionamiento de este framework desde el módulo Chequera en la funcionalidad Registrar Chequera. Todas las funcionalidades que están presentes en el flujo fueron implementadas en la clase RegistrarChequeraMultiAction, la cual fue explicada con anterioridad. La llamada a los métodos implementados no especifica los parámetros correspondientes ya que si reciben únicamente el RequestContext como parámetro Spring WebFlow es capaz de reconocerlo.

Para la funcionalidad implementada se inicia la declaración del flujo de la siguiente manera:

```
<var name="chequeraComand"
    class="cu.uci.finixubnc.titulosvalores.chequera.web.webflow.command.ChequeraComand" />
```

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

Primeramente se inicia declarando una variable, la cual será representativa de la clase modelo perteneciente a la información que se maneje, en este caso es la variable chequeraCommand perteneciente al modelo ChequeraCommand, el cual será modificado en la vista para luego persistirlo.

La llamada al método *registrarEntradaFlujo* en la etiqueta *on-start* establece la variable chequeraCommand en el contexto del flujo.

Este flujo se realiza a partir de la vista *registrarChequera***,** al mismo tiempo son especificados un grupo de transacciones que serán ejecutadas a partir de los diferentes eventos que pueden ser generados por el usuario.

```
<view-state id="registrarChequera" model="chequeraComand" view="registrarChequera">
    <transition on="aceptar" to="primeraValidacion" />
    <transition on="cancelReg" to="endReg" />
    <transition on="cancelV" to="endReg" />
    <transition on="aceptarM" to="contabilizar"/>
    <transition on="cancelarM" to="registrarChequera"/>
    <transition on="verAsientos" to="verAsientosState"/>
    <transition on="actualizarDatosComisiones" validate="false" bind="false" to="registrarChequera">
        <evaluate expression="comisionesMultiAction.actualizarDatosComisiones" />
    </transition>
    <transition on="gestionarComision" validate="false" to="gestionarComisionSubflow">
        <evaluate expression="registrarChequeraMultiAction.limpiarMsgContabilizar" />
        <evaluate expression="registrarChequeraMultiAction.refrescarDesdeSesion" />
        <evaluate expression="comisionesMultiAction.registrarCallMetodo" />
    </transition>
</view-state>
```

Desde esta vista se pueden generar eventos tales como *aceptar* en el cual se validan los asientos pertenecientes a la chequera haciendo uso de las funcionalidades implementadas en RegistrarChequeraMultiAction.

```
<action-state id="primeraValidacion">
    <evaluate expression="registrarChequeraMultiAction.primeraValidacion" />
    <transition on="success" to="resultadoPrimeraValidacion" />
</action-state>

<decision-state id="resultadoPrimeraValidacion">
    <if test="flowScope.codigoMensaje=='m0'" then="segundavalidacion"
        else="registrarChequera" />
</decision-state>
```

.

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

<decision-state id="resultadoSegundaValidacion">
    <if test="flowScope.codigoMensaje=='m0'" then="contabilizar"
        else="registrarChequera" />
</decision-state>
```

En el caso de existir algún error en la primera validación es cancelado el proceso y se vuelve a la página inicial. En el caso de que ocurra en la segunda validación se muestra un mensaje a partir del cual pueden ser generados dos eventos. En el case de *cancelarM* se oculta el mensaje de error y se queda en la propia vista permitiendo realizar los cambios deseados en los datos. En el caso del evento *aceptarM* se contabiliza sin tener en cuenta el error mostrado al usuario, finalizando el flujo y quedando en la propia vista brindando la opción de volver a registrar otra chequera.

En el caso del evento *verAsientos* se mostraría una vista con los asientos creados y sus cuentas asociadas para la futura contabilización.

También se puede generar el evento *cancelReg* el cual pasara al *end-State* del flujo y llevando la vista hasta la pagina principal del subsistema.

3.3. Validación de la implementación de los módulos Chequera y Transferencia.

La calidad de un producto de software es el indicador que permite determinar si los procesos de construcción de software fueron apropiados, por lo que se define por un conjunto de cualidades que permiten caracterizar dicho producto y determinar su existencia y utilidad, siendo sinónimo de eficiencia, corrección, flexibilidad, portabilidad, usabilidad y seguridad. La calidad del software puede medirse después de elaborado el producto, pero esto puede resultar muy costoso si se detectan problemas derivados de imperfecciones en el diseño, por lo que es imprescindible tener en cuenta tanto la obtención de la calidad así como su control durante todas las etapas del ciclo de vida del software. Durante la implementación existe la posibilidad de que ocurran errores, por lo que se hace necesario detectar a tiempo los mismos con objetivos de eliminarlos y evitar afectaciones en etapas venideras.

La actividad fundamental para esta etapa es determinar a partir de pruebas realizadas al sistema en qué medida los módulos Chequera y Transferencia cumplen con las expectativas del cliente partiendo de las restricciones y los requisitos establecidos inicialmente. Para llevar a cabo esta tarea se tiene como objetivos fundamentales:

- Verificar la implementación de los módulos Chequera y Transferencia.
- Verificar la implementación de manera correcta de los requisitos establecidos previamente.
- Identificar los errores existentes, llevando a cabo la corrección de los mismos de manera eficiente.

3.3.1 Pruebas de Software.

Las pruebas de software son los procesos que permiten comprobar la calidad de un producto de software. Son llevadas a cabo con el objetivo de determinar posibles fallos de implementación, calidad o usabilidad de un software.

El sistema Quarxo fue probado desde el punto de vista funcional garantizando el cumplimiento de los requisitos funcionales definidos inicialmente.

3.3.1.1 Aplicación de las pruebas de Caja Blanca o Estructural

Esta prueba consiste específicamente en diseñar Casos de prueba[19] atendiendo al comportamiento interno del sistema y a la estructura del programa, concentrándose en la lógica interna del sistema sin considerar los aspectos de rendimiento. Asociadas a este método existen cuatro técnicas de prueba; Condición, Flujo de Datos, Bucles y Camino Básico, de las cuales será aplicada esta última, debido a que permite obtener una medida de la complejidad lógica del diseño y usar la misma como guía para la definición de un conjunto de caminos básicos, garantizando que durante la prueba se ejecute al menos una vez cada sentencia del programa.

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

A continuación se analizan y enumeran las sentencias de código de uno de los métodos contenidos en la clase GestionarChequeraManagerImpl, específicamente: contabilizarChequera, el cual construye una lista de asientos que posteriormente serán usados en la contabilización.

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

```
public List<Asiento> contabilizarChequera(ChequeraComand chequera) {

    List<Asiento> asientos = new ArrayList<Asiento>();                    // 1
    List<Asiento> listaAux = new ArrayList<Asiento>();                    // 1
    if (chequera.getLoteComisionesComand() != null) {                     // 2
        NError error = contabilizarFacade.construirAsientosComisiones(   // 3
                chequera.getLoteComisionesComand(), chequera              // 3
                        .getReferenciaCorriente(), chequera               // 3
                        .getReferenciaOriginal(), "", "05012A", chequera// 3
                        .getFechaContable(), chequera.getFechaValor(),    // 3
                listaAux);                                                // 3
        if (error.getCodigo() == 0) {                                     // 4
            asientos.addAll(listaAux);                                    // 5
        }                                                                 // 6
    }                                                                     // 7

    return asientos;                                                      // 8
}
```

Figura 13: Método contabilizarChequera.

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

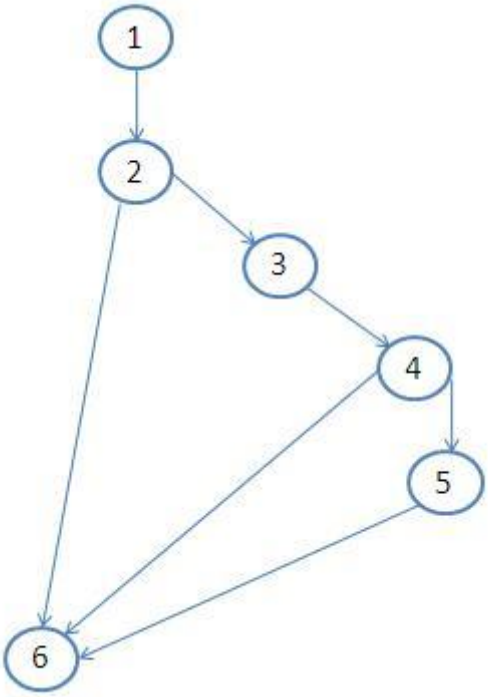

Figura 14: Grafo de flujo asociado al algoritmo contabilizarChequera.

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

1. V (G) = (A – N) + 2

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

V (G) = (7 - 6) + 2

V (G) = 3.

2. V (G) = P + 1

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

V (G) = 2 + 1

V (G) = 4.

3. V (G) = R

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

V (G) = 3

El cálculo efectuado mediante las fórmulas antes presentadas muestran una complejidad ciclomática de valor 3, de manera que existen siete posibles caminos por donde el flujo puede circular, este valor representa el número mínimo de casos de pruebas para el procedimiento tratado.

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

Camino básico #1: 1 – 2 -- 6
Camino básico #2: 1 – 2 – 3 – 4 -- 6
Camino básico #3: 1 – 2 – 3 – 4 – 5 – 6
Se procede a ejecutar los casos de pruebas para cada uno de los caminos básicos determinados en el grafo de flujo. Para definir los casos de prueba es necesario tener en cuenta:

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

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

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

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

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

Descripción: Se deben determinar los asientos para contabilizar en caso de que se seleccione la comisión a pagar.

Condición de ejecución: La chequera debe tener todos sus datos.

Entrada:
Los datos de la chequera, no se selecciona la opción de cobrar comisiones.
Resultados esperados: Teniendo en cuenta que no fue seleccionado la opción de cobrar comisiones la lista de asientos contables debe devolverse vacía.
El resultado obtenido fue correcto.

2- Caso de prueba para el camino básico # 2.

Descripción: Se deben determinar los asientos para contabilizar en caso de que se seleccione la comisión a pagar.

Condición de ejecución: La chequera debe tener todos sus datos. Se debe seleccionar la opción de cobrar comisiones, y la cuenta a la que se le cobrara comisiones debe tener crédito disponible, así como no debe estar cancelada

Entrada:
Los datos de la chequera, Lote de comisiones con cuenta asociadas con saldo disponibles. (CUC32100260800).
Resultados esperados: Se debe generar un listado de asientos contables asociados a las cuentas que se le cobraran las comisiones.
El resultado obtenido fue correcto.

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

Descripción: Se deben determinar los asientos para contabilizar en caso de que se seleccionaran las comisiones a pagar.

Condición de ejecución: La chequera debe tener todos sus datos. Se debe seleccionar la opción de cobrar comisiones, y la cuenta a la que se le realiza el débito de la comisión no debe tener saldo disponible o debe estar cancelada.

Entrada:
Los datos de la chequera, lote de comisiones con cuentas asociadas sin saldos disponibles o deben estar canceladas. (CUC32100000500).
Resultados esperados: No debe ser generado el listado de asientos a contabilizar por existir un error en el proceso de crear los asientos contables.
El resultado obtenido fue correcto.

3.3.1.2 Aplicación de las pruebas de Caja Negra o Funcional.

Estas pruebas se realizan sobre la interfaz del usuario, se centran fundamentalmente en la parte funcional del software, o sea, los casos de prueba van orientados a demostrar que las funciones del sistema son operativas, las mismas aceptan los valores de entradas produciendo una salida correcta sin tener en cuenta lo que pueda estar haciendo el software internamente.

Las pruebas de caja negra no constituyen una alternativa a las pruebas de Caja Blanca, son enfocadas de manera complementaria con el objetivo de identificar errores de tipos diferentes a los encontrados en estas.

Con las pruebas de Caja negra se intenta encontrar errores de los siguientes tipos:

- Funciones incorrectas o ausentes.
- Errores de Interfaz.
- Errores en estructuras de datos o en acceso a las bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Para llevar a cabo correctamente este tipo de pruebas existen diferentes técnicas:

- *Partición de Equivalencia:* Divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- *Análisis de Valores Límites:* Prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
- *Grafos de Causa-Efecto:* Permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

Para llevar a cabo la realización de las pruebas de Caja Negra se empleo la técnica Partición de Equivalencia, la cual constituye una de las más efectivas pues permite examinar los valores válidos e inválidos de las entradas que existen en el software, descubriendo de forma inmediata una clase de errores que de otro modo requerirían que se ejecutaran varios casos de prueba antes de detectar el error genérico. Este tipo de pruebas dirige la definición de casos de pruebas que descubran clases de errores, reduciendo el número de clases de prueba que hay que desarrollar.

A continuación se muestra el caso de prueba aplicado a la funcionalidad *Registrar Chequera,* en el cual las celdas de la tabla contienen que V, I, o N/A. V indica válido, I indica inválido, y N/A no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante.
En este caso de prueba se tiene como condiciones de ejecución las siguientes:

- El usuario debe estar autenticado con los permisos necesarios para realizar la acción.
- Debe existir conexión con la Base de Datos.
- Registrar.
- El Cheque que se genera del registro y confección de la Chequera queda con estado “Circulando”.

En este caso de prueba se ejecutara la sección Registrar Chequera, para la cual se tendrán en cuenta variables que se mencionan a continuación con sus respectivas características.

Tabla 8: Descripción de variables.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción

Número de cuenta	Número de cuenta	Campo de texto	No.	Se selecciona una cuenta de Cliente de
Modelo de impresión	Modelo de Impresión	Lista desplegable	No.	Se selecciona el tipo de modelo de Cheque que
Comisión	Comisiones Bancarias	Selección	No	Se selecciona la comisión a cobrar.

Para esta sección del caso de prueba se manejo la siguiente matriz de datos.

Tabla 8: Matriz de datos para el caso de prueba.

Escenario	Modelo de Impresión	Comisión	Respuesta del Sistema	Resultado de la Prueba	Flujo Central

Escenario	Modelo de Impresión	Comisión	Respuesta del Sistema	Resultado de la Prueba	Flujo Central
EC 1.1: Registrar Chequera	Modelo de cheque no minal	Fija	Que la chequera quede registrada en el sistema.	Satisfactorio.	1-El usuario introduce la URL en el navegador. 2- Selecciona la opción Ingresar. 3-Selecciona el subsistema Títulos valores. 4-En el menú principal selecciona la opción Gestionar Chequera. 5-Selecciona la opción Registrar. 6- El sistema muestra los campos. 7-El usuario introduce los valores y selecciona Aceptar. 8-El sistema valida los datos, registra la Chequera y muestra el mensaje: “Operación realizada satisfactoriamente”. 9-El usuario selecciona Aceptar. 10-El sistema va al menú principal.

Escenario	Modelo de Impresión	Comisión	Respuesta del Sistema	Resultado de la Prueba	Flujo Central
EC 1.1: Registrar Chequera	vacío Inválido	vacío Inválido	Que la chequera quede registrada en el sistema.	Satisfactorio.	1-El usuario introduce la URL en el navegador. 2- Selecciona la opción Ingresar. 3-Selecciona el subsistema Títulos valores. 4-En el menú principal selecciona la opción Gestionar Chequera. 5-Selecciona la opción Buscar. 6-Introduce el o los criterios de búsqueda, o da clic en el botón Buscar. 6.2 En caso de que el usuario no llene los campos seleccionados el sistema muestra el mensaje 'El campo es obligatorio.' 7-El sistema muestra las Chequeras que coinciden con el criterio especificado. 8-El usuario selecciona Aceptar.

En la sección de los anexos se encuentra el resto de los diseños de casos de pruebas para la validación de las funcionalidades especificadas en los requisitos detectados. Los resultados obtenidos al aplicar los diseños de pruebas especificados anteriormente a los módulos Chequera y Transferencia como parte de las pruebas internas realizadas fueron satisfactorios desde el punto de vista funcional, atendiendo al correcto funcionamiento del mismo ante diferentes situaciones.

Los módulos Chequera y Transferencias del sistema Quarxo, una vez probados y liberados por parte del departamento de calidad perteneciente a la universidad (CALISOFT), entraron a las pruebas de aceptación correspondientes en el BNC, en el cual fueron aprobados por parte del cliente por lo que actualmente se encuentran en completa capacidad para su explotación como parte del sistema.

3.4 Conclusiones Parciales.

Este capítulo centra sus elementos fundamentales en la implementación y validación de los módulos Chequera y Transferencia teniendo como premisa la calidad de la solución propuesta. En este sentido fueron abordados temas tales como los estándares de codificación empleados en el desarrollo además de ser especificadas las relaciones de estos módulos con los componentes generales del proyecto.

Relacionado a la calidad de la solución fueron realizadas pruebas de software en el nivel de unidad mediante casos de pruebas, las cuales fueron orientadas a comprobar el correcto funcionamiento del sistema para los diferentes requisitos funcionales definidos inicialmente.

De manera general se obtienen los módulos Chequera y Transferencia, los cuales cumplen desde el punto de vista funcional con los requisitos funcionales definidos a partir de las expectativas y las necesidades del cliente.

CONCLUSIONES

Con el desarrollo del trabajo de diploma se cumplieron con los objetivos propuestos:

- La caracterización de la metodología, lenguajes, tecnologías y herramientas definidas para el desarrollo del sistema Quarxo y la valoración de sistemas contables nacionales e internacionales asociados a los procesos de Chequeras y Transferencias , establecieron las bases teóricas para darle solución al problema planteado.

- La concepción de los artefactos dentro del desarrollo de los flujos de Diseño e Implementación a partir de las funcionalidades definidas previamente, permitió obtener una solución para gestionar las Chequeras y las Transferencias dentro del sistema Quarxo.

- La validación del diseño mediante la aplicación de las pruebas realizadas por el equipo de desarrollo del proyecto SAGEB y la liberación del mismo por Calisoft, demostró que la solución del subsistema Vencimiento cumple con los requerimientos, la eficiencia y la estabilidad necesaria para ser desplegado en el Banco Nacional de Cuba.

RECOMENDACIONES

Se recomienda aprovechar los estudios realizados para mejorar la solución presentada en futuras versiones del sistema además de incrementar el uso de las tecnologías y herramientas en el desarrollo de otros sistemas de gestión bancaria.

BIBLIOGRAFÍA

COMUNITY, JBOSS. 2001. HIBERNATE. [En línea] 2001. [Citado el: 27 de 3 de 2011.] http://www.hibernate.org/.

CréditosPERÚ. [En línea] [Citado el: 27 de 3 de 2011.] http://www.creditosperu.com.pe/gltransferenciabancaria.php.

Gutierrez, Javier J. ¿Qué es un framework web? [En línea] [Citado el: 27 de 3 de 2011.] http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf.

INC, CollabNet. SUBVERSION. [En línea] 2000. [Citado el: 27 de 3 de 2011.] http://subversion.apache.org/.

IvAr Jacobson, J Rumbaugh. *El Lenguaje Unificado de Modelado Manuel de Referencia.* s.l. : Addison Wesley.

Jacobson, Ivar. 2000. *El proceso Unificado de Desarrollo De SoftWare.* Madrid : Addison Welsley, 2000.

Lara, Rafael Bello. 2010. *Diseño e Implementación del Subsistema Cartas de Créditos Del Proyecto SAGEB.* Cuidad Habana : s.n., 2010.

LLORENTE, M. D. L. Á. 2002. Los Bancos como intermediarios financieros. [En línea] 2002. [Citado el: 27 de 3 de 2011.] http://www.eleconomista.cubaweb.cu/2002/nro153/153_267.html..

Microsoft. 2011. Microsoft SQL SERVER. [En línea] 2011. [Citado el: 27 de 3 de 2011.] http://www.microsoft.com/spain/sql/productinfo/overview/default.mspx.

Muñoz, Javier Martinez. 2008. *Definición De Los Requerimientos Funcionales Del Módulo Cuentas de Clientes y Órdenes de Pago Inmediato Del Proyecto Banco Nacional.* Cuidad Habana : s.n., 2008.

NÚÑEZ, JUAN MANUEL BARRIOS. 2003. *INVESTIGACIÓN DE LA PLATAFORMA J2EE Y SU APLICACIÓN PRÁCTICA.* Chile : s.n., 2003.

Rodríguez, Yoan Antonio López. 2008. *DEFINICIÓN DE LOS REQUERIMIENTOS FUNCIONALES DEL MÓDULO TESORERÍA, PRÉSTAMOS Y DEPÓSITOS DEL PROYECTO BANCO NACIONAL.* Cuidad Habana : s.n., 2008.

RODRÍGUEZ, YOAN ANTONIO LÖPEZ. 2008. *DEFINICIÓN DE LOS REQUERIMIENTOS FUNCIONALES DEL MÓDULO TESORERÍA, PRÉSTAMOS Y DEPÓSITOS DEL PROYECTO BANCO NACIONAL.* CUIDAD HABANA : s.n., 2008.

SARDIÑA, L.B.M. 1998. *Principales características del Sistema Contable del Banco Central de Cuba.* 1998.

Sun . 2011. Apache Tomcat. [En línea] 2011. [Citado el: 27 de 3 de 2011.] http://tomcat.apache.org/.

Visual Paradigm For Uml - Presentation Transcript. [En línea] [Citado el: 26 de 3 de 2011.] http://www.slideshare.net/vanquishdarkenigma/visual-paradigm-for-uml..

Zanchez, Juan y Guilarte, Calero. 2002. La cuenta Corriente y la Transferencia Bancaria. [En línea] 2002. [Citado el: 27 de 3 de 2001.] http://eprints.ucm.es/6428/1/9.Cuenta_corriente.pdf.

GLOSARIO DE TÉRMINOS

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

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

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

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

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

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

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

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

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

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

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

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

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

ANEXOS

Anexo 1: Clases del diagrama de negocio del módulo Chequera.

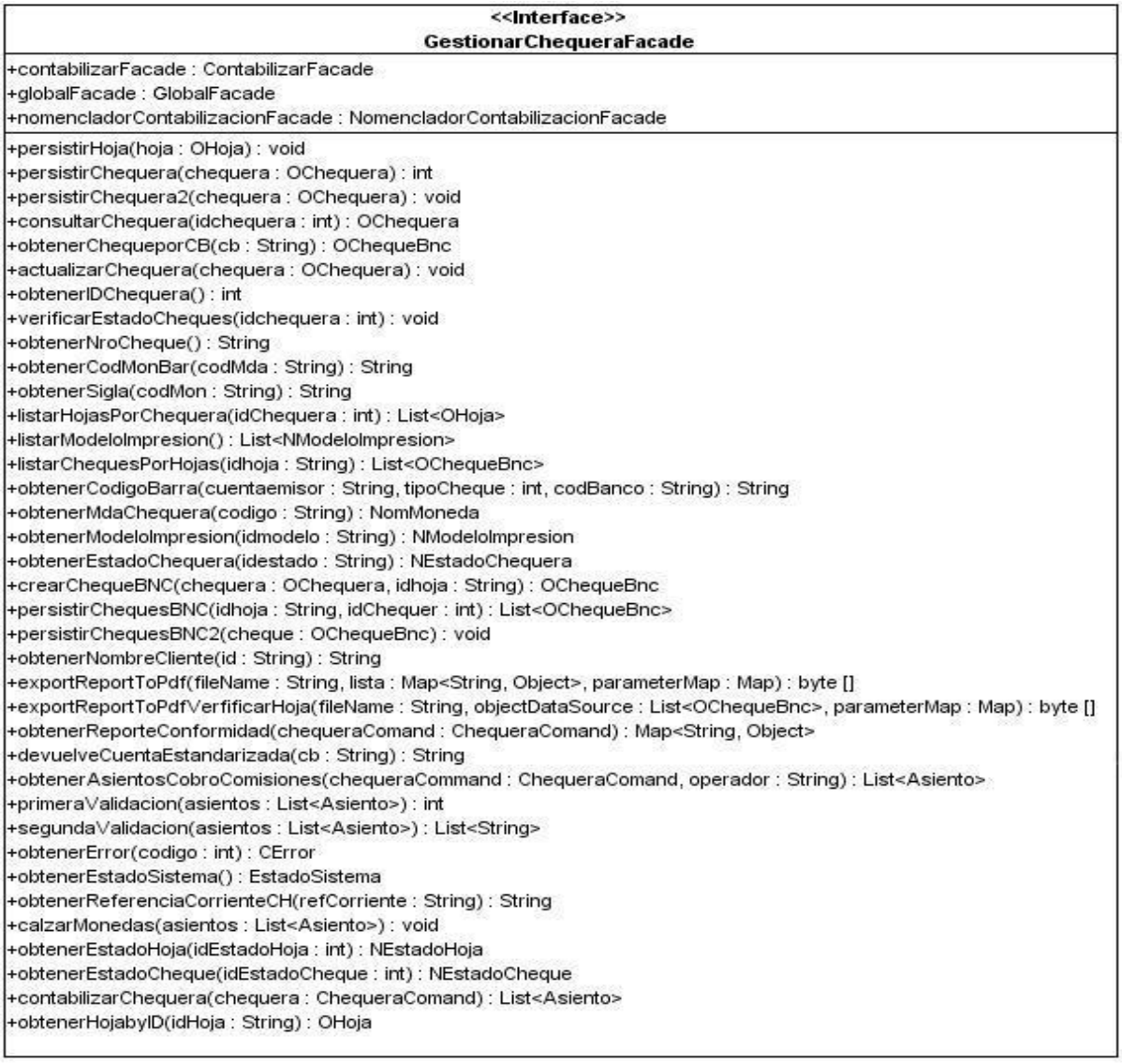

Figura 15: Funcionalidades definidas en la clase GestionarChequeraFacade

Figura 16: Funcionalidades definidas en la clase GestionarChequeraManager.