Universidad de las Ciencias Informáticas

Título: Diseño e implementación de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio del sistema Quarxo.

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

Autor: Yovani Royero Mena
Tutor: Ing. Yulier Matías León

La Habana, Junio 2012
"Año 54 de la Revolución"

"Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber."

Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaro ser el único autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____ _____

Yovani Royero Mena Ing. Yulier Matías León

Tutor:

Ing. Yulier Matías León

Graduado de Ingeniero en Ciencias Informáticas y profesor de la Universidad de Ciencias Informáticas (UCI), con 5 años de experiencia en el desarrollo de software.

e-mail: ymatias@uci.cu

Agradecimientos

A mi mamá por todos estos años de esfuerzo, sacrificio y dedicación para que yo haya podido tener y ser todo lo que hasta hoy he logrado. Gracias por todo lo que me has dado mami, te quiero mucho.

A mi papá por enseñarme a ser una mejor persona, por enseñarme a superarme cada día.

A mis hermanos Katia y Yasser por apoyarme siempre y ser ejemplos para mí, por demostrarme que todo es posible.

A mis tías Susa, Lurde y Chicha, a mis tíos, a mis primos, a toda mi familia en general por ser parte importante de mi vida y sentirse siempre orgullosos de mí.

Al amor de mi vida, mi chinita Nany, por estar siempre junto a mí, por demostrarme que cada día es diferente y especial. Por darme su apoyo incondicional, por amarme, por ser mi amiga, mi confidente, mi consejera. Te amo.

A Humberto y Maricela por estar presentes, por aceptarme en su familia y hacerme sentir uno más de ellos. A Humberto Laguardia por sus consejos y experiencias, por considerarme un nieto más. A toda la familia en general.

A Yoandy por ser como un hermano más para mí, por poder contar con él siempre.

A mis amigos de siempre casi hermanos, Reynier, Arlhey, Franier, Yandy y Walter. A los nuevos, Alain y Cervela. A Daylenis por su ayuda incondicional.

Al equipo de desarrollo del proyecto, a Manuel, Veranes, Emilio, Wilfredo, Bode, Vladimir, Dany, Puig, Nelson, Evelio, a todos en general con lo que he compartido todos estos años de la UCI y fuera de ella.

A los profesores, A mi tutor Matías por toda su ayuda y preocupación en todo momento, a los profesores Giselle, Yanirys, Catherine, que de una forma u otra contribuyeron a que me formara como ingeniero con el desarrollo de esta investigación. A Adolfo por enseñarme a buscar nuevas soluciones en lo profesional.

A la Universidad de Ciencias Informáticas por permitirme crecer profesionalmente

¡A todos muchas gracias!

Dedicatoria

A mi mamá le dedico este trabajo y todo lo que pueda llegar a ser en la vida, por ser mi guía, por enseñarme a superar los obstáculos de la vida, por no dejarme caer, por apoyarme siempre y ayudarme a ser hoy un profesional...

A mi hermana por poder contar contigo siempre, por sus consejos por enseñarme a ser mejor cada día, por ser un ejemplo siempre para mi…

A mi novia Danae por ser parte importante de mi vida, por extenderme sus manos incondicionalmente y estar a mi lado en los momentos que más lo he necesitado, por ser el amor de mi vida…

Resumen

Como parte de la modernización del sistema bancario cubano, la Universidad de las Ciencias Informáticas (UCI) está desarrollando un sistema contable informático llamado Quarxo para el Banco Nacional de Cuba (BNC) debido a que el actual sistema en explotación presenta dificultades para la gestión de los diferentes procesos bancarios.

Entre los procesos de contabilidad en el BNC encontramos la gestión de las comisiones, las comisiones a transacciones y las tasas de cambio, que en la actualidad presentan dificultades debido a que su automatización no contempla operaciones de suma importancia para su gestión efectiva.

El presente trabajo de diploma se basa en el diseño y la implementación de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio del sistema Quarxo en vista de desarrollar una solución eficaz que respondiera a los requisitos definidos en etapas anteriores. Para ello se realizó el diseño de los módulos basándose en patrones de diseño, contribuyendo a una mejor implementación con la ayuda de las diferentes herramientas, lenguajes y tecnologías estudiadas para el desarrollo de la misma. Por último se comprobó el correcto funcionamiento de los módulos realizando pruebas unitarias para luego ser probado y liberado por el departamento de calidad de la UCI y aceptado por el BNC.

Palabras Claves: Quarxo, Comisión, Comisión a transacción, Tasa de cambio, Diseño, Implementación, Prueba.

ÍNDICE

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

Introducción

El intercambio de bienes y servicios, la solicitud de creación de cuentas, la solicitud de préstamos, cobros, pagos de cheques entre otras, son actividades frecuentes en la actualidad, donde el banco juega el papel principal. El consumo de los servicios bancarios por las personas, empresas y entidades permite el desarrollo continuo de cada una de ellas y a su vez el de la sociedad donde se desenvuelven. Desarrollo que se logra al proporcionar seguridad, rapidez y eficacia en el uso de los recursos financieros de las partes, al establecerse compromisos entre ellas a través de los medios de pagos, el establecimiento de las tasas de cambio; la realización de transacciones entre otras, que son las aseguran el completamiento exitoso de la actividad realizada por los clientes.

Por su función de intermediarios o fuentes de préstamos de dinero, los bancos cobran una gran cantidad de conceptos por comisiones, estas son fijadas por ellos pero a su vez pueden ser negociadas con los clientes. Estas comisiones son una de las principales fuente de ingreso de estas entidades, por ello su exitosa gestión es un factor importante en su desarrollo y fortalecimiento.

Con el objetivo de brindar un mejor servicio cada país ha ido aplicando en cada una de sus entidades bancarias los avances tecnológicos. Esto ha elevado la rapidez, la calidad, la comunicación y la seguridad en la realización de los procesos tanto internos como los de los servicios brindados a los clientes.

Cuba, nunca ajena al desarrollo, se inserta en los procesos de mejora continua, tratando de prestar un mejor servicio en este sector financiero y para ello el Banco Nacional de Cuba (BNC) ha utilizado diferentes versiones del Sistema Automatizado para la Banca Internacional de Comercio (SABIC). Sin embargo el cambio de la actividad bancaria en la actualidad lo convierte en un sistema decadente ante los nuevos retos al no permitir la interacción de procesos como la gestión de las comisiones, las comisiones a transacciones y las tasas de cambio los que son de vital importancia para el funcionamiento interno del Sistema Bancario Cubano debido a que forman parte básica de los procesos de gestión de entidades y a su vez son utilizados por el resto de las operaciones, con procesos externos que se desarrollan en los demás bancos del país, no permite el registro manual de las tasas de cambio, no asocia las comisiones a transacciones, tampoco define si es obligatorio o no esta asociación, carece de una correcta validación de los datos de entrada además de estar desarrollado con tecnología MS-DOS, obsoleta en la actualidad.

Como parte de la modernización del Sistema Bancario Cubano el BNC solicitó a la Universidad de las Ciencias Informáticas (UCI) el desarrollo de un software que fuera extensible y abarcara los servicios que hoy brinda solucionando a su vez, las deficiencias que presenta el software en explotación. Para el cumplimiento de la solicitud realizada por el BNC la UCI está desarrollando un software llamado Quarxo sobre el núcleo del SABIC, el cual incluye los procesos de gestión de las comisiones, las comisiones a transacciones y las tasas de cambio.

Para el desarrollo de la presente investigación se plantea como ***problema a resolver*** teniendo en cuenta la situación problemática existente en él: ¿Cómo mejorar los procesos de gestión de las comisiones, las comisiones a transacciones y las tasas de cambio en el Banco Nacional de Cuba para erradicar posibles errores, agilizar el proceso y minimizar el uso de recursos humanos y materiales? Para ello el ***objeto de estudio*** se centra en los procesos de gestión de las comisiones, las comisiones a transacciones y las tasas de cambio en entidades financieras bancarias, especificado en el ***campo de acción*** los procesos de gestión de las comisiones, las comisiones a transacciones y las tasas de cambio en el Banco Nacional de Cuba.

El ***objetivo general*** que se persigue es diseñar e implementar los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio para el sistema Quarxo.

Para guiar la investigación se define como ***idea a defender*** la implementación de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio para el sistema Quarxo contribuirá a mejorar los procesos de gestión de comisiones, comisión a transacción y tasas de cambio en el Banco Nacional de Cuba.

Definiendo como ***tareas a realizar*** lo siguiente:

- Estudio y comprensión de los procesos relacionados con la gestión de las comisiones, las comisiones a transacciones y tasas de cambio en las entidades financieras.
- Caracterización de las tecnologías y herramientas a utilizar en el diseño e implementación de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio.
- Diseño de las clases de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio.
- Realización del modelo de componentes de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio del sistema Quarxo.
- Implementación las funcionalidades de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio.

- Realización de pruebas unitarias a la implementación de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio.

La presente investigación utiliza métodos científicos para facilitar el desarrollo del trabajo, métodos teóricos y empíricos. Entre los utilizados está el Histórico – Lógico que permite hacer un estudio de algunos sistemas contables desarrollados anteriormente. Otro método utilizado fue la modelación para la construcción de los modelos de diseño y componentes de los módulos anteriormente mencionados. Se utilizó también el método de observación a través del cual se pudo percibir y planificar como quedaría concebido el sistema.

Estructura del documento:

Capítulo 1: Fundamentación teórica, se abordan los principales conceptos del dominio del problema, así como las tecnologías, metodologías y lenguajes de programación empleados en la construcción del sistema.

Capítulo 2: Se explica el modelo del diseño, así como los patrones utilizados para dar respuesta a la solución planteada. Se realiza la estructuración de los módulos, evidenciando los diagramas de clases del diseño, los diagramas de secuencia y la descripción de las clases y sus atributos.

Capítulo 3: Se enfoca en la implementación de la solución explicando los aspectos principales de la implementación. Se realiza una descripción de las clases y funcionalidades de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio, además de la aplicación de pruebas unitarias para la comprobación del buen funcionamiento de dichos módulos.

Capítulo 1. Fundamentación teórica

1.1 Introducción

El presente capítulo se centra en los aspectos fundamentales que servirán de soporte teórico de la investigación, se aborda sobre los principales conceptos del dominio del problema para su mejor comprensión, se investigan algunos sistemas que automatizan los procesos tanto a nivel mundial como nacional. Además de las herramientas, tecnologías, metodologías y lenguaje de programación empleados en la construcción del sistema.

1.2 Conceptos fundamentales

1.2.1 La Contabilidad

Diferentes han sido las definiciones realizadas sobre la contabilidad. A continuación se brinda algunos de estos conceptos.

El Instituto Americano de Contadores Públicos plantea que: "La contabilidad es el arte de registrar, clasificar y resumir en forma significativa y en términos de dinero, las operaciones y los hechos que son cuando menos de carácter financieros, así como el de interpretar sus resultados". (1)

Juan C. Torres Tovar define que: “La contabilidad como polo opuesto a la información empírica, es el medio que, por sus métodos y técnicas, permite controlar y visualizar, a través de estados financieros, información clara, veraz y oportuna de todos los recursos de la entidad y sólo así poder tomar decisiones, conscientes de sus consecuencias”. (2)

Resumiendo la contabilidad es un proceso de naturaleza económica que tiene por objetivo formular juicios basados en la información de todas las operaciones financieras que tienen lugar en una entidad para la toma de decisiones en beneficio de esta.

1.2.2 Contabilidad Bancaria

La contabilidad bancaria es aquella que tiene relación con la prestación de servicios monetarios y registra todas las operaciones de cuentas en depósitos o retiros de dinero que realizan los clientes. Ya sea de cuentas corrientes o ahorros, también registran los créditos, giros tanto al interior o exterior, así como otros servicios bancarios. (3)

Otra definición de contabilidad bancaria es que se ocupa de la capacitación, la medición y la valoración de todos aquellos elementos financieros que circulan internamente en un banco, con el propósito de suministrar a los gerentes la información necesaria para la toma decisiones, además de realizar un análisis técnico de todas las actividades que se desarrollan dentro de este. (4)

Teniendo en cuenta lo antes planteado se puede resumir que la contabilidad bancaria ayuda a conocer el estado financiero de los bancos, al realizar un análisis de las operaciones económicas y financieras que se realizan en ellos, además de permitir la toma de decisiones en función de su desarrollo.

1.2.3 Los Bancos

Según Olomedo "un banco es una organización financiera cuya principal función es la Intermediación Financiera. Esto se entiende como el proceso mediante el cual obtienen (captan) fondos del público mediante diferentes tipos de depósito (productos pasivos) para realizar operaciones de crédito a través de varias clases de operación (productos activos) según las necesidades del solicitante". (5)

También lo definen como entidades que se organizan de acuerdo a leyes especiales y que se dedican a trabajar con el dinero, para lo cual reciben y tienen a su custodia depósitos hechos por las personas y las empresas, y otorgan préstamos usando esos mismos recursos. (6)

Se concluye que los bancos son entidades que guían los recursos financieros en la economía a través de la atracción de depósitos y la realización de préstamos, además del uso de diferentes instrumentos los cuales son un estímulo para el cliente tanto para ahorrar dinero como para pedir prestado.

Los Bancos Cubanos

El desarrollo económico de Cuba ha ido en ascenso y para ello su política económica también ha evolucionado para adaptarse a las nuevas exigencias de las actividades financieras donde la banca juega un papel primordial.

Dentro de ella se encuentra el BNC, quien es la entidad bancaria encargada de obtener y otorgar créditos en moneda nacional y libremente convertible; de centralizar las relaciones con las entidades extranjeras de seguro de crédito oficial a la exportación según lo decida el Banco Central de Cuba; de mantener el registro, el control, el servicio y la atención a la deuda externa que el Estado cubano y el Banco Nacional de Cuba tienen contraída con acreedores extranjeros hasta la fecha de entrada en vigor del Decreto Ley No.172, del BCC[1]. (7)

[1]BCC: Banco Central de Cuba

Dentro de sus procesos bancarios se encuentran la gestión de las comisiones, las comisiones a transacciones y las tasas de cambio que permiten tener control de las diferentes operaciones que impliquen varias monedas que se realizan dentro de la institución donde estas son utilizadas.

1.2.4 Tasa de cambio

Cada país cuenta con su moneda nacional la cual está hecha de una pieza de oro, plata u otro metal que con forma de disco y acuñada con diversos motivos para acreditar su valor y legitimidad, se utiliza como medio de intercambio (dinero). El *dinero* es un medio de intercambio, por lo general en forma de billetes y monedas, que es aceptado por una sociedad para el pago de bienes, servicios y todo tipo de obligaciones (8). El dinero cumple con tres características básicas:

- Se trata de un medio de intercambio que es fácil de almacenar y transportar.
- Es una unidad contable, ya que permite medir y comparar el valor de productos y servicios que son muy distintos entre sí.
- Es un refugio de valor, que posibilita el ahorro.

Para llevar a cabo el comercio entre los distintos países fue necesario definir un sistema que interviniera en el intercambio monetario el cual es conocido a nivel mundial como tasa o tipo de cambio.

La tasa o tipo de cambio representa el valor de la moneda base comparada con cada una de las monedas internacionales decidida por factores como la fortaleza de la economía, la disponibilidad de recursos y la confianza internacional. (9)

El tipo de cambio se presenta en dos direcciones ya que existe un precio para el vendedor y otro para el comprador. Donde el precio de compra siempre es menor que el precio de venta pues la diferencia entre estos dos valores representa la ganancia de la entidad intermediaria. Estos dos tipos de precios pueden ser establecidos de dos formas diferentes según la moneda que tomemos como base la de un país o la del otro: (10)

- **Forma directa:** Consiste en enunciar el valor de una unidad monetaria extranjera en términos de moneda nacional.
- **Forma indirecta:** Consiste en manifestar el valor de una unidad monetaria nacional con respecto a cada una de las monedas extranjeras.

Cada entidad financiera escoge una de estas formas de expresar el tipo de cambio para su posterior publicación y uso en los servicios requeridos.

Las tasas de cambio utilizadas en el BNC son emitidas por el BCC, quien es la entidad encargada de gestionar esta información para los bancos del país usando las diferentes formas de expresar las tasas de cambio. La información es enviada al BNC en archivos, los cuales son cargados en el SABIC para luego ser utilizados en las operaciones contables. Sin embargo las tasas de cambio luego de ser registradas en el sistema de la entidad no pueden ser actualizadas manualmente lo que imposibilita la corrección de un error o modificación de una tasa de cambio específica ya que dependen totalmente de esos archivos y tampoco permite la aplicación el registro manual de las tasas de cambio.

Con el fin de eliminar estas deficiencias se automatizarán los procesos de gestión de las tasas de cambio con el desarrollo del módulo Tasa de cambio para el sistema Quarxo.

1.2.5 Transacciones contables

Las transacciones contables son movimientos de saldos que producen efectos financieros sobre los recursos y fuentes de donde proceden los cuales se registran en los libros contables. Las transacciones deben de estar fundamentadas o apoyadas en documentos comerciales. Estos documentos constituyen la fuente de datos para los procesos contables.

Transacciones bancarias

Las transacciones bancarias son el movimiento de flujo de efectivo de abono al banco o retiro del banco, el retiro puede darse en efectivo, cheque de caja, etc.

1.2.6 Comisiones bancarias

Las comisiones bancarias son las cantidades que las entidades de crédito cobran en compensación por sus servicios (por ejemplo, enviar una transferencia, cambiar divisas, administrarle una cuenta, estudiar un préstamo, darle una tarjeta de crédito, etc.). Estas pueden cobrarse juntas, como un solo cargo genérico (caso de las tarifas planas) o separadas como un cargo individualizado por cada servicio prestado. (11)

Las comisiones bancarias son libres, pueden ser fijadas arbitrariamente por las entidades bancarias o ser negociadas por los clientes cumpliendo siempre con lo estipulado.

Requisitos de las comisiones bancarias (12)

i. Ser realmente necesarias: El servicio cobrado debe responder a un servicio efectivo y no recogido por otro producto ya contratado por el cliente.

ii. No ser sorpresivas**:** El precio de la comisión debe ser comunicado de forma clara al consumidor.
iii. Ser legales**:** Aunque el cobro está liberalizado, debe respetar diversas normas que regulan la materia.
iv. No resultar abusivas: Deben cubrir los gastos de la entidad y su margen de beneficios; ello implica que no puedan cobrar comisiones cuya cuantía sea proporcional al nominal de la operación. También es ilícito cobrar comisiones por seguimiento y administración de un préstamo, pues son actuaciones que el banco realiza por su propio interés.
v. Modificaciones de las comisiones: Para modificarlas deben haber transcurrido dos meses desde su publicación en el Tablón de Anuncios de la Entidad.
vi. No pueden repercutir en el cliente los gastos generados por la insuficiente diligencia de la entidad (gastos que se ocasionen por la no presentación dentro de plazo de recibos u efectos, como consecuencia de la actuación negligente del banco o por fallos en el sistema informático).
vii. Tampoco pueden cobrarse comisiones no previstas en el contrato.
viii. No se puede cobrar ninguna comisión por descubierto cuando este se produzca como consecuencia de normas de valoración a la hora de imputar un abono o adeudo en una determinada fecha. En todo caso, las comisiones por descubierto, si proceden, deben ser proporcionales al descubierto generado.

Tipos de comisiones:

Las comisiones bancarias se pueden clasificar de varias formas dependiendo de las acciones realizadas. Algunos de los conceptos más frecuentes que se aplican en el BNC por los que pueden cobrar comisión son:

Comisión de apertura: Es la tarifa que la entidad bancaria prestamista cobra al inicio del préstamo a los prestatarios, para llevar a cabo la formalización de un préstamo personal (11).

Comisión de cancelación anticipada: Es la cantidad de dinero que se le cobra al cliente cuando se lleva a cabo una cancelación adelantada del préstamo. En el momento en que el prestatario desea cancelar en su totalidad el préstamo, debe pagar además de la comisión total anticipada, los otros gastos que generó el préstamo. Estos últimos están determinados por el tipo de préstamo que se pactó. También es importante considerar que la entidad bancaria cobra una comisión por lo que dejó de ganar, tras el no cambio su curso normal. Entre los otros gastos que se pueden cobrar junto a la comisión por anticipación están: (11)

- Los honorarios del corredor de comercio, de ser un requisito en la póliza.
- Un seguro, tiene que ver con la necesidad de tener algún tipo de seguridad (esto se acuerda en la póliza).

Comisión por anticipación parcial anticipada: Es la remuneración que queda para el prestamista, cuando se da un adelanto de un pago en el préstamo; ya que este hecho, bajo ciertas circunstancias dadas, podría significar pérdidas. Las entidades bancarias suelen cobrar un porcentaje sobre la cantidad que se va a pagar en adelantado, y de esta manera se protegen y cobran los gastos extras que puede generar una anticipación en el pago. El pago de este tipo de comisión se realiza al mismo tiempo que se hace el pago del importe. Así mismo, las entidades exigen una cantidad mínima del importe a amortizar, con el fin de evitar gastos por múltiples amortizaciones de cuantías sin mayor importancia (11).

Comisión por cambio de condiciones modificación de contrato o cambio de garantías: Es el monto adicional que se le cobra al prestatario, cuando éste hace algún tipo de cambio al contrato inicial. La comisión cubriría todas las condiciones tales como el trámite y el análisis de riesgo en primera instancia. También puede ser vista esta comisión como la retribución que queda para el ente bancario prestamista, en el caso de que el cliente desee otro tipo de acuerdo (11).

Movimientos de fondos: Es el importe que se cobra cada vez que se realiza una transferencia o gestionan un ingreso por cheque, letra o pagaré. La comisión dependen de las siguientes variables: la forma en la que se formula la orden de pago (manualmente es más cara que en soporte magnético o informático), la urgencia de la transmisión y el destino de la operación. (11)

La aplicación de las distintas comisiones a las transacciones bancarias es muy frecuente en la actualidad y por ello las instituciones bancarias han ido fijando las comisiones de acuerdo a los servicios solicitados por el cliente. Esto acelera su gestión al estar asociadas a las transacciones.

Teniendo en cuenta las ventajas que brinda la asociación de las comisiones a las transacciones surge la necesidad de automatizar en el BNC la gestión de este proceso así como también la gestión de las comisiones que actualmente en la entidad se registran directamente en la base de datos. Para ello se desarrollaran los módulos Gestionar comisión y Comisión a transacción para el sistema Quarxo que darán solución a estas deficiencias que presenta el SABIC.

1.3 Sistemas informáticos contables

Un sistema de contabilidad puede verse como una combinación de programas, procedimientos, datos y equipamiento, utilizados de manera coherente en el procesamiento de la información. Son

capaces de registrar y procesar las transacciones históricas que se generan en una empresa, para ello solo hay que ingresar la información requerida como las pólizas contables, ingresos y egresos y hacer que el programa realice los cálculos necesarios. (13)

1.3.1 En el mundo

SAP para la banca

Sistemas, Aplicaciones y Productos para Procesamiento de Datos (SAP por sus siglas en inglés de Systems Applications Products in Data Processing) es un ERP[2] que proporciona soluciones para mejorar la coordinación estratégica y la eficacia de los procesos financieros, operativos y de capital humano en una empresa. Dentro de estas se encuentra SAP para la banca (SAP for banking) que procesa la información comercial y procesos financieros bancarios para entidades financieras bancarias, proporcionándoles herramientas necesarias para maximizarlos. Ofrece procesos transaccionales bancarios de los depósitos y los préstamos que ayudan a agilizar y simplificar las operaciones a través de líneas de productos y servicios. Permite gestionar el ciclo completo de una cuenta o contrato, facilita el establecimiento de intereses, costes y fechas valores. Procesa de manera eficaz balances de cuentas, concentración de efectivo, extractos y otras tareas claves, permite una gestión total de los riesgos bancarios tanto para la gestión comercial como no comercial, permite planificar el negocio a través de la gestión de activos y pasivos, posibilita una administración y mantenimiento de los contratos, así como la gestión de los procesos contables en las diferentes áreas. (14)

Aspel-BANCO

Es el Sistema de Control Bancario que controla eficientemente los ingresos y egresos de cualquier tipo de cuenta bancaria, ofreciendo información financiera precisa en todo momento. (15)

Aspel-BANCO permite:

- Manejar cuentas en moneda nacional y extranjera.
- Programar movimientos periódicos.
- Realizar inversiones en plazo fijo y en acciones.
- Hacer conciliación electrónica con las principales instituciones financieras.
- Consultar en cualquier momento todas las operaciones financieras.

Sin dejar de tener en cuenta las funcionalidades que brindan estos sistemas hay que señalar que ambos son aplicaciones contables privativas lo cual conlleva a un gran gasto económico su

[2]ERP: Planificación de Recursos Empresariales (Enterprise Resource Planning en inglés)

adquisición y mantenimiento lo que resulta poco conveniente. Tampoco sus funcionalidades pueden ser adaptadas a las necesidades existentes en BNC ya que no se tiene acceso a su código fuente por lo cual no se utilizará ninguna de sus funcionalidades en el desarrollo de los módulos para el sistema Quarxo.

1.3.2 En Cuba

Versat-Sarasola

Versat-Sarasola es un software desarrollado en Cuba por la entidad TEICO de Villa Clara, para automatizar prácticamente todas las actividades de Planificación, Control y Análisis Económico de cualquier tipo de Entidad, ya que es configurable. Presenta dos variantes para su instalación: la variante típica que incluye todos los subsistemas del VERSAT-Sarasola y la variante personalizada en la que solo estarán los subsistemas seleccionados por el usuario. Permite llevar el control y el registro contable individual de todos los hechos económicos que se originan en las estructuras internas de las entidades y obtener los Estados Financieros y Análisis Económicos y Financieros en estos niveles. Aparece el Subsistema de Contabilidad General como rector del sistema que maneja el resto de los subsistemas: (16)

- Costos y Procesos
- Cobros y pagos
- Activos Fijos
- Finanzas
- Banco

Permite controlar las operaciones que se realizan sobre las diferentes cuentas bancarias definidas, permitiendo**:**

- Definir las cuentas bancarias en diferentes monedas.
- Definir los talonarios de los diferentes instrumentos de pagos por cada cuenta bancaria, incluyendo además las letras de cambio.
- Procesar los estados de cuentas a partir de la información que se recibe en el mismo y las operaciones registradas con anterioridad en la entidad.
- Realizar conciliaciones de cada una de las cuentas bancarias.
- Procesar los cobros y pagos automáticos que se muestran en los estados de cuentas.
- Emitir reportes referidos a documentos en tránsito, registro de cheques emitidos, registro de disponibilidad, etc.

Este sistema contable es una de las soluciones informáticas más utilizadas actualmente en el país, que en sus últimas versiones tiene en cuenta la dualidad de monedas y la multientidad; define detalladamente todos y cada uno de los procesos que comprende el ajuste al costo en las entidades eficientemente de forma automática, sin embargo, su subsistema bancario no abarca todos los servicios bancarios como es la gestión de las tasas de cambio y las comisiones entre otras, las cuales son algunas de las funcionalidades que pudiera prestar una institución bancaria y esto se debe a que dicho sistema está destinado principalmente a entidades no bancarias imposibilitando esto su aplicación en el BNC, además de estar soportado sobre tecnología privativa.

SABIC

SABIC (Sistema automatizado para la Banca Internacional de Comercio) es un sistema diseñado y desarrollado por la Dirección de Sistemas Automatizados del Banco Central de Cuba. Entre sus principales características están: la contabilización en tiempo real (permite mantener actualizados los ficheros contables) y la contabilización multimoneda (permite registrar los activos y pasivos en las monedas orígenes sin tener que realizar en el momento del registro las conversiones de monedas). Además, las operaciones contables se pueden realizar a través de transacciones tipificadas generando los asientos contables de forma automática. (17)

SABIC comprende las regulaciones para la aplicación de las normas y proceso electrónico de datos de operaciones tales como remesas, transferencias, cuentas bancarias, ingresos, pagos, etc. Entre otras cuestiones, el sistema apoya el aumento de la eficacia y eficiencia en los servicios bancarios, la seguridad de la información contable, la interoperabilidad entre las sucursales del banco, la actualización de sus bases de datos en tiempo real, etc. (17)

A pesar de las ventajas que posee el SABIC, está soportado sobre MS-DOS el cual no está preparado para dar acceso remoto a los usuarios. Además de no ser compatible con Windows, sistema operativo que tienen instalado los puestos de trabajo del BNC, no permite el registro manual de las tasas de cambio, no asocia las comisiones a transacciones, tampoco define si es obligatorio o no esta asociación y carece de una correcta validación de los datos de entrada.

Teniendo en cuenta las deficiencias que presenta el SABIC no se utilizarán las funcionalidades que este brinda para los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio ya que no posee una correcta estructura definida para la realización de estos servicios pues las comisiones son registradas directamente a la base de datos y las tasas de cambio se cargan desde un fichero sin ofrecer información sobre los datos cargados además de no permitir la introducción de los valores de las tasas manualmente.

1.4 Tecnologías y Herramientas utilizadas en el desarrollo

Una adecuada selección de las tecnologías y herramientas que serán utilizadas en el desarrollo del software está dada por el conocimiento del ambiente donde será implantado favoreciendo esto en el aprovechamiento al máximo del tiempo así como la calidad del producto a desarrollar.

La selección de las tecnologías y metodologías utilizadas en la desarrollo de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio fue hecha por el grupo de arquitectura del sistema Quarxo, por ello no se definen dichas herramientas, sino que se expone una abreviada caracterización de cada una de ellas y de los beneficios que aportan en el proceso de diseño e implementación de los módulos implicados.

1.4.1 Metodología de desarrollo

Proceso Racional Unificado (RUP)

RUP (por sus siglas en inglés de Rational Unified Process) constituye una de las metodologías más estandarizadas utilizada para el desarrollo de sistemas orientados a objetos basados íntegramente en Lenguaje Unificado de Modelado (UML) como soporte a la metodología. Según Jacobson RUP es un proceso que se caracteriza por ser: (18)

- **Dirigido por casos de uso.** Los casos de uso reflejan lo que los usuarios futuros necesitan y desean y se usan para determinar el alcance de cada iteración y el contenido de trabajo de cada persona del equipo de desarrollo.
- **Centrado en la arquitectura.** La arquitectura representa la forma del sistema, la cual va madurando en su interacción con los casos de uso hasta llegar a un equilibrio entre funcionalidad y características técnicas.
- **Iterativo e incremental.** RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. Las iteraciones hacen referencia a pasos en los flujos de trabajo, y los incrementos, al crecimiento del producto.

RUP propone varias fases de desarrollo para la elaboración de los sistemas, entre ellas se encuentran Modelado del negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas, entre otras. Cada una de las iteraciones de estas fases añade funcionalidades al producto de software o mejora las existentes.

El presente trabajo se basa en el desarrollo de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio y para ello es necesario apoyarse en las disciplinas de Análisis y Diseño, e Implementación.

Disciplina Análisis y Diseño:

Tiene como función principal describir como el programa será realizado y define como será programado. Teniendo así entre sus objetivos (19):

- Transformar los requisitos al diseño del futuro sistema.
- Desarrollar una arquitectura para el sistema.
- Adaptar el diseño para que sea consistente con el entorno de implementación, diseñando para el rendimiento.

Disciplina Implementación:

En esta disciplina se describe como los elementos del modelo del diseño se implementan en términos de componentes y como se organizan. Además de la realización de pruebas unitarias a los componentes y la integración de los resultados del trabajo de implementadores individuales en un sistema ejecutable. (19)

1.4.2 Patrones de diseño

Los patrones de diseño son una solución a un problema de diseño no trivial que es efectiva y reusable. Facilitan la comunicación entre diseñadores, pues establecen un marco de referencia, ayudan a especificar las interfaces, identificando los elementos claves en las interfaces y las relaciones existentes entre distintas interfaces. (20)

A continuación se mencionan los patrones a utilizar que ayudarán de manera significativa en la obtención de un buen diseño e implementación de los módulos, al permitir la declaración de interfaces y sus relaciones, así como la definición de responsabilidades y comportamiento de las clases a diseñar e implementar.

Patrones de Asignación de Responsabilidades (GRAPS)

GRASP (General Responsabilty Assignment Software Patterns por sus siglas en inglés), es un conjunto de patrones que permite la asignación de responsabilidad en parámetros útiles para el diseño del producto (19). Entre ellos se encuentran:

Patrón Experto: ¿Cómo asignar responsabilidades de la forma más eficiente? Asigna una responsabilidad al experto en la información, la clase que cuenta con la información necesaria

para cumplir la responsabilidad. 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.

Patrón Creador: ¿Quién debería ser responsable de crear una nueva instancia de alguna clase? Guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento.

Patrón Controlador: ¿Quién debería encargarse de atender un evento del sistema? El controlador es un intermediario entre la interfaz de usuario y el núcleo de las clases donde reside la lógica de la aplicación. El controlador no realiza mucho trabajo por sí mismo; más bien coordina la actividad de otros objetos. Asignar la responsabilidad del manejo de mensajes de los eventos del sistema a una clase que represente alguna de las siguientes opciones:

- El sistema global.
- La empresa u organización global.
- Algo activo en el mundo real que pueda participar en la tarea.
- Un manejador artificial de todos los eventos del sistema de un caso de uso.

Patrón Bajo Acoplamiento: ¿Cómo dar soporte a una mínima dependencia y a un aumento de la reutilización? 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.

Patrón Alta Cohesión: Es la meta principal que ha de buscarse en todo momento. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño. Una clase con mucha cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizarla. Su alto grado de funcionalidad, combinada con una reducida cantidad de operaciones, también simplifica el mantenimiento y los mejoramientos. La ventaja que significa una gran funcionalidad también soporta un aumento de la capacidad de reutilización.

Patrones Banda de Cuatro (GoF)

GoF (Gang of Four) son patrones de comportamiento que se encargan de establecer la forma en que los objetos se comunicarán en una aplicación. (20)

Fachada (Facade): Este patrón sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Utilizado para reducir la dependencia entre clases, ya que ofrece un punto de acceso al resto de las clases. Si

estas cambian o se sustituyen por otras solo hay que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones clientes. El patrón Fachada es sencillo y se utiliza ampliamente.

Singular (Singleton): Este patrón garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. El acceso a datos en el sistema se realiza mediante una única instancia evitando que se creen múltiples conexiones innecesarias.

1.4.3 Ambiente de desarrollo

Plataforma Java 2 Edición Empresarial (J2EE)

J2EE (Java 2 Enterprise Edition) es una plataforma abierta y estándar de Java que proporciona técnicas específicas que describen el lenguaje, pero, además, provee las herramientas para implementar productos de software de alto rendimiento, escalabilidad y seguridad. Define una arquitectura utilizando un modelo multicapa, permitiendo así realizar aplicaciones distribuidas complejas.

Lenguaje Java

Java es un lenguaje de programación simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico. Es un lenguaje multiplataforma, que proporciona un conjunto de clases para su uso en aplicaciones de red, permitiendo abrir sockets y establecer conexiones con servidores o clientes remotos. Fue diseñado para crear software altamente fiable.

Eclipse 3.5 IDE

Es un entorno de desarrollo integrado (IDE por sus siglas en inglés de Integrated Development Environment) de código abierto basado en Java, desarrollado por IBM cuyo código fuente fue puesto a disposición de los usuarios. En sí mismo Eclipse es un marco y un conjunto de servicios para construir un entorno de desarrollo a partir de componentes conectados (plugin[3]) que a través de ellos se puede hacer uso de esta herramienta para distintos lenguajes como C/C++, Cobol, C#.

Contenedor Web (Tomcat 7.0)

Apache Tomcat implementa las especificaciones de los servlets y las páginas servidoras de Java (JSP) de Sun Microsystems. Al ser desarrollado en Java, funciona en cualquier sistema operativo

[3] Un **plugin** es un módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande.

que tenga instalado la máquina virtual de Java. Puede funcionar como servidor web por sí mismo, es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Control de Versiones (SubVersion 1.6.6)

Es un software de sistema de control de versiones de código abierto y gratuito. Soporta el manejo de ficheros y directorios a través del tiempo. Permite que varios miembros de un equipo puedan comprobar la última versión del código de un repositorio, hacer sus cambios en el código, y luego confirmar de nuevo los archivos al mismo tiempo. Se integra con el eclipse mediante el plugin SubEclipse.

Base de Datos (SQL Server 2005)

SQL Server 2005 provee herramientas sólidas y conocidas reduciendo la complejidad de la creación, despliegue, administración y uso de aplicaciones analíticas y de datos empresariales en plataformas que van desde los dispositivos móviles hasta los sistemas de datos empresariales. A través de un conjunto global de características, la interoperabilidad con sistemas existentes y la automatización de tareas rutinarias, SQL Server 2005 ofrece una solución completa de datos para empresas de todos los tamaños.

El uso de este gestor de base de datos a pesar de ser un software privativo se debe a que es utilizado por el SABIC y por acuerdos entre el cliente y la dirección del proyecto se decidió utilizarlo. La selección de SQL Server 2005 privó el uso del gestor de base datos Postgres el cual también está basado en tecnología libre además de ser multiplataforma y un gestor confiable.

La selección del IDE Eclipse además de ser una herramienta libre, se debe a que es uno de los menos consumidores de recursos que utilizan Java como lenguaje de programación. Lenguaje destinado para la creación de aplicaciones altamente fiables por sus características como el servidor web Tomcat el cual será utilizado por su gran estabilidad en entornos de mucho tráfico. Además de estas herramientas libres se seleccionó el controlador de versiones SubVersion que permite conservar las diferentes versiones durante todo el proceso de desarrollo de los módulos por integrarse fácilmente al IDE de desarrollo.

1.4.4 Herramientas de modelado

UML

Es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software y está compuesto por diversos elementos gráficos que se combinan para conformar diagramas.

Visual Paradigm 3.4

Visual Paradigm es una herramienta CASE[4] que permite visualizar y diseñar elementos de software, para ello utiliza UML y ofrece una gama de facilidades para el modelado de aplicaciones. Está orientada a la creación de diseños usando el paradigma de programación orientada a objetos. Apoya las notaciones de Java y UML. Provee soporte para la generación de código, tiene integración con diversos IDEs como NetBeans, JDeveloper, Eclipse, JBuilder, así como la posibilidad de realizarse la ingeniería inversa para aplicaciones realizadas en JAVA, .NET, XML e Hibernate. También tiene disponibilidad en múltiples plataformas y soporta BPMN (Notación y Modelado de Procesos de Negocio.

Con el uso de la metodología RUP apoyada en UML se logrará modelar con la herramienta Visual Paradigm los diferentes diagramas pertenecientes a las fases de Diseño e Implementación lo cual permitirá desarrollar el código de la aplicación posteriormente de forma satisfactoria.

1.4.5 Frameworks

Framework

Un Framework se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta. Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

Hibernate 3.5

Hibernate es una capa de persistencia objeto/relacional y un generador de sentencias SQL de código abierto. Con este framework, se puede diseñar objetos persistentes que podrán incluir relaciones, colecciones, polimorfismo, y un gran número de tipos de datos. Con Hibernate se logra una abstracción total del gestor de base de datos a utilizar. De una manera muy rápida y optimizada se puede realizar consultas contra cualquiera de los entornos soportados: Oracle, DB2, SQL Server, etc. Se adapta a cualquier proceso de desarrollo, ya sea comenzando un diseño desde cero o trabajando con una base de datos existente.

Spring Framework 2.0

Spring es un framework de código abierto para el desarrollo de aplicaciones en la plataforma Java. Es el más popular de todos los frameworks de peso ligero en la actualidad, respecto al desarrollo

[4] El acrónimo CASE en inglés significa Computer Aided Software Engineering y se traduce como Ingeniería de Software Asistida por Computadoras.

de aplicaciones web en Java. Además presenta una arquitectura muy flexible. La gran ventaja que presenta la utilización de este framework recae en la presencia de varios módulos completamente independientes unos de otros.

Dojo Toolkit 1.3

Es una biblioteca Javascript de código abierto, que contiene APIs[5] y controles (widgets) para facilitar el desarrollo de aplicaciones Web que utilicen tecnología AJAX. Provee un conjunto de componentes de interfaz gráfica de usuario como calendarios y menús, que se pueden insertar de manera sencilla en páginas HTML.

El uso de las diferentes tecnologías y herramientas antes mencionadas permite crear un ambiente de desarrollo favorable para el diseño e implementación de los módulos apoyados con el uso de los diferentes frameworks, que nos brindan una estructura fiable y flexible como es el caso del framework Spring, se facilita el trabajo con la base de datos mediante Hibernate, además de crear una interfaz amigable con el uso de las librerías de Dojo.

Conclusiones Parciales

Para el desarrollo del trabajo, se han estudiado diferentes conceptos y sistemas contables así como las principales características de las comisiones y tasas de cambio en los bancos cubanos como en bancos internacionales. Además de familiarizarse con las principales herramientas y tecnologías definidas por el equipo de arquitectura de sistema Quarxo que se utilizarán en el diseño e implementación de los módulos, entre ellas se encuentran:

- RUP como metodología de desarrollo del software y UML como lenguaje de modelado.
- Visual Paradigm como herramienta CASE para el diseño.
- Eclipse como IDE de desarrollo por ser una herramienta altamente provechosa, el cual permite la implementación del sistema con varios frameworks además de ser soportado por la plataforma J2EE.

Con el uso de estas tecnologías y herramientas se minimiza grandemente el costo de desarrollo al estar basadas en software libre.

[5] **API**: API, abreviatura de **A**pplication **P**rogramming **I**nterface. Un API es una serie de servicios o funciones que se le ofrece al programador.

Capítulo 2. Arquitectura y Diseño de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio

2.1 Introducción

En el presente capítulo se describe la arquitectura del sistema Quarxo y el diseño de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio, el cual tiene como objetivo generar las entradas adecuadas para la fase posterior. Se generan en este capítulo el modelo de diseño, diagramas de paquetes, diagramas de clases, diagramas de secuencias y los modelos de datos.

2.2 Descripción de la arquitectura

La arquitectura definida para el desarrollo del sistema Quarxo está dividida en tres capas: la Capa de Presentación, la Capa de Negocio y la Capa de Acceso a Datos, además de una cuarta capa transversal a ellas que se define como la Capa de Dominio.

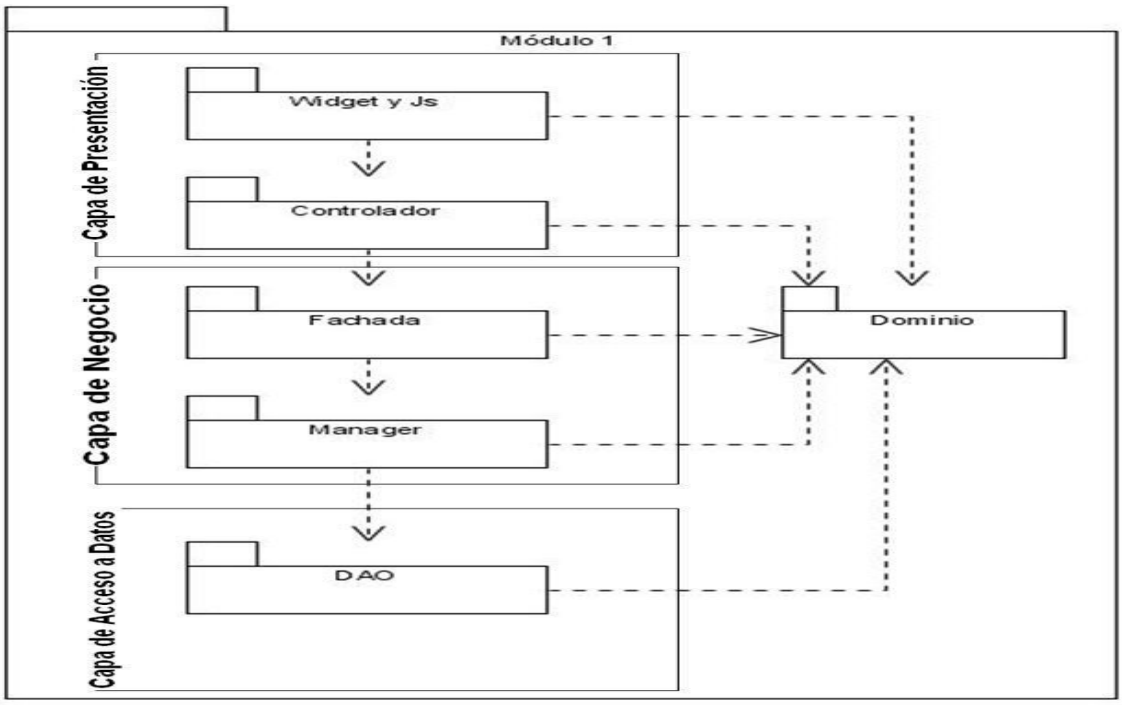

Ilustración 1 Arquitectura de un módulo de Quarxo

2.2.1 Capa de Presentación

Esta capa se encuentra dividida en dos subcapas, una del lado del cliente que utiliza la librería de Dojo para realizar las peticiones asíncronas que serán atendidas por la subcapa del lado del servidor, la cual utiliza el framework Spring MVC para atender dichas peticiones, también es la

encargada de controlar el flujo de presentación y enviar las respuestas correspondientes solicitadas por la interfaz de usuario. Además de ello se relaciona con la Capa de Negocio y la Capa de Dominio.

2.2.2 Capa de Negocio

Esta capa también se encuentra divida en dos subcapas, la primera es la Fachada que será el punto de intercambio de la Capa de Negocio con la Capa de Presentación al englobar todos los métodos que necesitará esta última, no realiza la lógica de negocio delegando así a la subcapa Manager la realización de toda la lógica de negocio a través de toda su jerarquía de clases, la comunicación con la Capa de Acceso a Datos para el trabajo con los datos y con la Capa de Dominio para la generación de los objetos de dominio.

2.2.3 Capa de Acceso a Datos

En esta capa se implementarán los métodos encargados de interactuar con la Base de Datos. Esta capa tendrá solamente dependencia con la Capa de Dominio y la interacción con la Capa de Negocio se realizará mediantes interfaces. Para el desarrollo de esta capa se utilizará el patrón Objetos de Acceso a Datos (DAO, Data Access Object en inglés).

2.2.4 Capa de Dominio

En esta capa se declararán todas las clases que representan entidades del negocio. Estas clases de dominio estarán presentes en todas las capas anteriormente descritas.

2.3 Diseño de los módulos

Durante el desarrollo de un software el diseño se realiza con el fin de traducir los requisitos a una estructura que describe como implementar el sistema, siendo lo suficientemente específica para que no existan ambigüedades, y ofreciendo la posibilidad de descomponer los trabajos de implementación en componentes más manejables, visualizándolos mediante una notación común. La importancia del diseño se puede traducir en una sola palabra, calidad, ya que sin este se corre el riesgo de construir un sistema inestable, que fallará cuando se lleven a cabo cambios; que pueden resultar difícil de comprobar. (18)

2.3.1 Descripción de los módulos

Tomando como artefactos de entrada los requerimientos definidos y la descripción de los casos de usos por los analistas del sistema QUARXO los módulos a diseñar son:

- Gestionar comisión

- Comisión a transacción
- Tasa de cambio

Gestionar comisión

El módulo le permite al usuario gestionar las comisiones que el banco aplicará en las transacciones contables que lo requieran. Estas comisiones pueden ser fijas o variables en dependencia de si se desea que cobren un importe fijo o cobren un importe variable. Al crear una comisión esta no puede ser eliminada del sistema, debido a que puede haber sido utilizada en alguna transacción contable pasada. Los saldos de las comisiones están expresados en función de la moneda base del sistema.

Casos de usos:

- Registrar Comisión
- Actualizar Comisión
- Consultar Comisión
- Buscar Comisión

Comisión a transacción

El módulo le permite al usuario gestionar las comisiones que están asociadas a una transacción contable existente. Se brindan las funcionalidades que permiten registrar una comisión a una transacción, actualizarlas, eliminarlas y buscarlas. Las interfaces utilizadas en estas funcionalidades son amigables y valida todas las entradas de datos. Tener asociadas las comisiones a las transacciones permite que el sistema valide el cobro de estas en el momento que se esté creando una transacción.

Casos de usos:

- Registrar Comisión a transacción
- Actualizar Comisión a transacción
- Eliminar Comisión a transacción
- Buscar Comisión a transacción

Tasa de cambio

El módulo permite la gestión de las tasas de cambio que utiliza el sistema. Estas son utilizadas en las operaciones donde intervienen varias monedas, como por ejemplo: las transferencias, los pagos en monedas extranjeras, etc. Por lo que el sistema necesita acceder a dicha información

para realizar dichas operaciones y así lograr la equivalencia entre dos saldos de distintas monedas.

Casos de usos:

- Registrar Tasa de cambio manual
- Registrar Tasa de cambio desde archivo
- Actualizar Tasa de cambio
- Buscar Tasa de cambio

2.3.2 Modelo de diseño

2.3.2.1 Modelo de paquete

Para el desarrollo de software resulta conveniente agrupar las clases y ficheros por diferentes criterios para lograr una organización y facilitar la comprensión del código de la aplicación, resultando conveniente para ello el uso de paquetes. Estos reflejan la arquitectura de alto nivel de un sistema: su descomposición en subsistemas y sus dependencias. Una dependencia entre paquetes resume las dependencias entre los contenidos del paquete (18). Los modelos de paquetes de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio tienen la misma estructura, motivo por el cual se presentó solo el modelo de paquetes del módulo Gestionar comisión que se muestra a continuación.

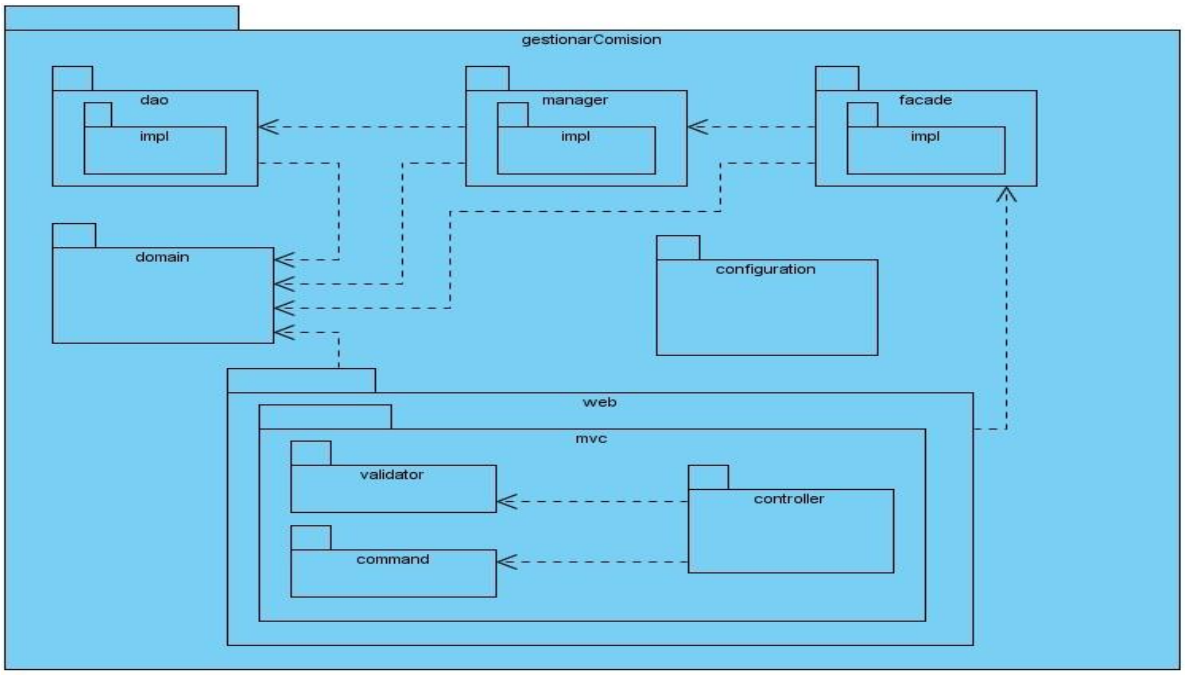

Ilustración 2 Diagrama de paquete del módulo Gestionar comisión

A continuación se mencionan las características de cada paquete:

Paquete configuration: en este paquete están los ficheros XML de configuración de los diferentes contextos de Spring:

- -dataaccess.xml: contexto de acceso a datos
- -business.xml: contexto de negocio
- -servlet.xml: contexto de Spring
- -webflow.xml: contexto de Spring Webflow

Paquete dao: en este paquete están las interfaces y las clases que implementan el acceso a datos del módulo, además de los ficheros de mapeos de Hibernate.
Paquete manager: en este paquete se encuentran las interfaces e implementaciones del negocio del módulo correspondiente.
Paquete facade: en este paquete se encuentra la interfaz del módulo hacia la Capa de presentación y la implementación de las funcionalidades que se le brindaran a la presentación.
Paquete domain: en este paquete están las clases del dominio del módulo.
Paquete web: este paquete es el contenedor de los paquetes *mvc* y *webflow*.
Paquete mvc: aquí estarán los paquetes que contienen la lógica de presentación en el servidor para Spring MVC.
Paquete command: contiene clases que representan objetos a manipular en los formularios.
Paquete controller: contiene las diferentes clases que heredan de los controladores de Spring.
Paquete propertyEditor: contiene las clases para convertir un grupo de datos en objetos.
Paquete validator: contiene las clases encargadas de validar los datos del lado del servidor.

Para un mayor entendimiento del diseño de los módulos se realizaron los diagramas de clases del diseño y solamente se muestran en este capítulo los diagramas correspondientes al módulo Gestionar comisión, el resto de los diagramas de los demás módulos se exponen en los Anexos del documento.

2.3.2.2 Diagrama de clases del diseño

Un diagrama de clases del diseño muestra y describe gráficamente las especificaciones de las clases de software y de las interfaces de la aplicación. (18)

A continuación se describe el diseño correspondiente a cada capa definida para el desarrollo del sistema.

Diseño de la Capa de Presentación

En la capa de presentación de cada módulo se hace uso del framework Spring MVC para atender las peticiones realizadas al servidor. Seguidamente se muestra el diseño de dicha capa en el módulo Gestionar comisión el cual es similar para el resto de los módulos.

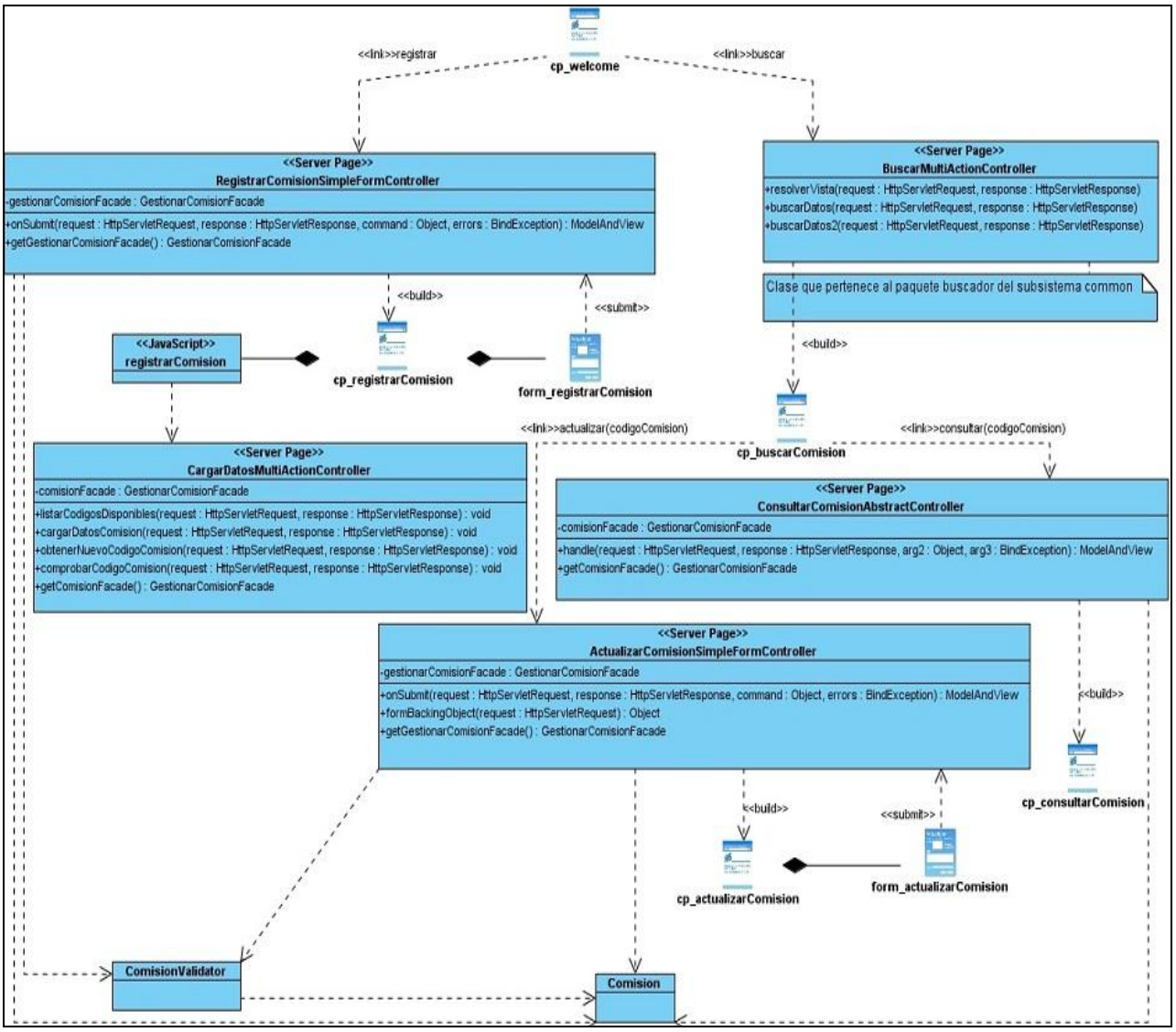

Ilustración 3 Diseño de la capa de presentación del módulo Gestionar comisión

Página cliente (cp por sus siglas en inglés de Client page): representan páginas web encargadas de mostrar los formularios de información al usuario.

Formulario (Form): representa una colección de campos de entrada de datos, es parte de una *página cliente.*

Server Page: son las clases que heredan de los controladores de Spring MVC y se encargan de atender las peticiones realizadas desde el cliente al servidor.

Diseño de la Capa de Negocio

Cada módulo tiene una *Fachada* la cual se encarga de brindar las funcionalidades a la capa de presentación aislando de esta forma la capa de presentación de la implementación de la lógica del negocio.

Las clases que componen la capa de negocio se encuentran declaradas en el *Manager* y sus funcionalidades para solucionar el negocio están implementadas en el ManagerImpl.

Global: en este paquete se encuentran funcionalidades comunes a diferentes subsistemas.

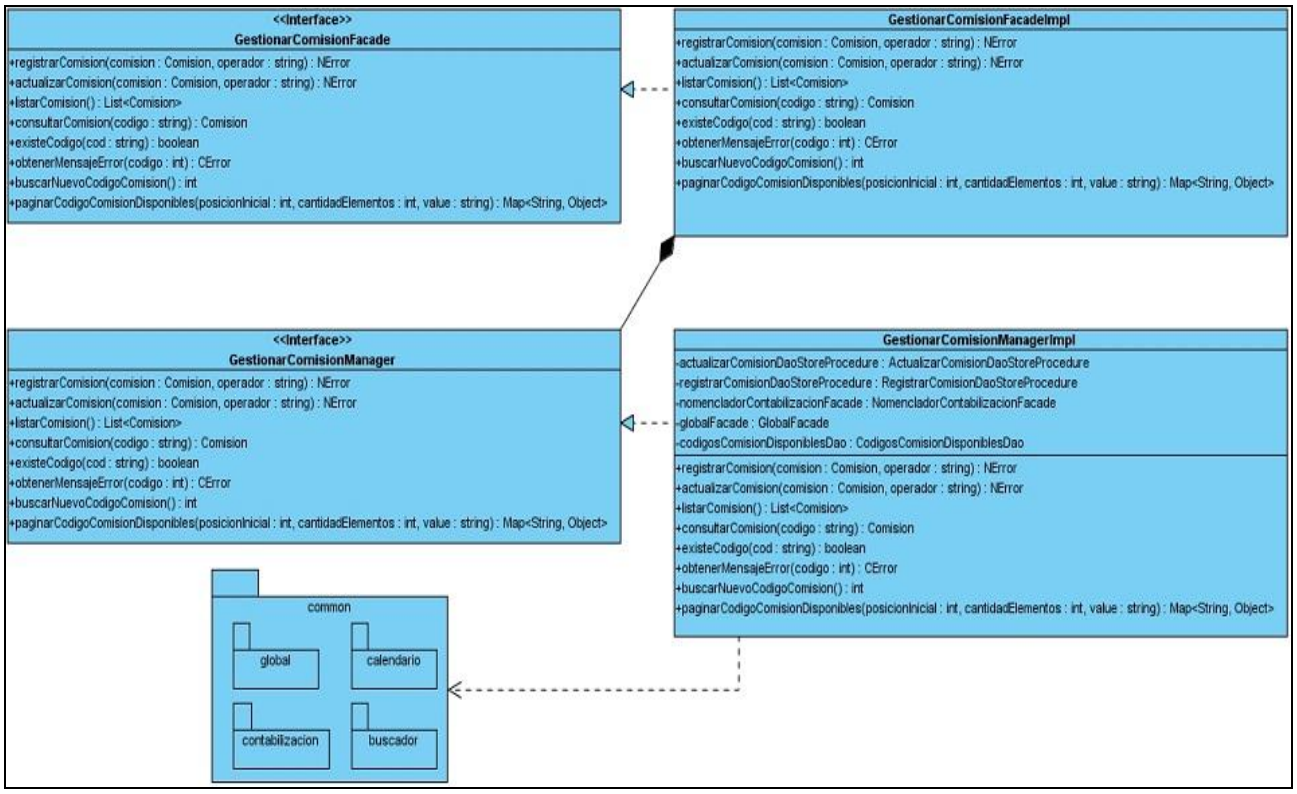

Ilustración 4 Capa de Negocio del Módulo Gestionar comisión

Diseño de la Capa de Acceso a Datos

Con el uso del patrón DAO y de los componentes DAO genéricos utilizados en la arquitectura del sistema compuesto por la interfaz FinixuBaseDAO y la clase FinixuAbstractBaseDAO se facilita el trabajo para el acceso a datos ya que estas clases ofrecen funcionalidades básicas a realizar con las clases persistentes como son: persistir, actualizar, listar, buscar por identificador, eliminar. Además se utiliza la clase genérica StoredProcedureDAOSupport que ofrece funcionalidades para trabajar con los procedimientos almacenados de la base de datos.

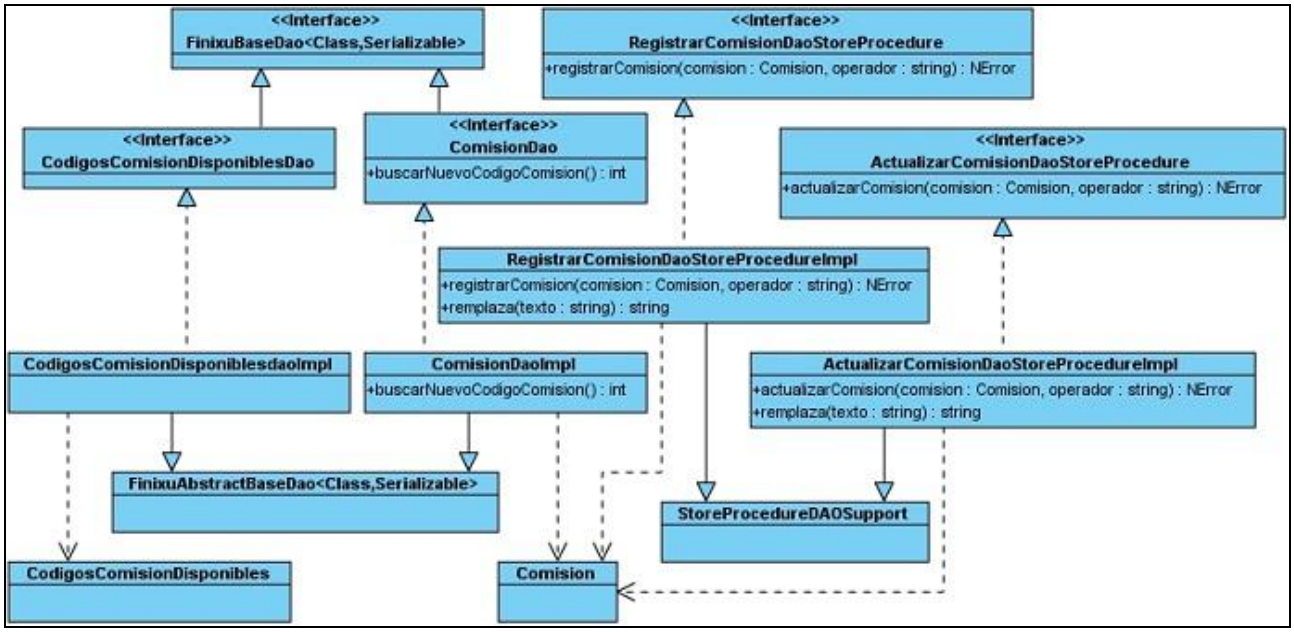

Ilustración 5 Modelo de Acceso a Datos del Módulo Gestionar comisión

Diseño de la Capa de Dominio

En esta capa se encuentran las clases que representan entidades del negocio. Estas clases están presentes en todas las capas anteriormente descritas.

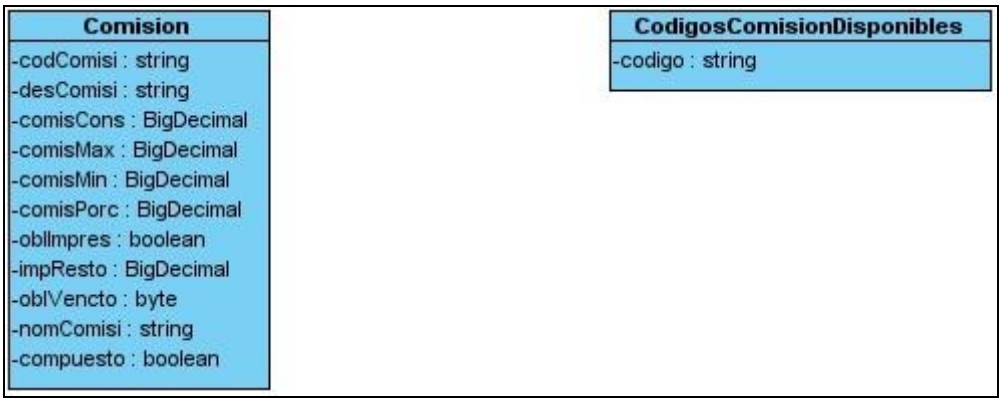

Ilustración 6 Modelo Dominio del Módulo Gestionar comisión

2.3.2.3 Diagrama de Interacción del módulo Gestionar comisión

Por cada diagrama de clases del diseño se modela un diagrama de interacción. Como diagrama de interacción se seleccionó el diagrama de secuencia. En este diagrama se incluyen las interacciones entre las clases del diseño, a través de mensajes. Un mensaje significa una operación en la clase a la que va el mensaje.

A continuación se muestra el diagrama de secuencia para el caso de uso Registrar comisión.

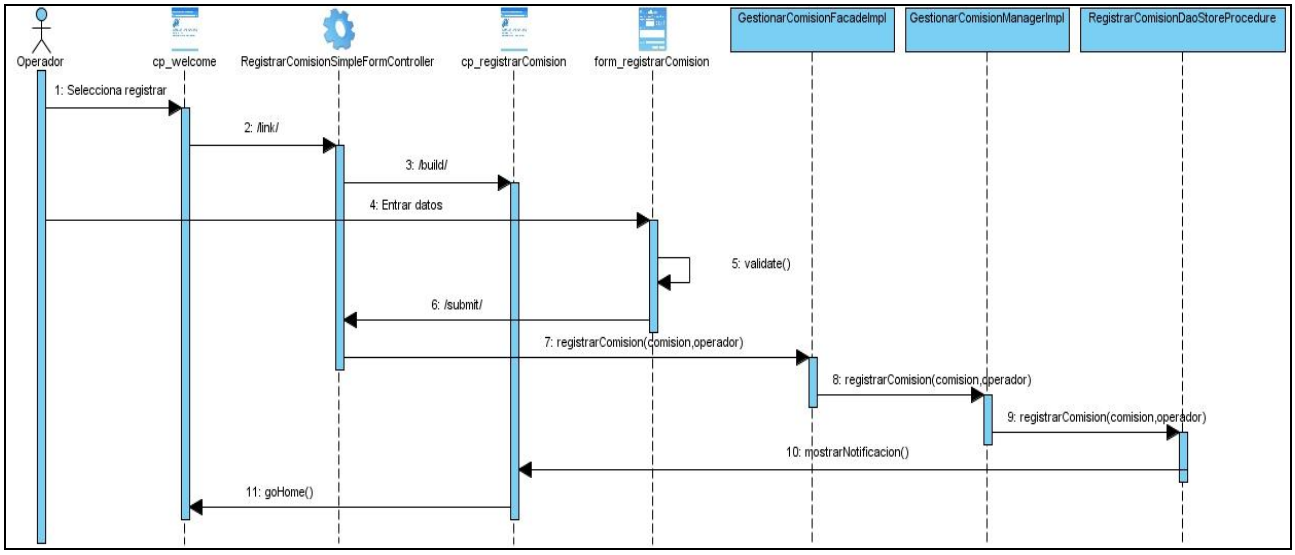

Ilustración 7 Diagrama de secuencia: Registrar Comisión

2.3.3 Modelo de datos

El modelo de datos es un modelo abstracto que describe como deben ser representados y usados los datos. Describe la estructura de la base de datos, así como los datos, sus relaciones y las restricciones que deben cumplirse entre ellos. (18)

La representación del modelo de datos de los módulos diseñados pertenecen a un sistema que ya cuenta con una base de datos en la cual sus tablas no se encuentran relacionadas, para definir las relaciones entre ellas se utiliza el framework Hibernate que permite que se comporten como si estuvieran relacionadas entre sí.

A continuación se representa el Modelo de datos correspondiente al módulo Gestionar comisión.

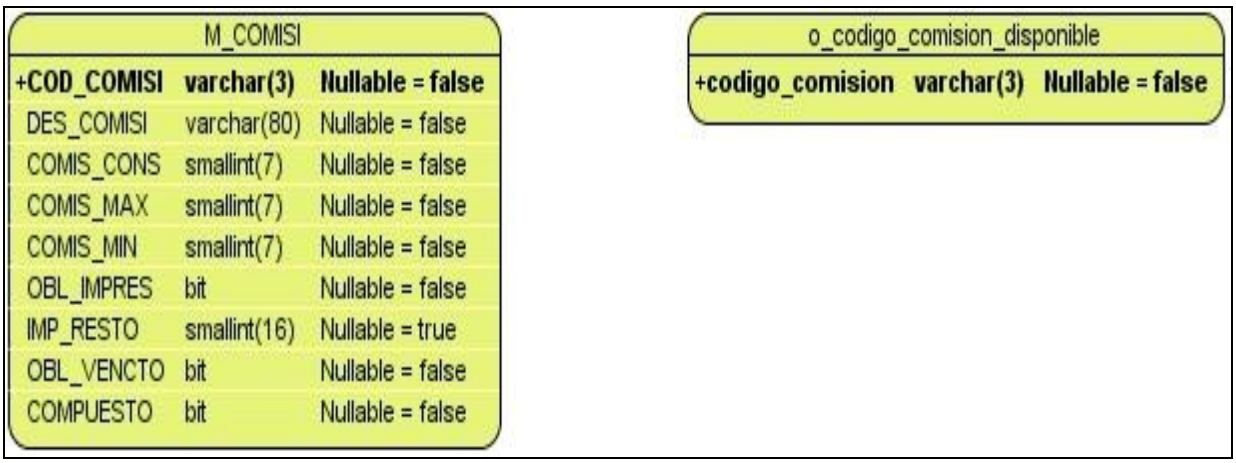

Ilustración 8 Modelo de Datos del módulo Gestionar comisión

2.3.4 Patrones de diseño empleados

El uso de patrones en la elaboración del diseño es muy frecuente en aras de lograr los objetivos esperados. Estos son considerados soluciones simples y experimentadas a problemas específicos y comunes del diseño en el desarrollo de software.

Para el diseño de la solución propuesta, se tuvieron en cuenta varios patrones, dentro de los que se destacan los ***GRASP***, que describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

Los ***patrones GRASP*** utilizados son los siguientes:

Controlador: Las clases controladoras de los módulos, constituyen un ejemplo de la aplicación de este patrón. Las mismas tendrán la responsabilidad de escuchar y responder a las peticiones realizadas por la presentación y de comunicarse con la capa de negocio.

Experto: Este patrón lo podemos encontrar en diferentes clases en el sistema, como son los dominios, las cuales son expertos en tener los atributos que se le definen, encontramos también a las clases DAO como por ejemplo la clase ComisionDAO, la cual es responsable de efectuar la mayoría de las operaciones de listar que se muestran en la página.

Alta cohesión: El sistema en general fue diseñado en módulos definidos a partir de que en los mismos se manejaran la menor cantidad de entidades posibles, de manera tal que las clases pertenecientes a las diferentes capas se le asignaran las responsabilidades necesarias y bien delimitadas para garantizar una alta cohesión generando por ende **bajo acoplamiento**.

Bajo acoplamiento: Este patrón se evidencia a la hora de definir interfaces e implementaciones, como por ejemplo la interfaz GestionarComisionFacade y su implementación GestionarComisionFacadeImpl, las cuales permiten que las clases de la presentación CargarDatosMultiActionController, RegistrarComisionSimpleFormController y otros controladores se relacionen exclusivamente con ellas para realizar sus operaciones, disminuyéndose de esta forma el impacto de cambios posteriores en el negocio del sistema.

Otros patrones utilizados en el diseño son los ***patrones GOF,*** dentro de ellos se encuentra:

Fachada: El manejo de este patrón se evidencia en la definición de la interfaz GestionarComisionFacade, la cual hace función de intermediaria entre la presentación y una interfaz o grupos de interfaces más complejas que se encargan de la lógica de negocio y el acceso a datos.

Patrones ***J2EE*** utilizados:

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

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

Conclusiones del capítulo

La descripción de la arquitectura del sistema Quarxo y el estudio de los patrones de diseño contribuyeron a desarrollar exitosamente el diseño de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio obteniéndose además del estilo arquitectónico utilizado:

- Los diagramas de clases del diseño.
- Los diagramas de secuencia correspondientes a cada caso de uso.
- El Modelo de Datos de los módulos.

Todos estos diagramas proporcionan una entrada apropiada como punto de partida a las actividades de implementación con lo que se logra una mayor organización.

Capítulo 3. Implementación y Prueba

3.1 Introducción

En el presente capítulo se explican y se presentan las implementaciones de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio, modelándose de esta forma los artefactos pertenecientes al flujo de trabajo de implementación. Se muestran fragmentos de código de los principales flujos de la implementación, así como los métodos más relevantes. Se probará la solución mediante las pruebas de unidad con el objetivo de encontrar errores que imposibiliten el cumplimiento de los requisitos funcionales definidos anteriormente.

3.2 Modelo de implementación

La implementación es el principal flujo de trabajo en la fase de construcción. En él se describe como el modelo de implementación transforma el resultado del modelo de diseño en el código final del sistema. Se describe además como los elementos de diseño se implementan en componentes y son organizados de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en los lenguajes de programación utilizados así como la dependencia entre ellos. (18)

3.2.1 Diagrama de componentes

Un diagrama de componentes muestra las dependencias lógicas entre componentes de software, sean éstos fuentes, binarios o ejecutables. También describe los elementos físicos del sistema y sus relaciones muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Este diagrama representa todos los tipos de elementos de software que entran en la fabricación de aplicaciones informáticas, pueden ser simples archivos, paquetes, bibliotecas cargadas dinámicamente, entre otros. (27)

El diagrama de componentes de cada uno de los módulos es similar por ello solo se hará referencia al diagrama del módulo Gestionar comisión. En este diagrama se muestra la interacción de cada uno de sus componentes dentro del módulo y a su vez la interacción del módulo con el resto de los componentes del sistema de los cuales consume servicios.

Dentro de los componentes con los que interactúa el módulo se encuentra el componente **Contabilización** donde se encuentran las clases necesarias para las operaciones de contabilización, el componente **Buscador** el cual es un motor de búsqueda que filtra por diferentes criterios según el módulo donde se emplee, el componente **Global** donde se encuentran las clases con las funcionalidades que son comunes para todos los módulos del

sistema. También el módulo interacciona con el componente de **Seguridad** el cual se encarga de definir los permisos del usuario así como las operaciones a las que tiene acceso y por último el componente **dataAccess** el cual tiene la función de brindar todos los elementos necesarios para llevar a cabo la conexión a la base de datos por lo que los módulos dependen de él para realizar el acceso a datos.

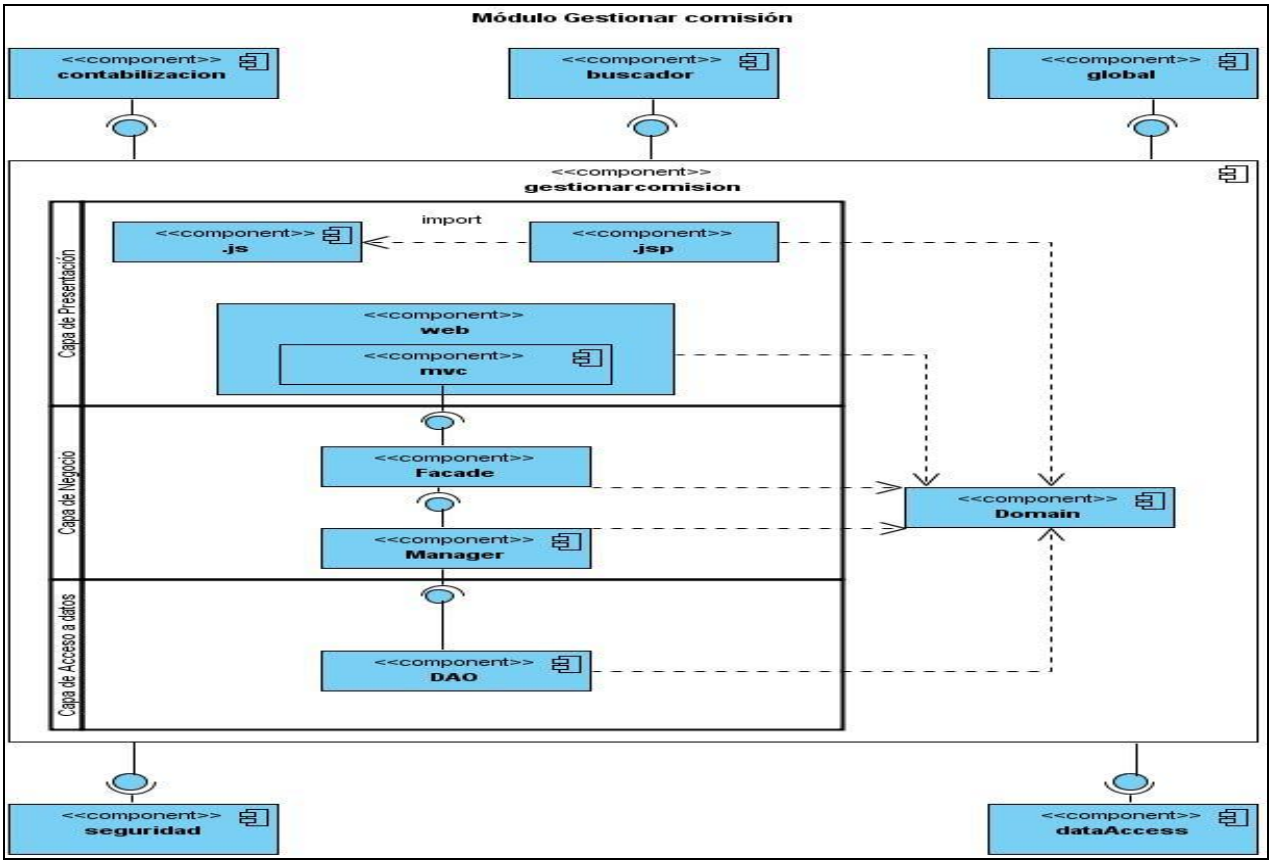

Ilustración 9 Diagrama de componentes del módulo Gestionar comisión

3.3 Estándares de codificación

Un estándar de codificación es una pauta de programación que no está enfocada a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. Un estándar de programación define la nomenclatura de los objetos, variables, métodos y funciones, teniendo que ver además, con el orden y legibilidad del código escrito. (28)

El uso de estándares asegura la legibilidad del código entre distintos desarrolladores, permite la guía para el mantenimiento o actualización del sistema, además de que facilita la portabilidad entre plataformas y aplicaciones.

Con el fin de lograr uniformidad en el desarrollo de la aplicación, se definió una convención de código en cada una de las capas presentes.

3.3.1 Convención de código general

- Los nombres de los paquetes deben escribirse en letra minúscula, representar sustantivos y describir de alguna forma su contenido. Ej. **manager**.
- Las clases e interfaces deben nombrarse comenzando por letra inicial mayúscula, en caso de estar formado por palabras compuestas el inicio de cada palabra deberá comenzar con mayúscula. Ej. **GestionarComisionFacade**.
- Las variables y los nombres de los métodos comienzan siempre con minúscula, en caso de estar compuesto por más de una palabra, estas empezarán con mayúscula exceptuando la primera letra. Ej. **codigo** y **buscarNuevoCodigoComision()**.
- Las constantes se escribirán en letras mayúsculas, si están formadas por más de una palabra se separarán por el signo guión bajo "_". Ej. **FILE_EXTENSIONS**.

3.3.2 Capa de presentación

Las clases pertenecientes al sub-paquete ***mvc*** se nombrarán de la siguiente manera:

- Paquete ***command:*** [nombre de la clase] + [***Command***]. Ej. ***TasaCambioCommand***.
- Paquete ***controller*****:** [nombre de la clase] + [nombre del controlador que se hereda]. Ej. ***CargarDatosMultiActionController***.
- Paquete ***validator:*** [nombre de la clase] + [***Validator***]. Ej. ***ComisionValidator***.
- Paquete ***propertyEditor:*** [nombre de la clase] + [***PropertyEditor***]. Ej. ***ContratoPropertyEditor***.

3.3.3 Capa de Negocio

Las clases se nombrarán de la siguiente manera:

- Paquete ***facade:***
    - *Interfaces:* [nombre del módulo] + [***Facade***]. Ej. ***GestionarComisionFacade***.
    - *Implementación de las interfaces:* [nombre del módulo] + [***FacadeImpl***]. Ej. ***GestionarComisionFacadeImpl***
- Paquete ***manager:***
    - *Interfaces***:** [nombre del negocio] + [***Manager***]. Ej. ***GestionarComisionManager***.

- Clases que implementan las interfaces*:* [nombre del negocio] + [***ManagerImpl***] Ej. ***GestionarComisionManagerImpl***.

3.3.4 Capa de Acceso a Datos

Las clases se nombrarán de la siguiente manera:

- Paquete ***dao***:
  - *Interfaces:* [nombre de la clase dominio que representa] + [***DAO***]. Ej. ***ComisionDAO***.
  - *Clases que implementan las interfaces*: [nombre de la clase dominio que representa] + [***DAOImpl***]. Ej. ***ComisionDAOImpl***.

Si el módulo trabaja con procedimientos almacenados las clases en el paquete ***dao*** se nombrarán:

- *Interfaces*: [nombre de la clase] + ***DaoStoreProcedure.***
  Ej. ***RegistrarComisionDaoStoreProcedure.***
- *Clases que implementan las interfaces*: [nombre de la clase] + ***DaoStoreProcedureImpl***. Ej. ***RegistrarComisionDaoStoreProcedureImpl.***

3.4 Aspectos Principales de la Implementación

3.4.1 Utilización de Spring MVC Framework

Como se propone en el capítulo anterior se utiliza el framework Spring MVC por las ventajas que este brinda para manejar la información de cada uno de los módulos. Su uso facilita el envío de los objetos y otros datos entre los controladores y las páginas *.jsp. Este framework permite encapsular los valores de los campos del formulario dentro de un objeto el cual se envía al controlador y se realiza el registro y actualización de este objeto a través de las otras capas que intervienen en este proceso.

A continuación se describirá brevemente algunos casos de uso de los módulos puesto que no en todos los módulos se procede igual al registrar los datos.

Descripción del caso de uso Registrar comisión del módulo Gestionar comisión

A través de la página registrarComision.jsp se introducirán los datos, los que serán encapsulados en un objeto comisión para ser enviado al controlador el cual se encargará del registro de dichos datos. Este encapsulamiento se logra por medio de la relación de los campos de entrada con los atributos del objeto mediante la etiqueta de Spring `<form:input` y su atributo `path` como se muestra a continuación.

```
<form:form id="formulario" commandName="comision">…

<form:input path="nomComisi" id="nombre"  cssClass="texto"/>…
<form:input path="codComisi" id="codComisi" cssClass="texto"/>…
```

Ilustración 10 Formulario Registrar comisión

Luego de llenar todos los campos necesarios para el registro de la comisión y presionar el botón aceptar estos datos son enviados al controlador **RegistrarComisionSimpleFormController** el cual en el método **onSubmit(…)** se encarga de recibir el objeto enviado con los datos y realizar las validaciones del lado del servidor además de llamar al método de la fachada para completar el registro de los datos y verificar si ocurrió algún error mientras se realizó la operación o si se realizó de manera satisfactoria, devolviendo una notificación al usuario como se muestra en el bloque de código del controlador.

```
@Override
protected ModelAndView onSubmit(HttpServletRequest request,
HttpServletResponse response, Object command, BindException errors)throws
Exception {

	Comision comision=(Comision) command;
	comision.setImpResto(new BigDecimal("0"));
	if(comision.getComisPorc()==null)
	comision.setComisPorc(new BigDecimal("0"));
	if(comision.getComisMax()==null)
	comision.setComisMax(new BigDecimal("0"));
	if(comision.getComisMin()==null)
	comision.setComisMin(new BigDecimal("0"));
	if(comision.getComisCons()==null)
	comision.setComisCons(new BigDecimal("0"));
	String userName = UserHandler.getUser().getCodUsuari();
	NError e =gestionarComisionFacade.registrarComision(comision,
username);

	Map<String,String> result = new HashMap<String,String>();
	if(e.isError()){
	CError error =
gestionarComisionFacade.obtenerMensajeError(e.getCodigo());
	result.put("respuesta", error.getTextoError());

	}else
		result.put("respuesta", "ok");
	return new ModelAndView("../common/responseView",result);
}
```

Ilustración 11 Implementación de la funcionalidad Registrar comisión

Descripción del caso de uso Actualizar comisión del módulo Gestionar comisión

Para actualizar una comisión se utiliza el mismo patrón para la implementación, utilizando la asociación de los campos de entrada con los atributos del objeto a actualizar como en el caso de uso Registrar comisión. Así como también se utiliza un controlador que hereda del controlador SimpleFormController del framework Spring MVC. Sin embargo cambia el flujo de la acciones ya que primeramente se selecciona una determinada comisión a actualizar y se envía el código al controlador y este es el encargado de llamar el método especializado en buscar la comisión con ese código seleccionado para luego retornar el objeto comisión con sus valores al formulario de la página actualizarComision.jsp. A continuación se muestra el código del método encargado de realizar esta operación.

```
@Override
protected Object formBackingObject(HttpServletRequest request)
throws Exception {
 String codigo=request.getParameter("codigo");
 if(codigo != null){
  Comisión comision =gestionarComisionFacade.consultarComision(codigo);
      return comision;
 }
 return super.formBackingObject(request);
}
```

Ilustración 12 Implementación de la funcionalidad Actualizar comisión

Luego de mostrar el objeto el usuario puede actualizar los campos de la comisión seleccionada y luego salvar los nuevos valores introducidos. El resto de esta operación se realiza de la misma forma que se registra una nueva comisión.

Esta implementación de los caso de usos del módulo Gestionar comisión es similar en el módulo Comisión a transacción por ello no se hará referencia a su implementación ya que sigue las mismas pautas de implementación.

Descripción del caso de uso Registrar tasa de cambio del módulo Tasa de cambio

En este módulo el registro de las tasas de cambio puede ser de 2 formas diferentes, la primera de forma manual donde el usuario introduce los valores y la segunda desde un fichero determinado donde se encuentran los tipos de cambios a introducir.

Registrar tasa de cambio manual

Luego de introducir los valores de las tasas a registrar en la página registrarTasaCambio.jsp se envían los datos a la clase controladora que hereda del controlador AbstractController de Spring

MVC el cual en el método **handleRequestInternal(…)** procesa la información y devuelve a la vista una notificación de si ocurrió algún error o si la operación se realizó de forma satisfactoria como se muestra a continuación.

```
@Override
protected ModelAndView handleRequestInternal(HttpServletRequest request,
HttpServletResponse response) throws Exception {
int cont=Integer.parseInt(request.getParameter("cont"));
String [] auxDatos=request.getParameterValues("datos");
String fecha=request.getParameter("fecha");
SimpleDateFormat format=new SimpleDateFormat("dd/MM/yyyy");
Date fechaActualizacion=null;
fechaActualizacion=format.parse(fecha);
String col=request.getParameter("campos");
String [] campos=col.split(",");
Map<NomMoneda,List<TasaDeCambio>> map=new HashMap<NomMoneda,
List<TasaDeCambio>>();
List<NomMoneda> monedas=tasaDeCambioFacade.listarMonedas();
int pos=cont+1;
(…)
NError error=tasaDeCambioFacade.registrarTasas(userName,
fechaActualizacion, map);
JSONObject json = new JSONObject();
if(error.getCodigo()==0){
json.put("error", "");
}else{
String textoError="";
CError cError = tasaDeCambioFacade.obtenerError(error.getCodigo());
if(cError != null)
textoError = cError.getTextoError();
json.put("error", textoError+"  Detalles: "+error.getDetalles());
}
try {
      ResponseUtil.escribirDatosEnElResponse(response, json);
      } catch (IOException e) {
            e.printStackTrace();}
      return null;}
```

Ilustración 13 Implementación de la funcionalidad Registrar tasa de cambio manual

Registrar tasa de cambio desde archivo

Después de introducir los valores de los campos y cargar la dirección del fichero donde se encuentran las diferentes tasas a registrar se procede al procesamiento de los datos. Para ello en la clase **CargarDatosTasaCambioMultiActionController** el método **cargarArchivo(…)** se encarga de verificar si la extensión del fichero es la correcta así como los datos que contiene son válidos y por último se muestra el éxito o no de la operación. Seguidamente se muestra la implementación de dicha funcionalidad.

```
public void cargarArchivo(HttpServletRequest request,
HttpServletResponse response) {
logger.info("Processing file request...");
boolean isMultipartContent =
ServletFileUpload.isMultipartContent(request);
String fech = request.getParameter("fecha");
String userName = UserHandler.getUser().getCodUsuari();
SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
Date fecha = null;
try {
fecha = sdf.parse(fech);
} catch (ParseException e1) {
      e1.printStackTrace();
}
JSONObject json = new JSONObject();
if (isMultipartContent) {
try {
logger.info("Starting load...");
List fields = upload.parseRequest(request);
Iterator it = fields.iterator();
while (it.hasNext()) {
FileItem fileItem = (FileItem) it.next();
boolean isFormField = fileItem.isFormField();
if (!isFormField) {
      if (isDBFFile(fileItem)) {
FicheroTipoCambiofichero =
cambioFacade.obtenerFichero(fileItem.getName().substring(0,
fileItem.getName().indexOf(".")),
fileItem.getName().substring(fileItem.getName().lastIndexOf('.') + 1));
      if (fichero != null) {
      DBFReader db = new DBFReader(fileItem.getInputStream());

      (…)

json.put("error", mensaje);
} else {
json.put("error", "ok");
}} else {
json.put("error2", "archivoNoReconocido");// archivo no reconocido
}break;
}else
json.put("error2", "archivoNoValido");
}}
logger.info("Load terminated...");
} catch (FileUploadException e) {
json.put("error2", "archivoNoValido");
} catch (DBFException e) {
json.put("error2", "archivoNoValido");
} catch (IOException e) {
json.put("error2", "archivoNoValido");
}}
datos = json;
```

Ilustración 14 Implementación de la funcionalidad Registrar tasa de cambio desde archivo

La implementación del caso de uso Actualizar tasa de cambio es similar al resto de los actualizar por lo que no se hace mención a dicho código por seguir el mismo patrón de implementación usando el SimpleFormController de Spring MVC y la asociación de los atributos del objeto con los campos del formulario.

3.5 Descripción de las clases y las funcionalidades de los módulos

A continuación se describirán las clases de mayor peso de los módulos del punto de vista funcional.

Módulo Gestionar comisión

Nombre: CargarDatosMultiActionController	
Tipo de clase: Controladora	
Atributos	**Tipo**
comisionFacade	GestionarComisionFacade
Para cada responsabilidad	
Nombre:	**Descripción:**
listarCodigosDisponibles(HttpServletRequest request, HttpServletResponse response)	Devuelve los códigos disponibles para las comisiones.
comprobarCodigoComision(HttpServletRequest request, HttpServletResponse response)	Comprueba si el código seleccionado ya está asociado a una comisión o no.
cargarDatosComision(HttpServletRequest request, HttpServletResponse response)	Devuelve el nombre de las comisiones existentes.
obtenerNuevoCodigoComision(HttpServletRequest request, HttpServletResponse response)	Devuelve un nuevo código de comisión.

Nombre: GestionarComisionManagerImpl	
Tipo de clase: Manager	
Atributos	**Tipo**
actualizarComisionDAOStoreProcedure	ActualizarComisionDaoStoreProcedure
registrarComisionDaoStoreProcedure	RegistrarComisionDaoStoreProcedure
nomencladorContabilizacionFacade	NomencladorContabilizacionFacade
codigosComisionDisponiblesDAO	CodigosComisionDisponiblesDAO
globalFacade	GlobalFacade
Para cada responsabilidad	
Nombre:	**Descripción:**
registrarComision(Comision comision, String operador)	Recibe una comisión por parámetros y la registra en la BD a través de ssp_OpenFilaTable y devuelve un objeto NError con el error correspondiente en caso de que exista.

actualizarComision(Comision comision, String operador)	Actualiza la comisión pasada por parámetros a través de actualizarComisionDAOStoreProcedure.
consultarComision(String codigo)	Devuelve la comisión con el código pasado por parámetro.
existeCodigo(String cod)	Comprueba si el código seleccionado ya está asociado a una comisión o no.
obtenerMensajeError(int codigo)	Devuelve el mensaje de error dado el código pasado, lo obtiene de globalFacade.
paginarCodigoComisionDisponibles(int posicionInicial, int cantidadElementos, String value)	Devuelve un paginado de los códigos de las comisiones disponibles.
listarComision()	Devuelve todas las comisiones existentes.
buscarNuevoCodigoComision()	Devuelve un nuevo código de comisión.

Módulo Comisión a transacción

Nombre: CargarDatosMultiActionController	
Tipo de clase: Controladora	
Atributos	**Tipo**
gestionarComisionATransaccionFacade	GestionarComisionATransaccionFacade
Para cada responsabilidad	
Nombre:	**Descripción:**
listarTransacciones(HttpServletRequest request, HttpServletResponse response)	Devuelve todas las transacciones existentes.
listarDesglose(HttpServletRequest request, HttpServletResponse response)	Devuelve los desgloses asociados a una transacción determinada.
listarComisiones(HttpServletRequest request, HttpServletResponse response)	Devuelve todas las comisiones existentes.
listarConceptos(HttpServletRequest request, HttpServletResponse response)	Devuelve todos los conceptos existentes.
eliminarComisionATransaccion(HttpServletRequest request, HttpServletResponse response)	Elimina la asociación de una comisión a una transacción.
consultarIdComisionATransaccion(HttpServletRequest request, HttpServletResponse response)	Comprueba si el id seleccionado para una ComisionATransaccion ya está registrado.

Módulo Tasa de cambio

Nombre: CargarDatosTasaCambioMultiactionControlller	
Tipo de clase: Controladora	
Atributos	**Tipo**
cambioFacade	TasaDeCambioFacade
upload	ServletFileUpload
FILE_EXTENSIONS[] = { "dbf", "DBF" }	String
datos	JSONObject
Para cada responsabilidad	
Nombre:	**Descripción:**
listarTiposDeCambio(HttpServletRequest request, HttpServletResponse response)	Devuelve los diferentes tipos de cambios.
listarMonedas(HttpServletRequest request, HttpServletResponse response)	Devuelve las monedas y los valores de los tipos de cambios seleccionados para cada una de las monedas.
cargarArchivo(HttpServletRequest request, HttpServletResponse response)	Carga y procesa el fichero de las tasas de cambio.
devolverDatos(HttpServletRequest request, HttpServletResponse response)	Devuelve el resultado de la operación de carga y procesamiento del fichero de las tasas de cambio.
mostrarTiposAsociados(HttpServletRequest request, HttpServletResponse response)	Muestra los tipos de cambios asociados a las monedas en el fichero de las tasas de cambio.
isDBFFile(FileItem file)	Comprueba si el fichero a cargar es un fichero DBF.

3.6 Realización de pruebas

En el desarrollo de un software es necesario comprobar si cumple con cada uno de los requisitos funcionales además de verificar si existen errores o no en la implementación final. Una actividad eficiente para detección de errores es la realización de pruebas.

3.6.1 Niveles de prueba de software

Las pruebas están divididas en diferentes niveles que se aplican en diferentes momentos del ciclo de vida del software, estos son:

- **Unidad:** son aplicables a módulos del software. Este tipo de pruebas se simplifican cuando se diseña con un alto grado de cohesión. Se procede a comprobar las interfaces de usuarios, la estructura de los datos locales, las condiciones límites, los caminos independientes y los caminos de manejo de errores.
- **Integración:** se orientan a detectar fallos provocados por una incorrecta comunicación entre módulos. El software se puede ejecutar en un contexto de hardware concreto, por lo que la Prueba de Sistema es la que se encarga de buscar errores en este ensamblaje software/hardware. Finalmente el usuario ha de realizar la Prueba de Aceptación final sobre el sistema completo.
- **Sistema:** se centran principalmente en verificar la interacción entre los actores y el sistema, por lo que a menudo los casos de pruebas se obtienen a partir de las descripciones de los casos de uso.
- **Aceptación:** la mayoría de los desarrolladores de productos de software llevan a cabo un proceso denominado pruebas Alfa o Beta para descubrir errores que considera solo el usuario final puede descubrir. La prueba Alfa es llevada a cabo por el cliente en el lugar de desarrollo. Las pruebas Betas se llevan a cabo por los usuarios finales en sus puestos de trabajo, y a diferencia de la prueba Alfa el desarrollador no está presente, por lo que el cliente debe registrar todas las inconformidades y tramitárselas al desarrollador. (29)

Las pruebas desarrolladas para la implementación de los módulos se enfocan en el nivel de unidad utilizando los diferentes tipos de prueba, ya sea las pruebas estructurales o de caja blanca o las pruebas funcionales o de caja negra.

3.6.1.1 Diseño de los casos de prueba para caja blanca

Las pruebas de caja blanca se centran en como diseñar los casos de prueba atendiendo al comportamiento interno y la estructura del programa. Se examina así la lógica interna del programa sin considerar los aspectos de rendimiento. El objetivo de la técnica es diseñar casos de prueba para que se ejecuten, al menos una vez, todas las sentencias del programa, y todas las condiciones tanto en su vertiente verdadera como falsa. (29)

Para el desarrollo de esta prueba se utilizó JUnit el cual es una herramienta gratuita y multiplataforma, de fácil integración con el IDE de desarrollo Eclipse, además de permitir probar la interacción de los módulos con los demás módulos del sistema y poseer una interfaz sencilla que informa los posibles resultados de las pruebas.

Para la realización de las pruebas se diseñó primeramente los casos de prueba definiendo las funcionalidades a probar y seguidamente los datos de entrada y el resultado esperado. A continuación se realizaron las configuraciones de los ficheros para la comunicación de las diferentes capas de la aplicación y de los módulos y por último se llevaron a cabo las diferentes pruebas diseñadas.

A continuación se muestra el diseño del caso de prueba de la funcionalidad a probar en el módulo Gestionar comisión.

Caso de prueba para la funcionalidad Consultar comisión	
Descripción	Debe consultar una comisión y devolver su nombre.
Condición de ejecución	El código de la comisión no puede estar vacío
Entrada	codComisi = 025
Resultados esperados	Se obtiene el nombre de la comisión: "CIERRE DE CUENTA" satisfactoriamente.

Tabla 1 Caso de prueba para el caso de uso Consultar comisión del módulo Gestionar comisión

Seguidamente se mostrarán diferentes imágenes con la implementación de la prueba realizada y su descripción.

```
public class PruebaModulosTest extends TestCase {

    private GestionarComisionFacadeImpl facadeImpl;
    private GestionarComisionATransaccionFacadeImpl transaccionFacadeImpl;
    private TasaDeCambioFacadeImpl tasaDeCambioFacade;
    private SessionFactory sf;

    @Before
    protected void setUp() throws Exception {

        ApplicationContext context = new ClassPathXmlApplicationContext(
                "classpath:cu/uci/finixubnc/contabilidad/test/*.xml");

        facadeImpl = (GestionarComisionFacadeImpl) context
                .getBean("gestionarComisionFacade");
        transaccionFacadeImpl = (GestionarComisionATransaccionFacadeImpl) context
                .getBean("gestionarComisionATransaccionFacade");
        tasaDeCambioFacade = (TasaDeCambioFacadeImpl) context
                .getBean("tasaDeCambioFacade");

        sf = (SessionFactory) context.getBean("finixuSessionFactory");
        TransactionSynchronizationManager.bindResource(sf, new SessionHolder(sf
                .openSession()));

        super.setUp();
    }
```

Ilustración 15 Declaración e inicialización de los atributos de la clase PruebaModulosTest

En la imagen número 15 se muestra la clase creada que hereda de la clase *TestCase* de la librería de JUnit para realizar las pruebas. Se declaran los diferentes atributos como son las fachadas de cada módulo y el objeto *SessionFactory* para la inicialización de cada uno de los atributos a partir de los beans declarados en los diferentes contextos de Spring. El método ***setUp()*** es el encargado de inicializar cada atributo declarado.

Luego de inicializar los atributos se procede a la implementación del método de prueba definido en el caso de prueba.

```
@Test
public void testConsultarComision() {
    String cod = "025";
    String esperado = "CIERRE DE CUENTA";
    String var = facadeImpl.consultarComision(cod).getNomComisi();
    assertEquals(esperado, var);
}
```

Ilustración 16 Método de prueba para el caso de uso Consultar Comisión del módulo Gestionar comisión

Primeramente se define en el método el código de la comisión a consultar y seguidamente el texto esperado como resultado. Seguidamente se implementa la llamada del método de la fachada el cual pasándole el parámetro código debe devolver una comisión de la cual se le asigna a la variable *var* el nombre de la comisión el cual es comparado internamente por el método ***assertEquals(…)***.

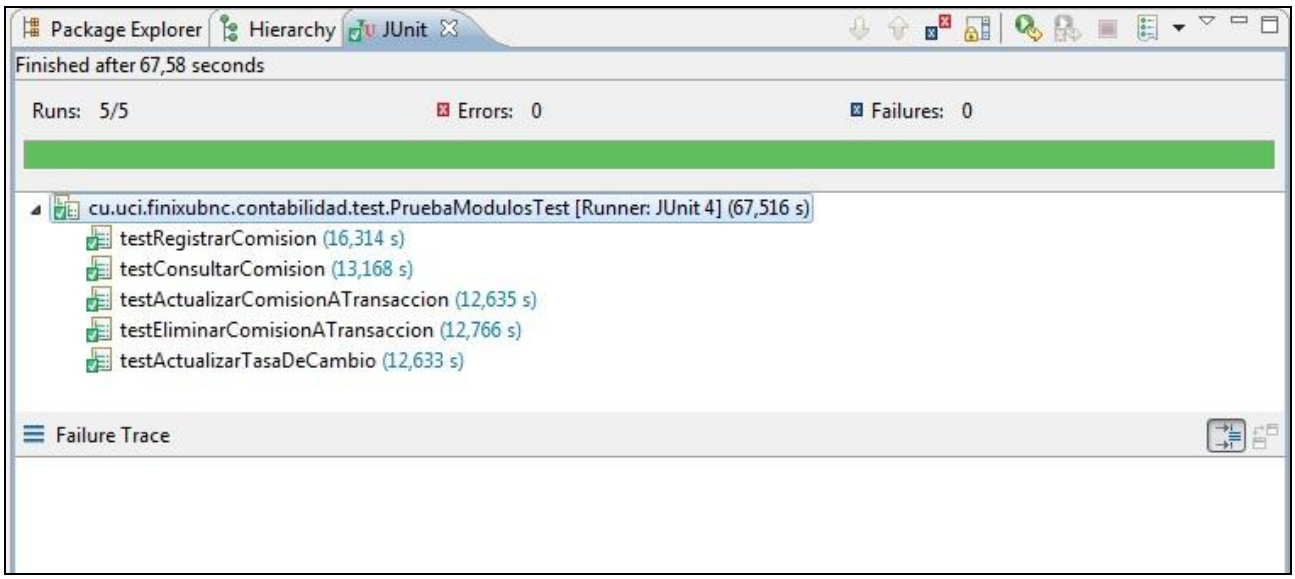

Ilustración 17 Resultado de las pruebas realizadas con el framework JUnit

El resultado de la ejecución del método de prueba es mostrado en una consola de la herramienta JUnit como se muestra anteriormente. En este caso se muestra el resultado de 5 pruebas realizadas a los módulos el cual fue satisfactorio. El resto de las pruebas pueden ser consultadas en los anexos de documento.

3.6.1.2 Diseño de casos de prueba para caja negra

Estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba. El componente se ve como una "Caja Negra" cuyo comportamiento sólo puede ser determinado estudiando sus entradas y las salidas obtenidas a partir de ellas. No obstante la realización de las pruebas con las diferentes posibles entradas y salidas de cada una de las funcionalidades resulta impracticable por lo que se selecciona solo un conjunto de datos con los que se realizarán las pruebas. (29)

Para seleccionar el conjunto de entradas y salidas sobre las que trabajar, hay que tener en cuenta que en todo programa existe un conjunto de entradas que causan un comportamiento erróneo en nuestro sistema, y como consecuencia producen una serie de salidas que revelan la presencia de defectos. Entonces, dado que la prueba exhaustiva es imposible, el objetivo final es encontrar una serie de datos de entrada cuya probabilidad de pertenecer al conjunto de entradas que causan dicho comportamiento erróneo sea lo más alto posible.

Para confeccionar los casos de prueba de Caja Negra existen distintos criterios. Algunos de ellos son:

- **Técnica de Partición de Equivalencia:** Esta técnica divide el dominio de entrada en clases de datos que tienden a ejercitar determinadas funciones de la solución informática y descubrir errores. Está representada por un conjunto de estados válidos y no válidos.
- **Técnica de Análisis de Valores Límites:** Prueba la habilidad de la solución informática para manejar datos que se encuentran en los límites aceptables. Es el desarrollo de pruebas para generar el número máximo o mínimo de datos.
- **Técnica de Grafos de Causa-Efecto:** Es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

Al módulo se le realizaron pruebas utilizando la técnica de Partición de Equivalencia ya que es una de las más efectivas pues permite definir casos de prueba que descubran clases de errores, reduciendo el número de casos de prueba a desarrollar.

Para la realización de las pruebas de caja negra se diseñaron los casos de pruebas con diferentes conjuntos de datos. En los anexos se especifica el diseño para un caso de uso del módulo Gestionar comisión.

Luego de la realización de los diferentes métodos de prueba se obtuvieron resultados satisfactorios tanto interno como funcional puesto que los módulos mantuvieron un correcto comportamiento ante los diferentes escenarios en los que se probaron.

Los módulos también fueron probados por el departamento de calidad de la UCI (CALISOFT) donde se le realizaron pruebas funcionales, de carga y estrés donde se encontraron 11 no conformidades en las 3 iteraciones realizadas las cuales permitieron corregir los errores encontrados y posteriormente liberar el producto. Luego de ser liberado pasó a las pruebas de aceptación del cliente que se realizaron en el Banco Nacional de Cuba donde los resultados arrojados son satisfactorios contribuyendo esto a un alto grado de aceptación por el cliente. Por tanto los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio se encuentran completamente disponibles para su explotación como parte importante del sistema Quarxo en el Banco Nacional de Cuba.

Conclusiones del capítulo

En el presente capítulo se obtuvo la solución de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio para el sistema Quarxo. Además se mostraron los elementos principales relacionados con la etapa de implementación y prueba en los cuales se obtuvo:

- El diagrama de componentes.
- La descripción de algunas de las funcionalidades implementadas.
- La descripción de los diseños de los casos de prueba tanto para caja blanca como caja negra y la implementación de las pruebas de caja blanca.

El desarrollo de estas actividades permitió obtener una correcta implementación así como la validación de la solución garantizando la satisfacción plena de las necesidades reales de los usuarios y demandas del cliente, cumpliendo con los objetivos trazados inicialmente.

Conclusiones Generales

Tras concluir el desarrollo del presente trabajo de diploma se obtuvo como resultado el logro de alcanzar el objetivo general planteado, además de las tareas definidas:

- ✓ Se identificó luego de realizar un estudio sobre los diferentes sistemas contables, las principales dificultades para su utilización como son los altos costos de sus licencias o la no existencia de todas las funcionalidades para cubrir los diferentes procesos de gestión. Determinándose así que era más factible y necesario para el país y el BNC el desarrollo de un sistema informático que permitiera automatizar los requerimientos actuales del BNC.

- ✓ Se estudiaron y caracterizaron un conjunto de tecnologías y herramientas a utilizar en el desarrollo del sistema además de la utilización de patrones de diseño que posibilitaron la reutilización del código, el desarrollo rápido y eficaz y la facilitación del mantenimiento futuro del sistema.

- ✓ Se realizó el diseño de los módulos Gestionar comisión, Comisión a transacción y Tasa de cambio a partir de los requisitos previamente definidos por los analistas además de las descripciones de los casos de usos, lo cual sirvió como punto de partida para el desarrollo de la implementación.

- ✓ Se implementaron las diferentes funcionalidades de cada módulo de forma exitosa lo que garantizará una eficiente gestión de los procesos de las comisiones, las comisiones a transacciones y las tasas de cambio en el BNC. Lo cual se pudo validar a través de las diferentes pruebas unitarias realizadas, las cuales arrojaron resultados positivos, demostrando que cada módulo cumple con los requisitos que permiten su correcto funcionamiento. Dándole cumplimiento de esta manera al problema científico planteado para esta investigación.

Recomendaciones

- ✓ Continuar los estudios del tema con el objetivo de encontrar nuevas funcionalidades para futuras versiones del sistema.
- ✓ Desarrollar una versión del sistema genérica, que se pueda utilizar en cualquier entidad bancaria mundial.
- ✓ Desarrollar una versión del sistema que utilice Postgres como gestor de base de datos.

Bibliografía

1. Estudio de la Contabilidad General. [Online] http://bibliodoc.uci.cu/pdf/contabilidad_g.pdf.
2. Banco de España. [En línea] http://www.bde.es/.
3. **Rahnema, Ahmad.** Sócrates: Investigación y Publicaciones de IEEM. [En línea] 2010. http://socrates.ieem.edu.uy/.
4. **Apache**. *Documentación del Servidor HTTP Apache 2.0.* [En línea] http://httpd.apache.org/docs/2.0/es/.
5. **A. Russell, Matthew.** *Dojo The Definitive Guide.* 2008. 978-0-596-51648-2.
6. **Christian Bauer, Gavin King.** *Hibernate in Action.* s.l.: Manning Publications Co, 2005. 1932394-15-X.
7. **Craig Walls, Ryan Breindenbach.** *Spring In Action.* s.l.: Manning Publications Co, 2005. 1-932394-35-4.
8. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns.* s.l.: Addison-Wesley Pub Co, 1995.
9. **Iglesias, Adolfo Miguel.** *Documento de Arquitectura, Modernización Sistema del Banco Nacional de Cuba.* 2009.
10. **Ivar Jacobson, Grady Booch, James Rumbaugh.** *El Proceso Unificado de Desarrollo de Software.* s.l.: Pearson Educación, S.A, 2000. 84-7829-036-2.
11. **Larman, Craig.** *UML y Patrones.* s.l.: Prentice Hall, 1999. 970-17-0261-1.
12. Manual de XHTML. [En línea] http://manual-xhtml.blogspot.com/2006/05/primer-documento-xhtml.html.
13. Microsoft SQL Server 2005. [En línea] http://www.microsoft.com/sqlserver/2005/en/us/product-information.aspx.
14. **Pérez, Javier Eguíluz.** *Introducción a JavaScript.* 2008.
15. Pruebas de Software. [En línea] http://lsi.ugr.es/~ig1/docis/pruso.pdf.

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

17. Visual Paradigm. [En línea] http://www.visual-paradigm.com.

18. Consumer. [En línea] Mayo de 2002. http://revista.consumer.es/web/es/20020501/pdf/informe.pdf.

Referencias bibliográficas

1. **Maldonador, R.** *Estudio de la Contabilidad General.* La Habana: Félix Varela, 2006.
2. **Torres Tovar, Juan Carlos.** *Introducción a la contabilidad I.*
3. **Medina María de los Ángeles.** Campos de la aplicación de la contabilidad. [En línea] http://www.scribd.com/doc/50595325/CAMPOS-DE-LA-APLICACION-DE-LA-CONTABILIDAD.
4. Gestión y Administración. [En línea] http://www.gestionyadministracion.com/contabilidad/contabilidad-bancaria.html.
5. **González Olomedo Paula.** El Prisma. [En línea] http://www.elprisma.com/apuntes/economia/monedabanca/.
6. **Banca fácil.** [En línea] http://www.bancafacil.cl/bancafacil/servlet/Contenido?indice=1.2&idPublicacion=1500000000000013&idCategoria=2
7. **BNC.** Manual de Instrucciones y procedimientos del Banco Nacional de Cuba. 2007.
8. Definición.de. [En línea] 2008. http://definicion.de/.
9. Gestiopolis. [En línea] 2008. http://www.gestiopolis.com/recursos/experto/catsexp/pagans/eco/15/tasacambioysmi.htm.
10. **Mascareñas, Juan.** Gaceta Financiera. [En línea] enero de 2001. http://www.gacetafinanciera.com/.
11. Banco de España. [En línea] http://www.bde.es/.
12. **Monetos.** *Monetos.* [En línea] http://www.monetos.es/financiacion/prestamos/comisiones-bancarias/tipos/.
13. **Tamargo, Lourdes Cerezal.** BETSIME. *La contabilidad en una nueva tecnología.* [En línea] Febrero de 2002. http://www.betsime.disaic.cu.
14. Global SAP. [En línea] http://www.sap.com/.
15. **Aspel.** *Aspel-Banco 3.0.* [En línea] 2004. http://www.aspel.com.mx/mx/productos/banco1.html.

16. **Cabrera González, Lic. Miguel P., y otros.** *XV FORUM DE CIENCIA Y TECNICA SISTEMA ECONÓMICO INTEGRADO VERSAT Sarasola.* 2004.

17. **Tamargo, Lourdes Cerezal.** [En línea] 2009. http://www.betsime.disaic.cu/secciones/tec_feb_02.htm#1.

18. **Jacobson, I.** *El proceso Unificado de Desarrollo de Software.*

19. **RSC.** *Rational Software Corporation.* 2002.

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

21. Sitio Oficial de Hibernate. [En línea] http://www.hibernate.org.

22. Sitio Oficial de Eclipse. [En línea] http://www.eclipse.org.

23. Sitio Oficial Visual Paradigm. [En línea] http://www.visual-paradigm.com.

24. Sitio Oficial Apache Tomcat. [En línea] http://tomcat.apache.org.

25. Sitio Oficial de Spring. [En línea] http://www.springframework.org.

26. IGP SQL Server .*Información General del Producto SQL Server 2005.* [En línea] http://www.microsoft.com/spain/sql/productinfo/overview/default.mspx.

27. **García Fernández Emilio De Jesús.** Diagrama de componentes y objetos. [En línea] 2010. http://ingenieriasoftwaredos.wikispaces.com/Diagrama+de+componentes+y+objetos.

28. **Ing. Javier Ramírez Hernández, Ing. Henry Ernesto Bermúdez Pérez.** Solución base para el desarrollo de sistemas de control logístico de CEDRUX. http://semanatecnologica.fordes.co.cu/ocs-2.3.2/public/site/280.pdf.

29. **Artola Luis.** *Tipos de pruebas automatizadas de software*. [En línea] 2009. http://www.programania.net/diseno-de-software/tipos-de-pruebas-automatizadas-de-software/.

Anexos

Anexo 1: Diseño del módulo Comisión a transacción.

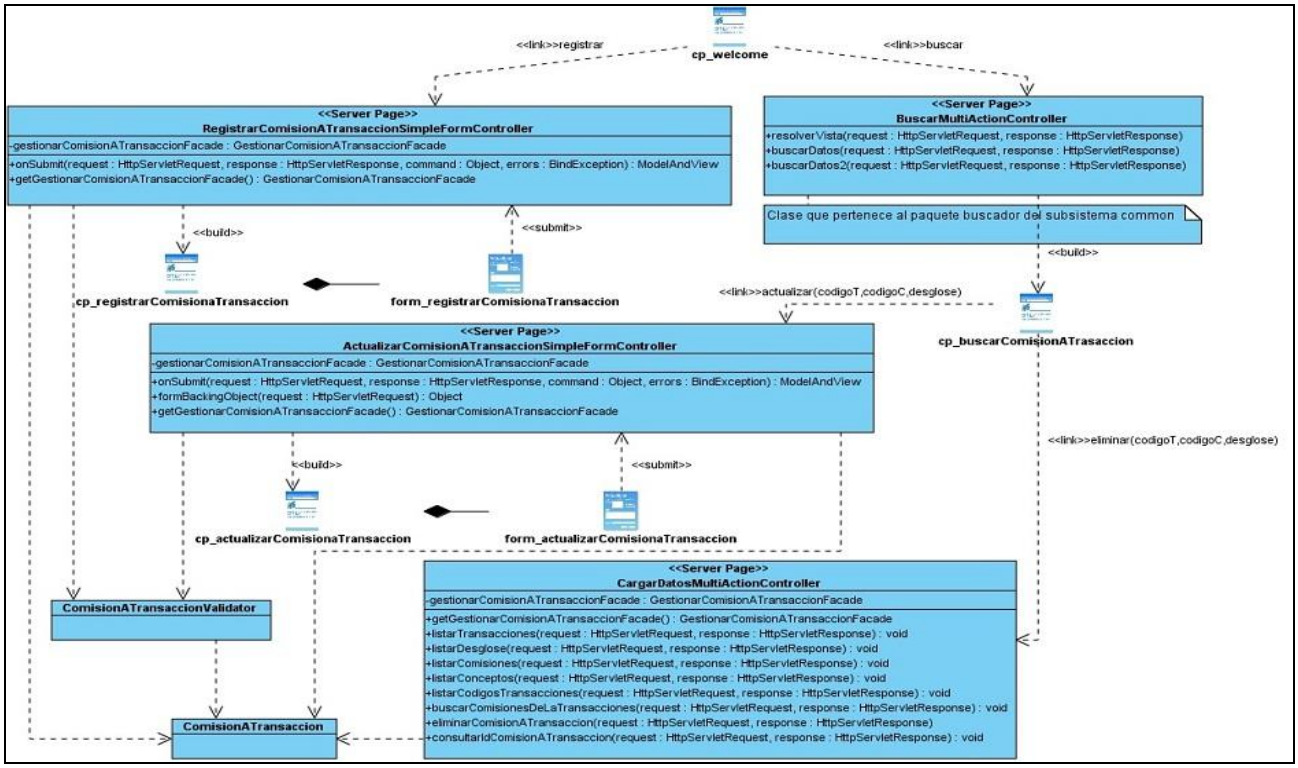

Ilustración 18 Diseño de la capa de Presentación del módulo Comisión a transacción

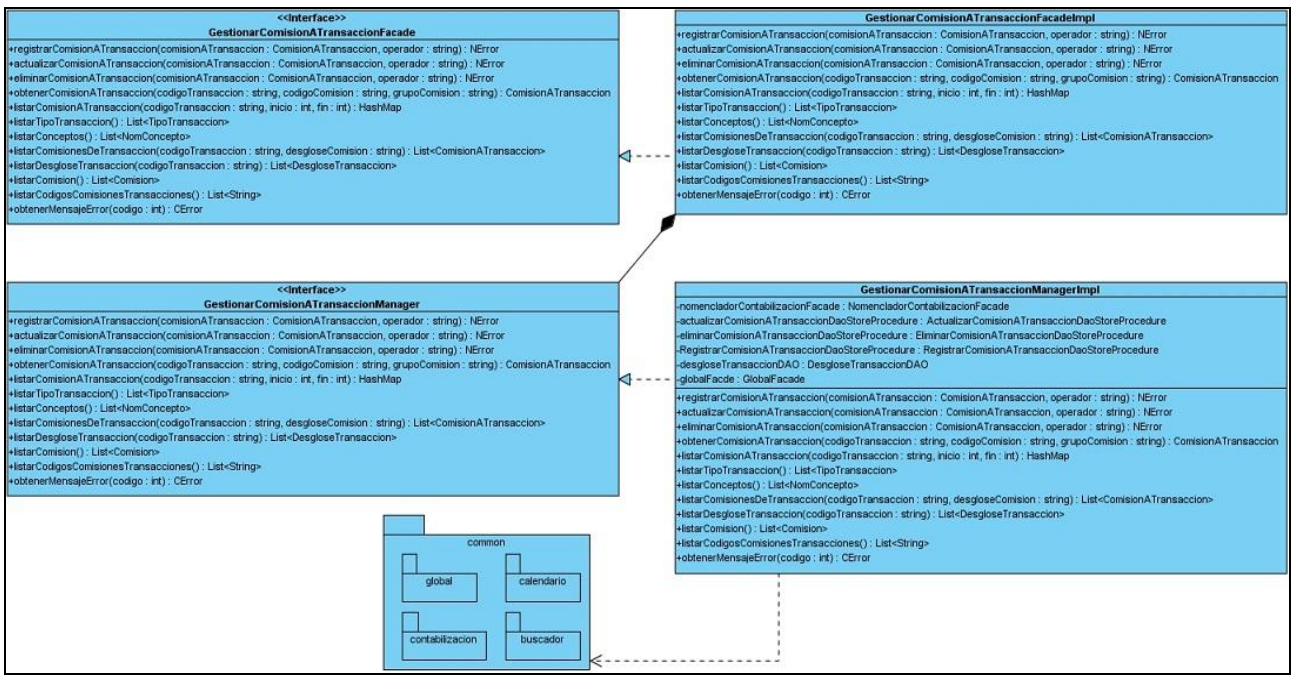

Ilustración 19 Diseño de la capa de Negocio del módulo Comisión a transacción

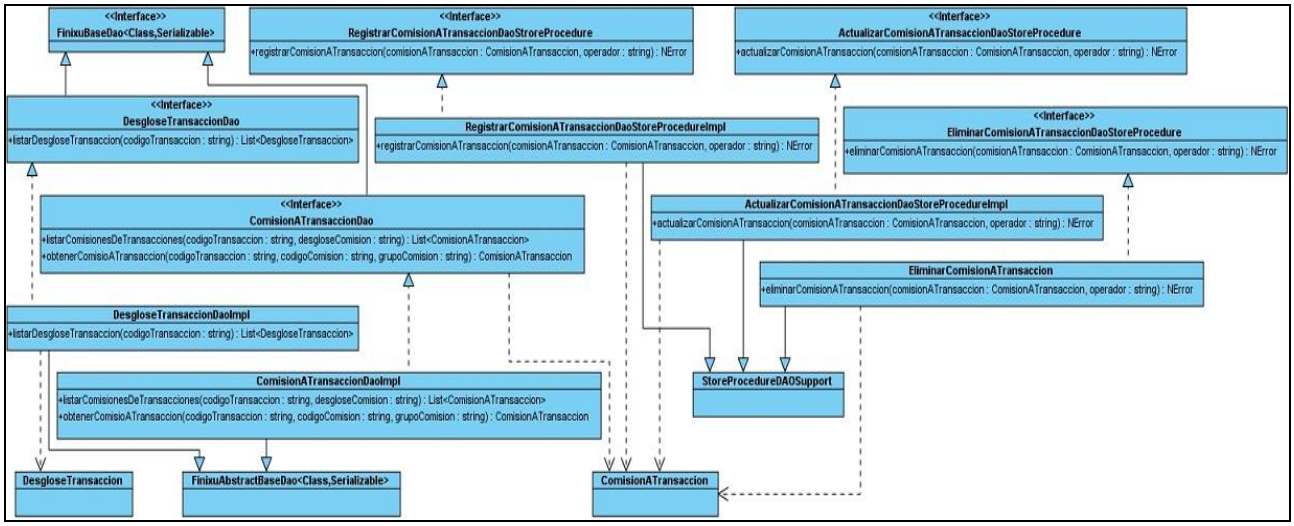

Ilustración 20 Diseño de la capa de Acceso a Datos del módulo Comisión a transacción

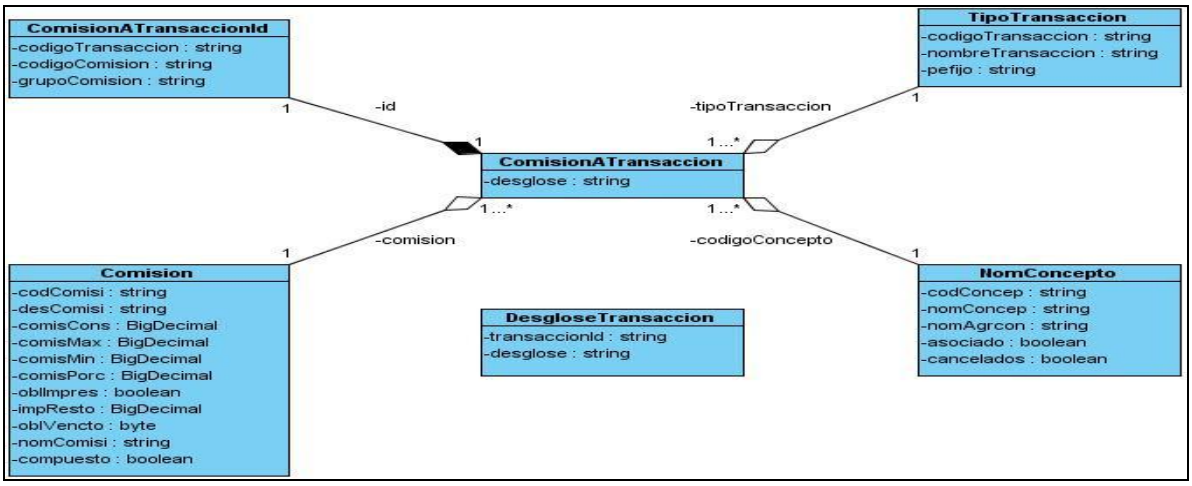

Ilustración 21 Modelo de Dominio del módulo Comisión a transacción

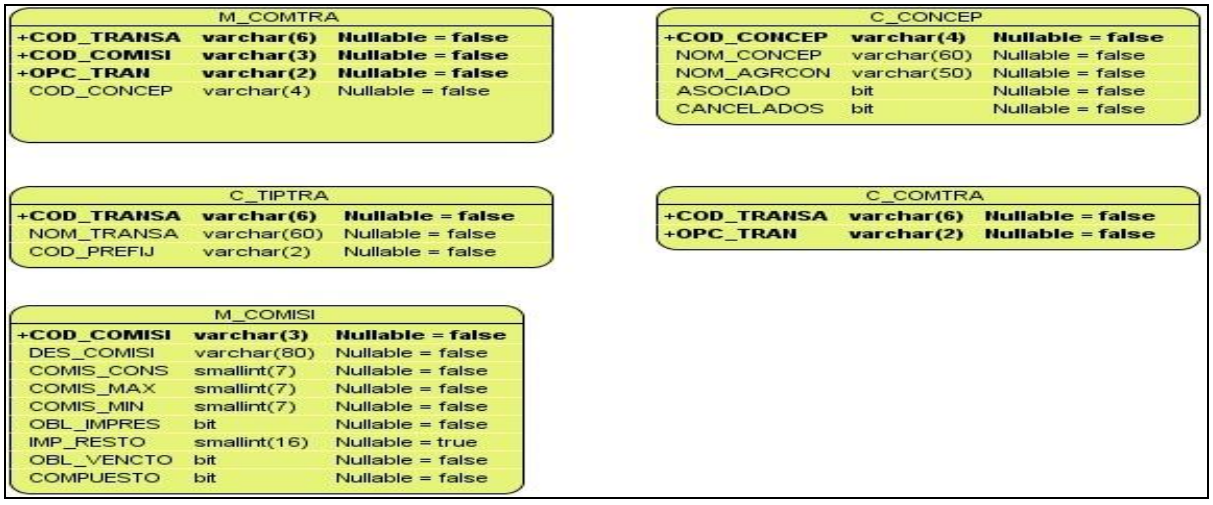

Ilustración 22 Modelo de datos del módulo Comisión a transacción

Anexo 2: Diseño del módulo Tasa de cambio

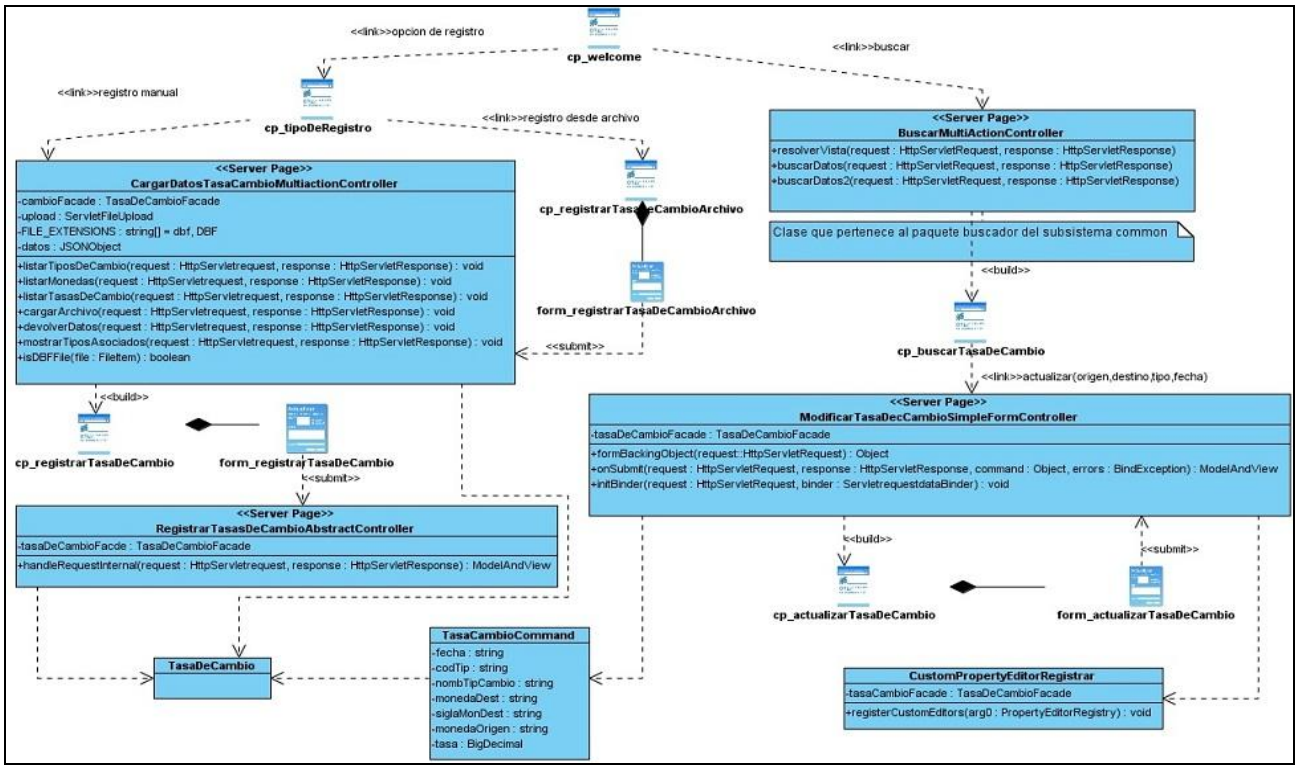

Ilustración 23 Diseño de la capa de Presentación del módulo Tasa de cambio

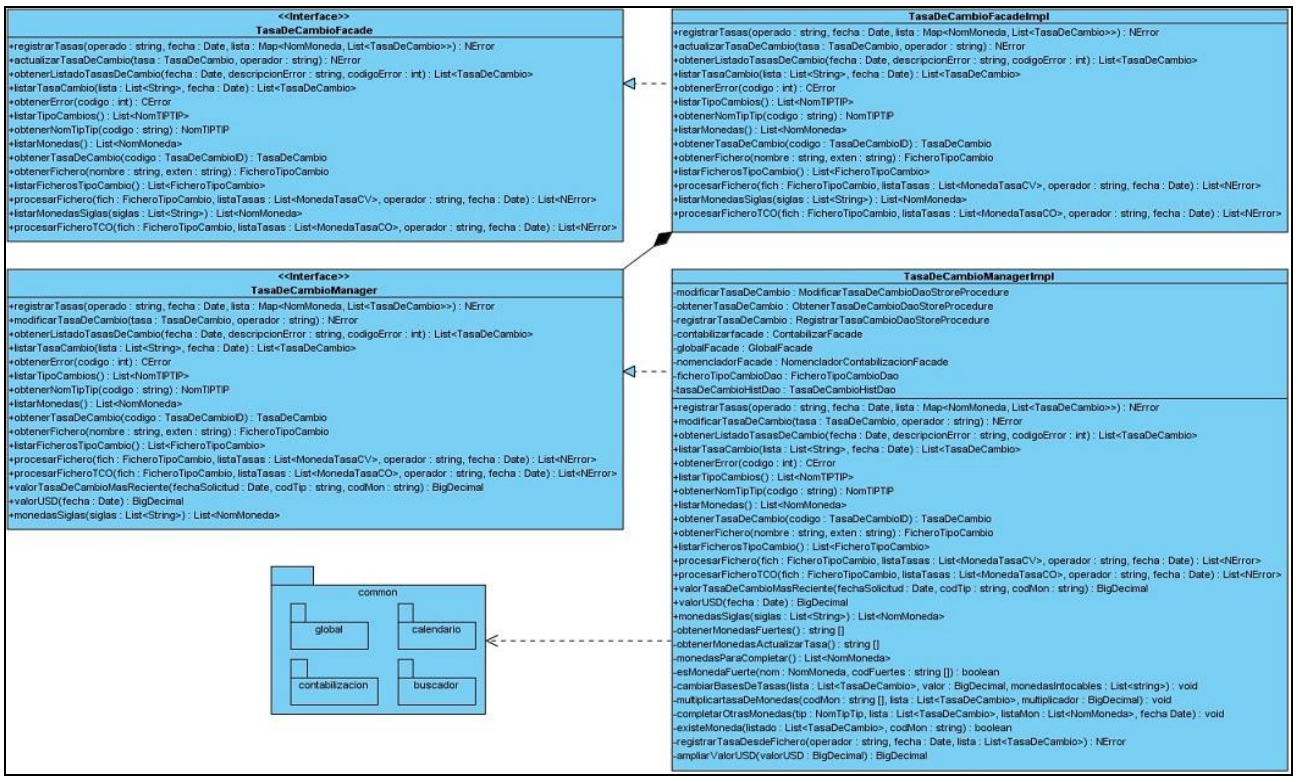

Ilustración 24 Diseño de la capa de Negocio del módulo Tasa de cambio

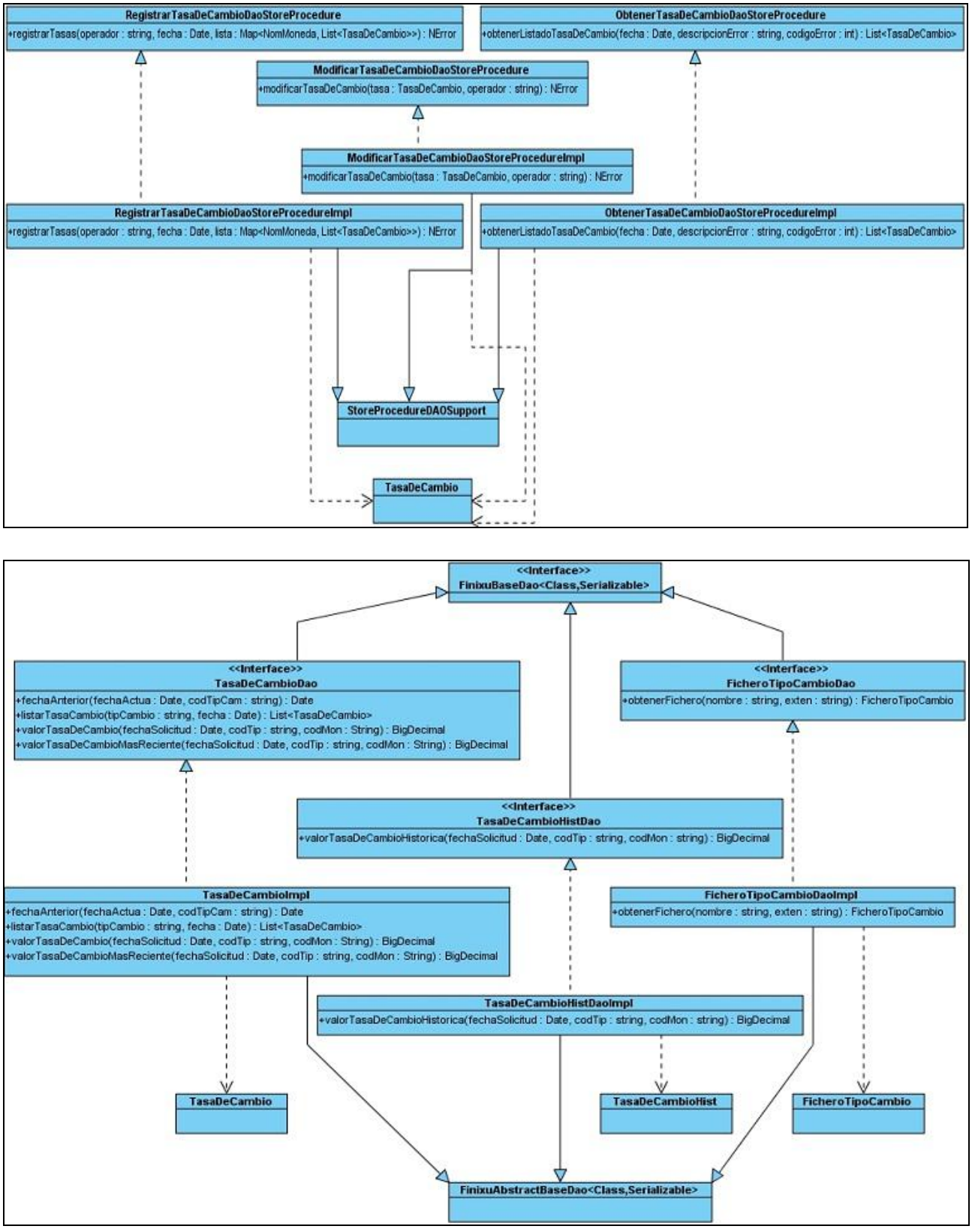

Ilustración 25 Diseño de la capa de Acceso a Datos del módulo Tasa de cambio

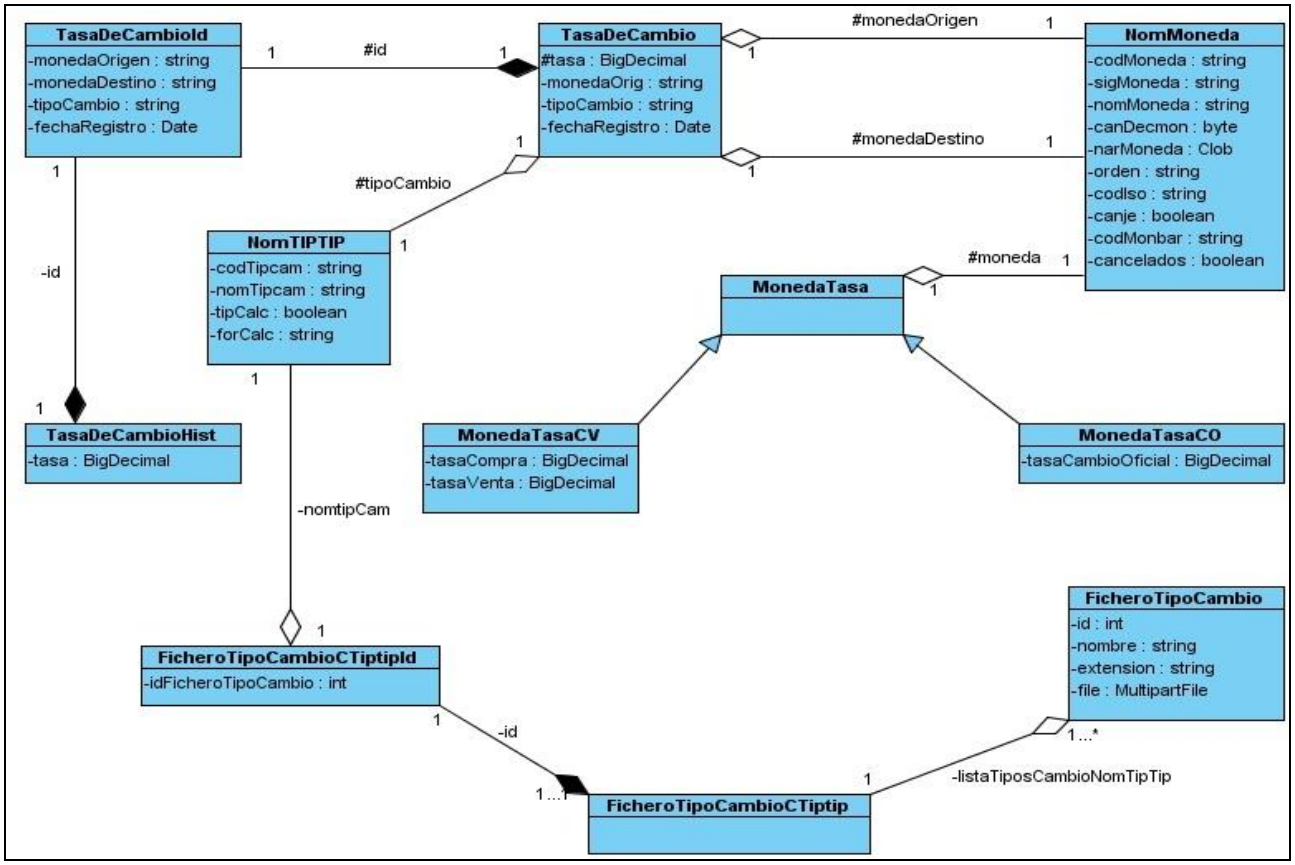

Ilustración 26 Modelo de Dominio del módulo Tasa de cambio

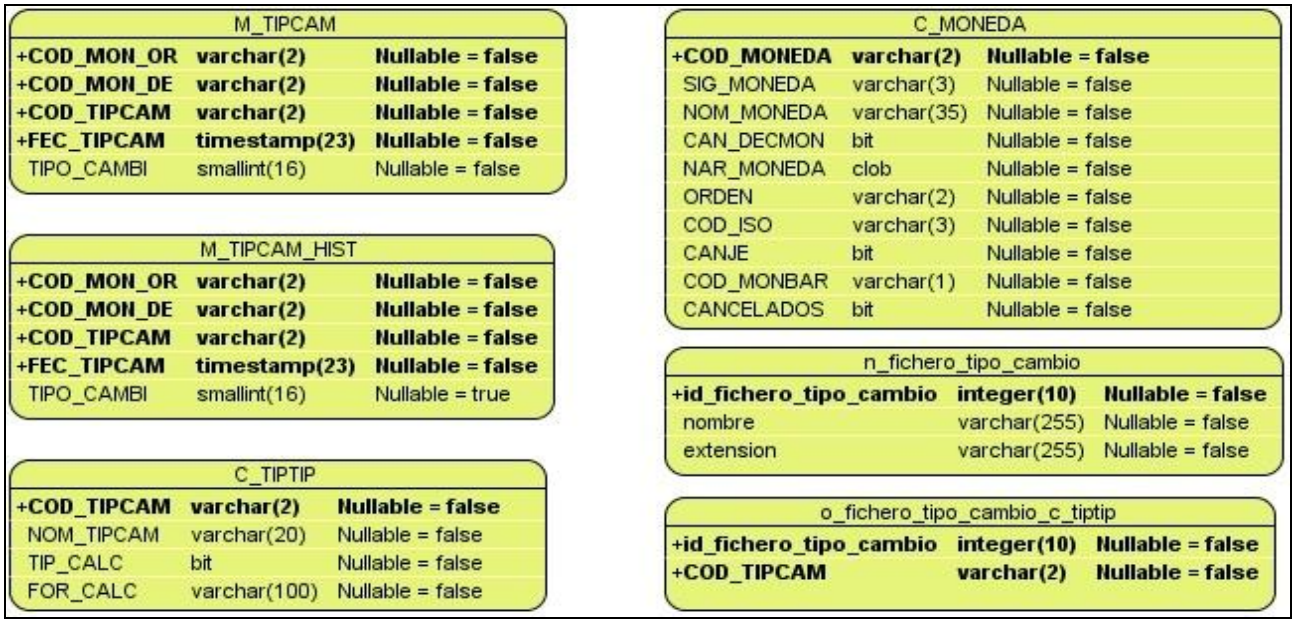

Ilustración 27 Modelo de datos del módulo Tasa de cambio

Anexo 3: Caso de prueba para caja blanca de la funcionalidad Eliminar comisión a transacción del módulo Comisión a transacción.

Caso de prueba para la funcionalidad Eliminar comisión a transacción	
Descripción	Debe consultar una comisión a transacción determinada y luego eliminarla.
Condición de ejecución	Los códigos y el grupo comisión no pueden ser nulos al igual que el operador.
Entrada	codigoTransaccion=04001 , codigoComision=021 ,grupoComision=00
Resultados esperados	Se elimina satisfactoriamente la comisión a transacción obteniéndose “ok” como texto de salida.

Tabla 2 Caso de prueba para el caso de uso Eliminar comisión a transacción del módulo Comisión a transacción

```
/**
 * Para probar el funcionamiento del eliminar comisión a transacción. Módulo
 * Comisión a transacción
 */
@Test
public void testEliminarComisionATransaccion() {
    boolean esperado = true;
    request.getParameter("codigo");
    controlHttpServletRequest.setReturnValue("040501");
    request.getParameter("codigoC");
    controlHttpServletRequest.setReturnValue("021");
    request.getParameter("desglose");
    controlHttpServletRequest.setReturnValue("00");
    controlHttpServletRequest.replay();
    boolean e = comisTransController.eliminarComisionATransaccion(request, response);
    assertEquals(esperado, e);

}
```

Ilustración 28 Método de prueba para el caso de uso Eliminar comisión a transacción del módulo Comisión a transacción

Anexo 4: Caso de prueba para caja blanca de la funcionalidad Actualizar tasa de cambio del módulo Tasa de cambio.

Caso de prueba para la funcionalidad Actualizar tasa de cambio	
Descripción	Debe consultar una tasa de cambio, sustituir el valor de la tasa y luego actualizar en la base de datos dicho cambio.
Condición de ejecución	El código de la comisión no puede estar vacío
Entrada	monedaOrigen =05, monedaDestino =40, fechaRegistro =11/10/2011,tipoCambio = 06, usuario=CCCCC, nuevaTasa=0.02

Resultados esperados	Se actualiza la tasa de cambio satisfactoriamente obteniéndose “ok” como texto de salida.

Tabla 3 Caso de prueba para el caso de uso Actualizar tasa de cambio del módulo Tasa de cambio

```
/**
 * Para probar le funcionamiento del actualizar tasa de cambio. Módulo Tasa
 * de cambio
 * @throws Exception
 */
@Test
public void testActualizarTasaDeCambio() throws Exception {
    boolean esperado = true;
    request.getParameter("origen");
    controlHttpServletRequest.setReturnValue("05");
    request.getParameter("destino");
    controlHttpServletRequest.setReturnValue("40");
    request.getParameter("fecha");
    controlHttpServletRequest.setReturnValue("11/10/2011");
    request.getParameter("tipo");
    controlHttpServletRequest.setReturnValue("06");
    controlHttpServletRequest.replay();
    BigDecimal t = new BigDecimal("0.02");
    TasaDeCambio tasa=(TasaDeCambio) tasaDeCambioControler.formBackingObject(request);
    tasa.setTasa(t);
    BindException errors=new BindException(tasa, "TasaDeCambio");
    boolean e=tasaDeCambioControler.onSubmit(request, response, tasa, errors);
    assertEquals(esperado, e);
}
```

Ilustración 29 Método de prueba para el caso de uso Actualizar tasa de cambio del módulo Tasa de cambio

Anexo 5: Diseño del caso de prueba para caja negra del caso de uso Registrar Comisión del módulo Gestionar comisión.

1. **Descripción General.**
   - ✓ Se realiza el registro de una comisión.
2. **Condiciones de Ejecución.**

- ✓ El usuario debe estar autenticado con los permisos necesarios para realizar la acción.
- ✓ Debe existir conexión con la Base de Datos.

3. Secciones a probar en el Caso de Uso.

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC Registrar comisión	EC 1.1 Registrar comisión con éxito.	Se registra la comisión introduciendo los datos de los campos mostrados en la interfaz, una vez introducidos los datos se selecciona Aceptar. El sistema valida los datos y realiza las operaciones pertinentes y muestra el mensaje: "Operación realizada satisfactoriamente."
	EC 1.3 Registrar comisión con datos incorrectos.	En el proceso de registrar una comisión pueden no haberse entrado todos los datos o que se insertaron con errores, el sistema señala los datos incorrectos y muestra para cada caso un mensaje de error en correspondencia con el campo con problemas.
	EC:1.2 Cancelar	Se pueden introducir o no los datos que conforman la operación de registrar una comisión pero una vez seleccionado el botón de cancelar, el sistema cancela la operación de registrar la comisión.

Tabla 4 Descripción del caso de prueba para el requisito Registrar comisión.

4. Descripción de variable.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
[1]	Nombre	Campo de texto	No	El campo es de 80 caracteres.
[2]	Código	Lista desplegable	No	Se carga automáticamente los códigos disponibles.
[3]	Porciento	Radiobutton/Campo de texto	Si	Campo desmarcado/ Campo numérico.
[4]	Mínimo	Campo de texto	Si	Campo numérico, depende de que el campo Porciento este seleccionado.

[5]	Máximo	Campo de texto	Si.	Campo numérico, depende de que el campo Porciento este seleccionado.
[6]	Constante	Radiobutton/Campo de texto	No	Campo seleccionado por defecto/Campo numérico.
[7]	Descripción	Campo de texto	Si	El campo es de 80 caracteres.

Tabla 5 Descripción de variables del caso de prueba para el requisito Registrar comisión.

5. Matriz de Datos.

✓ **Registrar comisión**

Escenario	Nombre	Código	Porciento	Mínimo	Máximo	Constante	Descripción	Respuesta del Sistema	Resultado de la Prueba	Flujo Central
EC 1.1 Registrar comisión con éxito.	AVALES BANCARIOS(V)	204 (V)	Null(V)	Null(V)	Null(V)	100(V)	$100,00 CADA UNO(V)	El Sistema Registra la comisión y se muestra un mensaje notificando el éxito de la operación.	Satisfactorio.	1. El usuario escribe la URL en el navegador. 2. Selecciona el botón Aceptar. 3. Selecciona el subsistema Contabilidad. 4. El Usuario selecciona la opción Registrar comisión en el menú principal. 5. El sistema muestra la interfaz con los campos necesarios para efectuar el registro. 6. El usuario introduce los datos y selecciona Aceptar. 7. El sistema valida los datos de entrada y muestra el mensaje: “Operación realizada satisfactoriamente.” 8. El usuario selecciona Aceptar. 9. El sistema va a la página del subsistema.

EC 1.2 Registrar comisión con datos incorrectos.	COMISION DE ENMIENDA (V)	104 (V)	Null(V)	Null(V)	Null(V)	100kInd (I)	$100	El sistema alerta la existencia de datos incorrectos.	Satisfactorio.	1. El usuario escribe la URL en el navegador. 2. Selecciona el botón Aceptar. 3. Selecciona el subsistema Contabilidad. 4. El Usuario selecciona la opción Registrar comisión en el menú principal. 5. El sistema muestra la interfaz con los campos necesarios para efectuar el registro. 6. El usuario introduce los datos y selecciona Aceptar. 7. El sistema valida los datos de entrada y muestra el mensaje: “El campo es numérico.” 8. El usuario corrige los datos de Entrada y selecciona Aceptar. 9. El sistema valida los datos de entrada y muestra el mensaje: “Operación realizada satisfactoriamente.” 10. El usuario selecciona Aceptar. 11. El sistema va a la página del subsistema.

	Null(I)	104 (V)	0.25 (V)	25(V)	75(V)	Null(V)	$100	El sistema alerta la existencia de datos incorrectos.	Satisfactorio.	1. El usuario escribe la URL en el navegador. 2. Selecciona el botón Aceptar. 3. Selecciona el subsistema Contabilidad. 4. El Usuario selecciona la opción Registrar comisión en el menú principal. 5. El sistema muestra la interfaz con los campos necesarios para efectuar el registro. 6. El usuario introduce los datos y selecciona Aceptar. *7.* El sistema valida los datos de entrada y muestra el mensaje: "El campo es obligatorio." 8. El usuario corrige los datos de Entrada y selecciona Aceptar. 9. El sistema valida los datos de Entrada y muestra el mensaje: "Operación realizada satisfactoriamente." 10. El usuario selecciona Aceptar. 11. El sistema va a la página del subsistema.

	ZDVSD GSDG KNSK NSIA GNAS SNA SFNG SDANJ KSDFN INSKD SKJDF KL GDSK KLGSD KLM KS AKLGS ALDK GL (I)	104 (V)	0.25 (V)	25(V)	75(V)	Null(V)	$100	El sistema alerta la existencia de datos incorrectos.	Satisfactorio.	1. El usuario escribe la URL en el navegador. 2. Selecciona el botón Aceptar. 3. Selecciona el subsistema Contabilidad. 4. El Usuario selecciona la opción Registrar comisión en el menú principal. 5. El sistema muestra la interfaz con los campos necesarios para efectuar el registro. 6. El usuario introduce los datos y selecciona Aceptar. 7. El sistema valida los datos de entrada y muestra el mensaje: Cantidad de caracteres: “El campo es de 80 caracteres.” 8. El usuario corrige los datos de entrada y selecciona Aceptar. El sistema valida los datos de entrada y muestra el

	COMISION DE ENMIENDA (V)	104 (I)	Null(V)	Null(V)	Null(V)	100(I)	$100	El sistema alerta la existencia de datos incorrectos.	Satisfactorio.	1. El usuario escribe la URL en el navegador. 2. Selecciona el botón Aceptar. 3. Selecciona el subsistema Contabilidad. 4. El Usuario selecciona la opción Registrar comisión en el menú principal. 5. El sistema muestra la interfaz con los campos necesarios para efectuar el registro. 6. El usuario introduce los datos y selecciona Aceptar. 7. El sistema valida los datos de entrada y muestra el mensaje: 8. Combo editable: “El valor especificado no es válido.” 9. El usuario corrige los datos de Entrada y selecciona Aceptar. 10. El sistema valida los datos de 11. Entrada y muestra el mensaje: “Operación realizada satisfactoriamente.” 12. El usuario selecciona Aceptar. 13. El sistema va a la página del subsistema.

EC 1.3 Cancelar el registro de la operación.	NA	NA	NA	NA	NA	NA	NA	El sistema cancela la operación de registrar la cuenta banco.	Satisfactorio.	1. El usuario escribe la URL en el navegador. 2. Selecciona el botón Aceptar. 3. Selecciona el subsistema Contabilidad. 4. El Usuario selecciona la opción Registrar comisión en el menú principal. 5. El sistema muestra la interfaz con los campos necesarios para efectuar el registro. 6. El usuario puede o no introducir los datos y selecciona el botón Cancelar. 7. El sistema muestra el mensaje: ¿Está seguro que desea cancelar la operación? 8. El usuario selecciona Aceptar. 9. El sistema va a la página del subsistema.

Tabla 6 Datos de prueba del caso de prueba para el requisito Registrar comisión

Anexo 6: Acta de liberación de los módulos Gestionar Comisión, Comisión a Transacción y Tasa de cambio del sistema Quarxo emitida por CALISOFT.

Universidad de las Ciencias Informáticas

Acta de Liberación de Productos Software

Fecha de liberación: 20 de diciembre de 2010.

Emitida a favor de: SAGEB.

1. Datos del producto

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

1

Ilustración 30 Acta de liberación del sistema Quarxo emitida por CALISOFT página 1

UCI Universidad de las Ciencias Informáticas

Acta de Liberación de Productos Software

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

Delvis Echeverría Perez
Nombre y Apellidos
Responsable Calisoft

Lissett Díaz Mesa
Nombre y Apellidos
Responsable Proyecto

2

Ilustración 31 Acta de liberación del sistema Quarxo emitida por CALISOFT página 2