Universidad de las Ciencias Informáticas

Facultad 3

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

Título: Diseño e implementación de los módulos Transacciones generales y Plan de cuentas del sistema Quarxo.

Autor: Adrian Veranes Fonseca

Tutor(es): Ing. Yulier Matías León, Ing. Lissett Díaz Mesa.

La Habana. "Año 53 de la Revolución"

Pensamiento

"Tu tiempo está limitado, así que no lo desaproveches viviendo la vida de algún otro. No te dejes arrastrar por los dogmas, que es lo mismo que vivir con los resultados del pensamiento de otras personas. No dejes que el ruido de las opiniones de otros ahoguen completamente tu voz interior. Y más importante, ten el valor de seguir a tu corazón y a tu intuición. Ellos, de algún modo, ya saben en lo que verdaderamente te quieres convertir. Todo lo demás es secundario."

Steve Jobs

Declaración de autoría

Declaro que soy el único autor del trabajo “Diseño e implementación de los módulos Transacciones generales y Plan de cuentas del sistema Quarxo” y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio. Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____	_____
Adrian Veranes Fonseca	Ing. Yulier Matías León
Autor	Tutor

Agradecimientos

A la Revolución Cubana y a su líder el Comandante en Jefe Fidel Castro Ruz por haberme dado la oportunidad de estudiar en una Universidad de Excelencia y por hacer de este país un lugar donde lo más importante es el bienestar de todos los hombres.

A todos los que de una forma u otra contribuyeron en mi formación, en especial a todos los profesores que me han impartido clases a lo largo de mi trayectoria como estudiante.

A mis tutores y demás miembros del proyecto que tanto me han ayudado en mi preparación y superación, en especial a Matias, Lisset, Yesenia, Javier, Yoan y Adolfo.

A todos mis amigos que a lo largo de estos cinco años tanto me han ayudado y apoyado, en especial a todos los que han compartido el aula conmigo y a mi amigo Addiel.

A mi familia completa, que tanto me quiere.

A Lili y a su familia por ser tan buenas personas.

Dedicatoria

A mis padres Dorita y Ricardito, por darme todo el apoyo que he necesitado, por su sacrificio y entrega en todo momento, por todo lo que soy en la vida, sin ellos no hubiera logrado llegar tan lejos. Por ser los mejores padres del mundo. Los quiero con todo mi corazón.

A mi hermana por ser tan buena estudiante, buena hija y buena hermana.

Resumen

Con el objetivo de contribuir con la modernización del sistema bancario nacional que está llevando a cabo el país, la Universidad de las Ciencias Informáticas está desarrollando el sistema Quarxo[1]. Dicho sistema automatiza los actuales requerimientos del Banco Nacional de Cuba por lo que remplazará al SABIC que se encuentra en explotación actualmente en dicha institución presentando algunos inconvenientes.

El presente trabajo de diploma se basa en el diseño y la implementación de los módulos Transacciones generales y Plan de cuentas del sistema Quarxo. En vista de desarrollar una solución eficaz que respondiera a los requisitos definidos anteriormente se realizó el diseño de los módulos basándose en patrones de diseño.

La solución abarca el estudio de diferentes elementos, donde se incluyen herramientas, lenguajes y tecnologías a utilizar en el desarrollo de la misma.

Palabras claves: Quarxo, Transacciones generales, Plan de cuentas, diseño, implementación.

[1] Quarxo es el nombre del sistema informático que se está desarrollando para el BNC.

Índice de Contenidos

Contenido

Índice de Contenidos

Índice de Contenidos

Índice de Contenidos

Introducción

Introducción

En Cuba como parte del proceso de mejoras que viene desarrollando la revolución se encuentra el proceso de modernización del sistema bancario cubano, donde se decidió desarrollar un sistema informático para el Banco Nacional de Cuba (BNC), bajo un acuerdo de colaboración entre el Banco Central de Cuba (BCC) y la Universidad de las Ciencias Informáticas (UCI).

Los bancos son entidades que se dedican a trabajar con el dinero, para lo cual reciben y tienen a su custodia depósitos hechos por personas y organizaciones, además otorgan préstamos usando esos mismos recursos, actividad que se denomina intermediación financiera. Estos permiten que el dinero circule en la economía, que el dinero que algunas personas y organizaciones tengan disponible pueda pasar a otras que no lo tienen y que lo solicitan. De esta forma facilita las actividades de estas personas y organizaciones y mejoran el funcionamiento de la economía en general. Por el papel que desempeñan, los bancos han ido adquiriendo una importancia vital en el desarrollo de la economía de un país por lo que se hace necesario que cuenten con tecnología de punta.

Los bancos deben llevar un registro actualizado de la actividad contable, dado que la contabilidad es la base sobre la cual se fundamentan las decisiones financieras de una empresa y está dirigida a resumir, registrar, clasificar y controlar las operaciones para interpretar sus resultados.

En el BNC para registrar las operaciones contables se crean cuentas que representan los diferentes tipos de saldos que se manejan. Las cuentas son el elemento básico y central en la contabilidad y en los servicios de pago. Estas cuentas se dividen según su naturaleza en varias clasificaciones, que pueden ser activos, pasivos, capital, cuentas de clientes, cuentas de bancos entre otras. En el BNC la cuenta está compuesta por una moneda, una subcuenta y una contrapartida. En esta estructura la subcuenta es un clasificador que representa el tipo de cuenta, es decir si es de naturaleza activa o pasiva, la finalidad de la misma, entre otras características. Dada la importancia que tienen estas subcuentas es necesario mejorar el proceso de gestión de las mismas.

Las operaciones que se realizan sobre las cuentas son llamadas transacciones. Estas transacciones pueden ser creadas de forma específica o general, la forma específica es donde se capturan solamente los datos necesarios permitiendo realizar operaciones que afectan a determinadas cuentas y la forma general es donde se especifican todos los datos de la transacción de forma manual pudiendo afectar a

cualquier tipo de cuenta. Los bancos trabajan constantemente creando transacciones contables, por lo que se hace necesario que cuenten con herramientas que faciliten este proceso.

Actualmente en el BNC para automatizar los procesos bancarios se encuentra en explotación el sistema SABIC (Sistema Automatizado para la Banca Internacional y de Comercio) en su versión sobre MS-DOS[2], desarrollado sobre FoxPro que es un lenguaje privativo propiedad de la Microsoft que dificulta la integración del SABIC con tecnologías modernas.

El SABIC presenta algunas limitaciones donde se incluye que el mismo se encuentra muy desactualizado, por lo que no automatiza algunos procesos que se llevan a cabo actualmente en el BNC, lo cual ha llevado al personal a hacer uso de otras herramientas como son las tablas Excel, para poder realizar la actividad contable, implicando tiempo y esfuerzo. El SABIC MS-DOS carece de validaciones y restricciones que eviten la introducción de información incorrecta al sistema, lo cual atenta contra el funcionamiento de la entidad y es necesario destinar recursos materiales y humanos para la detección de fallas. A raíz de estos problemas, se determina la necesidad de realizar una nueva solución informática que resuelva esas dificultades y se comienza a desarrollar un sistema llamado Quarxo.

Dada la situación problemática planteada anteriormente se identificó el siguiente ***problema a resolver****:*

¿Cómo lograr la creación de las transacciones generales y la gestión de las cuentas a partir de los módulos del sistema Quarxo?

Objeto de estudio: La creación de las transacciones generales y la gestión de las cuentas en entidades financieras bancarias.

Campo de acción: La informatización de las transacciones generales y la gestión de las cuentas en el Banco Nacional de Cuba.

[2] MS-DOS es un sistema operativo monotarea y monousuario para ordenadores personales.

Introducción

Objetivo general: Diseñar e implementar los módulos Transacciones generales y Plan de cuentas de Quarxo.

Posibles resultados: Diseño e implementación de una solución para la creación de las transacciones generales y la gestión de las cuentas en el Banco Nacional de Cuba para su posterior despliegue**.**

Tareas a cumplir:

1. Estudio y comprensión de los procesos relacionados con la creación de las transacciones generales y la gestión de las cuentas en el Banco Nacional de Cuba.
2. Estudio y preparación en las diferentes tecnologías que se usaran para desarrollar el sistema.
3. Realización del modelo de diseño de los módulos Transacciones generales y Plan de cuentas de Quarxo.
4. Realización del modelo de componentes de los módulos Transacciones generales y Plan de cuentas de Quarxo.
5. Implementación de las funcionalidades de los módulos Transacciones generales y Plan de cuentas de Quarxo.
6. Validación de los módulos Transacciones generales y Plan de cuentas de Quarxo.

Métodos Científicos:

La presente investigación está basada en métodos científicos para facilitar el desarrollo del trabajo, métodos teóricos y empíricos. Se utilizó el método Histórico – Lógico dado que se hizo un estudio de algunos sistemas similares desarrollados anteriormente. Un método muy utilizado fue la modelación que se puso en práctica en la construcción de los modelos de diseño y componentes de los módulos anteriormente mencionados. Se utilizó también el método de observación al observar detalladamente el desempeño del sistema.

Introducción

Estructura del Documento:

Capitulo 1: Se expone el estado del arte, donde se realiza la fundamentación teórica del tema. Al mismo tiempo se describe el objeto de estudio, los procesos fundamentales y otros detalles considerables. Se realiza un recorrido a través de diferentes sistemas informáticos contables que han sido implementados nacional e internacionalmente y que incluyen nuestro objeto de estudio. También se describen las tecnologías a considerar, el lenguaje de programación que se utiliza para la solución, las metodologías de desarrollo y las herramientas de desarrollo.

Capitulo 2: Se describe la arquitectura definida por la dirección y equipo de arquitectura del proyecto Quarxo y se transforman los requisitos funcionales en el diseño del futuro sistema con vista a conformar una entrada adecuada para la implementación, siendo esto último el objetivo principal de este capítulo. Además en este capítulo se crean los artefactos modelo de diseño, diagramas de paquetes, diagramas de clases, diagramas de secuencia y el modelo de datos.

Capitulo 3: Se enfoca en la implementación de los módulos Plan de cuentas y Transacciones generales de Quarxo. Se describen los estándares de codificación definidos para la escritura del código, se representa la interacción de los componentes mediante el diagrama de componentes y se detallan algunas de las clases más importantes del modelo de diseño presentado en el capítulo anterior. Finalmente se prueba la solución mediante pruebas de unidad en aras de encontrar errores que imposibiliten el cumplimiento de los requisitos funcionales definidos anteriormente.

Capítulo 1: Fundamentación Teórica

1.1. Introducción

El presente capítulo se centra en la realización de un estudio y análisis de diferentes temas necesarios para la realización de los módulos Plan de cuentas y Transacciones generales. Primeramente se abordarán los principales conceptos relacionados con el dominio del problema. Se realizará un recorrido a través de diferentes sistemas informáticos contables que han sido implementados nacional e internacionalmente y que incluyen nuestro objeto de estudio, para de ahí tomar los puntos más importantes que puedan ser utilizados en el sistema. Se abordará acerca de la metodología de desarrollo definida por la dirección del proyecto, la descripción de los patrones que se emplearán, finalizando con la especificación de las herramientas y tecnologías que se utilizarán en el desarrollo. El conocimiento teórico obtenido con este capítulo permitirá el buen desempeño en los próximos capítulos del presente trabajo.

1.2. Contabilidad

"La contabilidad es el arte de registrar, clasificar y resumir en forma significativa y en términos de dinero, las operaciones y los hechos que son cuando menos de carácter financiero, así como el de interpretar sus resultados" (1).

La ciencia que tiene por objeto el registro de las operaciones económicas efectuadas por una persona o entidad, con el fin de conocer sus resultados y la situación de la misma **(**2**)**.

Después de estudiar varias definiciones se concluye que la contabilidad está dirigida a registrar, clasificar, resumir y controlar las operaciones para interpretar sus resultados.

La contabilidad puede ser vista de varios tipos: Fiscal, Administrativa, Financiera, Privada, entre otras; cada una de ellas aplicadas a los diferentes sectores de la sociedad.

La Contabilidad Privada puede verse de varios tipos, entre ellas la Contabilidad Bancaria, siendo aplicada a los Bancos por presentar características diferentes al realizar su actividad contable con las demás entidades de la sociedad.

1.3. Contabilidad Bancaria

Es una rama de la contabilidad que se relaciona con la prestación de servicios monetarios, donde se registran todas las operaciones de cuentas en depósitos o retiros de dinero que realizan los clientes.

Registra los créditos, giros tanto al interior o exterior, así como otros servicios bancarios.

La contabilidad bancaria permite regular todas las operaciones y productos financieros que ofrece el Banco, pues facilita:

1. Proteger los activos de la organización mediante mecanismos que evidencien de forma automática y oportuna la malversación de fondos o sustracción de activos.

2. Explicar y justificar la gestión de los recursos, preparar los estados financieros.

3. Registrar y controlar las transacciones de la organización con exactitud y rapidez.

4. Proporcionar una imagen numérica de lo que sucede en la vida y en la actividad organizacional para la toma de decisiones.

1.4. Banco

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 (3).

La estructura del Sistema Bancario en Cuba se ha modificado con los años, en la actualidad se encuentra formada por un banco central, 8 bancos comerciales, 17 instituciones financieras no bancarias, 11 oficinas de representación de bancos extranjeros, y 4 instituciones financieras bancarias. Entre los Bancos Comerciales de Cuba se encuentra el Banco Nacional.

1.4.1. Banco Nacional de Cuba

Entidad encargada de obtener y otorgar créditos en moneda nacional y libremente convertible; centraliza las relaciones con las entidades extranjeras de seguro de crédito oficial a la exportación según se decida por el Banco Central de Cuba. Mantiene el registro, control, servicio y 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 Banco Central de Cuba (4).

Dentro del proceso contable en las entidades financieras, incluyendo en el Banco Nacional de Cuba se resalta la cuenta, elemento que permite tener un control estricto de las diferentes transacciones que se realizan dentro de la institución.

1.5. Cuenta

Es la unidad básica de registro dentro de la contabilidad. Se considera un instrumento de representación y medida de un elemento del patrimonio o de los resultados, que capta la situación inicial de éste y las variaciones que posteriormente se vayan produciendo en el mismo (5). Representa bienes, derechos y obligaciones con los que cuenta un individuo o entidad en una fecha determinada.

1.5.1. Las cuentas se clasifican en:

1- Reales (Activos, Pasivos y Capital): Constituye la masa patrimonial.

2- Nominales (Ingresos y Gastos): Constituye la masa operacional.

3- Mixtas (Participan las cuentas reales y nominales).

Además de la anterior clasificación, las cuentas se pueden agrupar según su tipo, de manera que las entidades (entre ellas el Banco Nacional de Cuba) poseen un clasificador con el objetivo de gestionarlas organizadamente. En dicho clasificador se puede precisar el número de la cuenta, la cuenta/subcuenta, así como una breve descripción de las mismas, posibilitando utilizarla ágilmente para la ejecución de los procesos bancarios.

1.6. Transacción contable

Una transacción contable es un conjunto de movimientos de saldos entre dos o más cuentas de la contabilidad donde intervienen operaciones de débitos y créditos que modifican el saldo de las mismas. En una transacción la suma de los importes de los débitos y los créditos debe ser igual a cero para una moneda y una fecha determinada, significando que la transacción esta cuadrada.

Son en la contabilidad como materias primas en una fábrica, por lo que juegan un papel primordial para conocer la capacidad financiera de la empresa mediante datos contables y estadísticos.

1.7. Registros generados durante la actividad contable

1.7.1. El Libro Diario

Libro conocido como “Libro de Entrada Original”, porque en él se registran por primera vez las transacciones que se realizan en una empresa y en un orden cronológico. Las anotaciones de los hechos contables en el Libro Diario se denominan asientos, los cuales exigen la identificación de los elementos afectados y la coordinación entre las cuentas que intervienen en los mismos. Los asientos contienen entradas de débito en una o más cuentas y crédito en otra(s) cuenta(s) de tal manera que la suma de los débitos sea igual a la suma de los créditos. Se garantiza así que se mantenga la ecuación de contabilidad Activo = Pasivo + Capital.

1.7.2. El Libro Mayor

El Libro Mayor es una consecuencia del Libro Diario. Su función es clasificar los hechos atendiendo a la naturaleza de los elementos que han intervenido, poniendo de manifiesto la situación de cada uno de los elementos por medio de los saldos de las cuentas (6). El libro Mayor permite recopilar sistemáticamente las operaciones inscritas en el libro de diario, es decir, que sirve para llevar control de cada cuenta contable.

Los bancos, por su ritmo de trabajo, necesitan de respuestas rápidas y seguras, siendo esto posible en la actualidad gracias a la creación de los sistemas contables que son capaces de gestionar toda la información que se necesita.

1.8. Sistemas Contables

Los sistemas contables deben permitir:

- Registrar las actividades contables, permitiendo de esta manera llevar un registro sistemático de las actividades financieras y comerciales diarias en términos económicos.
- Clasificar dicha información, ya que normalmente el volumen de información es tan grande y diverso que se convierte en información muy difícil de interpretar para la toma de decisiones, por lo que es necesario clasificarla en grupo y categorías.
- Resumir la información es la posibilidad más relevante que brindan estos sistemas automatizados, porque al resumir todos los hechos contables que una vez fueron registrados, será más fácil para poder interpretarlos y posteriormente actuar según las necesidades de la institución.

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

Sistemas Informáticos Contables

Los Sistemas Informáticos Contables son los programas de contabilidad o paquetes contables, destinados a sistematizar y simplificar las tareas de contabilidad. El software contable registra y procesa las transacciones históricas que se generan en una empresa o actividad productiva: las funciones de compra, ventas, cuentas por cobrar, cuentas por pagar, control de inventarios, balances, producción de artículos, nóminas, entre otras. Para ello sólo hay que ingresar la información requerida, como las pólizas contables, ingresos y egresos, y hacer que el programa realice los cálculos necesarios.

1.8.1. Sistemas informáticos contables existentes

En el mundo

SAP: Es un ERP[3] empresarial. Es un sistema informático basado en módulos integrados, que abarca prácticamente todos los aspectos de la administración empresarial. Entre sus soluciones se encuentra el SAP para bancos que gestiona los principales procesos de negocio de cualquier banco. SAP constituye una de las soluciones más amplia existente para la industria bancaria, entre sus funcionalidades se encuentran la gestión de cuentas y la creación de transacciones. La principal desventaja de SAP es su alto costo, por lo que a pesar de ser uno de los ERP más completos existentes actualmente muchas empresas no pueden adquirirlo.

SIBANCO (Solución Integral para Banca y Cooperativas) provee a bancos y cooperativas financieras funcionalidad óptima para el manejo y control de transacciones, fácil acceso a la información, eficiencia de las operaciones, seguridad y consolidación de informes contables, comerciales, operativo y tributario. Es una solución estable, flexible, moderna y segura que permite adaptarse rápidamente a los constantes cambios y requerimientos que surgen en el mundo financiero. SIBANCO permite automatizar la totalidad de las operaciones de una institución financiera, interactúa con redes de cajeros automáticos, y de punto de venta, terminales de autoservicios, kioscos multimedia, equipo de audio respuesta, banca por internet y por telefonía celular. El diseño y arquitectura de esta permite administrar centralizadamente los procesos y módulos.

Estos sistemas son propietarios y muy costosos por lo que es importante buscar alternativas más económicas para nuestras entidades financieras bancarias. Constituyen además un riesgo en cuanto a seguridad se refiere, debido a que no está disponible una bibliografía suficiente para un estudio completo de cómo fueron concebidos.

[3] ERP (Enterprise Resource Planing): Sistemas de Planeación de Recursos.

En Cuba

La Dirección de Sistemas Automatizados del Banco Central de Cuba (BCC) para satisfacer las necesidades de procesamiento de datos de bancos e instituciones financieras desarrolló el Sistema Automatizado para la Banca Internacional de Comercio (SABIC). Dentro de las principales características funcionales del SABIC se encuentran:

- La contabilización en tiempo real: permite que la extracción de dinero de una cuenta o el sobregiro de cualquier otra cuenta se realice de manera segura, controlando la existencia de los fondos requeridos para realizar la operación y haciendo posible que la institución mantenga sus ficheros contables actualizados permitiéndole saber en cualquier momento su situación financiera.
- La Contabilidad Multimoneda: permite poder registrar los activos y pasivos sin tener que hacer conversiones de moneda lo cual garantiza una mayor exactitud de la información sobre la situación financiera de la institución, al no tener que depender de las variaciones de los tipos de cambios.
- La característica multisucursal: se debe a que con su ayuda y utilizando una red de transmisión de datos X-25 o similares, se pueden enlazar entre sí todas las oficinas de un banco o institución financiera y realizar, también en tiempo real.
- La característica transaccional del sistema: se basa en la contabilización de operaciones mediante transacciones las cuales son el conjunto de asientos requeridos para registrar una operación. Al ser un sistema modular facilita la adaptabilidad, flexibilidad y evolución del sistema sin tener que efectuar cambios en sus programas generales. (7)

Debido a la evolución de las actividades que se llevan a cabo dentro de las instituciones financieras bancarias cubanas se hizo necesario la integración de otras funcionalidades al sistema por lo que existen varias versiones del SABIC:

La primera versión del SABIC utiliza el MS-DOS como sistema de explotación lo cual constituye una limitación al ser este último monotarea, es decir, el microprocesador solo puede atender un único proceso. Esta primera versión fue realizada en FoxPro y utiliza un servidor de ficheros lo que conlleva a un tráfico excesivo dentro de la red. Por esta razón se decidió realizar una nueva versión en un ambiente cliente-servidor.

La segunda versión del SABIC se realizó bajo la filosofía de tener en un corto tiempo un sistema que utilizara las ventajas de la técnica cliente-servidor; pero sin realizar un nuevo diseño del mismo, por lo que siguieron persistiendo problemas de no adecuación con todos los procesos que se llevan a cabo dentro de un banco. Se escogió como lenguaje para programar al cliente a Visual FoxPro y para el servidor SQL Server. Con respecto al cliente la selección se basó en que el sistema anterior está programado en FoxPro y utilizar Visual FoxPro era más productivo ya que se podía aprovechar parte del código escrito para la versión anterior.

Posteriormente a estas versiones al sistema se le han incluido algunas funcionalidades que dan soporte a la mayoría de los procesos que van surgiendo en la actividad financiera nacional o a las variaciones que han sufrido los mismos. El SABIC no permite la generación automática y la configuración de nuevos reportes por lo que se hace engorrosa esta actividad y resultan insuficientes sus funcionalidades para lograr que el BNC realice todas sus operaciones a través de él.

1.9. Elementos utilizados en el desarrollo

Es importante conocer el contexto donde se utilizará el software que se desarrolla ya que es necesario para una apropiada selección de las herramientas y tecnologías a utilizar, influyendo considerablemente en el tiempo de desarrollo y la calidad final del producto.

Las metodologías, tecnologías, herramientas y frameworks utilizados en la elaboración de los módulos en cuestión fueron definidos por el grupo de arquitectura del proyecto SAGEB, siendo esta la razón por la cual en este fragmento del capítulo no se definen dichos elementos, sino que se realiza una resumida caracterización de cada uno de ellos y de los beneficios que aportan en el trabajo de diseño e implementación de los módulos implicados.

1.9.1. Metodología de desarrollo de software

Una metodología no es más que el estudio de los métodos más apropiados que se emplean para desarrollar software de manera eficiente; o como precisan otros autores (8), define Quién debe hacer Qué, Cuándo, y Cómo debe hacerlo. Una metodología define un conjunto de pasos y procedimientos que deben seguirse para desarrollar software.

La importancia de utilizar una metodología radica principalmente en lograr un producto final con calidad. Existen metodologías tales como: XP, Métricas 3, SCRUM, RUP entre otras; siendo utilizada en el desarrollo de nuestro software la metodología RUP.

1.9.1.1. Rup (Rational Unified Process)

RUP es una metodología de desarrollo que de forma disciplinada asigna tareas y responsabilidades en un proyecto de desarrollo, con el objetivo de asegurar la producción de un software con calidad, dentro de plazos y presupuestos predecibles. Presenta tres características fundamentales:

1- **Iterativo e incremental**: principio de división en iteraciones pequeñas, tanto para el desarrollo de las funcionalidades, como para la planificación formal del proyecto. El trabajo se divide en piezas pequeñas; cada uno provee un subproducto incremental.

2- **Centrado en la arquitectura**: La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, las bases del sistema que son útiles para entenderlo, desarrollarlo y producirlo económicamente.

3- **Guiado por casos de uso**: Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso (cómo se llevan a cabo).

RUP divide su trabajo en cuatro fases de desarrollo:

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

[4] Release: Nueva versión de una aplicación informática.

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

1. Modelamiento del negocio: Este flujo identifica los procesos de negocio, los que estarán sujetos a automatización y quiénes intervienen en los mismos.
2. Requerimientos: Se identifican las restricciones que se imponen y lo que el sistema debe hacer.
3. Análisis y Diseño: Se trasladan los requerimientos dentro de la arquitectura de software.
4. Implementación: Se crean un software que se ajuste a la arquitectura y que tenga el comportamiento deseado.
5. Pruebas: Se asegura que el comportamiento requerido es el correcto y que todo lo solicitado está presente.
6. Configuración y administración del cambio: Se guardan todas las versiones del proyecto.
7. Administrando el proyecto: Se administran horarios y recursos.
8. Ambiente: Se administra el ambiente de desarrollo.
9. Distribución: Se hace todo lo necesario para la salida del proyecto.

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

Fase de Análisis y Diseño

En el análisis y diseño modelamos el sistema y encontramos su forma (incluida la arquitectura) para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se le suponen. Una entrada esencial en el diseño es el resultado del análisis, o sea el modelo de análisis, que proporciona una comprensión detallada de los requisitos. Además impone una estructura del sistema que debemos esforzarnos por conservar lo más fielmente posible cuando demos forma al sistema.

Concretamente podemos definir como propósitos del diseño:

Adquirir una comprensión de los aspectos relacionados con los requisitos no funcionales y restricciones

relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia y tecnologías de interfaz de usuario.

Crear una entrada apropiada y un punto de partida para actividades de implementación, capturando los requisitos o subsistemas individuales, interfaces y clases.

Descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo.

Capturar las interfaces entre los subsistemas antes en el ciclo de vida del software, lo cual es muy útil cuando utilizamos interfaces como elementos de sincronización entre diferentes equipos de desarrollo.

Fase de Implementación

En este flujo de trabajo se define la organización del código en paquetes de implementación. Tiene como objetivo principal dar solución a lo antes diseñado, mediante la implementación. Como resultado final de las tareas realizadas por los implementadores se obtiene el software.

1.9.2. Patrones de Diseño

Los patrones de diseño son soluciones probadas y exitosas a problemas comunes que se nos presentan en el desarrollo de software. A continuación se mencionarán un conjunto de ellos que contribuyen a que se realice el diseño de la solución de forma rápida y eficaz.

1.9.2.1. Patrones de Asignación de Responsabilidades. (GRAPS)

Los patrones GRASP (General Responsabilty Assignment Software Patterns), son patrones de software para la asignación de responsabilidades en parámetros útiles para el diseño del producto.

Entre los más útiles se encuentran:

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

Patrón Creador: El patrón creador ayuda a identificar quién debe ser el responsable de la instanciación o creación de nuevas clases u objetos. La clase podrá crear la nueva instancia si y sólo si tiene en cuenta al menos uno de los siguientes criterios:

- Tiene la información necesaria.
- Usa directamente las instancias creadas del objeto.
- Almacena o maneja varias instancias de la clase.
- Contiene o agrega la clase.

La visibilidad entre la clase creada y la clase creadora es una de las facilidades que se deriva del uso de patrón y además conduce a un bajo acoplamiento, lo cual supone facilidad de mantenimiento y reutilización así como mayor claridad.

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

Patrón Bajo Acoplamiento: Una clase con bajo acoplamiento no depende de muchas otras. Las clases con alto acoplamiento recurren a muchas otras y no es conveniente porque se hacen más difíciles de mantener, entender y reutilizar.

Patrón Alta Cohesión: El patrón Alta Cohesión es la meta principal que ha de tenerse en cuenta en cada momento en todas las decisiones de diseño. Indica que las clases deben tener un alto grado de funcionalidad, combinado con una reducida cantidad de operaciones lo que simplifica el mantenimiento de estas. Su objetivo es asignar una responsabilidad de modo que la cohesión siga siendo alta. Las clases con alta cohesión se caracterizan por tener responsabilidades estrechamente relacionadas y que no realizan un trabajo enorme.

1.9.2.2. Patrones GoF

Patrón Fachada: Patrón estructural que trata de simplificar la interfaz entre dos sistemas o componentes de software ocultando un sistema complejo detrás de una clase que hace de pantalla o fachada. La idea

principal es la de ocultar todo lo posible la complejidad de un sistema, el conjunto de clases o componentes que lo forman, de forma que solo se ofrezca un punto de entrada al sistema tapado por la fachada. Su uso aísla los posibles cambios que se puedan producir en alguna de las partes.

Patrón Cadena de Responsabilidad: La cadena de responsabilidad se encarga de evitar el acoplamiento del remitente de una petición a su receptor, dando a más de un objeto la posibilidad de manejar la petición.

Patrón Singleton: Patrón creacional diseñado para restringir la creación de objetos pertenecientes a una clase. Su objetivo es garantizar que una determinada clase sólo tenga una instancia y proporcionar un punto de acceso global a esta. Es un patrón muy sencillo de diseñar y a menudo es implementado por otros patrones.

1.9.2.3 Patrón J2EE (Core J2EE Patterns)

Patrón de Acceso a Datos (DAO): Plantea como solución, utilizar un *"Data Access Object"* (DAO) para abstraer y encapsular todos los accesos a la fuente de datos. Este maneja la conexión con la fuente de datos para obtener y almacenar datos.

El DAO implementa el mecanismo de acceso requerido para trabajar con la fuente de datos. Este oculta completamente los detalles de implementación de la fuente de datos a sus clientes. Como la interfaz expuesta por el mismo no cambia cuando cambia la implementación de la fuente de datos subyacente, este patrón permite adaptarse a diferentes esquemas de almacenamiento sin que esto afecte a sus clientes o componentes de negocio. Esencialmente este actúa como un adaptador entre el componente y la fuente de datos.

Composite View (Vistas Compuestas): Este patrón hace que la representación de vistas sea más manejable ya que gestiona los diferentes elementos de una página por medio de una plantilla. Tiene como propósito el crear Vistas Compuestas de varias sub-vistas de forma modular, flexible y extensible para construir vistas de páginas JSP para aplicaciones J2EE.El control de la vista se puede implementar de diferentes formas: utilizando etiquetas jsp estándar, como <jsp: include>, utilizando componentes JavaBeans, y también mediante etiquetas personalizadas.

1.9.3. Ambiente de Desarrollo

1.9.3.1. Plataforma JEE

JEE es una plataforma que ofrece soluciones para el desarrollo, definiendo estándares para desarrollar aplicaciones en el lenguaje Java. Las aplicaciones desarrolladas en esta plataforma tienden a ser portables, escalables, robustas y seguras. Define una arquitectura utilizando un modelo multicapa, permitiendo así realizar aplicaciones distribuidas complejas.

1.9.3.2. Lenguaje Java

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

1.9.3.3. Servidor de aplicaciones web

Tomcat funciona como un contenedor de servlets, implementa las especificaciones de servlets y JavaServer Page (JSP). Es desarrollado en un entorno abierto. Fue publicado bajo la licencia del software de Apache. Tomcat puede funcionar como servidor web por sí mismo o en combinación con el servidor web Apache. Es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.
Se utiliza Tomcat en su versión 6.

1.9.3.4. Gestor de Base de Datos

SQL Server 2005 provee herramientas sólidas y conocidas, reduciendo la complejidad de la creación, despliegue, administración y uso de aplicaciones de datos empresariales. Se caracteriza por brindar soporte para transacciones y procedimientos almacenados. Sus lenguajes de consulta son el SQL y el T-SQL (en inglés: Transact-SQL), siendo este último el principal medio de programación y administración del

servidor. Requiere para su funcionamiento el sistema operativo Microsoft Windows, representando esto una de sus principales desventajas.

1.9.3.5. Eclipse IDE

Eclipse es un IDE (Integrated Development Environment), en español Entorno de Desarrollo Integrado para Java muy potente. Fue creado por la IBM bajo la filosofía de software libre. Se está convirtiendo en el estándar de puntera de los entornos de desarrollo para Java. Es Eclipse un marco de trabajo que está compuesto por componentes que se pueden o no incluir en dependencia de las necesidades del desarrollador, a estos complementos se les llama (plugins). De hecho, existen complementos que permiten usar Eclipse para programar en otros lenguajes aparte del Java como son PHP, Perl, Python, C/C++.
Se utiliza Eclipse IDE en su versión 3.4.

1.9.3.6. Control de Versiones (SubVersion)

Es una herramienta desarrollada bajo tecnologías open source. Uno de sus principales objetivos es mantener íntegro el control de versiones, se integra con el eclipse mediante el plugin SubEclipse.
Se utiliza el SubVersion en su versión 1.6.6.

1.9.4. Lenguaje y herramienta de Modelado

1.9.4.1. UML

UML (en inglés: Unified Modeling Language) es un lenguaje de modelado visual que permite visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software, soportando además el paradigma orientado a objetos. Se caracteriza por dividir a los sistemas en una estructura estática y un comportamiento dinámico**.** (9)

1.9.4.2. Visual Paradigm

Esta es una herramienta que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño

orientados a objetos, construcción, pruebas y despliegue. El Visual Paradim contribuye a que las aplicaciones se construyan con una mejor calidad y menor coste, ya que permite:

Aplicar ingeniería inversa, la generación de código y editar detalles de casos de uso.
Se utiliza el Visual Paradigm en su versión 6.4.

1.9.5. Tecnologías y Frameworks

1.9.5.1. Framework.

Un framework es un término muy utilizado últimamente en el campo de la informática, se utiliza para referirse a un conjunto de bibliotecas, que se utilizan para implementar la estructura de un modelo para una aplicación. Esto se realiza con el objetivo de promover la reutilización de código, posibilitando que no sea necesario perder tiempo en reinventar la rueda. Existen diferentes tipos de framework, para diferentes propósitos, algunos orientados al desarrollo de aplicaciones web, o para un determinado sistema operativo o lenguaje. En un sentido muy amplio el framework que se utiliza determina la arquitectura del software.

1.9.5.2. Spring Framework

Spring es un framework bajo la concepción de código abierto diseñado para el desarrollo de aplicaciones de la plataforma J2EE. Ofrece mucha libertad a los desarrolladores en Java, brindando soluciones bien documentadas, seguras y robustas. Gracias a sus once grandes módulos.

Para su funcionamiento incluye inversión de control e inyección de dependencia. Apoyándose en el API de Java Reflection.

Se utiliza Spring Framework en su versión 2.5.

1.9.5.3. Spring WebFlow

Spring WebFlow es un framework que permite operar la navegación de la aplicación Web. Al ser limitado el flujo de páginas brindado por los Frameworks MVC clásico, surge el framework Spring

WebFlow. Los flujos Web en este framework son diseñados para ser auto-controlados, dando la

posibilidad de definir reglas de navegación múltiples y complejas.

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

Se utiliza Spring WebFlow en su versión 2.0.8.

1.9.5.4. Hibernate

Hibernate es un framework ORM (*Object-Relational Mapping*) para la plataforma Java disponible además para la plataforma .NET con el nombre de NHibernate. Es open source (código abierto) y gratis. Este permite y hace más viable el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) haciendo más fácil esta relación. Entre sus funciones esta permitir transacciones, asociaciones, polimorfismo, herencia, carga mediante referencia, la cual consiste en cargar sólo la referencia de donde se encuentran los datos, es llamada (carga lazy), persistencia transitiva, estrategias de fetching, además de poseer un potente lenguaje de consulta Hibernate Query Language (HQL) al dotarlo de un lenguaje invariante con respecto a la base de datos y además sintácticamente muy parecido al Structured Query Language (SQL).

Se utiliza Hibernate en su versión 3.5.

1.9.5.5. Dojo Toolkit

Esta herramienta es un framework que contiene funcionalidades que ayudan el desarrollo de aplicaciones web. Ofrece soporte para la internacionalización e incluye un gran cúmulo de componentes visuales basados en XHTML, CSS y Javascript que permiten enriquecer la interfaz de usuario. Dojo incorpora soporte para el trabajo con AJAX (en inglés: Asynchronous JavaScript and XML).

Se utiliza Dojo Toolkit en su versión 1.3.

1.10. Conclusiones del capítulo

El estudio de los principales conceptos asociados a las cuentas y transacciones y de los sistemas informáticos existentes similares al sistema que se quiere desarrollar permitió complementar los conocimientos necesarios para el desarrollo del presente trabajo.

También en este capítulo se llega a la conclusión de que resulta poco factible costear cualquiera de los sistemas ya existentes en el mundo que permitan la gestión de las cuentas y las transacciones por lo que se hace necesario la creación de un sistema que gestione estas. Por estas razones se decide llevar a cabo el diseño e implementación de los módulos Plan de cuentas y Transacciones generales del sistema Quarxo.

Las tecnologías que se utilizan para desarrollar el sistema en cuestión son las definidas por el equipo de arquitectos del sistema Quarxo. Se decide como ambiente de trabajo el IDE Eclipse por ser una herramienta eficiente que permite desarrollar en el lenguaje definido, java y facilita el desarrollo con Frameworks como son Hibernate, Spring, Dojo y otros abordados en el capítulo. Las herramientas y tecnologías seleccionadas para el desarrollo en su mayoría se basan en la filosofía de software libre y están eximidas de costo.

Capitulo 2: Diseño de los módulos Transacciones generales y Plan de cuentas.

2.1. Introducción

En el presente capitulo se describe la arquitectura definida por la dirección y equipo de arquitectura del proyecto Quarxo y se transforman los requisitos funcionales en el diseño del futuro sistema con vista a conformar una entrada adecuada para la implementación, siendo esto último el objetivo principal del presente capitulo. En este se crean los artefactos modelo de diseño, diagramas de paquetes, diagramas de clases, diagramas de secuencia y el modelo de datos.

2.2. Descripción de la arquitectura propuesta

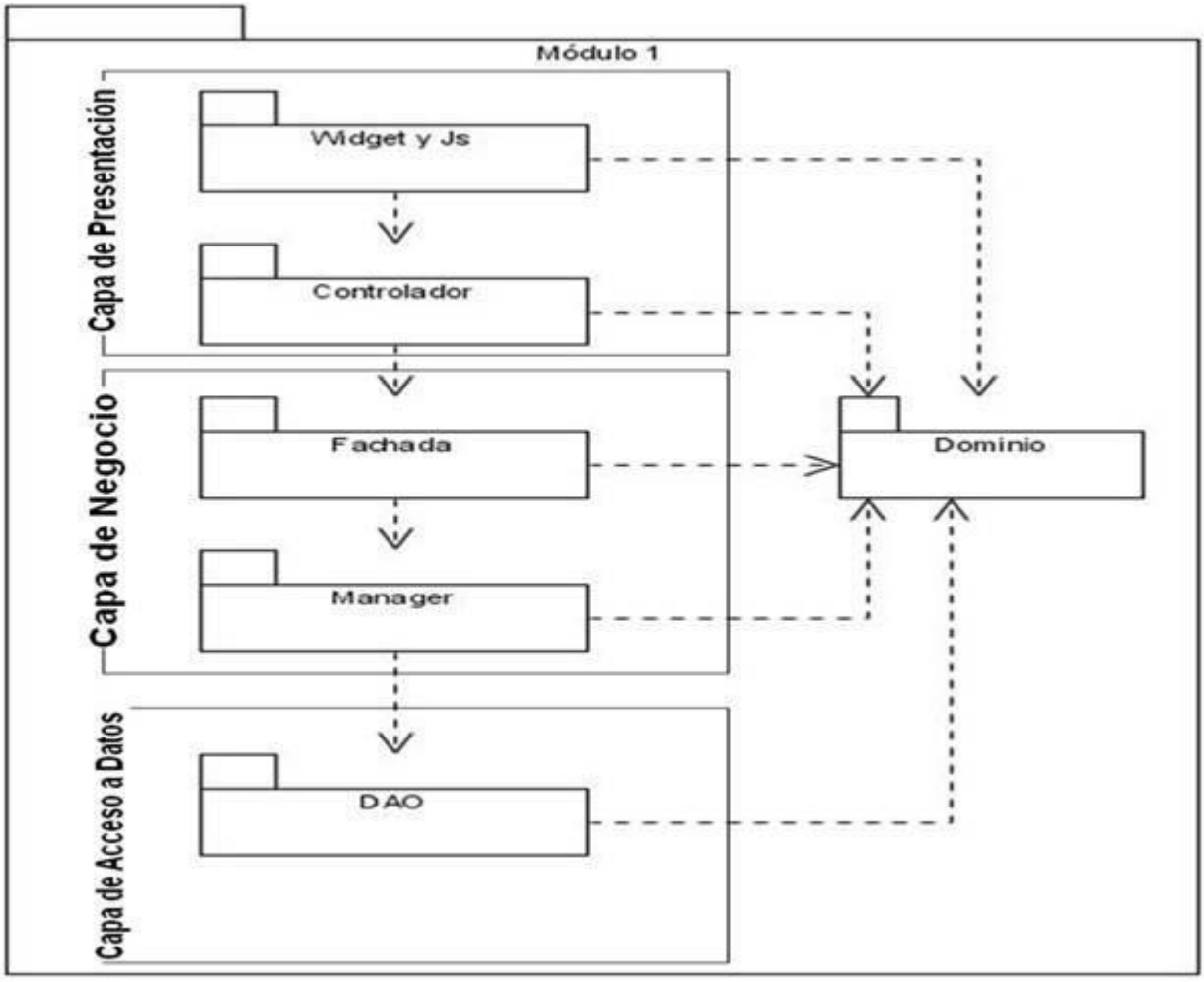
La arquitectura que se propuso en el proyecto y sobre la que se realiza el desarrollo del software está compuesta por tres capas lógicas principales:

Capa de Presentación.

Capa de Negocio.

Capa de Acceso a Datos.

Donde además existe una capa transversal a las otras capas con los objetos del dominio.



Estructura de las capas lógicas de la arquitectura.

Capa de Presentación

Esta capa estará dividida en dos subcapas, una del lado del cliente y otra del lado del servidor. La subcapa del lado del cliente utiliza el framework dojo para realizar peticiones asíncronas a la subcapa del lado del servidor. Y la subcapa del lado del servidor es la encargada de recibir todas las peticiones de la interfaz de usuario, controlar el flujo de presentación del sistema y enviar las respuestas correspondientes a la interfaz de usuario. La subcapa del lado del servidor estará relacionada con la capa de Negocios a través de la fachada y con la capa de Dominio. En la capa de presentación se destaca la utilización del

framework Spring MVC y Spring Web Flow.

Capa de Negocios

Esta capa también está divida en dos subcapas. En la Fachada se expondrán todas las funcionalidades que la capa de presentación necesitará. Esta subcapa invocará métodos de la subcapa Manager. En la subcapa Manager se implementará el negocio de los módulos en cuestión, y de aquí se accederá de ser necesario a la Capa de Acceso a Datos, a otras Capas de Negocios y a la Capa de Dominio.

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.

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.

La arquitectura definida utiliza el patrón MVC (Modelo-Vista-Controlador) el cual propone utilizar tres capas el Modelo, la Vista y el Controlador; el modelo es la representación del dominio o datos del sistema, la vista se encarga de presentar la interfaz al usuario. En la vista sólo se deben de hacer operaciones simples y el controlador es el encargado de escuchar los cambios en la vista y enviarlos al modelo, el cual le regresa los datos a la vista como un ciclo. Cuando se aplica este patrón el controlador es el que decide que vista se debe de mostrar y que información es la que se envía.

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

Subsistema: Los Subsistemas agruparán un conjunto de módulos relacionados con los procesos que ejecutan. Los Subsistemas, Módulos y Componentes se definirán según las funcionalidades identificadas en el levantamiento de requisitos.

Módulo: Los Módulos agruparán un conjunto de Casos de Uso relacionados con uno o más procesos

bancarios.

Componentes: Los Componentes son un conjunto de funcionalidades comunes que serán reutilizados por otros módulos del sistema. Estos componentes en algunas ocasiones se comportarán como módulos visuales en el sistema, y en otras ocasiones solamente recogerán funcionalidades del negocio. Con el objetivo de lograr un sistema flexible, robusto y adaptable, en el sistema se definirán componentes genéricos que encapsulen procesos o actividades generales.

Caso de uso: Elemento más bajo en la jerarquía, representa funcionalidades del sistema.

2.3. Diseño

El diseño es un proceso iterativo mediante el cual los requerimientos se traducen en un plano para construir el software, el cual se encuentra en el núcleo técnico de la ingeniería del software. 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. (10).

El diseño posee los siguientes propósitos:

- Adquirir una comprensión de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y de interfaz de usuario.
- Crear una entrada apropiada y un punto de partida para actividades de implementación, capturando los requisitos o subsistemas individuales, interfaces y clases.
- Descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo.
- Capturar las interfaces entre los subsistemas antes en el ciclo de vida del software, lo cual es muy útil cuando se utilizan interfaces como elementos de sincronización entre diferentes equipos de desarrollo. (10).

2.4. Diagrama de Paquetes

Durante el desarrollo de software resulta muy conveniente agrupar clases y ficheros por diferentes criterios para lograr la organización y facilitar la comprensión del código de la aplicación, resultando conveniente el desarrollo de los diagramas de paquetes, los cuales muestran cómo está dividido el sistema en agrupaciones lógicas mostrando las dependencias entre estas.

A continuación se muestra la estructura y dependencia de paquetes del módulo Plan de cuentas del Subsistema Contabilidad y una explicación de la composición de cada uno.

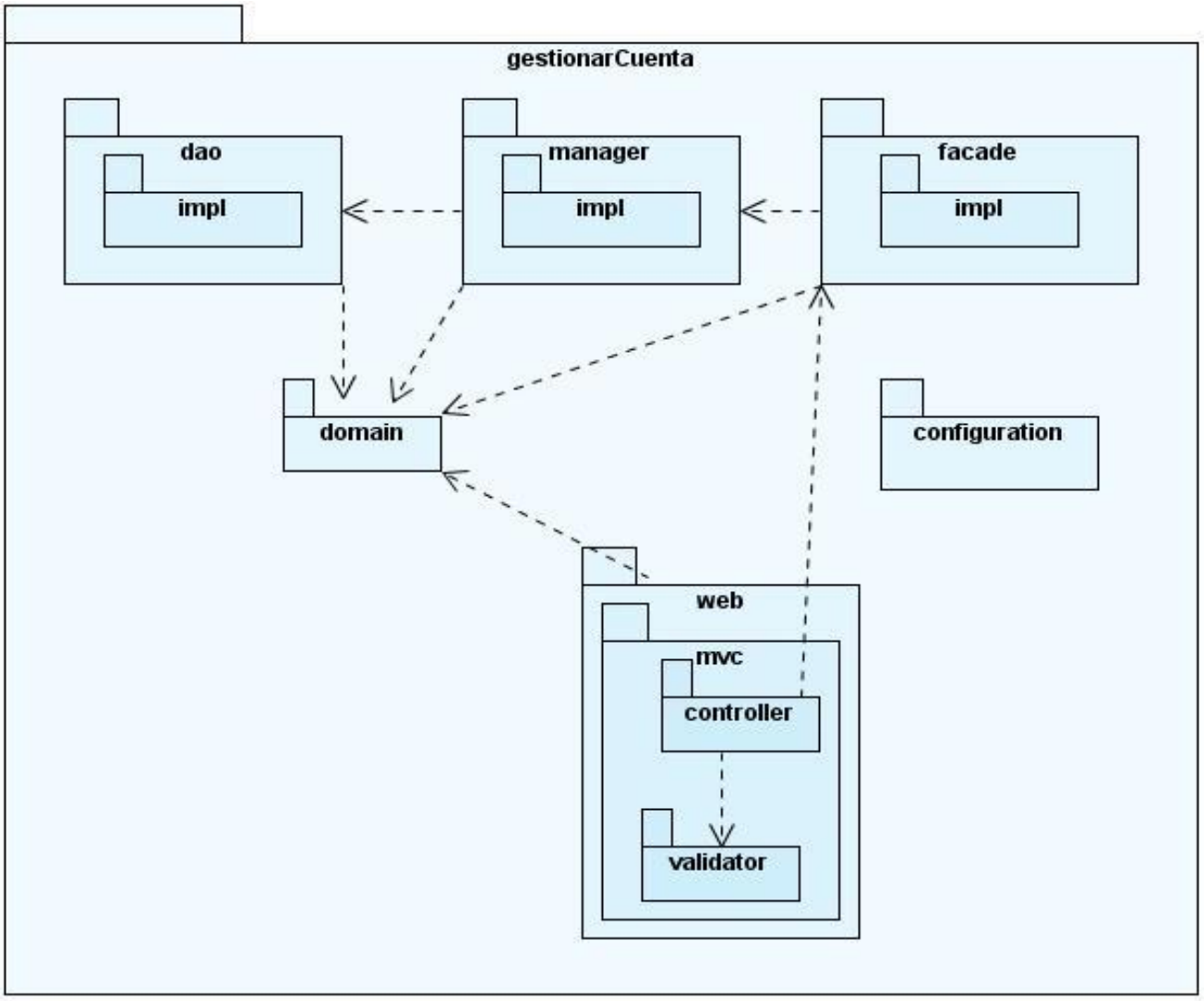


Diagrama de paquetes del módulo Plan de cuentas.

A continuación se explican los paquetes pertenecientes al módulo Plan de cuentas y el módulo Transacciones generales.

Paquete *configuration*: En este paquete se encuentran los ficheros de configuración en formato XML de los diferentes contextos de Spring para el módulo, estos son:

- -servlet.xml: Define el contexto de Spring (Preguntarle a Adolfo si pongo MVC).
- -bussiness.xml: Define el contexto para el negocio.
- -webflow.xml: Define el contexto para Spring WebFlow.
- -dataaccess.xml: Define el contexto para acceso a datos.

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

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

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

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

Paquete *web*: El paquete web agrupa un grupo de clases y paquetes que son los encargados de realizar todo lo referente a la presentación en el lado del servidor.

Paquete *mvc*: En este subpaquete se agrupa todo lo referente a la lógica de presentación cuando se hace uso de SpringMVC.

Paquete *webflow*: En este subpaquete se agrupa todo lo referente a la lógica de presentación cuando se hace uso de Spring WebFlow.

Paquete *controller*: En este paquete están las clases encargadas de responder a las peticiones realizadas por el cliente.

Paquete *serviceFlow*: Contiene las clases encargadas de establecer la comunicación entre el flujo y la

fachada del módulo.

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

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

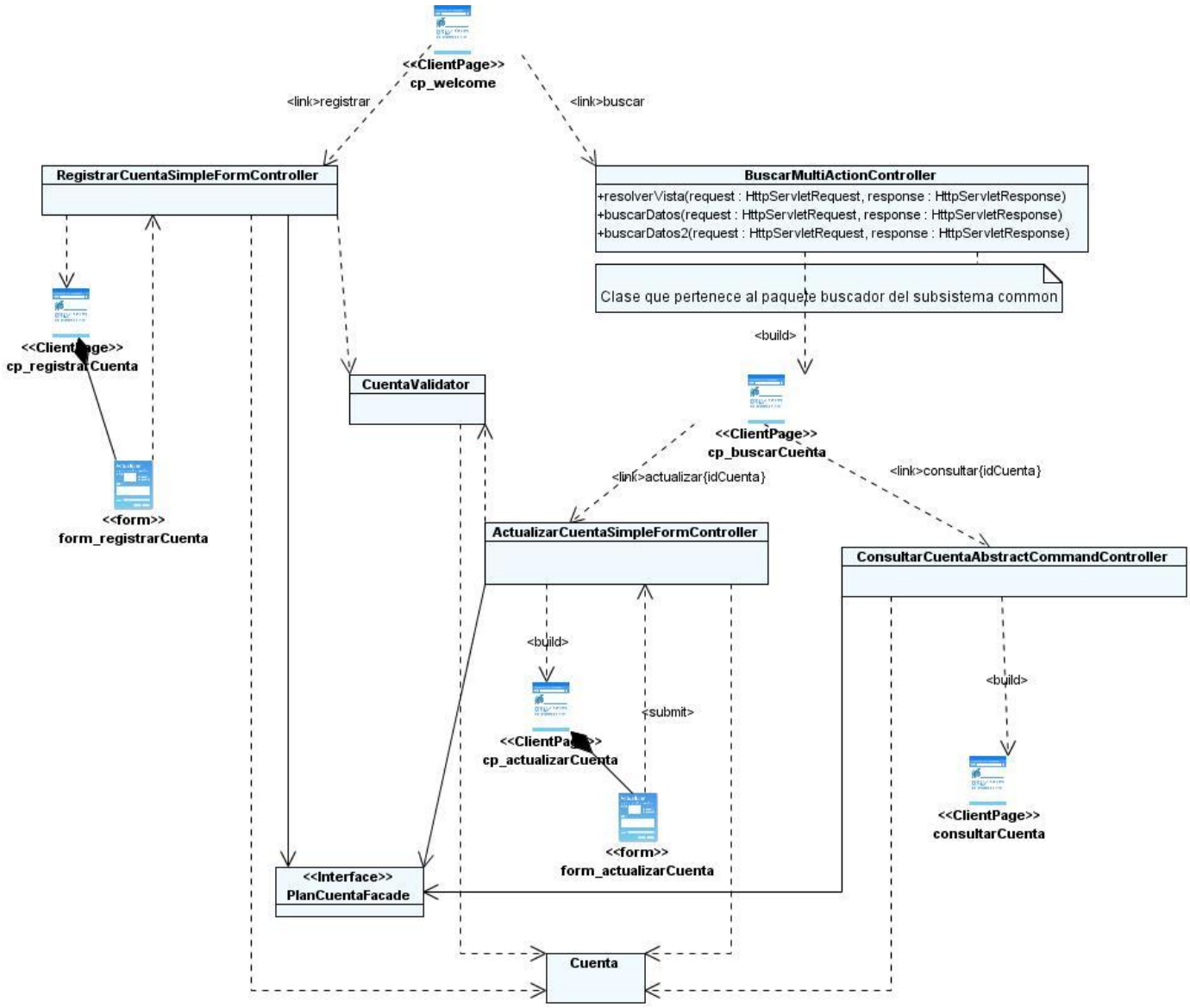
Paquete *propertyEditor*: Agrupa las clases que actúan como intermediarias para convertir datos de la interfaz de usuario en objetos y viceversa.

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

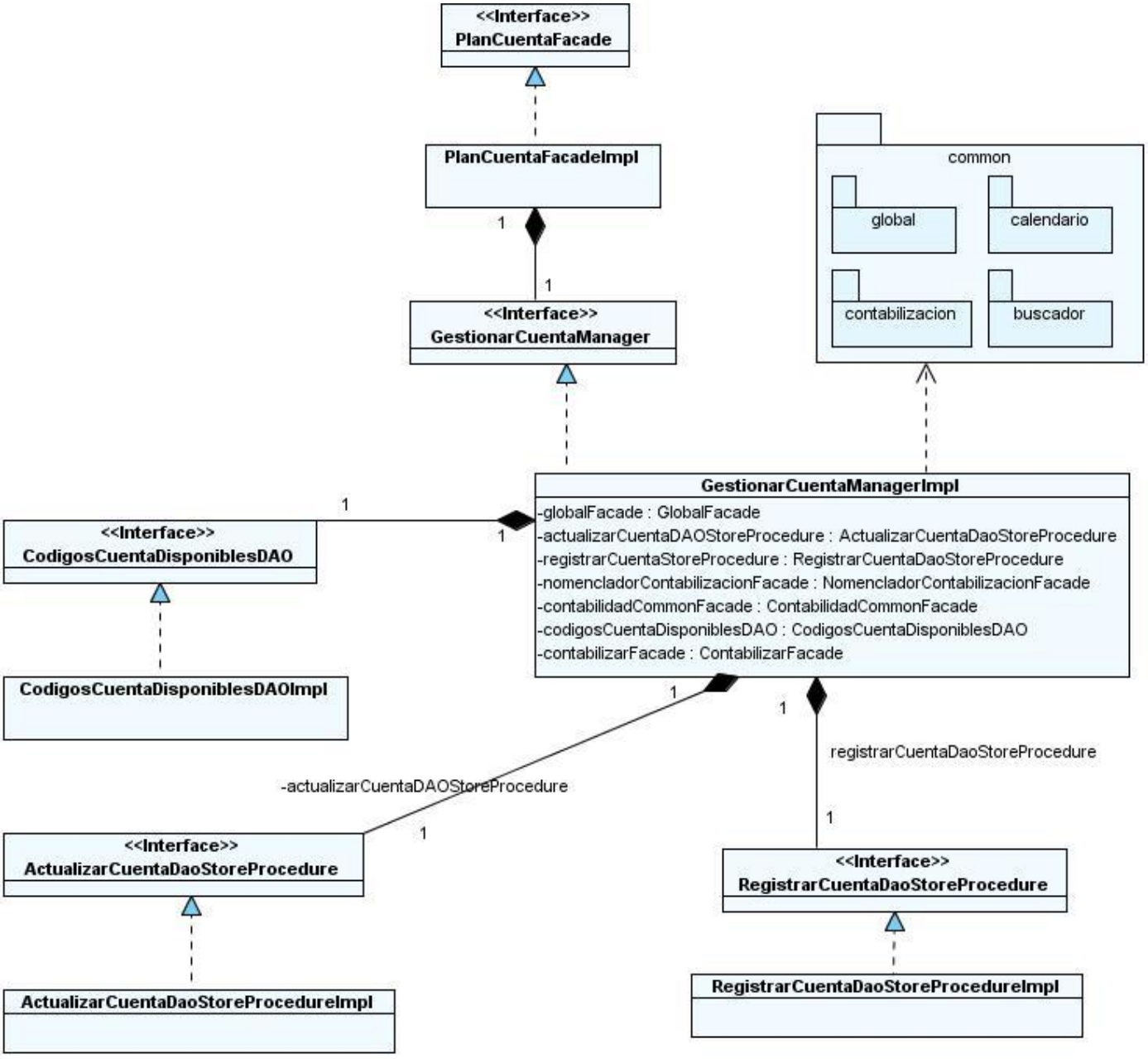
2.5. Diagrama de clases del diseño

El artefacto fundamental del flujo de trabajo diseño definido por RUP es el modelo de diseño el cual es un modelo de objetos que describe la realización de los casos de uso y al mismo tiempo constituye una abstracción del modelo de implementación y el código fuente, es una entrada esencial a las actividades de implementación y prueba. (8).

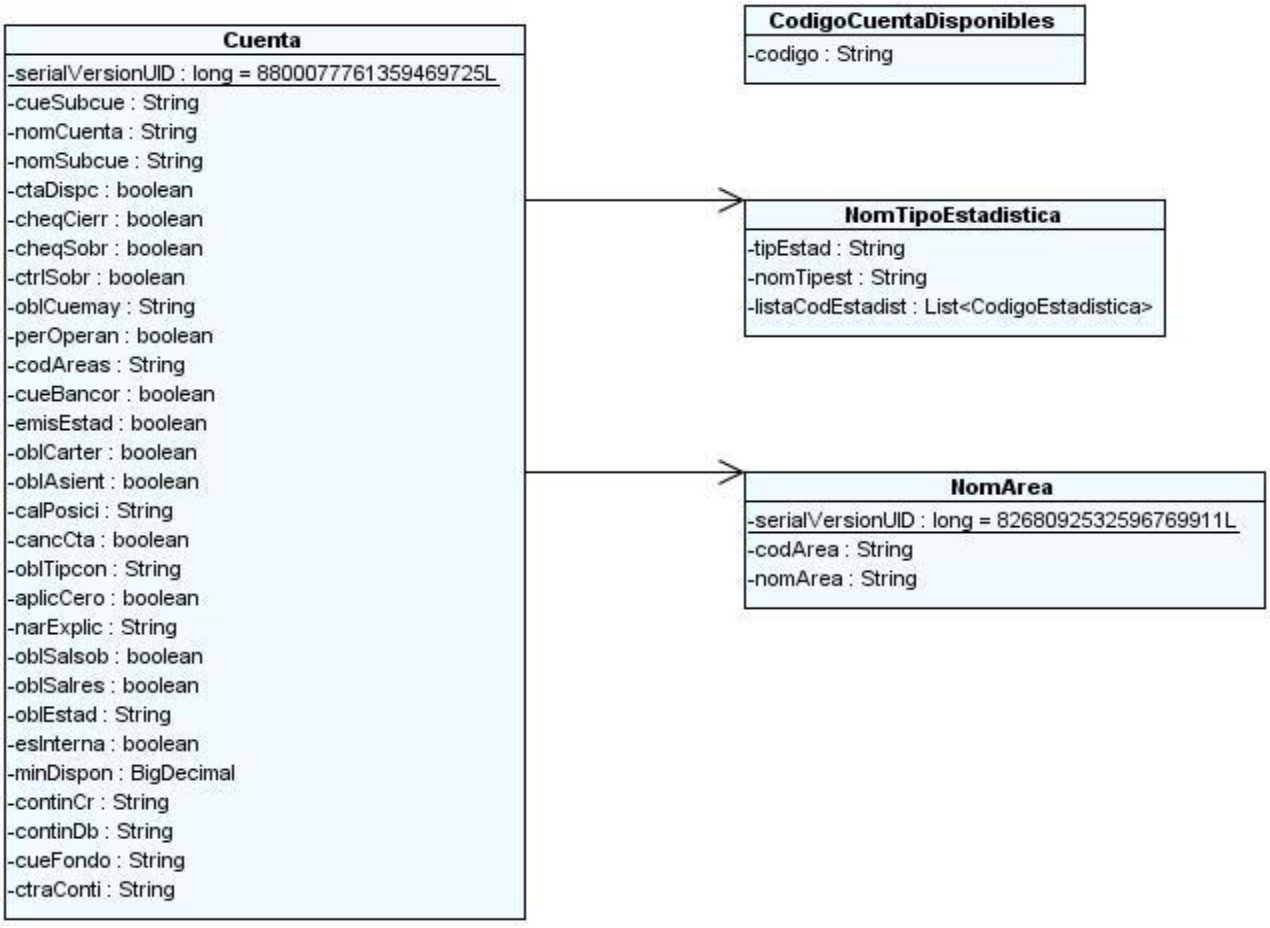
A partir de los patrones de diseño explicados anteriormente se define el siguiente modelo de diseño para los módulos Plan de cuentas y Transacciones generales.



Capa de presentación del diagrama de clases del diseño del módulo Plan de cuentas.



Capa de negocio y acceso a datos del diagrama de clases del diseño del módulo Plan de cuentas.



Capa de dominio del diagrama de clases del diseño del módulo Plan de cuentas.

2.6. Diagrama de interacción del diseño

En el flujo de diseño se utiliza fundamentalmente el diagrama de secuencia. En él 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. El objetivo de realizar diagramas de secuencia hace que se tenga en cuenta el orden y el momento en que se envían los mensajes a los objetos. Los mensajes que se envían entre las clases pueden ser síncronos, el tipo normal de llamada del mensaje donde se pasa el control al objeto llamado hasta que el método finalice, o asíncronos donde mantiene el control el objeto que realiza la llamada.

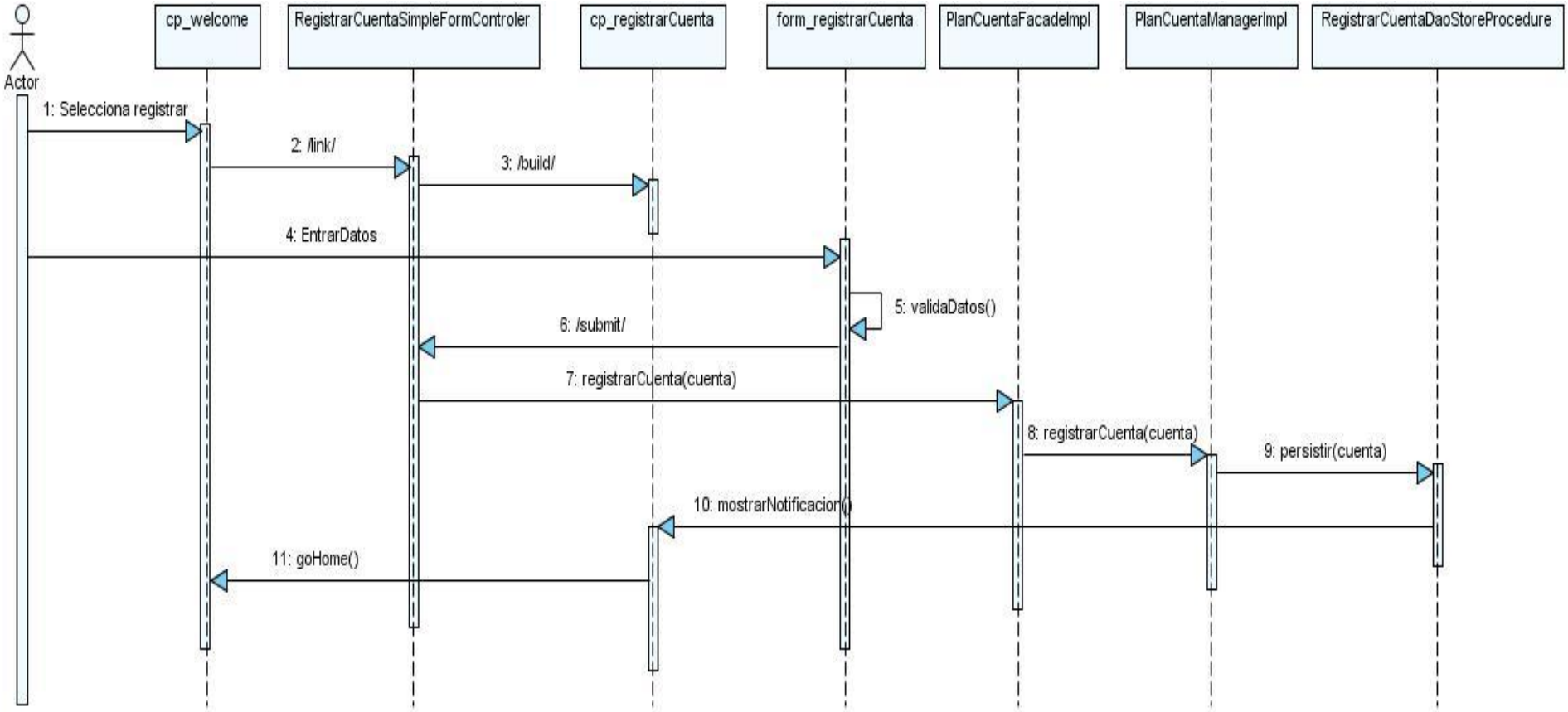
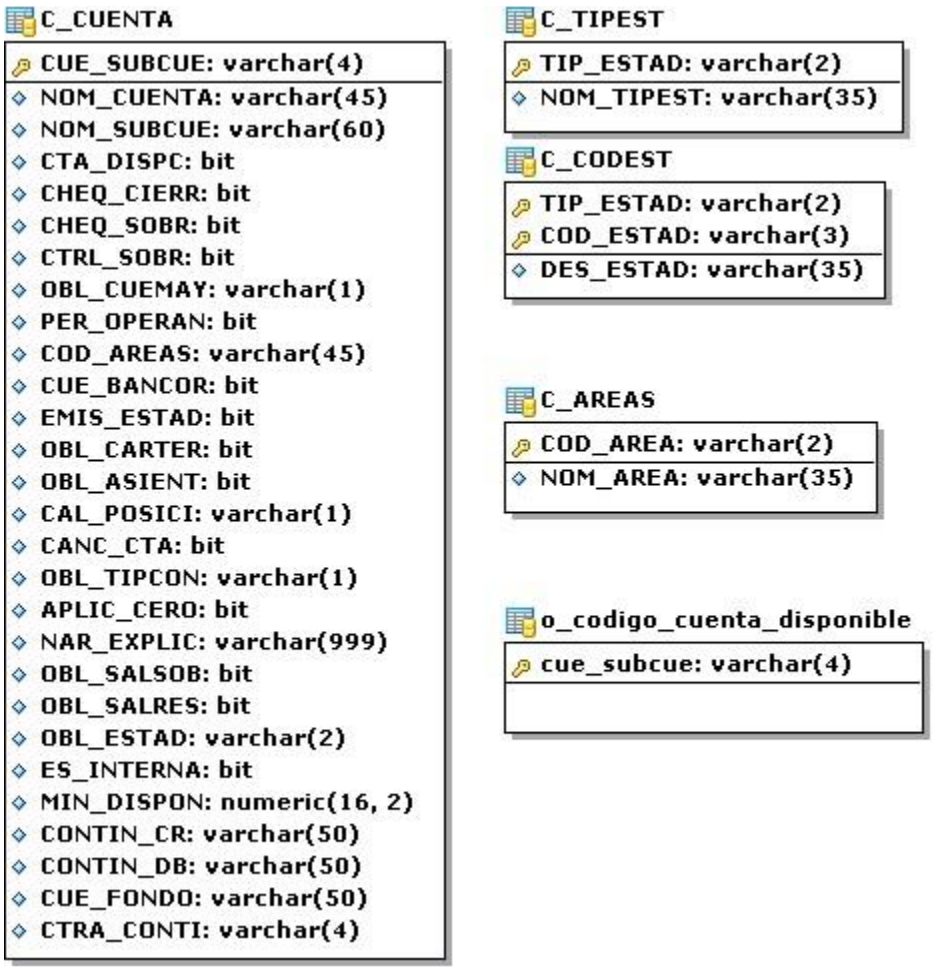


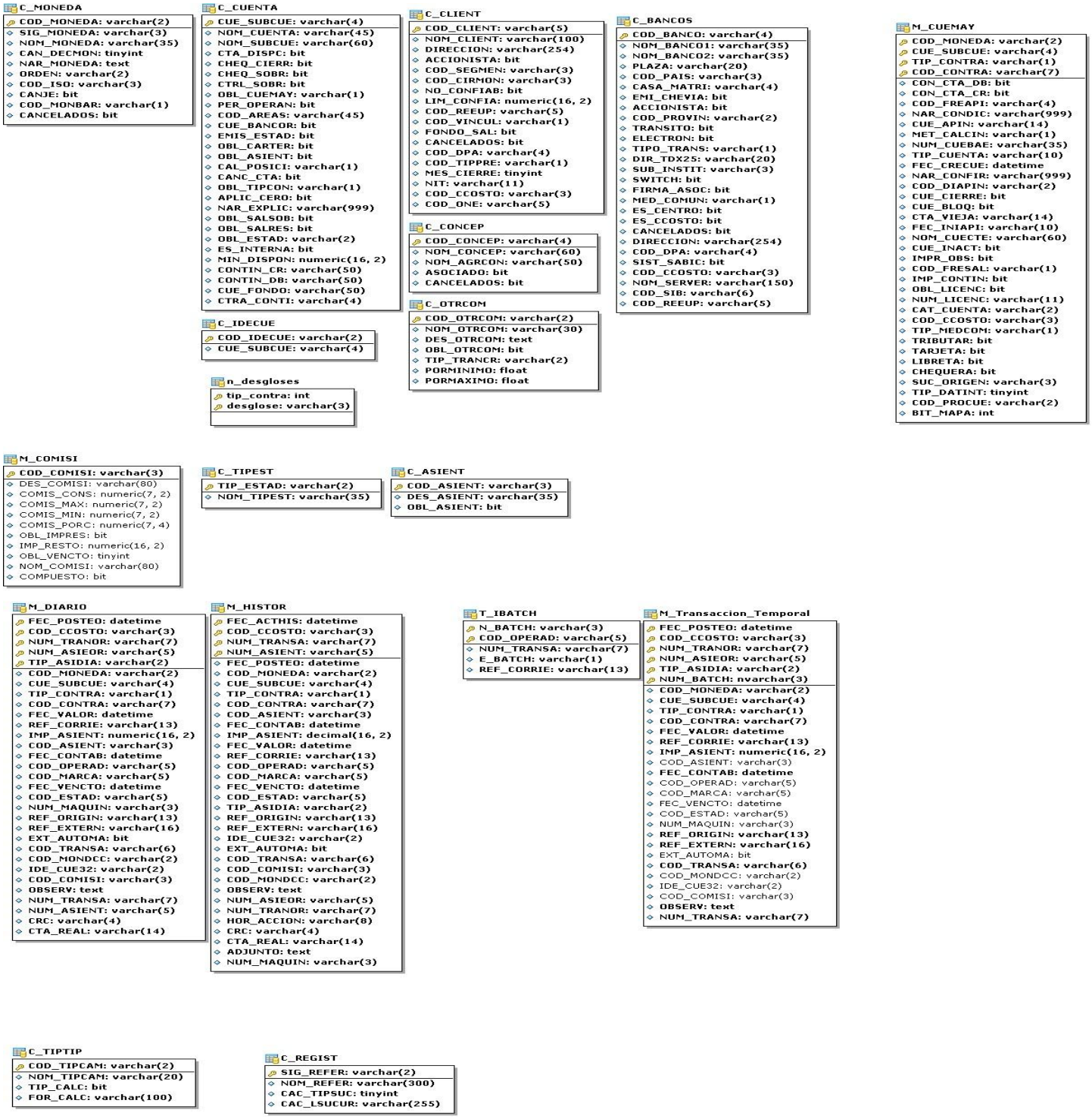
Diagrama de secuencia del caso de uso registrar cuentas del módulo Plan de cuentas.

2.7. Modelo de datos

Un modelo de datos es un modelo abstracto que describe cómo deben ser representados y usados los datos. Sirve para describir la estructura de la base de datos, así como los datos, sus relaciones y las restricciones que deben cumplirse entre ellos. Teniendo en cuenta que los módulos pertenecen a un sistema que ya cuenta con una base de datos, se representarán solamente en el modelo de datos las clases que compondrán la parte que le corresponde a los módulos Plan de cuentas y Transacciones generales. A continuación el modelo de datos físico diseñado para los módulos Plan de cuentas y Transacciones generales.



Modelo de datos del módulo Plan de cuentas.



Modelo de datos del módulo Transacciones generales.

2.8. Patrones de diseño empleados

El diseño fue elaborado siguiendo patrones, que de manera general constituyen soluciones simples y experimentadas a problemas específicos y comunes del diseño. En este caso se emplearon los patrones GRASP (en ingles General Responsability Assignment Software Patterns), los que describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Los patrones GRASP que se utilizaron son los siguientes:

Controlador: 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 fue utilizado en el diseño de los dos módulos de manera general ya que cada método se implementa en la clase que conoce toda la información relacionado con este.

Alta cohesión: Este patrón fue utilizado en el diseño de los dos módulos de manera general ya que se le da a cada clase un alto grado de funcionalidad combinado con una cantidad mínima de operaciones.

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

Durante el diseño de los módulos se emplearon patrones GOF, específicamente:

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

Cadena de Responsabilidad: Cuando desde la vista se solicita información que se encuentra en la base de datos, esta es atendida primeramente por los Controller, luego por la Facade, el Manager y finalmente el DAO, evidenciándose de esta manera la utilización de dicho patrón.

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 CodigoCuentasDisponiblesDAOImpl.

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

2.9. Conclusiones del capítulo

La descripción de la arquitectura del proyecto Quarxo y el estudio de los patrones de diseño contribuyeron a desarrollar exitosamente el diseño de los módulos Plan de cuentas y Transacciones generales que proporciona una entrada apropiada como punto de partida a las actividades de implementación.

Capitulo 3: Implementación de los módulos Plan de cuentas y Transacciones generales de Quarxo.

3.1. Introducción

En el último capítulo del presente trabajo se lleva a cabo la implementación de los módulos Plan de cuentas y Transacciones generales de Quarxo. En el mismo se describirán los estándares de codificación definidos para la escritura del código, se representará la interacción de los componentes mediante el diagrama de componentes y serán detalladas algunas de las clases más importantes del modelo de diseño presentado en el capítulo anterior. Finalmente se probará la solución mediante pruebas de unidad en aras de encontrar errores que imposibiliten el cumplimiento de los requisitos funcionales definidos anteriormente.

3.2. Estándares de codificación

El mantenimiento del código es la facilidad con que el software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores o mejorar el rendimiento. Aunque la legibilidad y el mantenimiento son el resultado de muchos factores, una faceta del desarrollo de software en la que todos los programadores influyen especialmente es en la técnica de codificación.

Un buen método para asegurar que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutina.

Los estándares de codificación que se emplearon en el desarrollo de los módulos en cuestión fueron definidos por el equipo de desarrollo para lograr generalización en la programación del software.

3.3. Convenciones de nomenclatura

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

Ejemplo: PlanCuentaManagerImpl. En este caso el nombre de la clase está compuesto por 4 palabras iniciadas cada una con letra mayúscula.

También se definió que los nombres de las clases tengan relación con la función de estas en el sistema. El nombre de cada clase generalmente concluye con el nombre del paquete al que pertenece, a continuación se muestran ejemplos.

facade: Las interfaces incluidas en este paquete, después del nombre se le agrega la palabra Facade y en el caso del subpaquete impl que contiene las implementaciones a dichas interfaces tendrán el mismo nombre de la interfaz que implementen más la palabra Impl.

Ejemplo: TransaccionGeneralFacade, TransaccionGeneralFacadeImpl.

manager: Las interfaces incluidas en este paquete, después del nombre se le agrega la palabra Manager y en el caso del subpaquete impl que contiene las implementaciones a dichas interfaces tendrán el mismo nombre de la interfaz que implementen más la palabra Impl.

Ejemplo: ContabilizarTransaccionGeneralManager y ContabilizarTransaccionGeneralManagerImpl.

dao: Las interfaces incluidas en este paquete, después del nombre se le agrega la abreviatura DAO y en el caso del subpaquete impl que contiene las implementaciones a dichas interfaces tendrán el mismo nombre de la interfaz que implementen más Impl.

Ejemplo: TransaccionTemporalDAO, TransaccionTemporalDAOImpl.

controller: Las clases incluidas en este paquete, después del nombre se le incorpora el nombre del controlador de Spring del cual hereda. Ejemplo: ActualizarCuentaSimpleFormController.

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

Ejemplo: TransaccionCommand.

validator: En este paquete la nomenclatura de las clases está determinada por el nombre de estas más la palabra Validator.

Ejemplo: CuentaValidator.

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

De forma general el nombre de los métodos, los atributos y las variables dentro de las clases se escriben comenzando con letra minúscula, en caso de que sea un nombre compuesto se empleará notación Camel

Casing, que es muy similar a Pascal Casing con la excepción de la letra inicial. El nombre de estos siempre debe sugerir la función que realizan. Lo mismo se aplica a los nombres de ficheros de código javascript, sus funciones y variables internas.

Con el objetivo de incrementar la legibilidad del código también se emplearon comentarios en las funciones más complejas.

3.4. Tratamiento de errores

Es necesario que el personal que opere con el sistema sea notificado cuando exista un error de entrada de datos o cualquier otro error generado por el sistema. En los módulos en cuestión se evitan, minimizan y tratan los posibles errores mostrando mensajes en un lenguaje de fácil comprensión para los usuarios.

3.5. Diagramas de componentes

Los diagramas de componentes describen los elementos físicos y sus relaciones.

Representan todos los tipos de elementos del software que entran en la fabricación de aplicaciones informáticas. Ejemplos de componentes pueden ser paquetes, archivos de datos, bibliotecas cargadas dinámicamente y ejecutables.

A continuación se explican de manera general los componentes utilizados.

El componente **Buscador** ofrece un conjunto de clases que constituyen el motor de búsqueda, presentando una interfaz gráfica mediante la cual se realiza la búsqueda de diferentes elementos en dependencia del módulo donde se emplee. Ejemplo en Plan de cuentas se utiliza para buscar cuentas, donde existen diversos criterios para refinar la búsqueda.

En el caso del componente **Contabilización** engloba un grupo de clases necesarias para realizar la contabilización de las operaciones. El componente Transacciones generales utiliza **Contabilización** principalmente para registrar las transacciones, es decir para contabilizarlas y Plan de cuentas lo utiliza para realizar validaciones necesarias antes de cancelar una cuenta.

El componente **Seguridad** como su nombre lo indica es el encargado de mantener la seguridad en todo el sistema. Este intercepta las peticiones y comprueba los permisos en dependencia del usuario que las realice, de aquí que sea necesaria la comunicación con el mismo para garantizar la seguridad de los módulos Transacciones generales y Plan de cuentas.

Impresión es un componente que tiene como función brindar las clases que contienen las funcionalidades mediante las cuales es posible imprimir determinada información y de forma automática las operaciones que realicen contabilización. Se utiliza en el módulo Transacciones generales después de contabilizar.

El componente **dataAccess** 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.

A continuación se muestra el diagrama de componentes de los módulos Transacciones generales y Plan de cuentas. Se puede apreciar en este diagrama que ambos módulos consumen servicios de los componentes contabilización, global, buscador, seguridad y dataAccess.

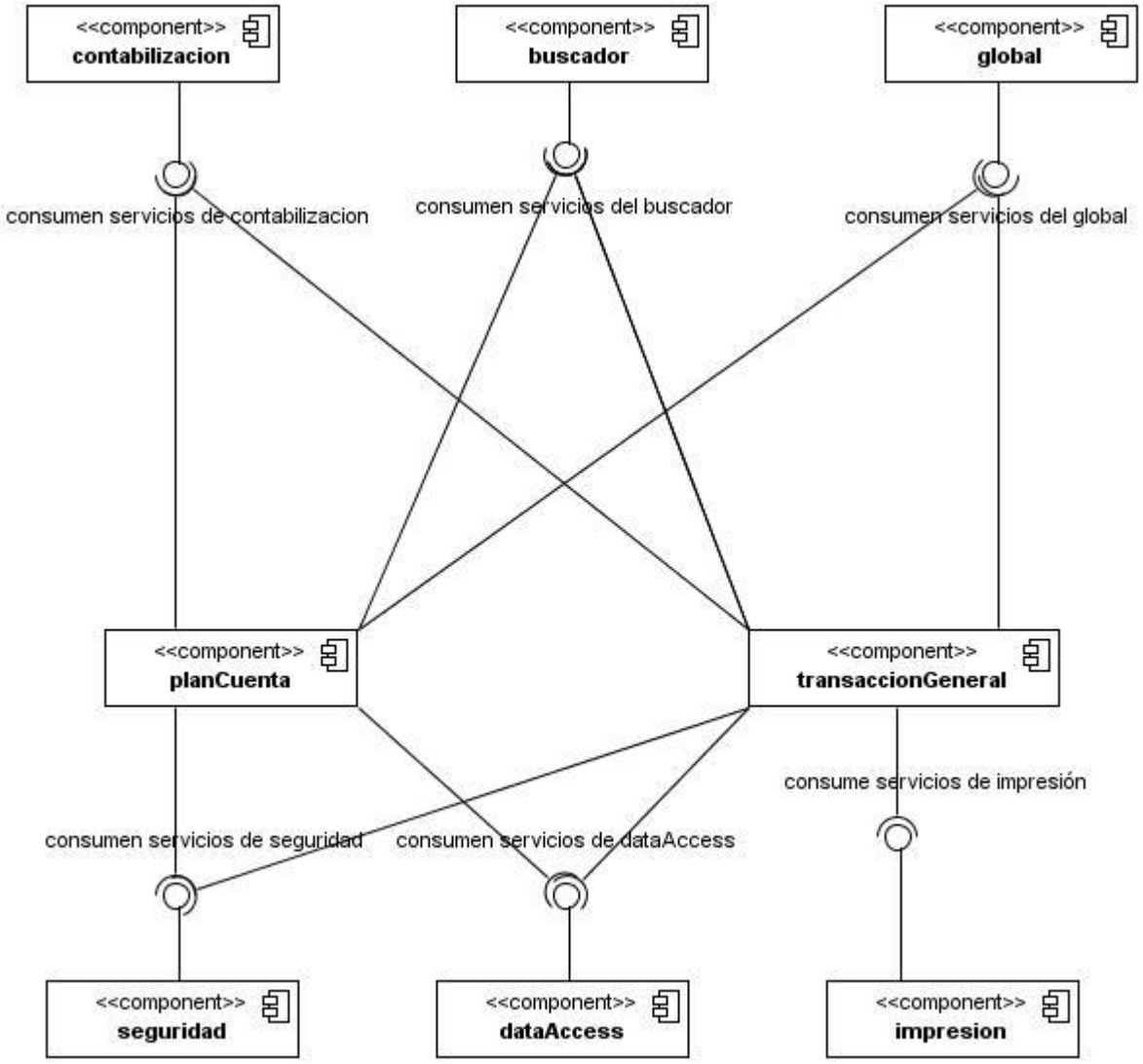


Diagrama de componentes de los módulos Plan de cuentas y Transacciones generales.

3.6. Descripción de algunas clases principales y sus funcionalidades.

A continuación serán descritos los atributos y métodos de algunas de las clases más importantes para los módulos Plan de cuentas y Transacciones generales desde el punto de vista funcional.

- **Módulo plan de cuenta**

Nombre: CargarDatosMultiActionController	
Tipo de clase: Controladora	
Atributo	**Tipo**
planCuentaFacade	PlanCuentaFacade
Para cada responsabilidad:	
Nombre:	**Descripción**:
cargarContingenciaCreditos(HttpServletRequest request, HttpServletResponse response)	Escribe en el response un JSON que contiene los nombres y códigos de las cuentas que son contingencia créditos para ser listadas en las vistas.
cargarContingenciaDebitos(HttpServletRequest request, HttpServletResponse response)	Escribe en el response un JSON que contiene los nombres y códigos de las cuentas que son contingencia débitos para ser listadas en las vistas.
cargarInformacionEstadistica(HttpServletRequest request, HttpServletResponse response)	Escribe en el response un JSON que contiene los nombres y tipos de estadísticas para ser listadas en las vistas.
cargarContrapartidaCuenta(HttpServletRequest request, HttpServletResponse response)	Escribe en el response un JSON con los nombres y códigos de las cuentas que son contrapartida de una cuenta cuyo código es obtenido del parámetro request.
cargarAreasAutorizadas(	Escribe en el response un JSON con los nombres y códigos

HttpServletRequest request, HttpServletResponse response)	de las áreas autorizadas a usar los tipos de cuentas.
cargarCuentasReales (HttpServletRequest request, HttpServletResponse response)	Escribe en el response un JSON con los nombres y códigos de las cuentas reales.
cargarTodasLasCuentas(HttpServletRequest request, HttpServletResponse response)	Escribe en el response un JSON con los nombres y códigos de todas las cuentas.
cargarNombresCuenta(HttpServletRequest request, HttpServletResponse response)	Escribe en el response un JSON con los nombres de todas las cuentas.
cargarSubCuenta(HttpServletRequest request, HttpServletResponse response)	Escribe en el response un JSON con los nombres de todas las subcuentas.
comprobarCodigoCuenta(HttpServletRequest request, HttpServletResponse response)	Dado un código de cuenta escribe en el response true en caso de que no exista una cuenta con ese código o false en caso contrario.
listarCodigosDisponibles(HttpServletRequest request, HttpServletResponse response)	Escribe en el response un JSON con los códigos disponibles para crear cuentas.

Nombre: GestionarCuentaManagerImpl
Tipo de clase: Manager

Atributo	**Tipo**
globalFacade	GlobalFacade
actualizarCuentaDAOStoreProcedure	ActualizarCuentaDAOStoreProcedure
registrarCuentaStoreProcedure	RegistrarCuentaStoreProcedure
nomencladorContabilizacionFacade	NomencladorContabilizacionFacade
contabilidadCommonFacade	ContabilidadCommonFacade
codigosCuentaDisponibleDAO	CodigosCuentaDisponibleDAO
contabilizarFacade	ContabilizarFacade
Para cada responsabilidad:	
Nombre:	**Descripción**:
registrarCuenta (Cuenta cuenta, String operador)	Obtiene una cuenta por parámetros y la registra en la BD a través de registrarCuentaStoreProcedure y devuelve un objeto NError con el error correspondiente en caso de que exista.
consultarCuenta (String codigo)	Devuelve la cuenta con el código pasado.
actualizarCuenta (Cuenta cuenta, String operador)	Actualiza la cuenta pasada por parámetros a través de actualizarCuentaDAOStoreProcedure.
listCuentas (String inicio, String fin, boolean contraparte)	Devuelve la lista de cuentas de forma paginada por los parámetros inicio y fin, la obtiene de nomencladorContabilizacionFacade.
listarArea()	Devuelve la lista de áreas autorizadas, la obtiene de globalFacade.
listarPais()	Devuelve la lista de países, la obtiene de globalFacade.
listarTipoEstadistica()	Devuelve la lista de tipos de estadística, la obtiene de

	contabilidadCommonFacade.
listarNombreCuentas()	Devuelve la lista de los nombre de las cuentas, la obtiene de nomencladorContabilizacionFacade.
listarNombreSubCuentas()	Devuelve la lista de los nombre de las subcuentas, la obtiene de nomencladorContabilizacionFacade.
listarCuentas()	Devuelve la lista de las cuentas, la obtiene de nomencladorContabilizacionFacade.
obtenerMensajeError(int codigo)	Devuelve el mensaje de error dado el código pasado, lo obtiene de globalFacade.
obtenerArea(String cod)	Devuelve el área autorizada dado el código pasado, lo obtiene de globalFacade.
obtenerEstadistica(String cod)	Devuelve la estadística dado el código pasado, la obtiene de contabilidadCommonFacade.
listarCodigoCuentasDisponibles()	Devuelve una lista con los códigos de las cuentas disponibles, la obtiene de globalFacade.
paginarCodigoCuentaDisponibles (int posicionInicial, int cantidadElementos, String value)	Devuelve una lista con los códigos de las cuentas disponibles de forma paginada, la obtiene de codigosCuentaDisponiblesDAO.
cantidadAsientosPorCueSubcue(String cueSubcue)	Devuelve la cantidad de asientos asociados a la cuenta con el código pasado, la obtiene de contabilizarFacade.

- **Transacciones generales**

Nombre: ContabilizarTransaccionGeneralManagerImpl	
Tipo de clase: Controladora	
Atributo	**Tipo**

accesoCAsientoDAO	CAsientoDAO
accesoTransaccionTemp	TransaccionTemporalDAO
accesoTBatch	TIbatchDAO
accesoGeneradorBatch	TIbatchDAOStoreProcedure
accesoTipoOperaciones	OperacionDAO
globalFacade	GlobalFacade
contabilizarFacade	ContabilizarFacade
nomencladorContabilizacionFacade	NomencladorContabilizacionFacade
contabilidadComonFacade	ContabilidadCommonFacade
Para cada responsabilidad:	
Nombre:	**Descripción**:
generarNuevaReferenciaCorriente(String TipoOperacion)	Devuelve la nueva referencia corriente para el tipo de operación pasado, la obtiene de contabilizarFacade.
listarCodigosAsientos()	Devuelve la lista de códigos de asientos, la obtiene de accesoCAsientoDAO.
listarIdentificCuentasCorrientes()	Devuelve la lista de identificadores de Cuentas corrientes, la obtiene de nomencladorContabilizacionFacade.
listarOperaciones()	Devuelve la lista de operaciones, la obtiene de accesoTipoOperaciones.
obtenerTransaccion(String numTransaccion, int inicio, int cantidad)	Devuelve la lista de los Asientos del histórico dado el número de una transacción, de forma paginada, la obtiene de contabilizarFacade.

salvarTransaccionTemporal(List<Asiento> lista, String operador, String numBatch)	Salva la transacción temporal registrando los asientos en la BD a través de accesoTransaccionTemp.
generarNumeroAsiento(int numero)	Devuelve el número pasado con 5 dígitos, agregándole ceros delante hasta completar.
listarReferenciaCorrientesDeAsientos(String tipAsidia)	Devuelve la lista de referencias corrientes de los asientos con un tipo de asiento igual al pasado por parámetros, la obtiene de nomencladorContabilizacionFacade.
listarCodContra(String codMoneda, String cueSubCue, String tipContra)	Devuelve la lista de códigos contrapartes de la BD con la moneda, cuenta y tipo de contraparte pasados por parámetros, la obtiene de nomencladorContabilizacionFacade.
listarDesgloses(String codMoneda, String cueSubCue, String tipContra, String codContra)	Devuelve la lista de desgloses de la BD con la moneda, cuenta, tipo de contraparte y código de contraparte pasados por parámetros, la obtiene de nomencladorContabilizacionFacade.
numeroBatchPorOperador(String operador)	Devuelve la lista de números Batch para el operador pasado
modificarFechaVencimiento(List<Asiento> asientos, String refCorriente, Date fechaContable, Date fechaValor, Date fechaVencimiento, String obsv, String tipoTransaccion)	Modifica los asientos pasados por parámetro asignándoles los restantes parámetros pasados a cada asiento.
modificarFechaValor(List<Asiento> asientos, String refCorriente, Date fechaContable,	Modifica los asientos pasados por parámetro asignándoles los restantes parámetros pasados a cada

Date fechaValor, String obsv, String tipoTransaccion)	asiento.
cancelarTransaccion(List<AsientoMHistor> asientos, String refCorriente, Date fechaContable, Date fechaValor, String obsv, String tipoTransaccion)	Permite cancelar la transacción pasándole todos sus parámetros.
listarAsientosDiario(String refCorriente, String tipAsidia, int inicio, int cantidad)	Devuelve una lista de los asientos del diario con la referencia corriente y tipo de asiento igual al pasado por parámetros, de forma paginada,
obtenerCantidadTransaccion(String numTransaccion)	Devuelve la cantidad de asientos del histórico, que pertenecen a una transacción pasada por parámetros.
fechaContableSistema()	Devuelve la fecha contable del sistema, la obtiene de globalFacade.
fechaPosteoSistema()	Devuelve la fecha de posteo del sistema, la obtiene de globalFacade.
incrementarDesglose(String refCorriente)	Devuelve el desglose incrementado contenido en la referencia corriente pasada.
obtenerCantidadTransaccionTipAsidia(String numTransaccion, String tipAsidia)	Devuelve la cantidad de asientos de una transacción con el tipo de asiento igual al pasado por parámetros.
primeraValidacion(List<Asiento> asientos)	Realiza la primera validación de los asientos a contabilizar y devuelve el código del error en caso de que se produzca alguno.
segundaValidacion(List<Asiento> asientos)	Realiza la segunda validación de los asientos a contabilizar y devuelve una lista de mensajes.
cifraDeCuadrade(List<Asiento> asientos)	Devuelve la cifra de cuadre de los asientos pasados

	por parámetros.
listarAsientosID(List<AsientoId> idAsientos)	Devuelve la lista de asientos que tienen el id incluido en los pasados por parámetros.
listarAsientosDiarioTotalFilas(String refCorriente, String tipAsidia, int inicio, int cantidad)	Devuelve la cantidad de asientos del diario dada una referencia corriente y un tipo de asiento.
convertirAsiento(AsientoMHistor asientoHistor)	Devuelve el Asiento resultado de convertir el AsientoMHistor pasado por parámetros a Asiento.
contabilizar(List<Asiento> asientos, String codigoOperador, String numBatch, DatosAdicionalesComand datosAdicionalesComand)	Contabiliza la transacción pasada por parámetros y devuelve el error en caso de existir.
listarMonedas()	Devuelve la lista de monedas, las obtiene de nomencladorContabilizacionFacade.
listarCuentas()	Devuelve la lista de cuentas, las obtiene de nomencladorContabilizacionFacade.
consultarCuenta(String codigo)	Devuelve la cuenta con el código pasado por parámetro, la obtiene de nomencladorContabilizacionFacade.
listarNomTipEstadistica()	Devuelve la lista de tipos de estadísticas, la obtiene de contabilidadComonFacade.
listarBancoNombreCodigo(List<String> ids)	Devuelve la lista de los nombres y códigos de los bancos con el código existente en la lista pasada por parámetros, la obtiene del globalFacade.
obtenerClienteJuridicoById(List<String> ids)	Devuelve la lista de los clientes jurídicos con el id existente en la lista pasada por parámetros, la obtiene

	del globalFacade.
obtenerListaNomConcepto(List<String> id)	Devuelve la lista de los conceptos con el id existente en la lista pasada por parámetros, la obtiene del nomencladorContabilizacionFacade.
obtenerListaNomOtroConcepto(List<String> id, String cueSubcue)	Devuelve la lista de los otros conceptos con el id existente en la lista pasada por parámetros y el código de cuenta igual al pasado por parámetro, la obtiene del nomencladorContabilizacionFacade.
esReservaOSobregiro(String cuesubcue)	Devuelve true si la cuenta es reserva o sobregiro, false en caso contrario.
listarBancoNombreCodigo()	Devuelve la lista de los bancos con el nombre y código solamente, los obtiene de globalFacade.
listarClienteJuridicoNombreCodigo()	Devuelve la lista de los clientes juridicos con el nombre y código solamente, los obtiene de globalFacade.
listarConceptos()	Devuelve la lista de los conceptos, los obtiene de nomencladorContabilizacionFacade.
listarOtroConcepto(String cuesubcue)	Devuelve la lista de los otros conceptos con el código de cuenta igual al pasado por parámetro, los obtiene de nomencladorContabilizacionFacade.
listarComisiones()	Devuelve la lista de las comisiones, los obtiene de nomencladorContabilizacionFacade.
obtenerComision(String codigo)	Devuelve la comisión con el código igual al pasado por parámetros.
obtenerIdentificCuentasCorrientes(String ideCue)	Devuelve la lista de los identificadores de cuentas corrientes, los obtiene de nomencladorContabilizacionFacade.

transaccionesTemporales(String operador, String numBatch)	Devuelve la lista de asientos temporales dado el numBatch.
CalcularTipoCambio(String Moneda1, String Moneda2, Date fecha, BigDecimal importe, boolean comision, String codCompra, String codVenta)	Devuelve el tipo de cambio de una moneda a otra, lo obtiene de globalFacade.
obtenerCentroCosto()	Devuelve el centro de costo, lo obtiene de contabilizarFacade.
calzarMonedas(List<Asiento> asientos)	revisar
estaCuadradaMonImpFVal(List<Asiento> asientos)	Devuelve true si la transacción esta cuadrada, false en caso contrario. Esta cuadrada cuando la suma de los importes es 0.
estaCuadradaMonImpFValFVen(List<Asiento> asientos)	Devuelve true si la transacción esta cuadrada agrupando por monedas, false en caso contrario. Esta cuadrada cuando la suma de los importes para los asientos con la misma moneda es 0.
convertirAsiento(List<AsientoTemporal> listado)	Devuelve una lista de asientos resultado de convertir la lista pasada de AsientoTemporal a lista de Asiento.
obtenerMonedaBase()	Devuelve la moneda base, obtenida a través de globalFacade.
eliminarTransaccionTemporal(String operador, String numeroBatch, boolean elimNumero)	Elimina la transacción temporal con el operador y numeroBatch pasado por parámetro.
obtenerTipoEstadistica(String codigo)	Devuelve el tipo Estadística dado el código, este tipo es obtenido a través de contabilidadComonFacade
obtenerVariableSistema(String nombre,	Devuelve la variable del sistema dado el nombre de la

String tipo)	variable y tipo de esta, esta es obtenida a través de globalFacade;
agruparAsientos(List<Asiento> asientos)	Agrupa los asientos por moneda y fecha valor y devuelve un Map<String, BigDecimal> con la suma de los importes por estos grupos.

3.7. Elementos importantes en la implementación

3.7.1. Utilización de Spring MVC Framework

Como se propone en el capítulo anterior se utiliza el framework Spring MVC por las ventajas que este brinda. En el módulo Plan de cuentas este es utilizado en todos los casos de usos, facilitando el envío del objeto cuenta y otros datos entre el Controlador y las paginas *.jsp. Este framework permite encapsular los valores de los campos del formulario dentro de un objeto Cuenta 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 presenta una parte del código del formulario de la página **RegistrarCuenta.jsp** donde se puede observar la forma de relacionar los campos de entrada con los atributos del objeto mediante la etiqueta `<form:input` y su atributo `path`.

```
<form:form id="formulario" commandName="registrarCuenta">…
   <form:input path="nomCuenta" id="nomCuenta"cssClass="texto"/>…
   <form:input path="cueSubcue" id="cueSubcue" cssClass="texto"/>…
```

A continuación se muestra un fragmento de un bloque de código del controlador **RegistrarCuentaSimpleFormController** donde se muestra la forma en que se obtiene el objeto que llega creado con todos los valores entrados desde la página al controlador.

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

3.7.2. Utilización de Spring WebFlow Framework

Este framework es utilizado en el módulo Transacciones generales dado que en este se realiza un proceso complejo principalmente para la creación y registro de las transacciones que se logra mediante un flujo en el que se va construyendo mediante diferentes interfaces dicha transacción y al estar construida ocurren varias etapas de validaciones que son necesarias para evitar el registro de transacciones incorrectas.

A continuación se describen elementos importantes del caso de uso Transaccion general.

En este fragmento de código se muestran los atributos de la clase **TransaccionCommand**, una instancia de esta clase contendrá los datos principales que navegarán en el flujo para la creación y registro de la transacción.

```
…
 private Date fechaContable;
 private String tipoOperacion;
 private String referenciaCorriente;
 private List<Asiento> asientos;
```

Aquí se muestra la declaración de la variable ***transaccionCommand*** que se define en el flujo **transaccioinGeneral-flow.xml** para la creación y registro de la transacción, este command representa una instancia de la clase **TransaccionCommand**.

```
…
<var name= "transaccionCommand" class = "...TransaccionCommand"/>

…

<on-start>
            <evaluate
expression="transaccionGeneralMultiAction.inicializaDatos"/>
     <set name="numero" value="0"></set>
</on-start>
…
```

El fragmento de código a continuación muestra un estado del flujo **transaccioinGeneral-flow.xml** donde se puede observar cómo se utiliza la variable ***transaccionCommand*** como modelo de la vista ***transaccionGeneral*** que es la vista principal de este flujo. En dicha vista se muestran los datos existentes en la transacción que se está creando y a partir de esta se puede acceder a las diferentes vistas que permiten crear, modificar o agregar asientos[5] a la transacción así como también dicha vista permite eliminar asientos de la transacción.

[5] Son los elementos por lo que están constituidas las transacciones. Representan acciones sobre una cuenta.

```
… <view-state id="inicio" model="transaccionCommand" view="transaccionGeneral">
   <transition on="verAsientos" to="verAsientosState" />
    <transition on="registrar" to="NuevoDbCr">
       <evaluate expression="transaccionGeneralMultiAction.registrar"/>
    </transition>
    <transition on="ExtraerCartera" to="ExtraerCartera">
        <evaluate
expression="transaccionGeneralMultiAction.compartirTransaccionSession"/>
    </transition>
    <transition on="CancelarPosteoAdelantado" to="CancelarPosteoAdelantado">
        <evaluate
expression="transaccionGeneralMultiAction.compartirTransaccionSession"/>
    </transition>
    <transition on="actualizar" to="actualizar">
        <evaluate
expression="transaccionGeneralMultiAction.construirVistaActualizarDBCR"/>
    </transition>
    <transition on="eliminar">
        <evaluate expression="transaccionGeneralMultiAction.eliminarAsiento"/>
    </transition>
        <transition on="aceptar" to="inicio">
             <evaluate
expression="transaccionGeneralMultiAction.prepararTransaccionGeneral" />
        <evaluate expression="transaccionGeneralMultiAction.guardarTransaccion" />
    </transition>
       <transition on="aceptarM" to="inicio">
        <evaluate expression="transaccionGeneralMultiAction.contablizar"/>
    </transition>
       <transition on="cancelar" to="fin">
        <evaluate expression="transaccionGeneralMultiAction.cancelarTransaccion"/>
       </transition> …
```

La vista Nuevo Debito Crédito posee también gran importancia en este flujo ya que mediante esta se pueden crear y agregar a la transacción los diferentes asientos.

3.8. Prueba de software

Las pruebas son los procesos que permiten verificar y revelar la calidad de un producto de software. Son utilizadas para identificar posibles fallos de implementación, calidad, o usabilidad de un software.

Para determinar el nivel de calidad de los módulos implementados se deben efectuar pruebas que permitan comprobar el grado de cumplimiento de las especificaciones iniciales del sistema.

Las pruebas que se le realizarán a los módulos corresponden al nivel de Unidad, las cuales están enfocadas al código fuente de los componentes para verificar todos los flujos de control, probando de manera individual las partes del sistema que han sido desarrolladas. Adecuadas a este nivel se aplicarán los métodos de prueba Caja Blanca para analizar la lógica interna del programa y Caja Negra con el objetivo de verificar que las funciones son operativas a través de la interfaz del software, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, manteniendo la integridad de la información externa.

3.8.1. Pruebas de Caja Blanca, de Cristal o Estructurales

Estas pruebas se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. En la realización de estas pruebas se escogen distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven los valores de salida adecuados. Específicamente consisten en diseñar los Casos de prueba[6] atendiendo al comportamiento interno y la estructura del programa, examinándose la lógica interna sin considerar los aspectos de rendimiento.

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

Asociadas a este método existen cuatro técnicas de prueba; Condición, Flujo de Datos, Bucles y Camino Básico, de las cuales será aplicada esta última, debido a que permite obtener una medida de la complejidad lógica del diseño y usar la misma como guía para la definición de un conjunto de caminos básicos, garantizando que durante la prueba se ejecute al menos una vez cada sentencia del programa.

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

1. V (G) = (A – N) + 2
2. V (G) = P + 1
3. V (G) = R

V (G): Complejidad del grafo.

A: Cantidad total de aristas del grafo.

N: Cantidad total de nodos del grafo.

P: La cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas).

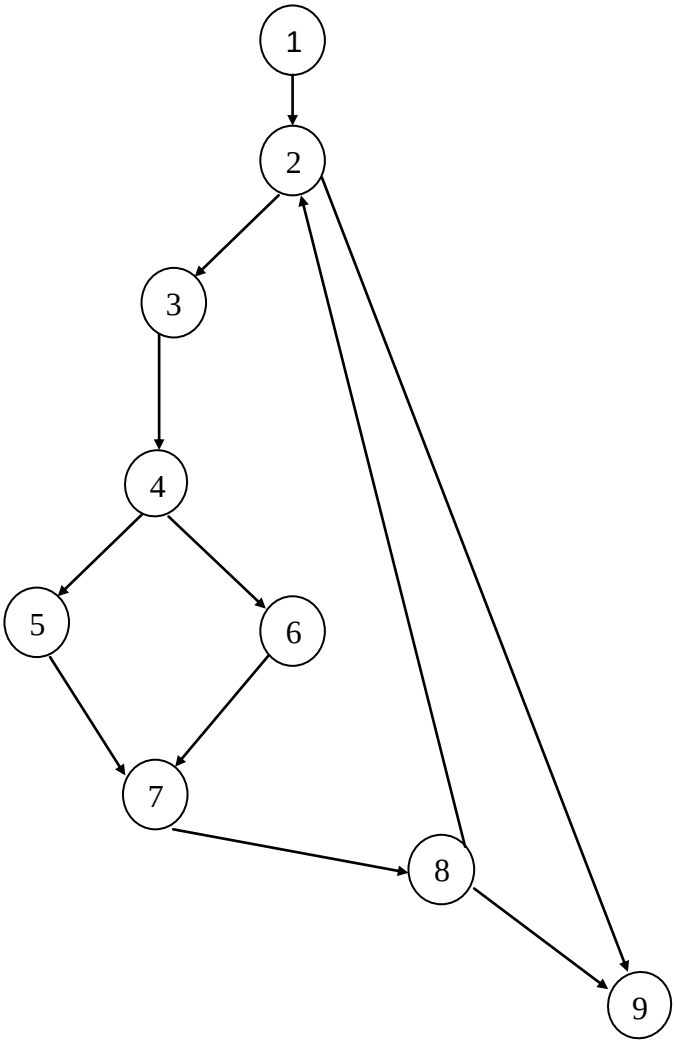
R: La cantidad total de regiones, se incluye el área exterior del grafo, contando como una región más.

3.8.1.1. Aplicación de las pruebas de Caja Blanca, de Cristal o Estructurales

A continuación se analizan y enumeran las sentencias de código del método agruparAsientosMonFValFVen de la clase ContabilizarTransaccionGeneralManagerImpl del módulo Transacciones generales. Este método hace una agrupación de los asientos por moneda, fecha valor y fecha de vencimiento y devuelve un Map con dichos grupos el cual contiene el tipo o identificador del grupo y los asientos por cada grupo. Este método sirve de apoyo al método que verifica si esta cuadrada la transacción por grupos.

```
private Map<String, BigDecimal> agruparAsientosMonFValFVen(
                  List<Asiento> listado) {

Map<String, BigDecimal> sumaImporte = new Hashtable<String, BigDecimal>();//1
Iterator<Asiento> it = listado.iterator();//1
SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy");//1

 while (it.hasNext()) {//2
      Asiento asiento = it.next();//3
      String moneda = asiento.getMoneda().getCodMoneda();//3
      String fecha = formato.format(asiento.getFechaValor());//3
      String fechaVen = formato.format(asiento.getFechaVencimiento());//3

      if (sumaImporte.containsKey(moneda + "_" + fecha + "_" + fechaVen))
{//4
            BigDecimal aux = sumaImporte.get(
                                    moneda + "_" + fecha + "_" +
fechaVen).add(
                                    asiento.getImporteAsiento());//5
            sumaImporte.put(moneda + "_" + fecha + "_" + fechaVen, aux);//5
      } else {
            sumaImporte.put(moneda + "_" + fecha + "_" + fechaVen, asiento
                                    .getImporteAsiento());//6
              }//7
   }//8

            return sumaImporte;//9
```

A continuación se presenta el grafo de flujo asociado al código antes presentado a través de nodos, aristas y regiones:



Dado el grafo de flujo anterior se determina la complejidad ciclomática del método en cuestión, el cálculo se demuestra mediante las 3 formulas especificadas anteriormente, arrojando el mismo resultado en cada caso:

1. V (G) = (A – N) + 2

   V (G) = (11 -9) + 2

   V (G) = 4.

2. V (G) = P + 1

V (G) = 3 + 1

V (G) = 4.

3. V (G) = R

V (G) = 4.

El cálculo efectuado muestran una complejidad ciclomática de valor 4, de manera que existen cuatro posibles caminos por donde el flujo puede circular, este valor representa el número mínimo de casos de pruebas para el procedimiento tratado.

Seguidamente es necesario especificar los caminos básicos que puede tomar el algoritmo durante su ejecución. En estas representaciones se marcan o especifican las aristas al incluirlas por primera vez que son las que determinan que el camino no estaba explorado.

Camino básico #1: ***1 – 2 – 3 – 4 – 5 – 7 – 8 - 9***

Camino básico #2: 1 – 2 – 3 – ***4 – 6 – 7*** – 8 - 9

Camino básico #3: 1 – 2 – 3 – 4 – 5 – 7 – ***8 - 2*** – 3 – 4 – 6 – 7 – 8 – 9

Camino básico #4: 1 – ***2 - 9***

Se procede a ejecutar los casos de pruebas para cada uno de los caminos básicos determinados en el grafo de flujo. Para definir los casos de prueba es necesario tener en cuenta:

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

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

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

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

Caso de prueba para el camino básico # 1

Entrada:

Lista con un único asiento, valores aleatorios Ejemplo:

# Asiento	Moneda	Fecha Valor	Fecha Vencimiento	Importe
1	CUP	15/05/2011	20/05/2011	10

Resultados esperados: Teniendo en cuenta los datos pasados por parámetro se espera como salida del algoritmo un objeto Map<String, BigDecimal> con un elemento como se muestra en la tabla siguiente:

# en la lista	Llave	Valor
1	“CUP_15/05/2011_20/05/2011”	10

El resultado obtenido fue correcto.

Caso de prueba para el camino básico # 2:

Este camino básico nunca ocurrirá, porque tiene una condición que no podrá ser verdadera la primera vez que se ejecute ya que el Map se encuentra vacio, para ejecutarlo se necesita ejecutar otro camino anteriormente y luego ejecutar este. El camino básico # 3 incluye las principales sentencias de código de este camino por lo que el siguiente caso de prueba puede usarse como caso de prueba para el camino básico # 2 también.

Caso de prueba para el camino básico # 3

Entrada:

Lista con dos asientos con igual moneda, fecha de vencimiento y fecha valor Ejemplo:

# Asiento	Moneda	Fecha Valor	Fecha Vencimiento	Importe
1	CUP	15/05/2011	20/05/2011	10
2	CUP	15/05/2011	20/05/2011	5

Resultados esperados: Teniendo en cuenta los datos pasados por parámetro se espera como salida del algoritmo un objeto Map<String, BigDecimal> con un solo elemento como se muestra en la siguiente tabla:

# en la lista	Llave	Valor
1	“CUP_15/05/2011_20/05/2011”	15

El resultado obtenido fue correcto.

Caso de prueba para el camino básico # 4

Entrada:

Lista de asientos vacía.

Resultados esperados: Teniendo en cuenta los datos pasados por parámetro se espera como salida del algoritmo un objeto Map<String, BigDecimal> sin ningún elemento:

El resultado obtenido fue correcto.

3.8.2. Prueba de Caja Negra o Funcional.

Este tipo de prueba se centra en lo que hace el sistema, pero sin dar importancia a cómo lo hace, por tanto no se precisa definir ni conocer los detalles internos de su funcionamiento. En este tipo de pruebas

los casos de prueba solo pretenden demostrar que las funciones del sistema son operativas, que los valores de entradas se aceptan adecuadamente y que se produce una salida correcta. Este tipo de prueba se realiza sobre la interfaz de usuario solamente.

La prueba de Caja Negra no es una alternativa a las técnicas de prueba de la Caja Blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la Caja Blanca.

Con este tipo de pruebas se intenta encontrar:

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

Para el correcto desarrollo de las pruebas existen diferentes técnicas que se muestran a continuación:

Partición de Equivalencia: Divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.

Análisis de Valores Límites: Prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.

Grafos de Causa-Efecto: Permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

3.8.2.1. Aplicación de la Prueba de Caja Negra o Funcional

Las pruebas de caja negra se le realizaron a los módulos usando la técnica Partición de Equivalencia ya que es una de las más efectivas pues permite examinar los valores válidos e inválidos de las entradas existentes en el software y descubre de forma inmediata clases de errores que de otro modo requieren la ejecución de muchos casos antes de detectar el error genérico.

Los resultados obtenidos a través de la realización de las pruebas expuestas anteriormente, fueron satisfactorios desde el punto de vista interno y funcional, dado que los módulos mantuvieron un correcto comportamiento ante las diferentes situaciones en que se probaron.

Los módulos Plan de cuentas y Transacciones generales una vez probados y liberados por parte del departamento de calidad de la universidad (CALISOFT), entraron a las pruebas de aceptación correspondientes en el Banco Nacional de Cuba, donde también fueron aprobados por parte del cliente por lo que dichos módulos se encuentran en completa capacidad para su explotación como elementos importantes del sistema Quarxo.

3.9. Conclusiones del capítulo

En el presente capitulo, se mostraron los elementos principales relacionados con la etapa de implementación de los módulos Plan de cuentas y Transacciones generales de Quarxo. Se abordaron temas referentes a los estándares de codificación que se emplearon en la implementación de estos módulos, los que además fueron definidos para todo el proyecto. Se mostraron las relaciones existentes entre los componentes de dichos módulos y algunos aspectos importantes que se tuvieron en cuenta en la implementación. Se mostraron además como se realizaron diferentes pruebas a los módulos implementados donde se obtuvieron resultados muy positivos.

Tomando en cuenta los resultados de las pruebas realizadas a los módulos en las diferentes etapas se puede afirmar que desde el punto de vista funcional cumple con los requerimientos capturados y especificados en etapas anteriores.

4. Conclusiones generales

El presente trabajo de diploma permite arribar a las siguientes conclusiones:

- A partir del análisis realizado acerca de los sistemas informáticos existentes que permitan gestionar las cuentas y crear transacciones generales, se identificó como principal dificultad para su utilización el alto costo que poseen sus licencias. Por lo que se determinó que era más factible y estrictamente necesario para el país y el BNC el desarrollo de un sistema que permitiera gestionar dichos procesos y automatizar los actuales requerimientos del BNC.
- La utilización de patrones de diseño posibilitó la reutilización del código, el desarrollo rápido y eficaz y facilitará el mantenimiento futuro del sistema.
- La realización del diseño y la implementación de los módulos Plan de cuentas y Transacciones generales a partir de los requisitos previamente definidos, se llevaron a cabo de forma exitosa. Esto se pudo validar mediante diferentes pruebas de unidad que generaron resultados muy positivos.

Como resultado del presente trabajo se obtuvo el diseño e implementación de los módulos Plan de cuentas y Transacciones generales que forman parte del sistema Quarxo. Los objetivos propuestos para este trabajo fueron plenamente cumplidos a través de las actividades realizadas.

5. 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 Postgrest como gestor de base de datos.

6. Referencias bibliográficas

1. **Maldonador, R. 2006**. Estudio de la Contabilidad General. La Habana: Félix Varela, 2006.
2. **CORTÉS., J**. *Contabilidad General*. Barcelona, Editorial: Mentensó., 1932. 2 p.
3. **González, Paula Olmedo.** El Prisma. [En línea] http://www.elprisma.com/apuntes/economia/monedabanca/.
4. **RUZ**, F. C. *Decreto Ley No 172.*, mayo de 1997. [Disponible en: http://www.cubagob.cu/des_eco/banco/espanol/regulaciones_bancarias/bcc-i-3.htm.
5. **Apellániz, Paloma.** El proceso contable. [En línea] 2000. [Citado el: 03 de 03 de 2011.] http://ciberconta.unizar.es/LECCION/cf002/450.HTM.
6. **Apellániz, Paloma.** El proceso contable. [En línea] 2000. [Citado el: 15 de 03 de 2011.] http://ciberconta.unizar.es/LECCION/cf002/520.HTM.
7. **SARDIÑA**, L. B. M. Principales características del Sistema Contable del Banco Central de Cuba. 1998.
8. **Ivar Jacobson, Grady Booch, James Rumbaugh.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Pearson Educación, S.A, 2000. 84-7829-036-2.
9. **Larman, Craig.** UML y Patrones. s.l. : Prentice Hall, 1999. 970-17-0261-1.
10. **Pressman, Roger. 2002**. Ingenieria de Software. Un enfoque practico. 2002.

7. Bibliografías consultadas

1. **Apache**. *Documentación del Servidor HTTP Apache 2.0.* [En línea] http://httpd.apache.org/docs/2.0/es/.
2. **A.Russell, Matthew.** *Dojo The Definitive Guide.* 2008. 978-0-596-51648-2.
3. **Christian Bauer, Gavin King.** *Hibernate in Action.* s.l. : Manning Publications Co, 2005. 1932394-15-X.
4. **Craig Walls, Ryan Breindenbach.** *Spring In Action.* s.l. : Manning Publications Co, 2005. 1-932394-35-4.
5. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns.* s.l. : Addison-Wesley Pub Co, 1995.
6. **Iglesias, Adolfo Miguel.** *Documento de Arquitectura, Modernización Sistema del Banco Nacional de Cuba.* 2009.
7. **Ivar Jacobson, Grady Booch, James Rumbaugh.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Pearson Educación, S.A, 2000. 84-7829-036-2.
8. **Larman, Craig.** *UML y Patrones.* s.l. : Prentice Hall, 1999. 970-17-0261-1.
9. Manual de XHTML. [En línea] [Citado el: 15 de febrero de 2011.] http://manual-xhtml.blogspot.com/2006/05/primer-documento-xhtml.html.
10. Microsoft SQL Server 2005. [En línea] [Citado el: 5 de abril de 2011.] http://www.microsoft.com/sqlserver/2005/en/us/product-information.aspx.
11. **Pérez, Javier Eguíluz.** *Introducción a JavaScript*. 2008.
12. Pruebas de Software. [En línea] [Citado el: 8 de abril de 2011.] http://lsi.ugr.es/~ig1/docis/pruso.pdf.

Bibliografía

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

14. **Saavedra Darias, Mavi y Sánchez Imbert, José Antonio.** *Definición De Los Requerimientos Funcionales Del Módulo Contabilidad Internacional Del Proyecto Banco Nacional.* 2008.

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

16. Visual Paradigm. [En línea] [Citado el: 15 de marzo de 2011.] http://www.visual-paradigm.com.