

Universidad de las Ciencias Informáticas

Facultad 3



Título: Diseño e implementación de los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos del sistema Quarxo.

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


Autor(es): Daileny C. Arias Pupo

Tutor(es): Ing. Yulier Matías León

Ing. Lisset Díaz Mesa

La Habana, Junio de 2011.

“Año 53 de la Revolución”



“La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica.”

Aristóteles

DECLARACIÓN DE AUTORÍA

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

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

Daileny C. Arias Pupo

Ing. Yulier Matías León

Ing. Lisset Díaz Mesa

Tutor (es):**Ing. Yulier Matías León**

Graduado de Ingeniero en Ciencias Informáticas y profesor de la Universidad de Ciencias Informáticas (UCI), analista del proyecto Banco con 2 años de experiencia en el desarrollo de software.

e-mail: [ymatias@uci.cu](mailto:yurias@uci.cu)

Ing. Lisset Díaz Mesa

Graduada de Ingeniera en Ciencias Informáticas y profesora de la Universidad de Ciencias Informáticas (UCI), Jefa del proyecto Banco con 3 años de experiencia en el desarrollo de software.

e-mail: ldiazm@uci.cu



Agradecimientos

A mis padres que son el más grande tesoro de mi vida, por ser los padres más geniales del mundo, por el amor que siempre me han inculcado, por confiar siempre en mí y apoyarme en todo lo que me he trazado en la vida, pues sin ellos nunca hubiera logrado hacer realidad este gran sueño que parecía inalcanzable, por ser lo más grande de mi vida: mi todo.

A mis hermanos Ale y Osmani por la confianza depositada en mí, por el apoyo que siempre he recibido de ellos y por ser mis dos hermanitos bellos.

A toda mi familia en especial a mis abuelitos por estar todos siempre pendientes de mí y por contribuir de una forma u otra en mi formación personal.

Al amor de mi vida Efra por darme siempre fuerzas para continuar y brindarme todo el amor del mundo.

A mi amigo Jorge por brindarme su apoyo cuando lo necesité.

A Anay y Efraín por su preocupación, por haberme brindado apoyo en todo momento y por acogerme en su familia como una hija más.

A mis tutores por la confianza depositada en mí y por su ayuda incondicional.

A mis amigos con los cuales he compartido estos 5 años, a los que están hoy aquí y a los que no están, muchas gracias por su amistad.

A mis compañeros de proyecto los cuales me ayudaron, apoyaron y a los cuales admiro mucho.

A la Universidad de Ciencias Informáticas por permitirme crecer profesionalmente y sobre todo como ser humano.

¡A todos muchas gracias!

***Dedicatoria***

A mis padres les dedico este trabajo y todo lo que pueda llegar a ser en la vida...

Resumen

Las Tecnologías de la Información y las Comunicaciones (TIC) han ido revolucionando con el paso de los años, lo cual ha traído consigo que las instituciones bancarias pongan en práctica los avances en el campo de las tecnologías en función de la automatización de sus procesos.

El Banco Nacional de Cuba, como sector clave de la economía cubana, en conjunto con la Universidad de las Ciencias Informáticas busca la obtención del Sistema Automatizado de Gestión Bancaria (SAGEB), utilizando el núcleo del Sistema Automatizado para la Banca Internacional de Comercio (SABIC).

Entre los procesos de contabilidad en el Banco Nacional de Cuba encontramos la gestión de los bancos y de las cuentas, que en la actualidad no cumplen con los objetivos esperados por la entidad y en su automatización no contemplan operaciones de suma importancia para una gestión efectiva de dichos procesos.

La presente investigación se basa en el diseño e implementación de los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos del sistema Quarxo. Se realizó por tanto un diseño basado en patrones que diera respuesta a los requisitos ya definidos, para así satisfacer las necesidades del cliente. Se utilizaron herramientas y tecnologías de la plataforma Java para la implementación, por las ventajas de que son software libre y tienen un excelente potencial para el desarrollo de aplicaciones web.

Al concluir el trabajo se comprobó el correcto funcionamiento de los módulos, realizando pruebas unitarias a cada módulo por separado.

Palabras Claves

Cuenta banco, Cuenta de otros conceptos, Banco, Quarxo, Diseño, Implementación, Prueba.

ÍNDICE

INTRODUCCIÓN	1
CAPÍTULO I. FUNDAMENTACIÓN TEÓRICA	5
1.1 INTRODUCCIÓN.....	5
1.2 CONCEPTOS BÁSICOS RELACIONADOS CON EL DOMINIO DEL PROBLEMA.....	5
1.2.1 Contabilidad.....	5
1.2.2 Banco.....	6
1.2.3 Cuenta.....	8
1.3 SISTEMAS INFORMÁTICOS CONTABLES.....	11
1.3.1 Informatización de la gestión de cuentas y de los bancos.....	11
1.4 METODOLOGÍAS DE DESARROLLO.....	15
1.4.1 Lenguaje y herramienta para el modelado	16
1.5 PATRONES DE DISEÑO	17
1.5.1 Patrón de Acceso a Datos (DAO).....	18
1.5.2 Patrones de Asignación de Responsabilidades (GRAPS)	18
1.5.3 Patrones GoF (Gang of Four)	19
1.6 AMBIENTE DE DESARROLLO.....	19
1.6.1 Lenguajes de programación	19
1.6.2 Frameworks	20
1.6.3 Herramientas de desarrollo	22
1.7 CONCLUSIONES DEL CAPÍTULO.....	23
CAPÍTULO 2- ARQUITECTURA Y DISEÑO DE LOS MÓDULOS GESTIONAR BANCO, CUENTA BANCO Y CUENTA DE OTROS CONCEPTOS	25
2.1 INTRODUCCIÓN.....	25
2.2 DESCRIPCIÓN DE LA ARQUITECTURA.....	25
2.2.1 Estructura del sistema	25
2.2.2 Capa de presentación	26
2.2.3 Capa de Negocio	26
2.2.4 Capa de acceso a datos.....	27
2.3 DISEÑO DE LOS MÓDULOS.....	27
2.3.1 Descripción de los módulos	28
2.3.2 Modelo de diseño.....	29
2.3.3 Modelo de Datos	37
2.3.4 Patrones de diseño empleados	38
2.4 CONCLUSIONES DEL CAPÍTULO.....	39
CAPÍTULO 3- IMPLEMENTACIÓN Y PRUEBA.....	40
3.1 INTRODUCCIÓN.....	40

3.2 MODELO DE IMPLEMENTACIÓN.....	40
3.2.1 Diagrama de Componentes.....	40
3.3 ESTÁNDARES DE CODIFICACIÓN.....	42
3.3.1 Convención de código general.....	42
3.3.2 Capa de presentación.....	43
3.3.3 Capa de Negocio.....	43
3.3.4 Capa de Acceso a Dato.....	44
3.4 ASPECTOS PRINCIPALES DE LA IMPLEMENTACIÓN.....	44
3.4.1 Utilización de Spring WebFlow Framework.....	44
3.5 DESCRIPCIÓN DE LAS CLASES Y FUNCIONALIDADES DE LOS MÓDULOS.....	50
3.6 REALIZACIÓN DE PRUEBA.....	53
3.6.1 Pruebas unitarias.....	53
3.7 CONCLUSIONES DEL CAPÍTULO.....	61
CONCLUSIONES GENERALES.....	62
RECOMENDACIONES.....	63
BIBLIOGRAFÍA.....	64
ANEXOS.....	67
GLOSARIO DE TÉRMINOS.....	69

ÍNDICE DE ILUSTRACIONES

ILUSTRACIÓN 1 ARQUITECTURA DE UN MÓDULO DE QUARXO.....	26
ILUSTRACIÓN 2 MODELO DE PAQUETES DEL MÓDULO GESTIONAR BANCO.....	30
ILUSTRACIÓN 3 DISEÑO DE LA CAPA DE PRESENTACIÓN DEL MÓDULO GESTIONAR BANCO.....	32
ILUSTRACIÓN 4 DISEÑO DE LA CAPA DE NEGOCIO DEL MÓDULO GESTIONAR BANCO.....	33
ILUSTRACIÓN 5 DISEÑO DE LA CAPA DE ACCESO A DATOS DEL MÓDULO GESTIONAR BANCO.....	34
ILUSTRACIÓN 6 MODELO DOMINIO DEL MÓDULO GESTIONAR BANCO.....	35
ILUSTRACIÓN 7 DIAGRAMA DE SECUENCIA: REGISTRAR BANCO (CARGAR DATOS).....	36
ILUSTRACIÓN 8 DIAGRAMA DE SECUENCIA: REGISTRAR BANCO (PERSISTIR).....	36
ILUSTRACIÓN 9 MODELO DE DATOS DEL MÓDULO GESTIONAR BANCO.....	37
ILUSTRACIÓN 10 DIAGRAMA DE COMPONENTES DE LOS MÓDULOS.....	41
ILUSTRACIÓN 11 CÓDIGO DEL MÉTODO OBTENER().....	55
ILUSTRACIÓN 12 GRAFO DE FLUJO ASOCIADO AL ALGORITMO OBTENER.....	56
ILUSTRACIÓN 13 DISEÑO DE LA CAPA DE PRESENTACIÓN DEL MÓDULO CUENTA BANCO.....	67
ILUSTRACIÓN 14 DISEÑO DE LA CAPA DE NEGOCIO DEL MÓDULO CUENTA BANCO.....	67
ILUSTRACIÓN 15 DISEÑO DE LA CAPA DE ACCESO A DATOS DEL MÓDULO CUENTA BANCO.....	68
ILUSTRACIÓN 16 MODELO DE DATOS DEL MÓDULO CUENTA BANCO.....	68

ÍNDICE DE TABLAS

TABLA 1 CLASE CUENTAAMPLIADAMULTIACION DEL MÓDULO CUENTA BANCO.....	51
TABLA 2 CLASE CUANTAAMPLIADA UTILIZADA POR LOS MÓDULOS CUENTA BANCO Y CUENTA DE OTROS CONCEPTOS.....	52
TABLA 3 CASO DE PRUEBA PARA EL ALGORITMO OBTENER: CAMINO BÁSICO #1.	57
TABLA 4 CASO DE PRUEBA PARA EL ALGORITMO OBTENER: CAMINO BÁSICO #2	58
TABLA 5 CASO DE PRUEBA PARA EL ALGORITMO OBTENER: CAMINO BÁSICO #3	58
TABLA 6 CASO DE PRUEBA PARA EL ALGORITMO OBTENER: CAMINO BÁSICO #4	58
TABLA 7 CASO DE PRUEBA PARA EL ALGORITMO OBTENER: CAMINO BÁSICO #5	59
TABLA 8 CASO DE PRUEBA PARA EL ALGORITMO OBTENER: CAMINO BÁSICO #6	59

Introducción

A través de los años los bancos por el papel que han jugado en el comercio mundial han sido considerados como uno de los sectores claves en la economía de un país, pues gran parte del ahorro, la inversión y el financiamiento en términos amplios se canaliza a través de ellos. Estos realizan diversas funciones, pero entre las principales podemos destacar: la intermediación¹ financiera, la producción de un conjunto complejo de servicios financieros, además de transmitir e implementar la política monetaria diseñada por el sector público.

En las instituciones bancarias se lleva el control de las actividades financieras mediante el uso de operaciones contables que permiten resumirlas de una forma útil para la toma de decisiones, esto se realiza gracias al uso de la contabilidad, la cual es considerada como *“la ciencia y/o técnica que enseña a clasificar y registrar todas las transacciones financieras de un negocio o empresa para proporcionar informes que sirven de base para la toma de decisiones sobre la actividad.”* [1]

Un elemento básico y central de la contabilidad lo representa la cuenta, la cual permite clasificar todas las transacciones comerciales que tienen una empresa o negocio. Representa bienes, derechos y obligaciones de un individuo o entidad en una fecha determinada.

Los bancos llevan un control de todas las cuentas que van creando, las cuales se desglosan en niveles o rubros² con el objetivo de organizar el trabajo, mejorar el desenvolvimiento de este al realizar su actividad contable y tener un control estricto de las diferentes transacciones que se realizan dentro de la institución. Entre los tipos de cuentas podemos encontrar las cuentas con otros bancos las cuales representan los saldos de un banco como cliente de otro banco, así como otras cuentas que son utilizadas para la contabilidad interna del banco.

Los bancos además mantienen relaciones financieras con otros bancos, con los cuales gestionan diferentes operaciones tales como: pagos, transferencias, negociaciones, etc. Por tal motivo, estos

¹ Intermediación financiera: Proceso mediante el cual una entidad, generalmente bancaria o financiera, traslada los recursos de los ahorrantes directamente a las empresas que requieren de financiamiento.

² Rubro: Título que agrupa a un conjunto de cuentas

gestionan la información de todos los bancos con los cuales tramitan operaciones financieras a través de acuerdos o contratos.

La creación de nuevos productos y servicios bancarios han hecho necesario contar con una infraestructura tecnológica que se adapte a la estrategia de negocios de la entidad, con el fin de agilizar la toma de decisiones, mantener un control estricto de los procesos y elevar la calidad del servicio ofrecido a los clientes.

El Banco Nacional de Cuba (BNC) desde hace ya varios años ha utilizado el Sistema Automatizado para la Banca Internacional de Comercio (SABIC), que se crea inicialmente para agilizar los procesos dentro de la institución.

El SABIC a pesar del papel tan fundamental que ha jugado en la automatización de los procesos del BNC³, se ha quedado atrás ante los nuevos cambios en el negocio y la tecnología, dificultando el trabajo de los funcionarios dentro de la entidad; dicho software se ejecuta sobre el sistema operativo MS-DOS que no es compatible con el Sistema de Comunicación (SISCOM) utilizado en la institución para el envío de mensajes con otros bancos del país y tiene como principal desventaja que es monotarea, lo cual entorpece el trabajo del que opera con dicha aplicación.

Las funcionalidades que ofrece el SABIC actualmente son insuficientes, de forma tal que en ocasiones impide registrar actividades contables importantes que se realizan en la institución, no ofrece una minuciosa validación de los datos de entrada, lo cual constituye una irregularidad que puede llegar a detectarse meses después, además de que las operaciones que se realizan no cumplan con todos los requisitos esperados, las operaciones de gestión de las cuentas y de los bancos son ejemplo de ello. Al gestionar una cuenta banco el control de la información referente a sus medios de comunicaciones así como de las personas autorizadas a operar sobre esas cuentas se realizan de forma engorrosa, el control se lleva en papel, así mismo pasa con la información de los medios de comunicaciones a la hora de realizar la gestión de los bancos.

³ BNC: Banco Nacional de Cuba

A raíz de esto y como parte de la modernización del Sistema Bancario Cubano, el BNC le solicitó a la Universidad de las Ciencias Informáticas (UCI) el desarrollo de un software que permita la gestión de todos los procesos dentro del banco, de forma tal que se resuelvan las dificultades descritas anteriormente.

Considerando lo analizado anteriormente, se define el siguiente **problema a resolver**:

¿Cómo lograr la gestión de las cuentas banco, cuenta de otros conceptos y de los bancos a partir de los módulos del sistema Quarxo?

A partir del problema planteado se define como **objeto de estudio** a los procesos de gestión de bancos y de las cuentas en las instituciones bancarias y el **campo de acción** que comprende la informatización de los procesos de cuentas de bancos, cuentas de otros conceptos y la gestión de los bancos en el Banco Nacional de Cuba.

Para darle solución al problema a resolver planteado se traza el siguiente **objetivo general**: Diseñar e implementar los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos para el sistema Quarxo.

Idea a Defender: La implementación de los módulos Cuenta banco, Cuenta de otros conceptos y Gestionar banco para el sistema Quarxo permitirá la gestión de las cuentas banco y cuenta de otros conceptos, así como la gestión de los bancos en el Banco Nacional de Cuba.

Tareas de la investigación

1. Caracterización de los procesos relacionados con la gestión de los bancos y de las cuentas en las entidades financieras.
2. Caracterización las tecnologías y herramientas a utilizar en el diseño e implementación de los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos del sistema Quarxo.
3. Realización del diseño de las clases de los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos del sistema Quarxo.
4. Implementación de las funcionalidades de los módulos Gestionar banco, Cuenta banco y Cuenta

de otros conceptos del sistema Quarxo.

5. Realización del modelo de componentes de los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos del sistema Quarxo.
6. Realización de pruebas unitarias a la implementación de los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos del sistema Quarxo.

Estructura del Documento:

Capítulo 1 Fundamentación teórica: Este capítulo comprende el estudio del estado del arte, brinda un breve acercamiento a los principales conceptos asociados al dominio del problema, además ubica al lector en el ambiente de desarrollo de los módulos justificándose las tecnologías, metodologías y herramientas que fueron utilizadas para el desarrollo de los procesos seleccionados.

Capítulo 2 Arquitectura y Diseño: En este capítulo se describe la arquitectura de los módulos, se explica el modelo del diseño, además del uso de los patrones utilizados para dar respuesta a la solución planteada. También se realiza la estructuración de los módulos donde se presentan los diagramas de clases del diseño, los de secuencia y el modelo de datos.

Capítulo 3 Implementación y Prueba: Este capítulo se enfoca en la construcción de la solución explicando los aspectos principales de la implementación. También se realiza una descripción de las clases y funcionalidades de los módulos Cuenta banco, Cuenta de otros conceptos y Gestionar banco, además de la aplicación de pruebas unitarias para la comprobación del buen funcionamiento de dichos módulos.

Capítulo I. FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se abordarán los aspectos fundamentales que servirán de soporte teórico para el desarrollo de toda la investigación. Además se analizarán diferentes sistemas existentes en el ámbito nacional e internacional que gestionen los procesos correspondientes a la investigación. Por último, se realizará una breve caracterización de las tecnologías, metodologías y herramientas utilizadas para la solución del problema planteado.

1.2 Conceptos básicos relacionados con el dominio del problema.

Para facilitar la comprensión de dicha investigación se deben entender primeramente los términos que se emplean en las instituciones bancarias, motivo por el cual se especifican algunos conceptos que no son comunes en el lenguaje cotidiano.

1.2.1 Contabilidad

Conceptos de diferentes autores:

“La contabilidad es un instrumento de comunicación de hechos económicos, financieros y sociales suscitados en una empresa, sujeto a medición, registración e interpretación para la toma de decisiones empresariales” [2]

“La Contabilidad es el arte de registrar, clasificar y resumir de manera significativa y en términos de dinero, transacciones y eventos que son en parte, por lo menos, de carácter financiero e interpretar los resultados de estos” [3]

Al analizar dichos conceptos se concluye que la contabilidad ofrece información de hechos económicos, financieros y sociales de una entidad a través de técnicas que se ocupan de registrar, clasificar y resumir las operaciones con el fin de interpretar sus resultados, de esta forma se considera una herramienta clave para la toma de decisión en materia de inversión.

La contabilidad según el tipo de actividades que se realicen, puede ser oficial o privada. La contabilidad

privada, se limita a registrar y analizar las operaciones económicas que puedan permitir las decisiones en campos financieros y económicos de las empresas, socios e individuos particulares; a su vez se clasifica en varios tipos, entre ellas la contabilidad bancaria aplicada a los bancos. [4]

1.2.1.1 Contabilidad bancaria

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

Ventajas de la contabilidad bancaria:

- ✓ Mantiene informado a la entidad de lo que debe y lo que le deben.
- ✓ Permite controlar los gastos y las inversiones.
- ✓ Ayuda a diferenciar los gastos de los propietarios con los de su negocio.
- ✓ Informa cuánto cuesta producir un artículo y en cuánto se puede vender.
- ✓ Permite conocer oportunamente cuánto se está ganando o perdiendo.
- ✓ Con una contabilidad organizada será más fácil conseguir préstamos y asesoría.
- ✓ Es orientadora, porque permite conocer en un momento dado, la situación financiera (Balance General) y situación económica (Estado de Resultados) del negocio.

1.2.2 Banco

Los bancos son entidades financieras o instituciones de tipo financiero que actúan de intermediarios entre los clientes y el dinero de forma legal, profesional y permanente. Se dedican a la administración de dinero que les deja en custodia sus clientes y hace uso de este para prestárselo a otros individuos o empresas, a los cuales se les aplica un interés, que es una de las variadas formas que tiene de hacer negocios e ir ampliando el dinero de sus arcas. Entre los servicios que brindan podemos encontrar los préstamos, créditos, cambios, etc.

Funciones del banco moderno:

- ✓ La intermediación de crédito.

- ✓ La intermediación de los pagos.
- ✓ La administración de los capitales.

1.2.2.1 Bancos Cubanos

En el proceso de transformaciones acontecidas en la economía cubana en las últimas décadas, un papel importante ha correspondido a la búsqueda de instrumentos idóneos que den respuesta a las condiciones del desarrollo económico que debía enfrentar el país y como uno de los pasos más relevantes encontramos la creación de diversas instituciones bancarias a lo largo de todo el país.

Actualmente la estructura del sistema bancario y financiero en Cuba está conformado por: 1 banco central, 8 bancos comerciales, 20 instituciones financieras no bancarias, 12 oficinas de representación de bancos extranjeros y 4 oficinas de representación de instituciones financieras no bancarias [5]. Entre los Bancos Comerciales de Cuba se encuentra el Banco Nacional, institución en la cual se enfocará el estudio de los procesos contables de las instituciones bancarias.

1.2.2.1.1 Banco Nacional de Cuba

El BNC como banco comercial, tiene entre sus operaciones bancarias autorizadas inherentes a la banca universal o de múltiples servicios:

- ✓ Obtener y otorgar créditos en moneda nacional y libremente convertible.
- ✓ Emitir garantías bancarias de todo tipo previa evaluación de la gestión económico-financiera de la entidad solicitante.
- ✓ Librar, aceptar, descontar, avalar y negociar, en cualquier forma, letras de cambio, pagarés, cheques y otros documentos mercantiles negociables denominados en moneda nacional y divisas.
- ✓ Fijar las tasas de interés que deberán aplicar a las operaciones que efectúe el Banco, dentro de los límites que establezca el BCC.
- ✓ Constituir entidades de seguro de crédito oficial a la exportación con arreglo a la legislación vigente en materia de seguros.
- ✓ Atender el registro, control y servicio de la deuda externa que el Estado cubano y el BNC contrajeron con acreedores foráneos hasta la fecha de entrada en vigor del Decreto Ley No. 172

de 1997, del Banco Central de Cuba.

Dentro del proceso contable en los bancos, incluyendo en el BNC se resalta la cuenta, elemento que permite tener un control estricto de las diferentes transacciones que se realizan dentro de la institución.

1.2.3 Cuenta

La cuenta es un instrumento de representación y medida de un elemento patrimonial que capta la situación inicial de éste y las variaciones que posteriormente se vayan produciendo en el mismo. [6] Representan las propiedades, derechos y deudas de una empresa en una fecha determinada; así mismo, los costos, los gastos y las utilidades en un período determinado.

Partes de una cuenta:

- **Titular:** Es el nombre de una cuenta, indica qué se registra en ella.
- **Debe:** Es la parte izquierda de la cuenta, donde se anota todo lo que el titular de la cuenta recibe, también se le dice debitar o adeudar a la que se registra por el debe.
- **Haber:** Es la parte derecha de la cuenta, donde se anota todo lo que el titular de la cuenta entrega; se dice abonar en la cuenta cuando se registra en el haber, también se puede decir acreditar.
- **Saldo:** Es la diferencia entre los movimientos deudor y acreedor. Existen dos clases de saldos, saldo deudor y saldo acreedor.

Las cuentas se pueden agrupar según su tipo, de manera que cada entidad crea su propia manera de agruparlas. El Banco Nacional de Cuba posee un clasificador de cuenta que les permite organizar el trabajo dentro de la entidad. En dicho clasificador se puede precisar el número de la cuenta, la subcuenta, así como una breve descripción de la misma, posibilitando que en la ejecución de los procesos bancarios se utilicen ágilmente.

1.2.3.1 Cuenta bancaria

La cuenta bancaria es como un acuerdo establecido entre una persona o cliente que coloca dinero en una entidad bancaria pudiendo rehacerse del monto mencionado en cualquier circunstancia, y al mismo tiempo permite que el banco seleccionado realice todas los procedimientos necesarios para su funcionamiento

como tal.

En el mundo existen varios tipos de cuentas, que cada país utiliza de acuerdo a las características y servicios que presta su sistema bancario. En los bancos cubanos las mismas se clasifican en:

- **Cuentas reales:** Este grupo está representado por los bienes, derechos y obligaciones de la empresa, es decir, lo integran el activo, pasivo y el capital, estas se denominan cuentas reales porque el saldo de estas cuentas representan lo que tiene una empresa en un momento dado.
- **Cuentas nominales:** Son cuentas temporales, estas duran abiertas lo que dura el ejercicio contable de la empresa, y al finalizar este, son cerradas y su resultado es traspasado a la cuenta capital quien es en definitiva la cuenta que va a ser afectada.
- **Cuentas mixtas:** Son cuentas cuyo saldo en una fecha determinada está formado por una parte real y otra parte nominal, sin embargo al cierre económico todo su saldo debe ser de naturaleza real.
- **Cuentas de orden:** Son cuentas que controlan ciertas operaciones o transacciones que no afectan al activo, el pasivo, el patrimonio o las operaciones del período, pero de una u otra forma, las transacciones que las generaron implican alguna responsabilidad para la empresa.
- **Caja:** Cuenta real de activo circulante. Refleja el dinero disponible en la empresa en un momento determinado.
- **Fondos fijos:** Cuenta real de activo circulante, llamada comúnmente Caja Chica, es una cantidad de dinero en curso legal determinado por la empresa para atender pagos menores de carácter general o de carácter previamente determinado.
- **Bancos:** Cuenta real de activo circulante. Comprende el efectivo que la empresa tiene depositado en instituciones bancarias o de crédito, siempre y cuando el mismo esté disponible. Generalmente este monto está representado por cuentas corrientes.
- **Plazos fijos:** Cuenta real de activo circulante. Período concedido por el pago de una deuda o para la presentación al pago de determinados documentos financieros.

La cuenta en el BNC está compuesta por los siguientes datos:

- Sigla de la moneda, se codifica utilizando 3 caracteres, recomendándose utilizar las siglas

definidas por el ISO.

- Código de la cuenta/subcuenta, se codifica utilizando 2 dígitos para la cuenta y 2 para la subcuenta, y tiene por objetivo permitir la clasificación contable de las operaciones que realiza un banco o institución financiera.
- Tipo de contraparte, se codifica utilizando un dígito, y permite definir la clase de contraparte (cliente, banco, etc.).
- Código de contraparte, se codifica utilizando 4 dígitos, y contiene en dependencia del tipo de contraparte un código de cliente, un código de banco, un código de concepto de ingreso/egreso, etc.
- Desglose de la cuenta, se codifica utilizando 2 dígitos, y se utiliza para aquellos casos en que sea necesario tener más de una cuenta con iguales sigla de moneda, código de cuenta/subcuenta, tipo de contraparte y código de contraparte.

Las cuentas en el BNC se agrupan en dependencia de la contrapartida o tipo de contraparte asociada a cada una, permitiendo de esta forma organizar el manejo de estas a la hora de realizar cualquier operación contable. Las mismas se agrupan en 4 grupos:

1: Cliente

2: Banco

3: Concepto

4: Otros Conceptos

Cuenta Banco

El BNC posee cuentas en otros bancos, localizados dentro o fuera del país, a dicha cuenta se le denomina internamente en el banco “Cuenta banco”, esta representa el efectivo que el banco tiene depositado en otras instituciones bancarias o de crédito como cliente de estos.

Al crearse dicha cuenta queda plasmado como se realizará el manejo de las cuentas, todas las instrucciones serán hechas por el cliente o personas autorizadas para girar contra los fondos disponibles en ella y dar instrucciones al banco, por medios de comunicación aceptables al banco de acuerdo con sus prácticas normales de operación.

Cuenta de otros conceptos

En los bancos se hace necesario el uso de la contabilidad interna para el cálculo de los costos y movimientos económicos y productivos dentro de estos, así como para la toma de decisiones en cuanto a producción y organización de la entidad. Para poder llevar un control sobre esa contabilidad interna, el BNC hace uso de cuentas a las cuales les denomina “Cuenta de otros conceptos”. Estas permiten controlar la contabilidad de los activos fijos tangibles, de los insumos, entre otros.

1.3 Sistemas informáticos contables

Son programas de contabilidad o paquetes contables, que tienen como función sistematizar y simplificar las tareas de contabilidad, además de registrar y procesar las transacciones históricas que se generan en una empresa, para ello solo hay que ingresar la información requerida como las pólizas contables, ingresos y egresos y hacer que el programa realice los cálculos necesarios.

Los sistemas informáticos contables son indispensables hoy en día para los bancos debido al gran volumen de información y de clientes con que operan, que manualmente sería casi imposible, además son imprescindibles para la toma de decisiones el conocer los estados financieros en la empresa.

1.3.1 Informatización de la gestión de cuentas y de los bancos.

Durante la investigación realizada sobre los sistemas informáticos contables existentes en la actualidad, se encontraron diferentes aplicaciones que cuentan con los procesos de gestión de cuentas y de bancos. Con el objetivo de enriquecer los conocimientos sobre las diferentes funcionalidades que debe ofrecer un sistema contable, a continuación se realiza una caracterización de los mismos.

En el mundo

SAP (Systems Applications Products in Data Processing): Es un ERP⁴ empresarial. Es un sistema informático basado en módulos integrados, que abarca prácticamente todos los aspectos de la

⁴ ERP (Enterprise Resource Planning): Sistemas de Planeación de Recursos.

administración empresarial. Entre sus soluciones se encuentra el **SAP para bancos**, comprende la solución más amplia existente para la industria bancaria, entre sus funcionalidades se encuentran la información de clientes, la gestión de cuentas y contratos, así como el establecimiento de fechas valores. [7]

SAP para bancos posee un módulo llamado Gestión de cuentas el cual maneja intereses y cargos de las cuentas, y permite definir la fecha efectiva de todas las transacciones, así como otras condiciones. Maneja todos los procesos del back-office para cuentas de cheques y ahorros, depósitos a plazo y esquemas de inversión. Estos procesos incluyen la administración y el monitoreo de la cuenta, transacciones de pago y la generación de los estados de cuenta.

Entre sus características principales se encuentran:

- ✓ Implementado en .NET y WebSphere.
- ✓ SAP también ofrece una nueva plataforma tecnológica denominada SAP NetWeaver, esta plataforma tecnológica convierte a SAP en un programa Web-enabled, lo que significa que estaría totalmente preparado para trabajar con él mediante la web.
- ✓ Trabaja sobre el sistema operativo Windows.
- ✓ Soporte para bases de datos Oracle.

Aspel-BANCO: Es el Sistema de Control Bancario que controla eficientemente los ingresos y egresos de cualquier tipo de cuenta bancaria, ofreciendo información financiera precisa en todo momento. Permite manejar movimientos y saldos en moneda nacional y extranjera, la programación de movimientos periódicos, el control de inversiones en plazo fijo y en acciones, así como la conciliación electrónica con las principales instituciones financieras. [8]

El Aspel-BANCO permite:

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

Dentro de este sistema el módulo Cuentas bancarias gestiona diferentes tipos de cuentas tales como:

cheques, inversiones, caja y crédito, entre otros. Además permite que en cualquier momento se pueda consultar el saldo actualizado de cada una de las cuentas bancarias, desglosando los montos en tránsito y en inversiones.

Sistema Bancario Financiero Byte: Dicho software es una solución para la industria bancaria y financiera, conformado por un conjunto de módulos independientes e integrables, que permiten la automatización de todos los departamentos, según sean las necesidades y posibilidades de inversión. Además contempla todas las operaciones que una institución financiera realiza, tanto administrativas como operativas, y esto lo hace con sus más de 80 módulos. [9]

Ventajas del Sistema Bancario Financiero Byte:

- ✓ Sistema altamente paramétrico, que permite a la institución reaccionar rápidamente ante los cambios y nuevas necesidades.
- ✓ Estándares de diseño en todos sus módulos, tanto en operación como en presentación, lo que permite fácil adaptación a los usuarios de los diferentes módulos.
- ✓ Ambiente gráfico que permite que el proceso de capacitación de los usuarios sea más rápido.
- ✓ Integración de las transacciones operativas al sistema contable, por medio de interfaces amigables y flexibles.
- ✓ Manuales en línea al alcance de todos los usuarios, que brinda una ayuda rápida y eficiente, reduciendo el requerimiento de personal especializado para soporte a usuarios.

El Sistema Bancario Financiero BYTE, está compuesto por diferentes módulos que en su conjunto conforman una solución integral. Uno de ellos se encarga de realizar la gestión de cuentas, dicho módulo se llama **Captaciones** y el mismo gestiona las: Cuenta Corriente, Cuentas de Traslado Automático, Cuentas Over Night, entre otras cuentas.

Características:

- ✓ Aplicación web.
- ✓ Es una aplicación multicapas.
- ✓ Presenta soporte para base de datos: Oracle, DB2, MS SQL Server, etc.
- ✓ Implementado completamente en el lenguaje Java.

En Cuba

SABIC (Sistema automatizado para la Banca Internacional de Comercio): Es un sistema diseñado y desarrollado para satisfacer las necesidades de procesamiento de datos de bancos e instituciones no bancarias, utilizando los medios técnicos de computación disponibles en el mercado.

Este sistema ha sido adaptado a los requerimientos de las operaciones propias del Banco Central y ha sido desarrollado para que los empleados que hagan uso de él puedan tramitar sus operaciones y realizar sus consultas sin necesidad de acudir a los archivos ni a la actividad manual. De esta forma se aumenta la seguridad, la eficiencia del trabajo y la productividad de los trabajadores.

Características principales del SABIC:

- ✓ Contabilización en tiempo real (que permite mantener actualizados los ficheros contables)
- ✓ Contabilización multimoneda (que permite registrar los activos y pasivos en las monedas orígenes sin tener que realizar en el momento del registro las conversiones de monedas, lo cual aumenta la exactitud de la información sobre la posición financiera de la institución). [10]

El SABIC ofrece la posibilidad de realizar la gestión de las cuentas y de los bancos. La gestión de las cuentas incluye: registros, modificaciones y consultas de las mismas, lo mismo incluye la gestión de los bancos.

Características del software:

- ✓ Utiliza MS-DOS como sistema de explotación.
- ✓ Desarrollado en lenguaje de programación FoxPro.
- ✓ Utiliza el gestor de base de datos SQL Server 2005.

A pesar de los beneficios aportados por el SABIC al sistema bancario cubano, el mismo presenta un conjunto de dificultades que afectan a los procesos que se realizan en el BNC. El mismo no ofrece una minuciosa validación de los datos de entrada, el control de muchas de las informaciones necesarias para realizar la gestión de las operaciones dentro de la entidad se llevan a cabo a través del uso del papel, como por ejemplo el control de los medios de comunicación y de las personas autorizadas, información necesaria que se utiliza al gestionar las cuentas y los bancos, entre otras dificultades que hacen engorroso el trabajo dentro de la entidad.

Valoración crítica

Luego de realizado el estudio y análisis de los sistemas contables existentes que incluyen de una forma u otra la gestión de las cuentas y de los bancos entre sus procesos, se arriba a la conclusión de que no existe un sistema altamente configurable que integre toda la información necesaria de las cuentas y de los bancos y se adapte a las necesidades propias del BNC.

En el actual contexto existen sistemas extranjeros como es el caso de SAP para bancos, Aspel-BANCO y el Sistema Bancario Financiero Byte, que comprenden los procesos de gestión de las cuentas entre sus módulos, pero dichos sistemas requieren de mantenimiento y actualización que implica grandes inversiones para el país. En el caso de Aspel-BANCO solo puede ser usado en empresas pequeñas que solo atiendan a lo sumo mil clientes, además que el mismo se ejecuta únicamente sobre Windows. En el caso de SAP para bancos, se necesita una persona al 100% con el conocimiento de todo el negocio, para que pueda hacerse cargo de administrar el SAP y dar solución a los problemas de forma rápida, requiere además de mantenimiento, actualización y fuertes inversiones en módulos complementarios lo cual lo hace muy costoso. El Sistema Bancario Financiero Byte tiene como característica que es privativo, por lo que se debe pagar altos precios para el mantenimiento del mismo. Otro hecho muy importante es que el sistema bancario financiero cubano tiene como característica el tener dos monedas base, algo que lo hace único ante los otros sistemas a nivel internacional, por lo tanto, los sistemas estudiados no cumplen este requisito que es fundamental para el trabajo en el BNC.

En el contexto nacional se encuentra el SABIC, que aunque comprenda los procesos de gestión de las cuentas y de los bancos, los mismos no cubren en su totalidad la información necesaria para una gestión completa de las cuentas y de los bancos, ya que la aplicación no permite gestionar la información de los medios de comunicación y de las personas autorizadas, información necesaria para la gestión de las cuentas y de los bancos.

En resumen, la complejidad que caracteriza al sistema bancario ha hecho de la globalización una realidad que desborda las fronteras de muchos sistemas nacionales y extranjeros. De ahí la importancia de desarrollar un sistema informático que contenga los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos, de forma tal que garanticen la gestión integral de los procesos en el BNC.

1.4 Metodologías de Desarrollo

Las metodologías de desarrollo de software especifican cómo se debe dividir un proyecto en etapas, qué

tareas se llevan a cabo en cada etapa, qué salidas se producen y cuándo se deben producir. Además, las metodologías especifican qué restricciones se aplican, qué herramientas se van a usar y cómo se gestionará y controlará un proyecto. Constituyen un conjunto de actividades necesarias para transformar los requisitos de los usuarios en un sistema software.

Las metodologías de desarrollos de software se dividen en dos grandes grupos: metodologías tradicionales (no ágiles), las cuales se centran en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, las herramientas y notaciones que se usarán; y las metodologías ágiles, las cuales son efectivas en proyectos con requisitos muy cambiantes y en donde se exige reducir drásticamente los tiempos de desarrollo manteniendo una alta calidad de los mismos. Algunas de las metodologías tradicionales (no ágiles) más empleadas en el mundo de la producción de software son: Rational Unified Process (RUP) y Microsoft Solution Framework (MSF). Por su parte las metodologías ágiles más utilizadas son: eXtreme Programming (XP), Scrum, Adaptive Software Development (ASD) y Familia de Metodologías Crystal entre otras.

Para el desarrollo de grandes proyectos que requieran mayor énfasis, planificación y control del proyecto, en especificación precisa de requisitos y modelado es recomendable utilizar las metodologías tradicionales, ya que las ágiles se dirigen a equipos de desarrollo pequeños con ciclos muy cortos de desarrollo. Entre las metodologías tradicionales encontramos a RUP, el cual es una de las metodologías más utilizadas actualmente en el mundo para el desarrollo de software, la misma es un proceso para el desarrollo de un proyecto de software que define claramente quien, cómo, cuándo y qué debe hacerse en el proyecto, con 3 características esenciales: centrado en la arquitectura, iterativo e incremental y dirigido por casos de uso. RUP toma en cuenta las mejores prácticas en el modelo de desarrollo de software. En particular, desarrollo de software en forma iterativa, manejo de requerimientos, utiliza arquitectura basada en componentes, modela el software visualmente (modela con UML), verifica la calidad del software y controla los cambios.

1.4.1 Lenguaje y herramienta para el modelado

El lenguaje propuesto por la metodología de desarrollo RUP para modelar elementos de software es UML (*Unified Modeling Language*), el cual es un sistema de notación que se ha convertido en estándar en el mundo del desarrollo de sistemas. El mismo es un lenguaje de modelado visual que se usa para

especificar, visualizar, construir y documentar artefactos de un sistema de software. Pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. Incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas.

Existen varios software para el modelado UML, son las denominadas herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador). La herramienta CASE es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo, su objetivo es acelerar el proceso para el que han sido diseñadas, automatizar o apoyar una o más fases del ciclo de vida del desarrollo de software. La principal ventaja de la utilización de estas herramientas es la mejora de la calidad de los desarrollos realizados y el aumento de la productividad. Entre las herramientas CASE para el modelado encontramos al Visual Paradigm, la cual es una eficaz herramienta para visualizar y diseñar elementos de software, para ello utiliza UML y ofrece una gama de facilidades para el modelado de aplicaciones. Está orientada a la creación de diseños usando el paradigma de programación orientada a objetos. Además ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite modelar diagramas de clases, código inverso y generar documentación. Provee soporte para la generación de código, tiene integración con diversos IDE's como NetBeans (de Sun Microsystems), JDeveloper (de Oracle), Eclipse (de IBM), JBuilder (de Borland), así como la posibilidad de realizarse la ingeniería inversa para aplicaciones realizadas en JAVA, .NET, XML e Hibernate.

1.5 Patrones de diseño

En la actualidad es muy común el uso de patrones en el desarrollo del software, los mismos proveen a los equipos de desarrollos de un mecanismo que les permitirá diseñar aplicaciones de alta calidad en el menor tiempo posible, debido a que constituyen soluciones a problemas específicos del diseño, promueven la reutilización de código y la agilización del proceso de desarrollo de software.

Un patrón de diseño es una solución a un problema de diseño. Se consideran como procedimientos para solucionar diversos problemas del mismo tipo y son la base para la búsqueda de soluciones a dificultades comunes en el desarrollo de software. A continuación se especifican algunos patrones de diseño:

1.5.1 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. El DAO 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. El DAO oculta completamente los detalles de implementación de la fuente de datos a sus clientes. Como la interfaz expuesta por el DAO no cambia cuando cambia la implementación de la fuente de datos subyacente, este patrón permite al DAO adaptarse a diferentes esquemas de almacenamiento sin que esto afecte a sus clientes o componentes de negocio. Esencialmente, el DAO actúa como un adaptador entre el componente y la fuente de datos. [11]

1.5.2 Patrones de Asignación de Responsabilidades (GRAPS)

Los patrones *GRASP* describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Los mismos constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable.

- ✓ **Patrón experto:** El uso de este patrón permite a los objetos utilizar su propia información para llevar a cabo las tareas, favorece la existencia de mínimas relaciones entre las clases, lo que permite contar con un sistema sólido y fácil de mantener.
- ✓ **Patrón creador:** El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, de forma tal que una instancia de un objeto sólo pueda ser creada por el objeto que contiene la información necesaria para ello. El uso de este patrón admite crear las dependencias mínimas necesarias entre las clases, lo que beneficia el mantenimiento del sistema y se brindan oportunidades de reutilización.
- ✓ **Patrón controlador:** Se considera para efectuar las asignaciones en cuanto al manejo de los eventos del sistema y definir sus operaciones. Es un intermediario entre la interfaz de usuario y el núcleo de las clases donde reside la lógica de la aplicación. El controlador no realiza mucho trabajo por sí mismo; más bien coordina la actividad de otros objetos.

- ✓ **Patrón de diseño Bajo Acoplamiento:** Dicho patrón soporta el diseño de clases que son más independientes, lo que reduce el impacto del cambio, es decir, que en caso de producirse una modificación en alguna de las clases, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización y disminuyendo la dependencia entre las clases.
- ✓ **Patrón de diseño Alta Cohesión:** La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a una clase que tiene una responsabilidad moderada en un área funcional y colabora con otras clases para llevar a cabo las tareas. Una clase con alta cohesión es ventajosa porque es relativamente fácil de mantener, entender y reutilizar.

1.5.3 Patrones GoF (Gang of Four)

- ✓ **Facade (Fachada):** Este patrón sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Utilizado para reducir la dependencia entre clases, ya que ofrece un punto de acceso al resto de las clases. Si estas cambian o se sustituyen por otras solo hay que actualizar la clase Fachada sin que el cambio afecte a las aplicaciones clientes. El patrón Fachada es sencillo y se utiliza ampliamente.
- ✓ **Singleton:** este patrón garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. El acceso a la Base de Datos en el sistema se realiza mediante una única instancia para evitar que se creen múltiples conexiones innecesarias.

1.6 Ambiente de desarrollo

1.6.1 Lenguajes de programación

Hoy en día la red de redes se ha convertido en el canal de comunicación más usado del mundo, por las grandes ventajas y potencialidades que brindan todos los sistemas que soporta, permitiendo la interacción con los usuarios y la personalización. Esto es posible por un conjunto de lenguajes de programación que le dan gran interactividad a las aplicaciones Web, tanto del lado del cliente como del lado del servidor.

Entre los **lenguajes de programación del lado del servidor** más usados en el desarrollo web encontramos PHP, JAVA y C#. PHP es un lenguaje gratuito e independiente de plataforma, rápido, con una gran librería de funciones y mucha documentación. En cuanto a C# es un lenguaje orientado a objetos, desarrollado y estandarizado por Microsoft como parte de la plataforma .NET; y el lenguaje JAVA es fácil de usar, multiplataforma, sumamente flexible, permite la creación de programas modulares y de códigos reutilizables. Dicho lenguaje además es compilado, utiliza el paradigma orientado a objeto y en la actualidad es muy utilizado, ya que permite el desarrollo de programas seguros y de alto rendimiento. Es uno de los lenguajes más utilizados en la actualidad para el desarrollo de proyectos de software bajo la especificación de Java Enterprise Edition (JEE), el cual es una tecnología que apunta a simplificar el diseño y desarrollo de aplicaciones empresariales robustas, escalables y seguras utilizando Java como lenguaje. Simplifica las aplicaciones empresariales basándolas en componentes modulares y estandarizados, ofreciendo un completo conjunto de servicios a estos componentes, y manejando muchos de las funciones de la aplicación de forma automática, sin necesidad de una programación compleja.

En cuanto a los **lenguajes de programación del lado del cliente** encontramos al JavaScript el cual es compatible con la mayoría de los navegadores modernos, permite crear efectos especiales en las páginas web y definir interactividades con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones JavaScript y ejecutarlas. Es un lenguaje bastante sencillo, rápido y fácil de aprender por personas de poca experiencia.

1.6.2 Frameworks

En el desarrollo de software, un framework es una estructura conceptual y tecnológica de soporte definido, normalmente, con artefactos o módulos de software concretos, en base a la cual otro proyecto de software puede ser organizado y desarrollado. Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

Para los lenguajes anteriormente mencionados se han creado diversos frameworks que facilitan el desarrollo de aplicaciones empresariales. Entre los frameworks que se utilizan para el trabajo con el acceso a datos de una aplicación se encuentran los frameworks ORM, entre los que se destacan

IBATIS, Hibernate, JDO, entre otros. Para el trabajo con la presentación, existen diversos frameworks que simplifican el desarrollo de interfaces de usuario en aplicaciones JEE, entre las que se destacan, Ext JS, Dojo Toolkit y Prototype. Además existen un conjunto de frameworks que posibilitan la aplicación del patrón MVC tales como Struts, JBoss Seam y Spring, es importante señalar que los dos últimos son utilizados en diversos proyectos productivos en la universidad. Debido al conjunto de facilidades que brinda el uso de frameworks en el desarrollo de aplicaciones, se decide hacer un estudio de alguno de los frameworks antes mencionados:

Spring Framework

Es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java. Por su diseño ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria. Las características fundamentales de este framework pueden emplearse en cualquier aplicación hecha en Java, existen muchas extensiones y mejoras para construir aplicaciones basadas en web por encima de la plataforma empresarial de Java (Java Enterprise Platform).

Spring WebFlow

Spring WebFlow es un framework y a su vez un subproyecto de Spring que se utiliza para definir los flujos de interfaces de usuario de aplicaciones web. Permite la definición de los flujos tanto en formato de clases Java, como utilizando un archivo de configuración XML. Se centra en proporcionar la infraestructura para construir y ejecutar aplicaciones web enriquecidas.

Además permite a los arquitectos de aplicaciones Web centrarse en las preocupaciones más abstractas de desarrollo de aplicaciones Web: la estructura lógica de los flujos de la Web y la definición de las condiciones de eventos Web.

Hibernate

Es una herramienta de código abierto dedicada al mapeo objeto-relacional para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.

Cuenta con el Hibernate Query Language (HQL), que es un lenguaje de consulta similar al SQL. Además brinda, de una manera muy rápida y mejorada, la posibilidad de generar bases de datos en cualquiera de los entornos soportados: PostgreSQL, Oracle, DB2, MySql, entre otros.

Dojo Toolkit

Es un framework que contiene APIs y controles para ayudar al desarrollo de aplicaciones web, utiliza Ajax para hacer trabajos asíncronos en la página y se compone de varios módulos: Dijit, Dojo y Dojox.

Dojo además proporciona variadas opciones en una sola biblioteca haciendo un mejor trabajo que sustenta los nuevos y viejos Browsers, resolviendo los problemas de compatibilidad entre los navegadores. Tiene múltiples puntos de entrada, es independiente del intérprete y unifica estándares de codificación.

1.6.3 Herramientas de desarrollo

Eclipse

Esta herramienta es un Entorno Integrado de Desarrollo (IDE, por sus siglas en inglés) multiplataforma, de código abierto y extensible. Ofrece la posibilidad de añadirle módulos o pluggins, que facilita el uso de JBoss Tools, para el desarrollo de aplicaciones con RichFaces, Seam, Hibernate y el JBoss AS. Además, brinda cobertura a otros lenguajes de programación aparte de Java, como pueden ser C/C++, Python, Cobol, Fortran, PHP, entre otros.

Contenedor Web (Tomcat)

El Apache Tomcat es un software de código abierto implementado para las tecnologías Java Servlet y Java Server Pages. Apache Tomcat es desarrollado en un entorno abierto, participativo y publicado bajo la licencia del software de Apache.

El mismo se ejecuta en varios sistemas operativos. Es un servidor configurable de diseño modular, con diversidad que permiten garantizar una elevada seguridad y buenas prestaciones. Además, brinda soporte para la plataforma de desarrollo JEE. Existe bastante documentación asociada al mismo, cuenta con una gran comunidad de desarrollo y es uno de los más usados internacionalmente.

Control de Versiones (SubVersion)

Subversion (SVN) es un software libre desarrollado para el control de versiones. Es un sistema centralizado para compartir información que permite realizar modificaciones atómicas y gestionar archivos, directorios y sus cambios a través del tiempo, lo que facilita las tareas administrativas. Su capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración.

Base de Datos

SQL Server 2005 es una plataforma global de base de datos que ofrece administración de datos empresariales con herramientas integradas de inteligencia empresarial (BI). El motor de la base de datos SQL Server 2005 ofrece almacenamiento más seguro y confiable tanto para datos relacionales como estructurados, lo que le permite crear y administrar aplicaciones de datos altamente disponibles y con mayor rendimiento para utilizar en su negocio.

1.7 Conclusiones del capítulo

Luego de realizarse un estudio de los sistemas existentes en la actualidad tanto nacional como internacional, se concluye que se hace necesario el desarrollo de una aplicación informática que realice una gestión integral de los procesos de gestión de cuentas y de bancos, pues ninguno de esos sistemas integra toda la información necesaria para una gestión completa de las cuentas y de los bancos, además que los procesos que los mismos automatizan no se adaptan a las necesidades propias del BNC. Dichos sistemas además son propietarios, por lo que resulta muy poco factible para la economía de nuestro país, el costear alguno de ellos.

Se realizó además un profundo estudio de algunas de las principales metodologías, herramientas y tecnologías llegando a las siguientes conclusiones, las cuales coinciden con la máxima dirección del proyecto: Con respecto a la metodología de desarrollo del software, se usará **RUP**, pues es la más adecuada dada la inmensidad del sistema y la complejidad del mismo, con la utilización de dicha metodología no sólo estamos garantizando aplicar buenas prácticas recomendadas, probadas y una arquitectura configurable, sino que se tendrá unificado todo el equipo de desarrollo de software, optimizando su comunicación y garantizando entendimiento entre sus integrantes, la misma modela al

software visualmente utilizando **UML**, lenguaje de modelado más utilizado en la actualidad ya que permite toda la modelación de los diferentes artefactos generados en los distintos flujos de trabajo presente en el ciclo de vida del desarrollo exitoso de un software. Se decidió utilizar además al **Visual Paradigm** como herramienta de modelado, por ser una herramienta multiplataforma que garantiza la calidad del software en todo el ciclo de vida, ya que permite la comunicación entre los desarrolladores mediante un lenguaje común para todos los roles que intervienen en el proceso de desarrollo del software. Como lenguaje de programación se propone el uso de **JAVA** debido a su robustez, seguridad, portabilidad, dinamismo e independencia de la arquitectura, dicho lenguaje se utilizará bajo la especificación de **Java Enterprise Edition (JEE)**, el cual reduce significativamente el esfuerzo necesario por los desarrolladores, proveyendo una arquitectura robusta. Sobre esta plataforma se han desarrollado múltiples framework que facilitan la programación de aplicaciones, ya que encapsulan operaciones complejas en instrucciones sencillas, motivo por el cual se decidió utilizar como framework núcleo de la aplicación a **Spring** y para el trabajo con los datos persistentes en la aplicación se utilizará el framework **Hibernate**, los cuales son de código abierto al igual que **Dojo**, utilizado como framework **JavaScript** o de presentación para crear interfaces de manera fácil. Como herramientas de desarrollo se utilizará el **IDE Eclipse**, herramienta altamente provechosa que admite la implementación con los frameworks antes mencionados y otros abordados en el capítulo. Como gestor de base de datos se utilizará el **SQL Server 2005**, por decisión del cliente; como contenedor web al **Apache Tomcat 6.0**, el cual es un software de código abierto que brinda soporte para la plataforma de desarrollo JEE y como controlador de versiones se utilizará al **SubVersion**, que es software libre y facilita las tareas administrativas. En conclusión los módulos se desarrollarán con herramientas y tecnologías en su mayoría libres y multiplataforma, lo cual trae como consecuencia positiva una reducción notable del costo del sistema por concepto de licencias de software y soporte.

Capítulo 2- Arquitectura y Diseño de los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos

2.1 Introducción

El presente capítulo enmarca su contenido en explicar la arquitectura del sistema Quarxo y el diseño de los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos. Se presentan el modelo de clases del diseño, el modelo de datos, así como los diagramas de interacción de las funcionalidades más significativas.

2.2 Descripción de la arquitectura

“La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución”. [12]

La Arquitectura propuesta por el proyecto está compuesta por tres capas: la Capa de Presentación, Capa de Negocio y la Capa de Acceso a Datos, además se utiliza una capa transversal a las otras con las clases del dominio. Dicha arquitectura fue establecida con el objetivo de separar las responsabilidades en cada una de las capas y lograr una mayor reutilización, escalabilidad y facilidad para el desarrollo del sistema. Las capas están bien delimitadas una de la otra, una capa superior interactúa con la inferior mediante interfaces que definen las funcionalidades que la misma debe brindar.

2.2.1 Estructura del sistema

El sistema está estructurado a través de subsistemas, módulos y componentes:

Los Subsistemas, Módulos y Componentes son definidos según las funcionalidades identificadas en la Captura de Requisitos. Los Subsistemas agrupan un conjunto de Módulos relacionados con los procesos que ejecutan. Los Módulos agrupan un conjunto de Casos de Uso que representan uno o más procesos bancarios estrechamente relacionados y por último los Componentes son un conjunto de funcionalidades comunes que son reutilizados por otros módulos del sistema.

A continuación se muestra una representación de las capas lógicas:

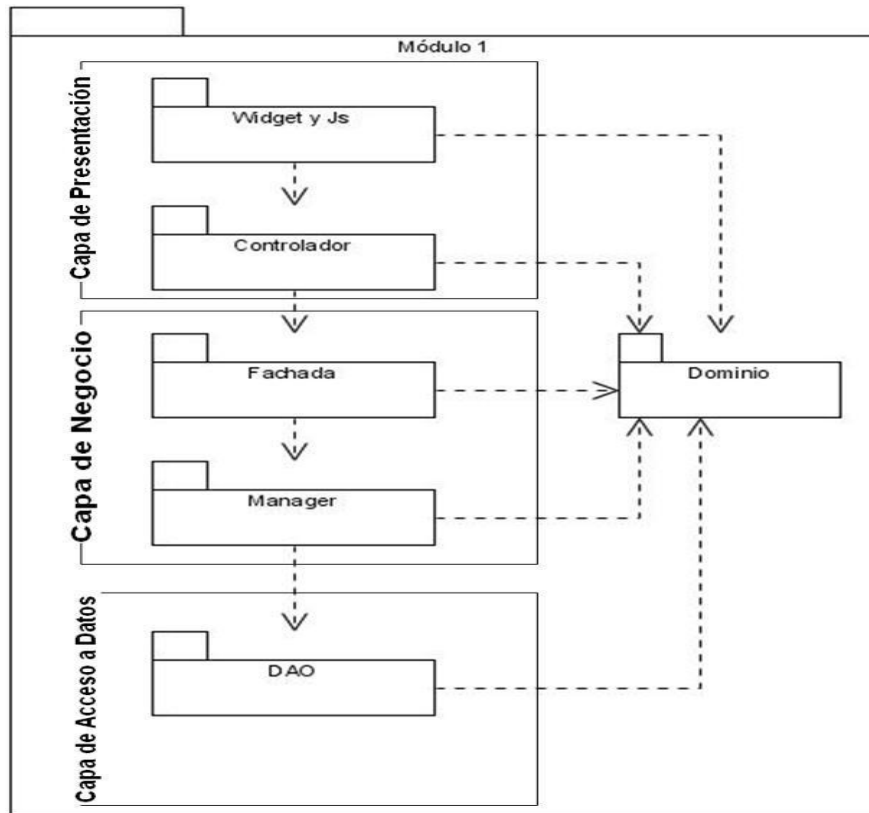


Ilustración 1 Arquitectura de un módulo de Quarxo

Para una fácil comprensión de las capas lógicas por las que está compuesto el sistema de acuerdo a la figura anterior, se realiza una breve descripción de cada una de ellas.

2.2.2 Capa de presentación

En esta capa se desarrollará la lógica de presentación. En el lado del servidor para recibir, controlar y enviar una respuesta a las peticiones realizadas desde el cliente, se utilizará Spring MVC. Para representar y controlar los flujos complejos y reutilizables de la aplicación se hará uso del Spring WebFlow. En el lado del cliente se utilizará la librería Dojo para generar las interfaces que interactuarán con el usuario. La Capa de presentación estará relacionada con la Capa de negocio y la Capa de dominio.

2.2.3 Capa de Negocio

Esta capa está dividida en dos subcapas:

- La subcapa Facade agrupará las funcionalidades según su naturaleza para que pueda ser invocada desde la capa de presentación, sin realizar la lógica de negocio, delegando de esta forma a la subcapa Manager la realización de la lógica del negocio. Esta será además, el punto de intercambio entre la Capa de presentación y la Capa de negocio.
- La subcapa Manager donde se realizará la lógica del negocio conteniendo la jerarquía de clases indicadas para su implementación. Además esta subcapa utilizará la Capa de acceso a datos para el trabajo con la persistencia de los datos y la Capa de dominio para la generación de los objetos del dominio.

2.2.4 Capa de acceso a datos

En esta capa se realizarán todas las operaciones relacionadas con el gestor de base de datos, permitiendo así la persistencia y el acceso a la información de la aplicación. La interacción con la Capa de Negocio se realizará a través de interfaces. Para el desarrollo de esta capa se utilizará el patrón DAO (Data Access Object en inglés), en español Objetos de Acceso a Datos.

Otra de las características de la arquitectura definida es la utilización del patrón Modelo Vista Controlador (MVC), separa la interfaz de usuario, la lógica de control y los datos de una aplicación, en tres componentes distintos: el Modelo administra el comportamiento y los datos del dominio de aplicación, la Vista maneja la visualización de la información y el Controlador interpreta las acciones del usuario sobre el sistema, informando al modelo y/o a la vista para que cambien según resulte apropiado. Esto ofrece varias ventajas, como la reusabilidad de componentes, además permite que modificaciones en las vistas afecten lo menos posible en la lógica de negocio o de datos.

2.3 Diseño de los módulos

Durante el desarrollo de un software el diseño se realiza con el fin de traducir los requisitos a una estructura que describe cómo implementar el sistema, siendo lo suficientemente específica para que no existan ambigüedades, y ofreciendo la posibilidad de descomponer los trabajos de implementación en componentes más manejables, visualizándolos mediante una notación común.

2.3.1 Descripción de los módulos

Tomando como artefactos de entrada los requisitos funcionales definidos previamente por los analistas del SAGEB, se realizará a continuación una breve descripción de cada módulo para comprender las características de cada uno de ellos, y se especificarán las funcionalidades que los mismos brindarán. Los módulos a diseñar son:

- Gestionar Banco
- Cuenta Banco
- Cuenta de Otros Conceptos

Gestionar Banco: permite al usuario gestionar todos los bancos que utiliza el sistema contable. Los bancos constituyen un tipo de contrapartida, que es utilizada en la creación de las cuentas ampliadas de 14 dígitos. Además se almacena información indispensable que posibilita la comunicación con los mismos.

Funcionalidades:

- Registrar Banco
- Actualizar Banco
- Consultar Banco
- Buscar Banco

Cuenta Banco: permite la gestión de las cuentas que el banco tiene en otros bancos. Algunos de los datos registrados tienen una gran utilidad y permiten su posterior utilización dentro del sistema contable, como por ejemplo el número de la cuenta en el otro banco, las personas asociadas y los medios de comunicación a los que se puede recurrir en caso de ser necesario.

Funcionalidades:

- Registrar Cuenta Banco
- Actualizar Cuenta Banco
- Consultar Cuenta Banco
- Cerrar Cuenta Banco
- Buscar Cuenta Banco

Cuenta de Otros Conceptos: permite la gestión de las cuentas de otros conceptos existentes en el sistema. Estas cuentas son cuentas del banco y son utilizadas para la contabilidad interna del banco.

Funcionalidades:

- Registrar Cuenta de Otros Concepto
- Actualizar Cuenta de Otros Concepto
- Consultar Cuenta de Otros Concepto
- Cerrar Cuenta de Otros Concepto
- Buscar Cuenta de Otros Concepto

2.3.2 Modelo de diseño

El modelo de diseño es un refinamiento y formalización adicional del modelo de análisis, donde se toman en cuenta las consecuencias del ambiente de implementación. El resultado del modelo de diseño son especificaciones muy detalladas de todos los objetos, incluyendo sus operaciones y atributos. [13] El modelo de diseño representa la primera entrada para la disciplina de implementación. Este modelo está formado por artefactos, en los que encontramos: los diagramas de paquetes y diagramas de clases.

Modelo de paquetes

Un paquete de diseño es usado para estructurar el modelo de diseño mediante su división en partes más pequeñas. Los paquetes son usados principalmente para la administración de configuración y organización del modelo.

Es de gran ayuda a la hora de realizar el diseño de un software el organizar por paquetes las clases y ficheros de un módulo. Los modelos de paquetes de los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos tienen la misma estructura, motivo por el cual se presentó solo el modelo de paquetes del módulo Gestionar banco.

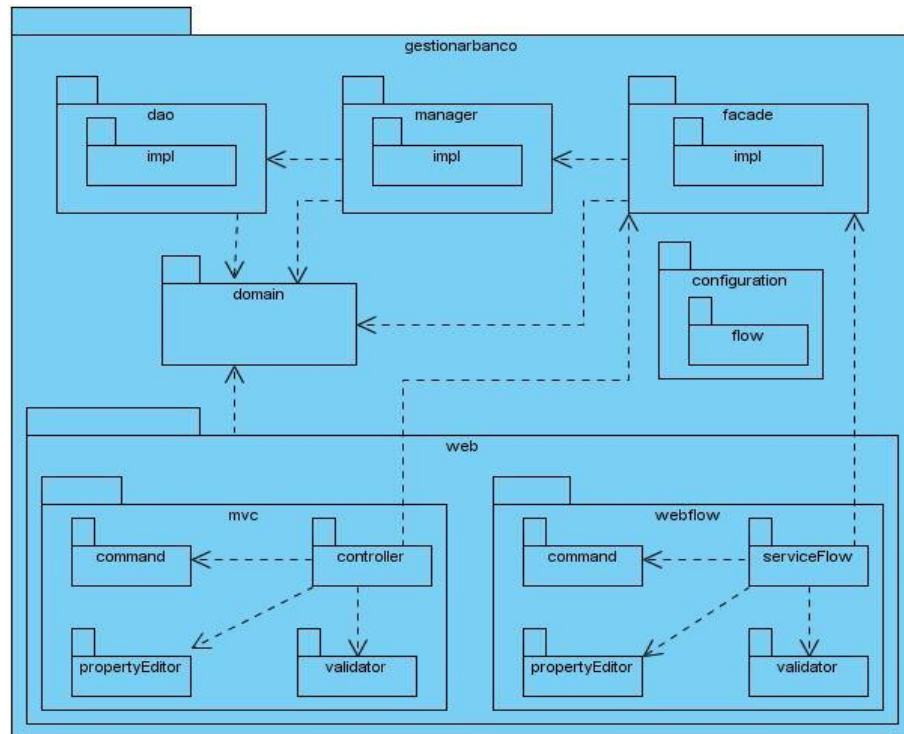


Ilustración 2 Modelo de paquetes del módulo Gestionar banco

A continuación se mencionan las características de cada paquete:

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

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

Paquete flow: en este paquete están presente los archivos XML que definen los flujos para Spring WebFlow.

Paquete dao: en este paquete están las interfaces y las clases que implementan el acceso a datos del módulo, además de los ficheros de mapeos de Hibernate.

Paquete manager: en este paquete se encuentran las interfaces e implementaciones del negocio del módulo correspondiente.

Paquete facade: en este paquete se encuentra la interfaz del módulo hacia la Capa de presentación y la implementación de las funcionalidades que se le brindarán a la presentación.

Paquete domain: en este paquete están las clases del dominio del módulo.

Paquete web: este paquete es el contenedor de los paquetes *mvc* y *webFlow*.

Paquete mvc: en este están los paquetes que contienen la lógica de presentación en el servidor para SpringMVC.

Paquete commad: contiene clases que representan objetos a manipular en los formularios.

Paquete controller: contiene las diferentes clases que heredan de los controladores de Spring.

Paquete propertyEditor: clases para convertir un grupo de datos en objetos.

Paquete validator: contiene las clases encargadas de validar los datos del lado del servidor.

Paquete webFlow: en este paquete están los paquetes que contienen la lógica de presentación en el servidor para Spring WebFlow.

Paquete serviceFlow: contiene las diferentes clases que median entre el flujo y la facade.

Para un mayor entendimiento del diseño de los módulos se realizaron los diagramas de clases del diseño y solamente se muestran en este capítulo los diagramas correspondientes al módulo Gestionar banco, el resto de los diagramas se exponen en los Anexo 1, Anexo 2 y Anexo 3 de la versión del documento de Tesis en formato digital.

Diagramas de Clases del Diseño

Un diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces de la aplicación. Sirve también para visualizar las relaciones entre las clases que involucran el sistema.

A continuación se describen el diseño correspondiente a cada capa:

Diseño de la Capa de Presentación

En la capa de presentación de cada módulo se hace uso mayormente del framework Spring WebFlow pues estos contemplan procesos con flujos complejos. En el caso del módulo Gestionar banco se utiliza este framework para darle solución a los requisitos que contemplan la gestión de los medios de comunicaciones.

Muestra del diseño de la capa de presentación:

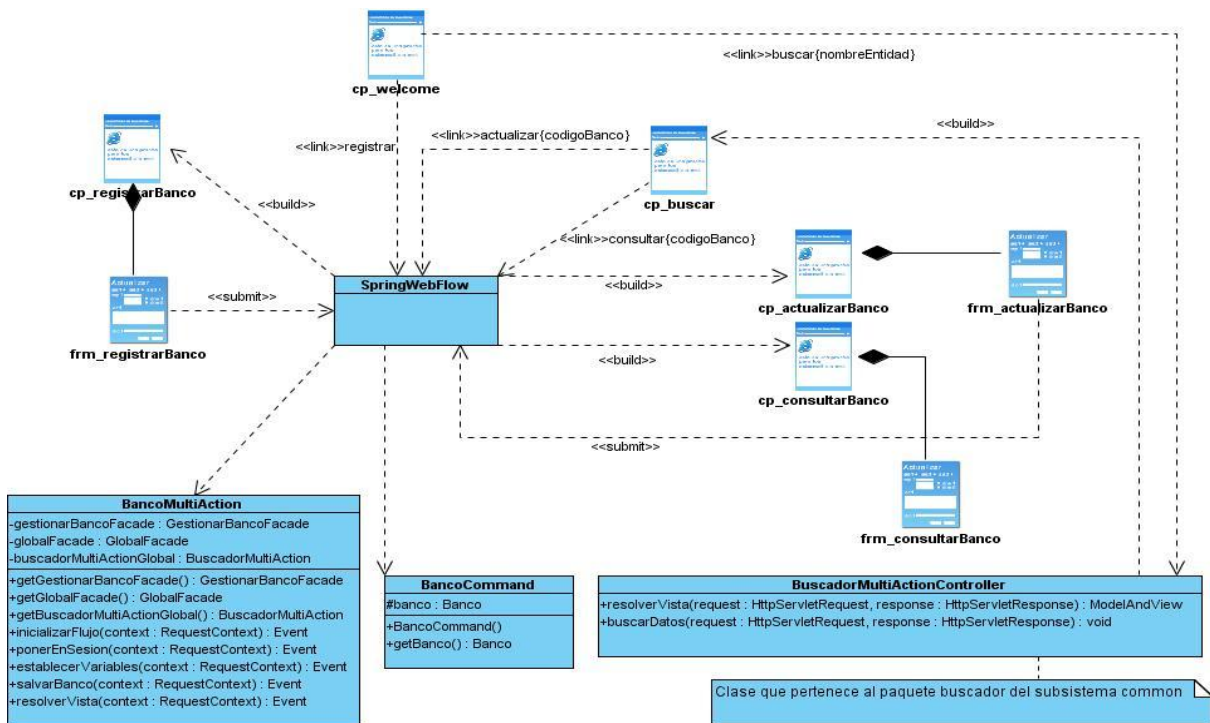


Ilustración 3 Diseño de la capa de presentación del módulo Gestionar Banco

Client page: Representan páginas web encargadas de mostrar los formularios de información al usuario.

Form: Representa una colección de campos de entrada de datos, es parte de una *Client Page*.

SpringWebFlow. En dicha clase se controlan los flujos por los que son guiados los usuarios en el sistema, además representa el mecanismo interno con el que SpringWebFlow gestiona las peticiones realizadas desde el cliente.

BancoMultiAction: En dicha clase se gestionan las operaciones asociadas a todos los flujos del módulo. Para su funcionamiento SpringWebFlow interactúa con ella.

BancoCommand: Dicha clase representa el objeto Banco que se manipula en los formularios.

Diseño de la Capa de Negocio

Los módulos tienen una fachada, la cual es la encargada de brindar funcionalidades a la capa de presentación, de esta forma dicha capa no conocerá la verdadera implementación de los métodos. La fachada de un módulo puede interactuar con otros módulos o subsistemas a través de las fachadas de los mismos.

Las clases manager contienen toda la lógica de la aplicación y es donde encontramos declaradas las clases que componen la capa de negocio; en el ManagerImpl encontramos implementadas sus funcionalidades para solucionar el negocio.

Global: Paquete en el que encontramos funcionalidades comunes a diferentes subsistemas.

Modelo de la capa de negocio del módulo Gestionar banco:

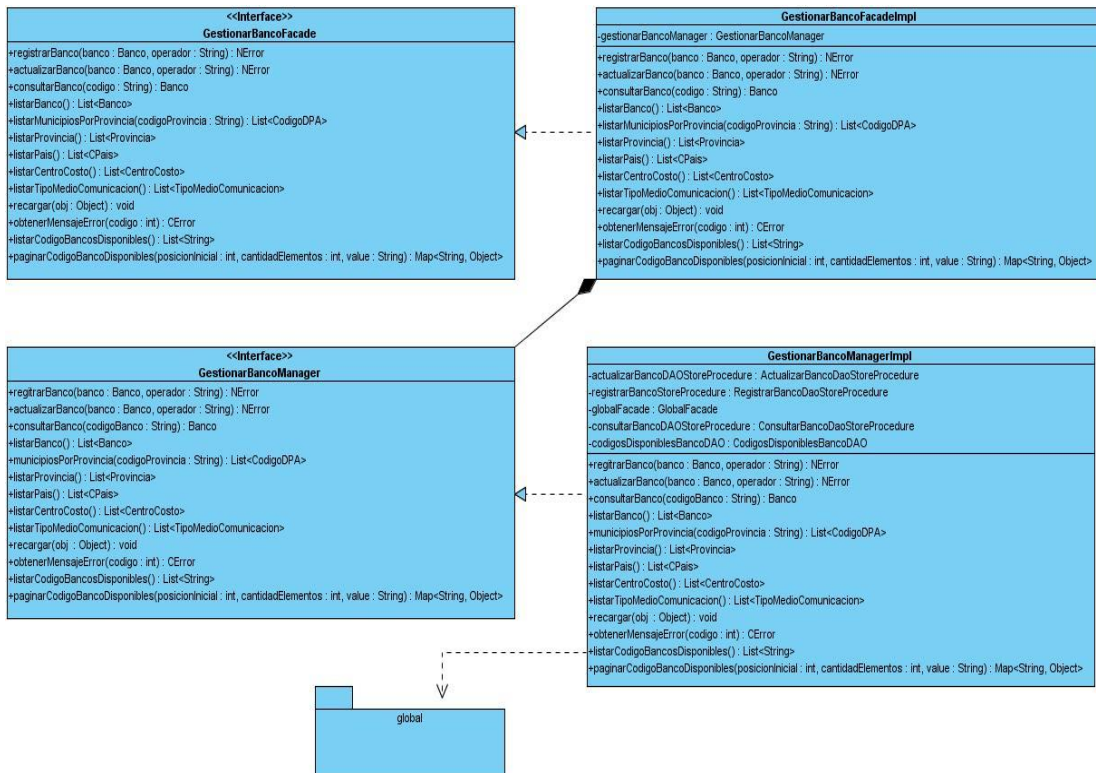


Ilustración 4 Diseño de la capa de Negocio del Módulo Gestionar Banco

Diseño de la Capa de Acceso a Datos

En dicha capa se hace uso del patrón DAO y de los componentes DAO genéricos utilizados en la arquitectura del proyecto compuesto por la interface *FinixuBaseDAO* y la clase *FinixuAbstractBaseDAO*. Estas clases ofrecen funcionalidades básicas a realizar con las clases persistentes como son: persistir, actualizar, listar, buscar por identificador, eliminar. Además se utiliza la clase *StoredProcedureDAOSupport*, la cual es un componente desarrollado por el proyecto para facilitar las invocaciones a funciones y procedimientos almacenados.

En el siguiente diagrama están declaradas e implementadas las funcionalidades encargadas de interactuar con la base de datos:

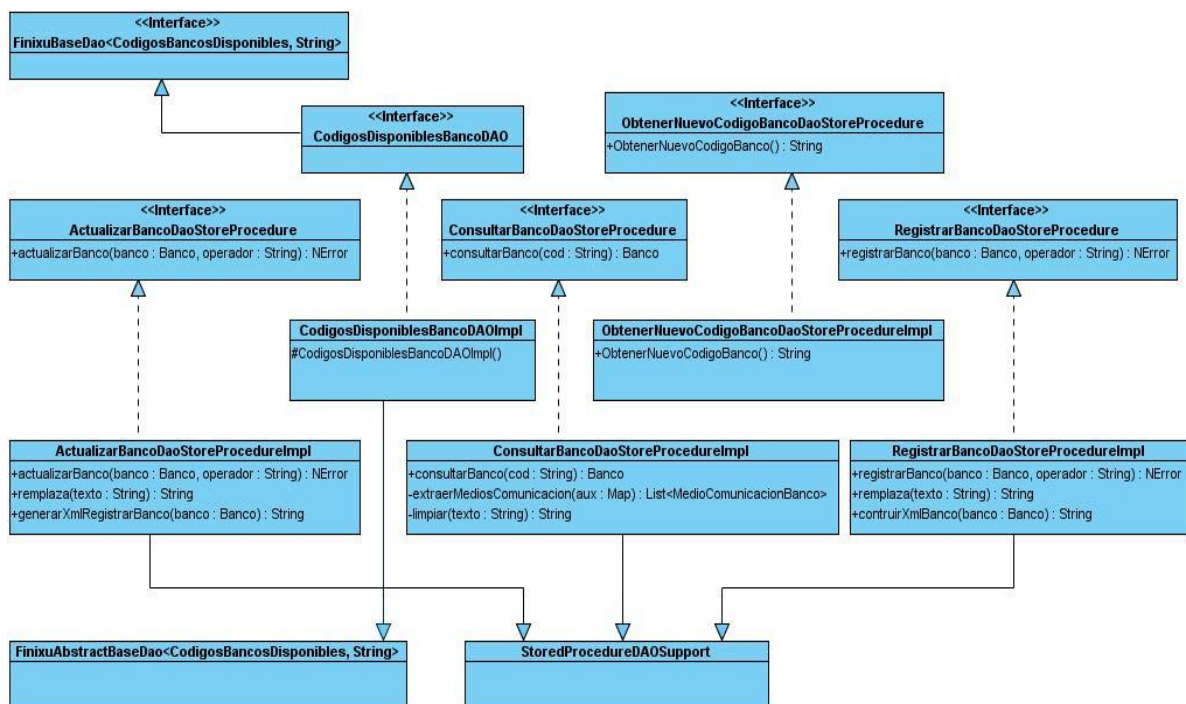


Ilustración 5 Diseño de la capa de Acceso a Datos del Módulo Gestionar Banco

Diseño de la Capa de Dominio

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

A continuación se presenta el modelo de dominio del módulo Gestionar banco:

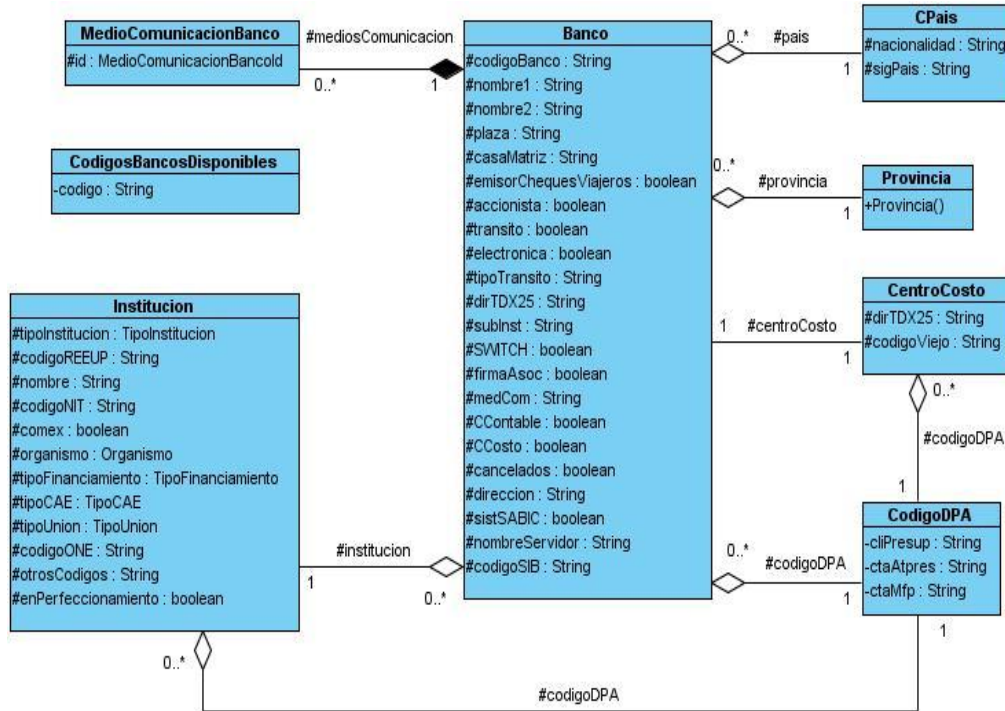


Ilustración 6 Modelo Dominio del Módulo Gestionar Banco

Diagramas de Interacción del Módulo Gestionar banco

Por cada escenario de los diagramas de clases del diseño es modelado un diagrama de interacción. Como diagrama de interacción fue seleccionado el de secuencia, el cual no es más que una representación que muestra, en determinado escenario de un caso de uso, los eventos generados por actores externos, su orden y los eventos internos del sistema. Contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos.

Se realizará un diagrama de secuencia para la petición de cargar datos y otra para la petición de persistirlos por la complejidad de los flujos presentes en los módulos.

Se muestran seguidamente los correspondientes diagramas asociados al escenario Registrar Banco.

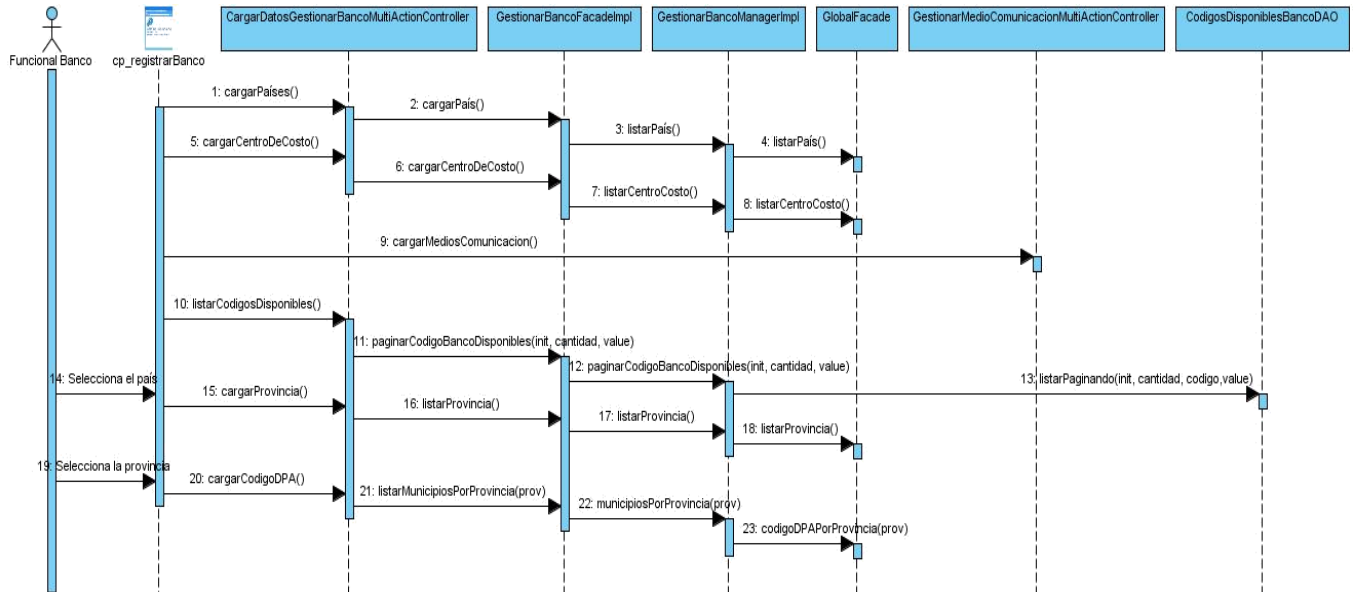


Ilustración 7 Diagrama de secuencia: Registrar Banco (Cargar Datos)

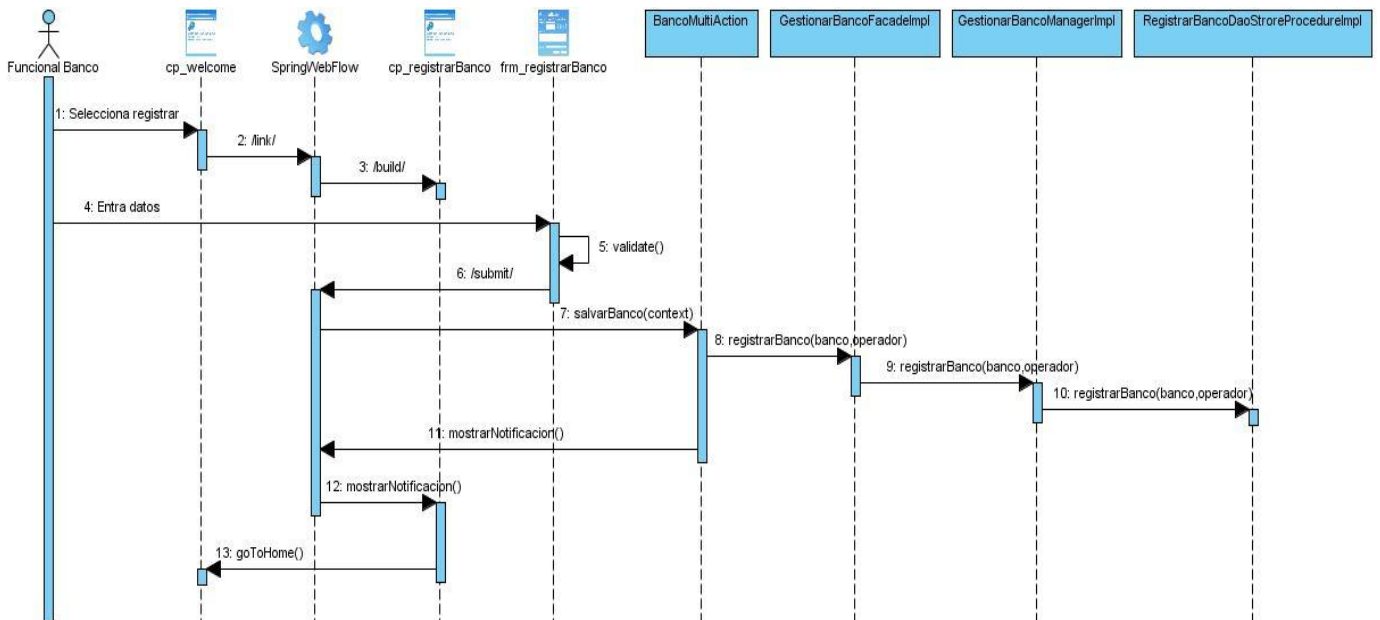


Ilustración 8 Diagrama de secuencia: Registrar Banco (Persistir)

2.3.3 Modelo de Datos

El Modelo de Datos se utiliza para la descripción de una base de datos. Este, por su parte, permite describir las estructuras de datos de la base (el tipo de datos que incluye la base y la forma en que se relacionan), las relaciones de integridad (las condiciones que los datos deben cumplir para reflejar correctamente la realidad deseada) y las operaciones de manipulación de los datos (agregado, borrado, modificación y recuperación de los datos de la base).

A continuación se representa el Modelo de datos correspondiente a los módulos Gestionar banco, el de los módulos Cuenta banco y Cuenta de otros conceptos se especificarán en el Anexo 4 de la versión del documento de Tesis en formato digital. Es importante resaltar que dichos módulos utilizarán una base de datos de un sistema que existe, en la cual las tablas no se encuentran relacionadas, por tanto se utilizará Hibernate para mapear las relaciones, permitiendo de esta forma que estas se comporten como si estuvieran relacionadas entre sí.

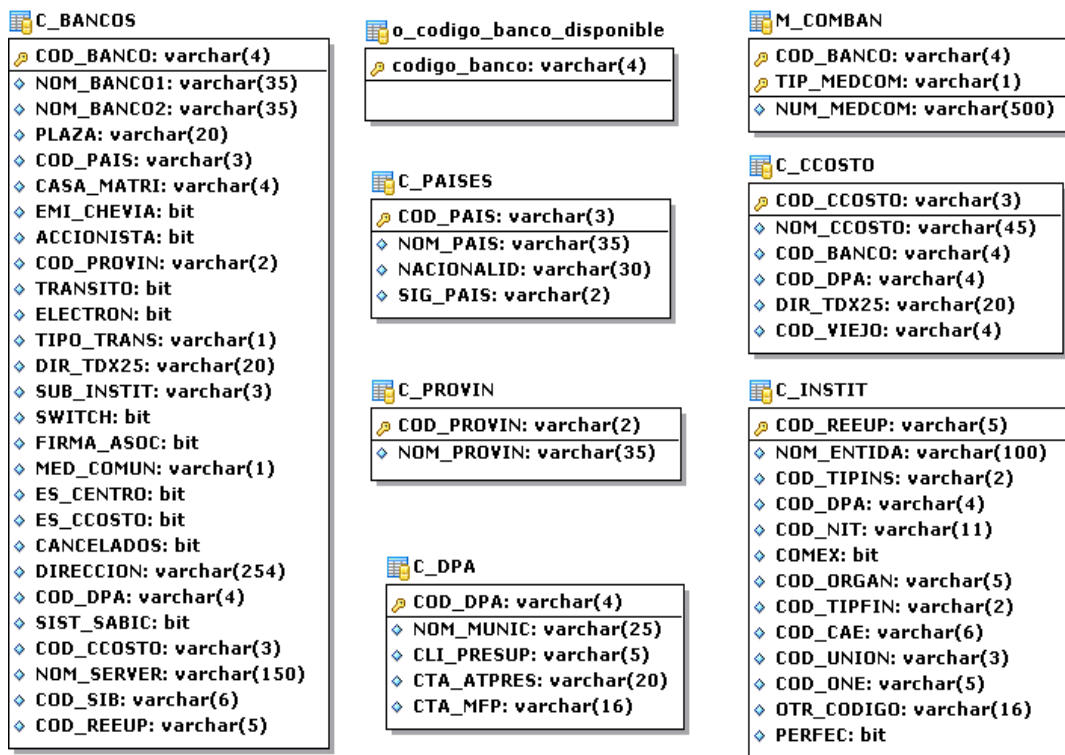


Ilustración 9 Modelo de Datos del módulo Gestionar banco

2.3.4 Patrones de diseño empleados

Generalmente, para elaborar el diseño se utilizan un grupo de patrones o modelos para lograr los objetivos esperados. Estos son considerados como procedimientos para solucionar diversos problemas del mismo tipo y son la base para la búsqueda de soluciones a dificultades comunes en el desarrollo de software.

Para definir el diseño de la solución propuesta se tuvieron en cuenta varios patrones, a continuación se especifican los patrones seleccionados:

DAO (Data Access Object): La utilización de dicho patrón se evidencia en la definición de las clases DAO, en las cuales se facilitan las funcionalidades específicas a realizar sobre la base de datos. De esta forma el negocio no se verá afectado de posibles cambios que puedan producirse en la lógica de acceso a datos y la fuente de datos.

Patrones GRASP (Patrones de Asignación de Responsabilidades)

Controlador: Un ejemplo de la aplicación de este patrón lo constituye la clase controladora CargarDatosGestionarBancoMultiActionController, dicha clase tendrá la responsabilidad de manejar los eventos que consisten en cargar los datos que serán mostrados al usuario.

Experto: Este patrón lo podemos encontrar en diferentes clases en el sistema, como son los dominios, las cuales son expertos en tener los atributos que se le definen, encontramos también a las clases DAO como por ejemplo la clase CodigosDisponiblesBancoDAO, la cual es responsable de listar los códigos de los bancos que no se hayan utilizado y a la vez que se utiliza un código de banco, lo elimina de la lista de bancos disponibles.

Alta cohesión: El sistema en general fue diseñado en módulos (Ej. Gestionar banco, Cuenta banco y Cuenta de otros conceptos), definidos a partir de que en los mismos se manejarán la menor cantidad de entidades posibles, de manera tal que a las clases pertenecientes a las diferentes capas se les asignarán las responsabilidades necesarias y bien delimitadas para garantizar una alta cohesión generando por ende **bajo acoplamiento**.

Bajo acoplamiento: Este patrón se evidencia a la hora de definir interfaces e implementaciones, como

por ejemplo la interfaz GestionarBancoFacade y su implementación, las cuales permiten que las clases de la presentación CargarDatosGestionarBancoMultiActionController y BancoMultiAction se relacionen exclusivamente con ellas para realizar sus operaciones, disminuyéndose de esta forma el impacto de cambios posteriores en el negocio del sistema.

Patrones GOF (Gang of Four)

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

2.4 Conclusiones del capítulo

Como resultado de este capítulo se realizó el análisis de la arquitectura base definida por el proyecto SAGEB y el diseño de los módulos Cuenta banco, Cuenta de otros conceptos y Gestionar banco mediante la correcta utilización de patrones que organizan el trabajo y posibilitan que el diseño sea entendible, flexible y sus componentes reutilizables. Lo realizado en este capítulo sirve de base fundamental en la implementación de los módulos pues se logra una mayor organización y uniformidad.

Capítulo 3- Implementación y Prueba

3.1 Introducción

En el presente capítulo se explican y se presentan las implementaciones de los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos, modelándose de esta forma los artefactos pertenecientes al flujo de trabajo de implementación. Se muestran fragmentos de código de los principales flujos de la implementación, así como los métodos más relevantes. Además se figuran las pruebas unitarias realizadas al sistema para dar cumplimiento a los requisitos especificados.

3.2 Modelo de Implementación

Según la metodología RUP el modelo de implementación toma el resultado del modelo de diseño para generar el código final del sistema. Describe además, cómo los elementos de diseño se implementan en componentes (ficheros de código fuente, de código binario, ejecutable, scripts), como se organizan dichos componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y como dependen de los componentes unos de otros.

3.2.1 Diagrama de Componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Permite modelar la vista estática del sistema mostrando las dependencias lógicas entre un conjunto de componentes de software. Los componentes pueden ser de código fuente, librerías, binarios o ejecutables y tienen relaciones de traza con los elementos del modelo que implementan.

El diagrama de componentes que se presenta a continuación, muestra las relaciones existentes entre los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos y los restantes componentes del sistema:

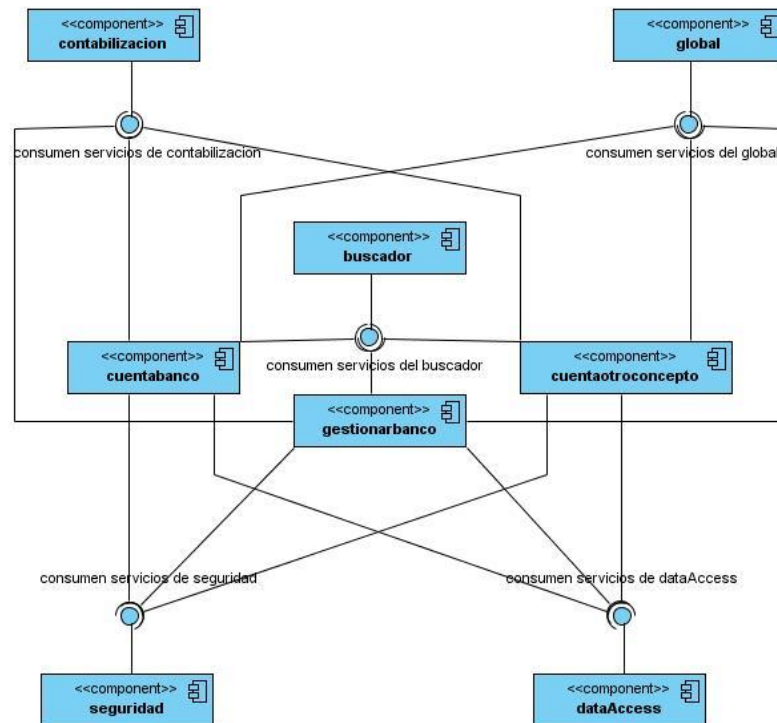


Ilustración 10 Diagrama de componentes de los módulos.

Seguidamente se explican los componentes de forma general:

buscador: Brinda un conjunto de clases que constituyen un motor de búsqueda, las cuales se muestran mediante una interfaz gráfica en la cual se especifican los criterios de búsqueda en dependencia del módulo que la use. Para realizar una gestión eficiente de la información en los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos se hace necesario la búsqueda tanto de cuentas bancos, cuentas de otros conceptos y de bancos respectivamente.

contabilización: Agrupa un conjunto de clases que contienen las funcionalidades necesarias para realizar la contabilización de determinadas operaciones que así lo requieran. Además de contener clases dominios que utilizan diferentes módulos para su funcionamiento. La mayoría de las clases dominios utilizadas por los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos están contenidas en el componente contabilización.

global: Contiene un conjunto de clases que implementan funcionalidades comunes a diferentes módulos,

como es el caso de las clases que se encargan de gestionar los medios de comunicaciones. Agrupa además diferentes clases dominios comunes. La gestión de los bancos y de las cuentas banco requieren en ocasiones la asociación de medios de comunicación los cuales se gestionan en este componente.

seguridad: Componente encargado de mantener la seguridad en todo el sistema mediante la comprobación de las peticiones y sus permisos en dependencia del usuario que la realice, de aquí la necesidad de que exista una comunicación con el mismo para garantizar la seguridad de los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos.

dataAccess: Encargado de brindar todos los elementos necesarios para llevar a cabo la conexión a la Base de Datos, por tal motivo todos los subsistemas y por ende los módulos dependen de el para poder realizar las funcionalidades que engloban.

3.3 Estándares de codificación

Un estándar de codificación comprende todos los aspectos relacionados con la generación de código, de tal manera que sea prudente, práctico y entendible para todos los programadores. Un código fuente completo debe reflejar un estilo armonioso y uniforme, como si un único programador hubiera escrito todo el código.

El desarrollo de una aplicación informática exige que se adopten estándares de codificación y estilo. El uso de estándares asegura la legibilidad del código entre distintos desarrolladores, permite la guía para el mantenimiento o actualización del sistema, además de que facilita la portabilidad entre plataformas y aplicaciones.

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

3.3.1 Convención de código general

- Los nombres de los paquetes deben escribirse en letra minúscula, representar sustantivos y describir de alguna forma su contenido. Ej. **controller**.
- Las clases e interfaces deben nombrarse comenzando por letra inicial mayúscula, en caso de estar formado por palabras compuestas el inicio de cada palabra deberá comenzar con mayúscula. Ej.

BancoCommand.

- Las variables y los nombres de los métodos comienzan siempre con minúscula, en caso de estar compuesto por más de una palabra, estas empezarán con mayúscula exceptuando la primera letra. Ej. **casaMatriz** y **registrarBanco()**.
- Las constantes se escribirán en letras mayúsculas, si están formadas por más de una palabra se separarán por el signo guión bajo “_”. Ej. **TIEMPO_ESPERA**.

3.3.2 Capa de presentación

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

- Paquete **command**: [Nombre de la clase] + [**Command**]. Ej. **BancoCommand**.
- Paquete **propertyEditor**: [Nombre de la clase] + [**PropertyEditor**]. Ej. **ContratoPropertyEditor**
- Paquete **serviceFlow**: [Nombre de la clase] + [**Action**] o [**MultiAction**]. Ej. **BancoMultiAction**.
- Paquete **validator**: [Nombre de la clase] + [**CommandValidator**]. Ej. **CuentaAmpliadaCommandValidator**.

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

- Paquete **command**: De la misma forma que en el sub-paquete **webFlow**.
- Paquete **controller**: [Nombre de la clase] + [nombre del controlador que se hereda]. Ej. **CargarDatosGestionarBancoMultiActionController**.
- Paquete **validator**: [Nombre de la clase] + [**Validator**]. Ej. **CartaCreditoValidator**.
- Paquete **propertyEditor**: De la misma forma que en el sub-paquete **webFlow**.

3.3.3 Capa de Negocio

Las clases se nombrarán de la siguiente manera:

- Paquete **facade**:
 - Interfaces: [nombre del módulo] + [**Facade**]. Ej. **GestionarBancoFacade**.

- Clases que implementan las interfaces: [nombre del módulo] + [**FacadeImpl**]. Ej.

GestionarBancoFacadeImpl

- Paquete **manager**:
 - Interfaces: [nombre del negocio] + [**Manager**]. Ej. **GestionarBancoManager**.
 - Clases que implementan las interfaces: [nombre del negocio] + [**ManagerImpl**] Ej.
GestionarBancoManagerImpl.

3.3.4 Capa de Acceso a Dato

Las clases se nombrarán de la siguiente manera:

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

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

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

3.4 Aspectos Principales de la Implementación

3.4.1 Utilización de Spring WebFlow Framework

En el capítulo anterior se especificaba el uso del framework Spring WebFlow por las ventajas que ofrece al definir y gestionar los flujos de páginas dentro de una aplicación web. Dicho framework se utiliza en los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos para simplificar la implementación de casos de usos agrupados por el patrón CRUD y manejar de forma satisfactoria la gestión de la información referente a los medios de comunicaciones y las personas autorizadas. En cada módulo se implementa un flujo genérico principal que responde a diferentes funcionalidades comunes de las vistas e

interactúa con el mismo objeto (**command**) en distintos estados. Dicho flujo dará respuestas a un conjunto de funcionalidades comunes de los casos de usos.

Se describirá brevemente a continuación el funcionamiento de los flujos, tomando como ejemplo al módulo Gestionar banco, ya que los flujos de los tres módulos tienen similar comportamiento, se representarán los estados más importantes, así como otros estados presentes en los otros dos módulos que poseen similar importancia.

Descripción del flujo de WebFlow

Lo primero que se hace en los flujos de los módulos es la declaración de la variable que representará a la clase que se encuentra en el paquete command, clase que representa el objeto que se manipula en los formularios. Después comenzará a ejecutarse el flujo con la llamada al método `inicializarFlujo()`, en el cual se establecerá el valor inicial del objeto command. Importante destacar que todas las funcionalidades presentes en el flujo se encuentran implementadas en la clase **BancoMultiAction**, la misma hace función de fachada, utilizándose para evitar la interacción directa desde el flujo con el negocio. Las funcionalidades que se exponen en el flujo no necesitan ser declaradas con los parámetros correspondientes, pues recibiendo el **RequestContext** como único parámetro, Spring WebFlow es capaz de reconocerlo.

```
var name="bancoCommand"  
class="cu.uci.finixubnc.contabilidad.gestionarbanco.web.webflow.command.Banco  
Command" />  
<on-start>  
<evaluate expression="bancoMultiAction.inicializarFlujo"/>  
</on-start>
```

Cada flujo tiene un subflujo encargado de realizar una llamada al principal:

```
<subflow-state id="subflowPrincipalBanco" subflow="subflowPrincipalBanco-flow">  
<input name="bancoCommand" />  
<input name="action" value="flowScope.action" type="java.lang.String" />  
<transition on="end" to="terminar"/>  
</subflow-state>
```

Dentro del subflujo principal encontramos:

La vista (`view-state`) es el estado más importante del flujo, en esta se activan recursos con tareas de

presentación en la aplicación, en este caso se activa *Java Server Pages* (jsp). Esta vista, además, es la que se le muestra al usuario y contiene las diferentes transiciones (`transition`) a partir de los posibles eventos que se puedan generar por el usuario.

```
<view-state id="principal" view="registrarBanco" model="bancoCommand">
  <on-render>
    <evaluate expression="bancoMultiAction.ponerEnSesion"/>
  </on-render>
  <transition on="registrarMC" to="medioComunicacionRegistrar">
    <evaluate expression="bancoMultiAction.establecerVariables"/>
  </transition>
  <transition on="actualizarMC" to="medioComunicacionRegistrar">
    <evaluate expression="bancoMultiAction.establecerVariables"/>
  </transition>
  <transition on="eliminarMC" to="medioComunicacionRegistrar">
    <evaluate expression="bancoMultiAction.establecerVariables"/>
  </transition>
  <transition on="salvar" to="principal">
    <evaluate expression="bancoMultiAction.salvarBanco"/>
  </transition>
</view-state>
```

Nótese que se pasa a un nuevo estado en el flujo como respuesta a cada evento:

```
<transition on="salvar" to="principal">
  <evaluate expression="bancoMultiAction.salvarBanco"/>
</transition>
```

Luego de generarse el evento `salvar` se invoca al método `salvarBanco()` el cual realiza el registro del objeto banco.

A continuación se muestra el método encargado de realizar dicha funcionalidad:

```
public Event salvarBanco(RequestContext context) {
    BancoCommand command = (BancoCommand) context.getFlowScope().get("bancoCommand");
    Banco banco = command.getBanco();
    String action = context.getFlowScope().getString("action");
    String userName = UserHandler.getUser().getCodUsuari();
    NError nerror=new NError();
    String mensaje=" ";

    if (action.equals("registrar")) {
        nerror=gestionarBancoFacade.registrarBanco(banco, userName);
    } else if (action.equals("actualizar")) {
        nerror=gestionarBancoFacade.actualizarBanco(banco, userName);
    }

    if (nerror.isError()) {
        context.getFlowScope().put("codigoMensaje", "m1");
        CError error = gestionarBancoFacade.obtenerMensajeError(nerror.getCodigo());
        mensaje = error.getTextoError();
        context.getFlowScope().put("detallesMensaje", mensaje);
    } else
        context.getFlowScope().put("codigoMensaje", "ok");

    return success();
}
```

En este método se realiza el registro o la actualización de la entidad en dependencia de la acción que se indique, además se verifica si ocurrió algún error al realizar la operación y se guarda en el flujo una variable que indicará esta respuesta.

Luego para comprobar si el registro se efectuó de forma satisfactoria o no, se vuelve a entrar al estado *principal*. En este caso volverá a la vista realizando comprobaciones en el java script para ver si existió algún error al persistir la entidad, en el caso de que exista se muestra el error al usuario, si no se persiste la entidad y se pasa al estado final del flujo:

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

En este flujo principal como en el flujo del módulo Cuenta banco un aspecto relevante es la utilización de un subflujo general que se encarga de gestionar los medios de comunicaciones, con esto se logra una

gran reutilización de código y transparencia a la hora de gestionar dichos medios. Se llega a este estado a partir de diferentes transiciones, en las cuales primeramente antes de pasar al subflujo, se evalúa el método `establecerVariables()` donde se establecen variables que se mantendrán en la memoria del flujo:

```
<transition on="registrarMC" to="medioComunicacionRegistrar">
  <evaluate expression="bancoMultiAction.establecerVariables"/>
</transition>
<transition on="actualizarMC" to="medioComunicacionRegistrar">
  <evaluate expression="bancoMultiAction.establecerVariables"/>
</transition>
<transition on="eliminarMC" to="medioComunicacionRegistrar">
  <evaluate expression="bancoMultiAction.establecerVariables"/>
</transition>
```

El subflujo requiere los parámetros de entrada (operación, listado, índice y propietario). Al terminar la ejecución de este flujo se regresa al estado de vista inicial.

```
<subflow-state id="medioComunicacionRegistrar" subflow="medio-comunicacion-flow">
  <input name="operacion" value="flowScope.operacion" type="java.lang.String"/>
  <input name="listado" value="bancoCommand.banco.getMediosComunicacion()"
    type="java.util.ArrayList"/>
  <input name="indice" value="flowScope.indice" type="java.lang.Integer"/>
  <input name="propietario" value="" type="java.lang.String"/>
  <transition to="principal"/>
</subflow-state>
```

El flujo principal del módulo Cuenta banco además contempla la utilización de un subflujo general que se encarga de gestionar las personas autorizadas, subflujo muy parecido al explicado anteriormente de los medios de comunicación.

```
<transition on="registrarPA" to="personaAsociada">
  <evaluate
    expression="cuentaAmpliadaMultiAction.establishVariablesPA"/>
</transition>
<transition on="actualizarPA" to="personaAsociada">
  <evaluate
    expression="cuentaAmpliadaMultiAction.establishVariablesPA"/>
</transition>
<transition on="eliminarPA" to="personaAsociada">
  <evaluate
    expression="cuentaAmpliadaMultiAction.establishVariablesPA"/>
</transition>
```

En este se establecen de igual manera variables que se mantendrán en la memoria del flujo `establishVariablesPA()` y al terminar la ejecución de este flujo se regresa al estado de vista inicial.

```
<subflow-state id="personaAsociada" subflow="permisocuentaampliada-flow">
  <input name="operacion" value="flowScope.operacion" type="java.lang.String"/>
  <input name="listado"
    value="cuentaAmpliadaCommand.cuentaBanco.getPersonasAsociada()"
    type="java.util.ArrayList"/>
  <input name="indice" value="flowScope.indice"
    type="java.lang.Integer"/>
  <transition to="principal"/>
</subflow-state>
```

El hecho de que los flujos de los medios de comunicación y de las personas autorizadas sean flujos globales para la aplicación, se hace necesario configurar su registro en el contexto de Spring WebFlow en cada módulo.

A continuación un ejemplo de cómo quedó configurado el módulo Cuenta banco. Para poder relacionarlos se especifica que el flujo de personas autorizadas es el padre del flujo de medios de comunicación y este a su vez es el padre del flujo del módulo Cuenta banco, con todo esto se logra la comunicación entre todos los flujos.

Registro del flujo de personas autorizadas

```
<webflow:flow-registry id="permisosFlowRegistry"
    flow-builder-services="permisosFlowBuilderServices">
<webflow:flow-location-pattern
    value="classpath:cu/uci/finixubnc/contabilidad/common/configuration/flow/permisocuentaampli
ada-flow.xml"/>
</webflow:flow-registry>
```

Registro del flujo de medios de comunicación

```
<webflow:flow-registry id="commonFlowRegistry"
    flow-builder-services="commonFlowBuilderServices" parent="permisosFlowRegistry">
<webflow:flow-location-pattern
    value="classpath:cu/uci/finixubnc/common/global/configuration/flow/medio-comunicacion-
flow.xml"/>
</webflow:flow-registry>
```

Registro del flujo del módulo Cuenta banco

```
<webflow:flow-registry id="flowRegistry" flow-builder-services="flowBuilderServices"
    parent="commonFlowRegistry">
<webflow:flow-location-pattern
    value="classpath:cu/uci/finixubnc/contabilidad/cuentaampliada/configuration/flow/*-flow.xml"/>
</webflow:flow-registry>
```

3.5 Descripción de las clases y funcionalidades de los módulos

Para un mayor entendimiento de la implementación, a continuación se describirán las clases de mayor peso de los módulos. De las clases que se encargan de manejar el flujo de cada módulo se describirá una, ya que las tres tienen similar comportamiento, en este caso será del módulo Cuenta banco la clase CuentaAmpliadaMultiAction. Se describirá además la clase que contiene los atributos asociados a una Cuenta banco.

Nombre: CuentaAmpliadaMultiAction	
Tipo de clase: Controladora	
Atributos	Tipo
cuentaAmpliadaFacade	CuentaAmpliadaFacade
globalFacade	GlobalFacade
buscadorMultiActionGlobal	BuscadorMultiAction
Para cada responsabilidad	
Nombre:	Descripción:
inicializarFlujo (RequestContext contex)	Guarda en memoria el objeto command de tipo Cuenta banco, el cual tendrá valor en dependencia de la acción que se esté ejecutando.
ponerEnSesion (RequestContext contex)	Permite poner en la sesión los medios de comunicación.
establecerVariablesMC (RequestContext contex)	Establece las variables con las que se trabajará en el flujo de medios de comunicación.
establecerVariablesPA (RequestContext contex)	Establece las variables con las que se trabajará en el flujo de personas autorizadas.
registrarCuentaBanco (RequestContext contex)	Registra o actualiza una cuenta banco en dependencia de la acción que se esté ejecutando.
resolverVista(RequestContext contex)	Desarrolla una vista en dependencia de la clase especificada.

Tabla 1 Clase CuentaAmpliadaMultiAction del módulo Cuenta banco.

En los módulos Cuenta banco y Cuenta de otros conceptos encontramos las clases CuentaBanco y CuentaOtroConcepto respectivamente, dichas clases extienden de una misma clase **CuentaAmpliada**, por tanto se describirá a continuación dicha clase:

Nombre: CuentaAmpliada
Tipo de clase:Modelo

Atributo	Tipo
fechaCreacion	Java.util.Date
inactiva	Boolean
cerrada	Boolean
bloqueada	Boolean
debito	Boolean
credito	Boolean
nombre	String
frecEstadoCuenta	<i>cu.uci.finixubnc.common.buscador.domain.Frecuencia</i>
eliminarObserv	Boolean
tipoMedioComunicacion	<i>cu.uci.finixubnc.common.global.domain.TipoMedioComunicacion</i>
permiteChequera	Boolean
firma	String
condicionDebitoCredito	String
personasAsociada	java.util.List
mediosComunicacion	java.util.List
categoriaCuenta	CategoriaCuenta
centroCosto	<i>cu.uci.finixubnc.common.global.domain.CentroCosto</i>
propositoCuenta	PropositoCuenta
identificador	CuentaAmpliadaId
monedaIdentificadora	NomMoneda
cuentaIdentificadora	Cuenta
tipoContraparte	Char
codigoContraparte	String

Tabla 2 Clase CuantaAmpliada utilizada por los módulos Cuenta banco y Cuenta de otros conceptos.

La clase Cuentabanco contiene además de todos estos atributos, el atributo:

numCuentaBancoCorresponsal	String
----------------------------	--------

3.6 Realización de Prueba

Cuando se desarrolla un software son innumerables las posibilidades de errores que podemos encontrar en el mismo, por eso la necesidad de desarrollar alguna actividad que sea capaz de garantizar que el producto final sea aceptable para el usuario. Una actividad muy eficiente lo constituye la realización de pruebas al producto, las cuales son un elemento crítico para la garantía de la calidad del software. Para comprobar el funcionamiento eficiente de los módulos implementados se desarrollaron pruebas unitarias, con las cuales se probó el correcto funcionamiento del código.

3.6.1 Pruebas unitarias

La **prueba unitaria** es la prueba enfocada a los elementos testeables más pequeño del software. Es aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera. Antes de iniciar cualquier otra prueba es preciso probar el flujo de datos de la interfaz del componente. Si los datos no entran correctamente, todas las demás pruebas no tienen sentido. El diseño de casos de prueba de una unidad comienza una vez que se ha desarrollado, revisado y verificado en su sintaxis el código a nivel fuente. [14]

Se aplicarán los métodos de pruebas adaptadas a este nivel, como son los métodos de prueba Caja Blanca para comprobar los caminos lógicos del software y Caja Negra para probar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene.

Diseño de casos de prueba para caja blanca

Las pruebas de Caja Blanca revisan la parte interna del software, específicamente sobre el código fuente. Se basan en el examen minucioso de los detalles procedimentales. Se comprueban los caminos lógicos del sistema generando casos de prueba que ejerciten las estructuras condicionales y los bucles. Existen varias técnicas de pruebas de la caja blanca que analizan diferentes partes del programa y se complementan entre sí para garantizar la calidad del sistema, entre los que encontramos: Camino Básico,

Condición, Flujo de Datos y Bucles. De todas las técnicas se selecciona la prueba del Camino Básico para obtener una medida de la complejidad ciclomática de los procedimientos, el número de caminos independientes y la cantidad mínima de casos de pruebas a realizar.

Para aplicar la técnica del Camino Básico es necesario conocer el conjunto de caminos independientes de un algoritmo determinado, para lo cual se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática. Los pasos que se siguen para aplicar esta técnica son:

- ✓ A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
- ✓ Se calcula la complejidad ciclomática del grafo.
- ✓ Se determina un conjunto básico de caminos independientes.
- ✓ Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Los métodos de prueba de la caja blanca permiten obtener casos de prueba que:

- ✓ Se ejerciten por lo menos una vez todos los caminos independientes para cada módulo.
- ✓ Se ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsas.
- ✓ Se ejecuten todos los bucles en sus límites y con sus límites operacionales.
- ✓ Se ejerciten las estructuras internas de datos para asegurar su validez.

Seguidamente se analizan y enumeran las sentencias de código de uno de los métodos de la clase **CuentaBancoManagerImpl** específicamente: **obtener ()**, este método internamente conforma una cuenta con todos los objetos asociados a ella como es el caso de las personas asociadas, la frecuencia, etc., y devuelve dicha cuenta y un atributo de tipo NError que devolverá un código de error en dependencia de que si ocurrió o no un problema originado en la búsqueda de dicha cuenta.

```
public CuentaAmpliada obtener(String codCuentaAmpliada, NError e) {

    CuentaAmpliada cuenta = getCuentaDAOStoreProcedure.getCuenta(codCuentaAmpliada, e);

    if (cuenta != null) {

        List<PersonaAsociada> personas = cuenta.getPersonasAsociada();
        Map<String, CPais> paises = new HashMap<String, CPais>();
        Map<String, TipoIdentificacion> tiposID = new HashMap<String, TipoIdentificacion>();
        Map<String, CategoriaPersona> categorias = new HashMap<String, CategoriaPersona>();
        for (PersonaAsociada personaAsociada : personas) {
            CPais pais = paises.get(personaAsociada.getPais().getCodigo());
            if (pais == null) {
                pais = globalFacade.obtenerPais(personaAsociada.getPais()
                    .getCodigo());
                paises.put(pais.getCodigo(), pais);
            }
            personaAsociada.setPais(pais);
            TipoIdentificacion tipo = tiposID.get(personaAsociada
                .getTipoIdentificacion().getCodigo());
            if (tipo == null) {
                tipo = globalFacade.obtenerTipoIdentificacion(personaAsociada
                    .getTipoIdentificacion().getCodigo());
                tiposID.put(tipo.getCodigo(), tipo);
            }
            personaAsociada.setTipoIdentificacion(tipo);
            CategoriaPersona categoria = categorias.get(personaAsociada
                .getCategoria().getCodigo());
            if (categoria == null) {
                categoria = nomencladorContFacade
                    .obtenerCategoriaPersona(personaAsociada.getCategoria()
                        .getCodigo());
                categorias.put(categoria.getCodigo(), categoria);
            }
            personaAsociada.setCategoria(categoria);
        }
        cuenta.setTipoMedioComunicacion(globalFacade
            .obtenerTipoMedioComunicacion(cuenta.getTipoMedioComunicacion()
                .getCodigo()));
        Frecuencia frecuencia = nomencladorContFacade
            .obtenerFrecuencia(cuenta.getFrecEstadoCuenta().getCodigo());
        cuenta.setFrecEstadoCuenta(frecuencia);

        Cuenta c = nomencladorContFacade.consultarCuenta(cuenta.getIdentificador().getCueSubcue());
        cuenta.setCuentaIdentificadora(c);
    }

    return cuenta;
}
```

Ilustración 11 Código del método obtener()

El primer paso consiste en dibujar el grafo de flujo asociado al código antes mostrado a través de nodos, aristas y regiones:

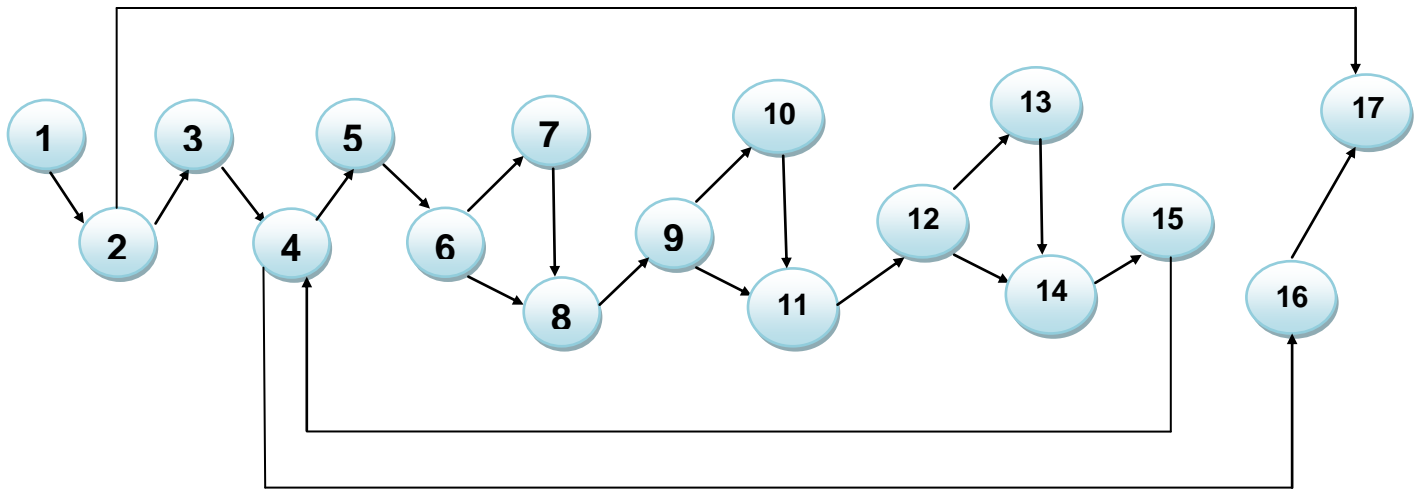


Ilustración 12 Grafo de flujo asociado al algoritmo obtener.

Luego de construido el gráfico se pasará a determinar la complejidad ciclomática mediante tres formas fundamentales que existen para calcular la complejidad:

1. La complejidad ciclomática, $V(G)$, se define como:

$$V(G) = A - N + 2$$

Donde: A es el número de aristas del grafo y N es el número de nodos.

$$V(G) = (21 - 17) + 2$$

$$V(G) = 6$$

2. La complejidad ciclomática, $V(G)$, también se define como:

$$V(G) = P + 1$$

Donde: P es el número de nodos predicado (nodos de los cuales parten dos o más aristas) contenidos en el grafo G.

$$V(G) = 5 + 1$$

$$V(G) = 6$$

3. La complejidad ciclomática, $V(G)$, también se define como:

$$V(G) = R$$

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

$$V(G) = 6$$

Acorde al resultado del cálculo de la complejidad ciclomática para el procedimiento, se determina que es necesario diseñar 6 casos de pruebas por el número de caminos básicos independientes a ejecutar, lo que significa que es necesario realizar como mínimo 6 pruebas al algoritmo.

A continuación se declararán los caminos básicos que puede tomar el algoritmo durante su ejecución, señalándose de cada camino, los elementos que los hacen independientes de los demás.

Camino básico #1: 1-2-17

Camino básico #2: 1-2-3-4-16-17

Camino básico #3: 1-2-3-4-5-6-8-9-11-12-14-15-4-16-17

Camino básico #4: 1-2-3-4-5-6-7-8-9-11-12-14-15-4-16-17

Camino básico #5: 1-2-3-4-5-6-8-9-10-11-12-14-15-4-16-17

Camino básico #6: 1-2-3-4-5-6-8-9-11-12-13-14-15-4-16-17

Seguidamente se procederá a confeccionar los casos de prueba para cada camino definido en el grafo de flujo.

Caso de prueba para el camino básico #1 (1-2-17)	
Descripción	Debe determinar una cuenta nula mostrándosele al usuario el error ocurrido.
Condición de ejecución	El código de la cuenta ampliada no puede estar vacío y a la misma vez que se ejecuta el método se cambia la cuenta en la Base de Datos ya sea porque se elimine o porque se le cambie el código.
Entrada	codCuentaAmpliada = 00100420136000, e = new NError()
Resultado esperado	Se obtiene una Cuenta banco nula.
Resultado	El resultado obtenido fue correcto.

Tabla 3 Caso de prueba para el algoritmo obtener: camino básico #1.

Caso de prueba para el camino básico #2 (1-2-3-4-16-17)	
Descripción	Debe determinar con precisión la cuenta que se mostrará al usuario para que pueda ser actualizada o consultada.

Condición de ejecución	El código de la cuenta ampliada no puede estar vacío y la cuenta asociada a ese código no tiene personas asociadas.
Entrada	codCuentaAmpliada = 00100720002000, e = new NError()
Resultado esperado	Se obtiene la Cuenta banco esperada satisfactoriamente.
Resultado	El resultado obtenido fue correcto.

Tabla 4 Caso de prueba para el algoritmo obtener: camino básico #2

Caso de prueba para el camino básico #3 (1-2-3-4-5-6-8-9-11-12-14-15-4-16-17)	
Descripción	Debe determinar con precisión la cuenta que se mostrará al usuario para que pueda ser actualizada o consultada.
Condición de ejecución	El código de la cuenta ampliada no puede estar vacío y la cuenta asociada a ese código tiene que tener una sola persona asociada.
Entrada	codCuentaAmpliada = 01121020033000, e = new NError()
Resultado esperado	Se obtiene la Cuenta banco esperada satisfactoriamente.
Resultado	El resultado obtenido fue correcto.

Tabla 5 Caso de prueba para el algoritmo obtener: camino básico #3

Caso de prueba para el camino básico #4 (1-2-3-4-5-6-7-8-9-11-12-14-15-4-16-17)	
Descripción	Debe determinar con precisión la cuenta que se mostrará al usuario para que pueda ser actualizada o consultada.
Condición de ejecución	El código de la cuenta ampliada no puede estar vacío y la cuenta asociada a ese código tiene que tener más de una persona asociada, cada una con distintos tipos de identificación y categoría, y que al menos dos de esas personas sean del mismo país.
Entrada	codCuentaAmpliada = 01121020013002, e = new NError()
Resultado esperado	Se obtiene la Cuenta banco esperada satisfactoriamente.
Resultado	El resultado obtenido fue correcto.

Tabla 6 Caso de prueba para el algoritmo obtener: camino básico #4

Caso de prueba para el camino básico #5 (1-2-3-4-5-6-8-9-10-11-12-14-15-4-16-17)	
Descripción	Debe determinar con precisión la cuenta que se mostrará al usuario para que pueda ser actualizada o consultada.
Condición de ejecución	El código de la cuenta ampliada no puede estar vacío y la cuenta asociada a ese código tiene que tener más de una persona asociada, cada una con distintos países y categoría, y que al menos dos de esas personas tengan el mismo tipo de identificación.
Entrada	codCuentaAmpliada = 00121220005000, e = new NError()
Resultado esperado	Se obtiene la Cuenta banco esperada satisfactoriamente.
Resultado	El resultado obtenido fue correcto.

Tabla 7 Caso de prueba para el algoritmo obtener: camino básico #5

Caso de prueba para el camino básico #6 (1-2-3-4-5-6-8-9-11-12-13-14-15-4-16-17)	
Descripción	Debe determinar con precisión la cuenta que se mostrará al usuario para que pueda ser actualizada o consultada.
Condición de ejecución	El código de la cuenta ampliada no puede estar vacío y la cuenta asociada a ese código tiene que tener más de una persona asociada, cada una con distintos países y tipo de identificación, y que al menos dos de esas personas tengan la misma categoría.
Entrada	codCuentaAmpliada = 40121020010000, e = new NError()
Resultado esperado	Se obtiene la Cuenta banco esperada satisfactoriamente.
Resultado	El resultado obtenido fue correcto.

Tabla 8 Caso de prueba para el algoritmo obtener: camino básico #6

Diseño de casos de prueba para caja negra

La prueba de Caja Negra se lleva a cabo sobre la interfaz del software, por lo que los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta, así como que la integridad de la información externa se mantiene. Además esta prueba examina algunos aspectos del modelo fundamentalmente del sistema sin

tener mucho en cuenta la estructura interna del software.

Estas pruebas no son una alternativa a las técnicas de prueba de la Caja Blanca, sino una visión adicional que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la Caja Blanca.

Los casos de prueba para caja negra demuestran que:

- ✓ Las funciones del software son operativas.
- ✓ Las entradas se aceptan de la forma adecuada produciendo el resultado correcto.
- ✓ La integridad de la información externa (por ejemplo archivos de datos) se mantiene.

Las técnicas de prueba no se hallan de forma aislada, sino como un conjunto integrado de acciones que combinadas permitirán verificar y evaluar la calidad de software. Entre las técnicas para desarrollar la prueba de caja negra encontramos:

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

De todas estas técnicas se aplicará la Partición de Equivalencia ya que es una de las más efectivas pues permite definir casos de prueba que descubran clases de errores, reduciendo el número de casos de prueba a desarrollar para demostrar que las funciones del software son operativas; las entradas se aceptan de forma adecuada y se producen salidas correctas.

Para verificar que la aplicación se comporta según los requerimientos establecidos por el cliente, se diseñan casos de pruebas usando el método de caja negra. En el anexo 5 de la versión del documento de Tesis en formato digital se especifica el caso de prueba de un requisito del módulo Cuenta banco.

Las pruebas realizadas a los módulos fueron satisfactorias desde el punto de vista interno y funcional, estas abarcaron requerimientos, funciones y lógica interna de cada módulo. Se aplicaron los métodos de caja negra y caja blanca para validar tanto la interfaz como el correcto funcionamiento interno del software. Combinar ambos enfoques permitió lograr mayor fiabilidad en el proceso de pruebas al diseñar los casos de prueba usando los dos tipos de técnicas a la vez.

En conclusión los tres módulos fueron probados y a su vez liberados por el departamento de calidad de la universidad (CALISOFT), además pasaron por las pruebas de aceptación por parte del cliente realizadas en el Banco Nacional de Cuba donde fueron aprobados. Por tanto los módulos Gestionar banco, Cuenta banco y Cuenta de otros conceptos se encuentran completamente disponibles para su utilización en el Banco Nacional de Cuba.

3.7 Conclusiones del capítulo

Al concluir el capítulo se obtuvo la solución implementada y a su vez validada tanto desde el punto de vista interno como funcional. Los módulos desarrollados permitirán llevar a cabo la gestión de los procesos relacionados con las cuentas banco, cuenta de otros conceptos y de los bancos en el BNC, a partir de la incorporación de los mismos al sistema Quarxo. Además que la utilización del framework Spring WebFlow brindó poderosos recursos que permitieron la integración de los medios de comunicación y de las personas autorizadas a los módulos, agilizándose el proceso de desarrollo de los mismos ahorrando código y tiempo del cronograma.

Conclusiones generales

Una vez culminado el trabajo y como resultado del mismo se logró alcanzar el objetivo general propuesto, tras cumplir todas las tareas trazadas a inicio de la investigación:

- A partir del análisis de los sistemas informáticos existentes que cuentan con los procesos de gestión de las cuentas y de los bancos, se demostró la necesidad e importancia de desarrollar un sistema informático que realice una gestión integral de los mismos, pues dichos sistemas son propietarios y por tanto a nuestro país se le dificulta el pagar altos precios por ellos.
- La implementación de la solución basada en tecnologías libres representa un gran aporte a la soberanía tecnológica por concepto de compra de licencias de software.
- La gestión de las cuentas banco y de los bancos a partir de la implementación de los módulos integrarán los medios de comunicación y las personas autorizadas, lográndose de esta forma la independencia del uso del papel para llevar el control de dicha información y que la gestión de esos procesos se realice de forma ágil y en menor tiempo.
- La realización de pruebas de unidad permitió validar la implementación de las funcionalidades desarrolladas corroborando la calidad de los módulos y demostrando además que los mismos están listos para su uso.
- La investigación realizada contribuye a la gestión de los procesos de cuentas y de los bancos en el BNC y constituye un aporte decisivo a la informatización y economía del país.

Recomendaciones

Tomando como base la investigación realizada y la experiencia acumulada durante el desarrollo del presente trabajo de diploma se recomienda:

- ✓ Continuar el estudio del tema con el objetivo de encontrar nuevas funcionalidades para futuras versiones de la aplicación.
- ✓ Extender el sistema hacia una nueva solución que sea genérica de manera tal que pueda ser utilizada por cualquier banco o entidad bancaria mundial.

Bibliografía

- [1]. **Ayaviri García, Daniel.** CONTABILIDAD BÁSICA Y DOCUMENTOS MERCANTILES. 1era. Argentina: N-DAG, pág. 10.
- [2]. **Gandarillas, Gonzalo J.** TEMAS DE CONTABILIDAD BASICA E INTERMEDIA. 4ta. s.l.: Educación y Cultura, pág. 4.
- [3]. *Instituto Americano de Contadores Públicos Certificados (AICPA).*
- [4]. Gestión y Administración. [En línea] [Citado el: 12 de Marzo de 2011.] <http://www.gestionyadministracion.com>
- [5]. Estructura y funciones del sistema financiero en Cuba. [En línea] [Citado el: 10 de Febrero de 2011.] www.estudios-economicos-cubanos.org/.../Estructura+y+Funciones+del+Sistema+Financiero+en+Cuba.pdf
- [6]. **Déniz Mayor, José Juan.** *Fundamentos de contabilidad financiera: teoría y práctica.* s.l.: Delta Publicaciones, 2008. pág. 83.
- [7]. **Urrestarazu C., Carolina.** *Modelos de Sistemas ERP: El Liderazgo del SAP.* 2003.
- [8]. Aspel-BANCO. [En línea] 2004. [Citado el: 10 de Marzo de 2011.] <http://www.aspel.com.mx/mx/productos/banco1.html>
- [9]. BYTE-Soluciones de Software para Industria Bancaria y Telecomunicaciones: Sistema bancario financiero Byte. http://www.bytesw.com/new/sistema_bancario_financierobyte.asp
- [10]. Revista BETSIME-La revista del empresario cubano. [En línea] [Citado el: 8 de Abril de 2011.] http://www.betsime.disaic.cu/secciones/tec_feb_02.htm
- [11]. **Lou Torrijos, Ricard.** Programación en castellano. [En línea] Diciembre 14, 2003. [Citado el: 20 de Abril de 2011.] <http://www.programacion.com/java/tutorial/patrones2/8>
- [12]. IEEE Std 1471-2000. [En línea] 2010. [Citado el: 24 de Abril de 2011.] <http://standards.ieee.org/findstds/standard/1471-2000.html>

- [13]. **Weitzenfeld, Alfredo.** *Ingeniería de software orientada a objetos con UML. Java e Internet.* s.l. : Cengage Learning Editores, 2005. pág. 678.
- [14]. **Jacobson, Ivar, Booch, Grady and Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2000.
15. Banco Central de la República Dominicana. [En línea] 2010. [Citado el: 10 de Noviembre de 2010.] <http://www.bancentral.gov.do/opiniones.asp?a=opiniones2006-2-28-1>
16. **SAP España.** *SAP: número uno en software ERP.* [En línea] 2007. [Citado el: 12 de Noviembre de 2010.] <http://www.sap.com/spain/solutions/business-suite/erp/index.epx>
17. Estudio de la Contabilidad General. [En línea] [Citado el: 16 de Noviembre de 2010.] http://bibliodoc.uci.cu/pdf/contabilidad_g.pdf
18. **Galíndez, Rodrigo.** Control de Versiones Usando Subversion. [En línea] [Citado el: 20 de Enero de 2011.] <http://www.rodrigogalindez.com/files/14.pdf>
19. **Romero Méndez, John Jairo .** Análisis Comparativo de las plataformas J2EE y .NET aplicado al desarrollo de servicios web. [En línea] Noviembre de 2008. [Citado el: 25 de Enero de 2011.] <http://es.scribd.com/doc/12759942/ANALISIS-COMPARATIVO-DE-LAS-PLATAFORMAS-J2EE-Y-NET-APLICADO-AL-DESARROLLO-DE-WEB-SERVICES>
20. **Larman, Craig.** *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* 2da.
21. **Rodríguez, María Chambi.** *Análisis y Diseño de Sistemas.* Universidad Salensiana de Bolivia Ingeniería de Sistemas. Bolivia : s.n., 2010.
22. **Gamma, Erich, y otros.** *Design Patterns. Elements of Reusable Object-Oriented Software.* 1st. s.l. : Addison-Wesley Pub Co, 1995.
23. [En línea] <http://www.consol.org.mx/2010/ponentes.php?idu=24>
24. [En línea] <http://www.slideshare.net/jhonatanalex/modelos-y-capas-de-la-ingenieria-de-software>

25. [En línea] [Citado el: 20 de Marzo de 2011.] <http://tikal.cifn.unam.mx/~jsegura/LCGII/java3.htm>
26. [En línea] [Citado el: 4 de Abril de 2011.] <http://es.scribd.com/doc/7844685/CONCEPTOS-DE-RUP>
27. [En línea] [Citado el: 10 de Marzo de 2011.] <http://bibdigital.epn.edu.ec/bitstream/15000/2314/1/CD-3058.pdf>
28. [En línea] [Citado el: 12 de Abril de 2011.]
<http://www.microsoft.com/spain/sql/productinfo/overview/default.msp>
29. [En línea] [Citado el: 26 de Enero de 2011.] <http://www.cuentas-bancarias.es/concepto-de-la-cuenta-bancaria.php>

Anexos

Anexo 1: Diseño del módulo Cuenta banco.

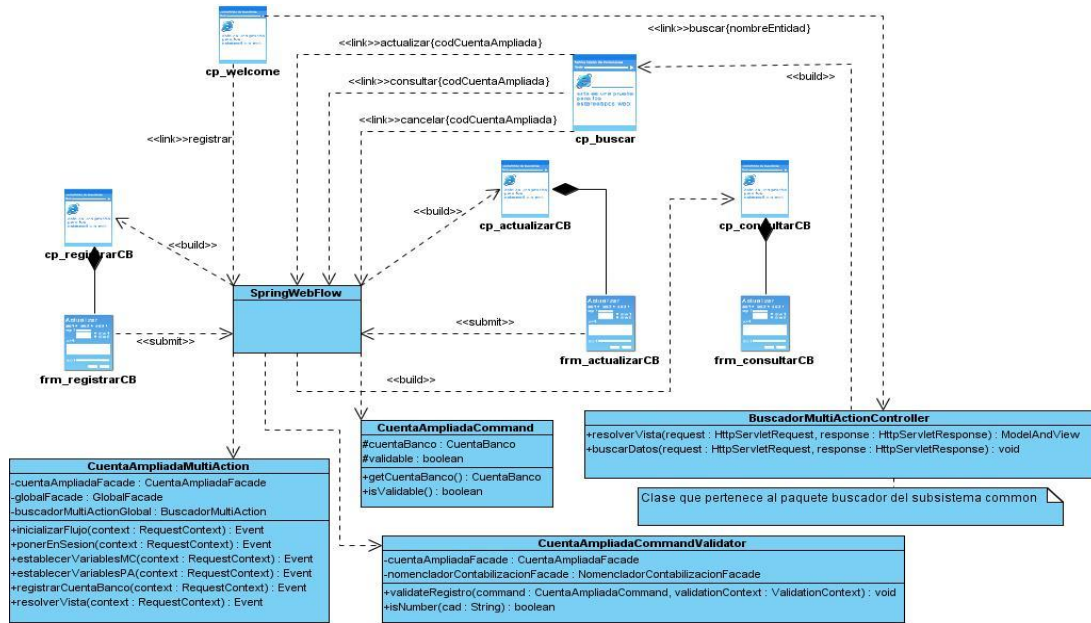


Ilustración 13 Diseño de la capa de Presentación del módulo Cuenta banco.

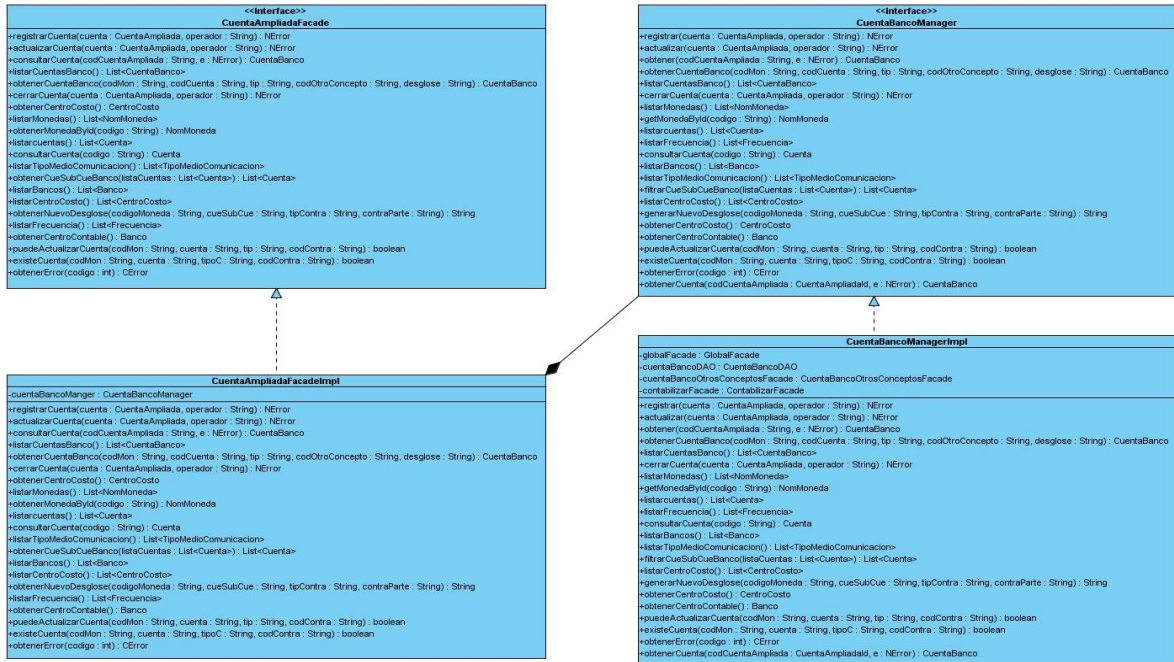


Ilustración 14 Diseño de la capa de Negocio del módulo Cuenta banco.

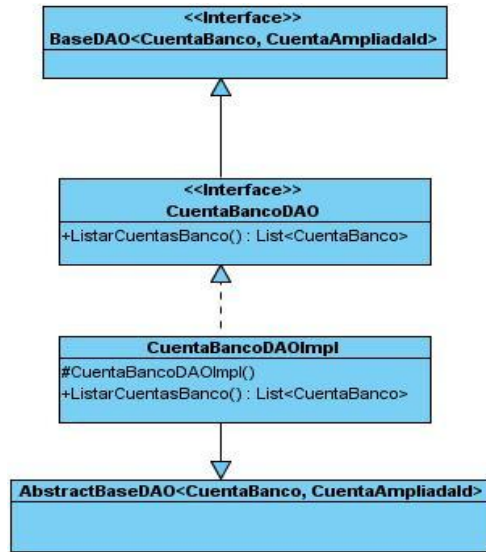


Ilustración 15 Diseño de la capa de Acceso a Datos del módulo Cuenta banco.

Anexo 4: Modelo de datos.

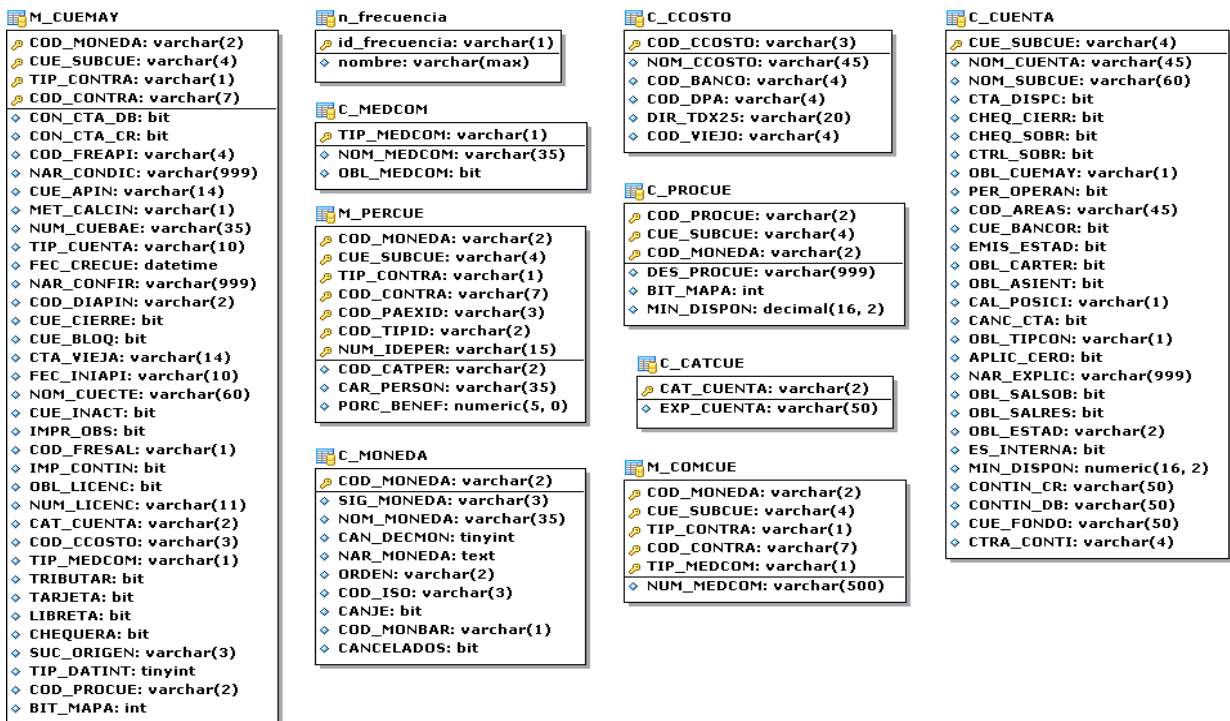


Ilustración 16 Modelo de datos del módulo Cuenta banco.

Glosario de términos

Entidad: organización administrativa, comercial, económica, productiva y de servicios de carácter estatal, cooperativa, privada o mixta, residentes en el territorio nacional; así como las organizaciones sociales y de masas del país.

Módulo: cada módulo es una parte del sistema, que se instala y funciona por separado, entrelazándose con otros módulos con los que intercambia información.

MS-DOS: Microsoft Disk Operating System, Sistema operativo de disco de Microsoft.

BCC: Banco Central de Cuba.

Cuenta/Subcuenta: Contiene el código de la cuenta en 4 dígitos, mostrando la categoría de la cuenta.

Software Propietario: Propietario significa que algún individuo o compañía retiene el derecho de autor exclusivo sobre una pieza de programación, al mismo tiempo que niega a otras personas el acceso al código fuente del programa y el derecho a copiarlo, modificarlo o estudiarlo. Sin embargo, el programa puede seguir siendo propietario aunque su código fuente se haya hecho público, si es que se mantienen restricciones sobre su uso, distribución o modificación.

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

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

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

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

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