Universidad de las Ciencias Informáticas

Título: Diseño e implementación del subsistema Vencimiento del sistema Quarxo para el Banco Nacional de Cuba.

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

Autores:

Daylen de la Caridad Barbán Reyes

Roberto Rodríguez Rodríguez

Tutor: Ing. Javier Martínez Muñoz.

Ciudad de La Habana, 2011.

Año 53 de la Revolución

DECLARACIÓN DE AUDITORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de junio del año 2011.

_____ _____

Daylen de La Caridad Barbán Reyes Roberto Rodríguez Rodríguez

_____

Ing. Javier Martínez Muñoz

DATOS DE CONTACTO

Ing. Javier Martínez Muñoz: Graduado en la Universidad de las Ciencias Informáticas. Instructor. Profesor de Matemática III y IV. 4 años de experiencia en el tema. 3 años de graduados. Analista principal de proyecto SAGEB.
Correo electrónico: jmmunoz@uci.cu

AGRADECIMIENTOS

De Daylen:

Agradezco a mi madre por todo el amor, apoyo y confianza que me ha brindado siempre; a mi tía por haber sido una segunda madre para mí, sin ella el camino hubiera sido más difícil; a mi abuela, que con su educación siempre supo sacar de mí lo mejor, a mi hermano y a mi tío Orlando por haber sido mis padres en todo momento en que necesite de uno.

Mis agradecimientos además para toda la gente linda que hizo de la universidad la mejor de las etapas de mi vida, en especial a mis amigos Yoan y Manuel y a mi mejor amiga Daneisy; a Arianna, Yelena, Yaslin y a las fantásticas niñas Roxi, Suamy y Mely.

Muchas gracias también a mi tutor, a mi compañero de tesis y al equipo de desarrollo del proyecto, por haber contribuido a mi superación profesional y personal; fue una linda experiencia haber trabajado con todos.

De Roberto

Agradezco a mi familia por todo el cariño y la confianza que han depositado siempre en mí, que me han servido de inspiración para superarme siempre un poquito más. A mi hermana Diagni y a Julio, que ha sido siempre como mi hermano mayor, por estar siempre ahí y darme su apoyo incondicional.

A mis padres, más que agradecer y dedicar, hago suyo todo los resultados obtenidos, por su infinito amor y cariño hacia mí y la forma tan certera en que me han sabido guiar por los caminos de la vida. A mis compañeros de carrera Yosmany, Yordanis, Manuel, Rolando y demás amigos con los que he compartido estos cinco años inolvidables.

Al magnífico equipo de trabajo del proyecto SAGEB, y en especial a aquellos que de una forma u otra han contribuido con la realización de este trabajo. Mis agradecimientos a mi compañera de tesis y mi tutor, por permitirme aprender tanto de ellos.

DEDICATORIA

De Daylen:

Dedico el presente Trabajo de Diploma a mi familia, en especial a mi mamita, a mi tía y a la personita más linda y que más alegría me proporciona: mi sobrino Carlos Alejandro. Los quiero mucho.

De Roberto

A mis padres, a mi hermana Diagnis y a mi sobri Isabel, que son lo más importante para mí.

RESUMEN

Con el crecimiento del desarrollo tecnológico y del dinamismo y complejidad de los procesos contables en las instituciones financieras, aumenta gradualmente la demanda de software bancario. Dentro de las instituciones que exige la creación de un nuevo sistema para realizar sus operaciones, se encuentra el Banco Nacional de Cuba (BNC) como consecuencia de las deficiencias que presenta la actual aplicación con la que se trabaja (SABIC[1]), para responder eficientemente a las necesidades actuales del mismo. Con el objetivo de satisfacer dicha necesidad, como parte de la modernización de los sistemas automatizados en los bancos cubanos se desarrolla el software Quarxo en el proyecto SAGEB[2] de la Universidad de Las Ciencias Informáticas (UCI).

Uno de los procesos más importantes que no está automatizado en el BNC con la organización y calidad requerida lo constituye la gestión de los vencimientos, haciendo que las transacciones involucradas se vuelvan lentas y trabajosas para los operarios.

El presente trabajo de diploma consiste en el Diseño e Implementación del Subsistema Vencimiento del sistema Quarxo. Con este objetivo y en aras de satisfacer las necesidades del cliente, el contenido del trabajo incluye la caracterización de la metodología, patrones de diseño, herramientas y tecnologías de la plataforma Java a utilizar por sus potencialidades como software libre para el desarrollo de aplicaciones web, la generación de los principales artefactos de salida de los flujos de trabajo del desarrollo de software involucrados y los elementos de validación de la solución propuesta.

Como resultado se obtienen los módulos Configuración y Gestionar Vencimiento, contribuyendo al logro de un manejo eficiente de los vencimientos en el Banco Nacional de Cuba.

Palabras claves:

Banco Nacional de Cuba, Desarrollo Tecnológico, Diseño, Eficiencia, Implementación, Modernización, Vencimiento.

[1] Sistema Automatizado para la Banca Internacional de Comercio.
[2] Sistema Automatizado de Gestión Bancaria.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

A raíz de los cambios en la economía mundial y la globalización, la información se ha convertido en uno de los principales recursos que posee actualmente toda empresa u organismo, haciendo necesaria su correcta y eficaz manipulación. Hoy está demostrado que las ventajas del manejo de la misma a través de sistemas computarizados, son significativamente superiores a las de dicho procedimiento realizado de forma manual. Estos sistemas automatizan los procesos operativos y suministran los datos necesarios para la toma de decisiones, contribuyendo al incremento de su capacidad de organización y eficiencia en sus procesos. Por esta razón se han ido incorporando los avances de la tecnología informática en función de brindar las herramientas necesarias para la implementación de sistemas de información confiables y eficientes.

Cuba no está exenta de esta realidad; como parte del proceso de informatización de la sociedad cubana se propone la modernización de los sistemas automatizados en las diferentes entidades o instituciones, haciendo uso de las tecnologías informáticas disponibles en el país. En este proceso los bancos constituyen una de las que se han priorizado, con el objetivo de enfrentar con éxito las transformaciones introducidas en el sistema bancario cubano en los últimos años.

Problemática

Formando parte del sistema financiero cubano se encuentra el Banco Nacional de Cuba, que ejerce funciones propias de la banca universal y atiende la deuda externa del Estado y del propio banco con acreedores extranjeros.

Actualmente esta entidad realiza gran parte de sus operaciones a través del SABIC, el cual está diseñado y desarrollado para satisfacer las necesidades de procesamiento de datos de bancos e instituciones no bancarias. Este sistema está centrado puramente en la contabilidad y no responde a todas las funcionalidades que requiere el BNC. Con el propósito de superar las deficiencias del SABIC y como parte del proceso de modernización de los sistemas informáticos en los bancos cubanos, en la UCI el proyecto SAGEB desarrolla un nuevo software llamado Quarxo, para responder con la calidad y rapidez requerida a la complejidad actual de los procesos que se llevan a cabo en el BNC. De ellos uno de los más importantes que no es gestionado eficientemente por el SABIC, por inconvenientes de tiempo de

realización del proceso, cantidad de operarios involucrados y de errores cometidos por los mismos, lo constituye el manejo de los vencimientos. Por esta razón es un requisito a tener en cuenta para la implementación de Quarxo.

De acuerdo al estudio de la problemática realizado se puede definir como ***problema a resolver*** la siguiente interrogante: ¿Cómo mejorar la gestión de los Vencimientos en el Banco Nacional de Cuba? Con vista a solucionarlo la realidad objetiva a estudiar (***objeto de estudio***) es la gestión de los vencimientos en los sistemas informáticos contables, y el ***objetivo general***, realizar el diseño y la implementación del subsistema Vencimiento del sistema Quarxo para el BNC. Atendiendo a situaciones particulares que constituyen parte del propósito general, habitualmente es recomendable plantear ***objetivos específicos*** de acuerdo con el problema a resolver. Fueron definidos los siguientes:

- ✓ Fundamentar la investigación mediante la elaboración del Marco Teórico.
- ✓ Realizar el diseño y la implementación del subsistema Vencimiento.
- ✓ Validar el diseño y la implementación del subsistema Vencimiento.

Como relación entre el problema, los objetivos y el objeto de estudio, limitando del mismo las partes que permitan desarrollar el proceso investigativo con que se alcanza el objetivo, surge como ***campo de acción*** la gestión de los vencimientos en el Banco Nacional de Cuba.

La ***idea a defender*** con la realización del presente trabajo es la siguiente: la implementación del subsistema Vencimiento permitirá mejorar el proceso de gestión de los vencimientos en el Banco Nacional de Cuba.

Para facilitar las vías para dar cumplimiento a los objetivos propuestos se definió el siguiente conjunto de ***tareas a cumplir***:

- ✓ Caracterización de las tecnologías, patrones y la arquitectura definida en el proyecto SAGEB.
- ✓ Valoración de los sistemas informáticos contables nacionales e internacionales que involucran la gestión de los vencimientos.
- ✓ Modelación del diagrama de clases del diseño del subsistema Vencimiento.
- ✓ Modelación de los diagramas de secuencia de cada funcionalidad del subsistema Vencimiento.
- ✓ Validación del diseño del subsistema Vencimiento.

- ✓ Obtención del modelo de datos del subsistema Vencimiento.
- ✓ Implementación de las funcionalidades del subsistema Vencimiento.
- ✓ Obtención del modelo de componentes del subsistema Vencimiento.
- ✓ Validación de la implementación del subsistema Vencimiento.

Con la realización de las mismas, el ***resultado esperado*** es la obtención del subsistema Vencimiento del sistema Quarxo.

ESTRUCTURA DEL DOCUMENTO

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En este capítulo se expone la fundamentación teórica del trabajo incluyendo conceptos fundamentales, estado del arte y una caracterización de las metodologías, tecnologías, lenguaje de programación y herramientas propuestas para darle solución al problema planteado.

CAPÍTULO 2: DISEÑO Y ARQUITECTURA DEL SUBSISTEMA VENCIMIENTO

Este capítulo incluye una descripción de la arquitectura de software utilizada por el proyecto SAGEB y los principales artefactos de salida del diseño del subsistema Vencimiento: diagrama de clases, diagramas de secuencia, modelo de paquetes y el modelo de datos, a partir de los requisitos funcionales capturados, así como los elementos de validación del diseño propuesto.

CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DEL SUBSISTEMA VENCIMIENTO

Este capítulo contiene como elementos fundamentales de la implementación del subsistema Vencimiento, la descripción de las clases definidas y sus funcionalidades y la explicación de la utilización de Spring WebFlow Framework, el diagrama de interacción de los componentes de la aplicación con el subsistema en cuestión, el modelo de despliegue del sistema y los elementos de validación de la implementación de Vencimiento.

CAPITULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

Las bases teóricas constituyen el corazón del trabajo en una investigación. Sin una buena base teórica todo instrumento seleccionado o técnica empleada en el estudio para el diseño e implementación de la solución a un problema determinado, carecerá de validez. Este capítulo contiene los elementos de la fundamentación teórica de este trabajo, dígase conceptos fundamentales relacionados con el proceso de negocio así como una panorámica sobre la informatización del mismo tanto a nivel internacional como nacional. Además se realiza una caracterización de las metodologías, técnicas, lenguaje de programación y herramientas que se proponen para el desarrollo de la solución.

1.2. Conceptos fundamentales

Banco

Los bancos actúan de intermediarios entre la oferta y la demanda de recursos financieros. Por esta razón su actividad influye en gran medida en la economía de un país y la vida de la sociedad. Se puede definir el banco como “una empresa constituida bajo la forma asociativa, cuya actividad se dirige a facilitar las operaciones de pago, negociar con valores y colectar capitales ociosos dándoles colocación útil. El banco moderno tiene que cumplir tres grandes funciones: la Intermediación de Crédito, la Intermediación de los pagos y la Administración de los capitales”. **(1)**

Sistema financiero cubano

El sistema financiero cubano estaba integrado en 1994 por el Banco Nacional de Cuba, el Banco Popular de Ahorro, el Banco Financiero Internacional y el Banco Internacional de Comercio, entre otros. Conjuntamente con la introducción de moderna tecnología de procesamiento de datos y la superación profesional, se llevó a cabo la ampliación de las atribuciones de los bancos existentes y la creación de nuevos bancos y entidades financieras no bancarias. De esta manera hoy el sistema financiero cubano está integrado por varias instituciones entre las que se encuentran:

- El Banco Central de Cuba (BCC) como órgano rector del sistema bancario financiero, cuyas funciones consisten en emitir la moneda nacional y mantener su estabilidad, proponer e implementar la política monetaria del país y supervisar las entidades integrantes del mismo.
- El Banco Nacional de Cuba (BNC) que existe con el carácter de banco comercial, autorizado a ejercer funciones inherentes a la banca universal.

Banco Nacional de Cuba

El Banco Nacional de Cuba fue creado como Banco Central del Estado con facultad orgánica, personalidad jurídica independiente y patrimonio propio. Al inicio se encargaba de centralizar las reservas monetarias, vigilar y regular el crédito, además de actuar como consejero económico y órgano emisor del Estado.

A partir de 1960 el BNC comenzó a realizar funciones de instituciones financieras que fueron desintegradas, quedando como único organismo financiero del país. Con la reorganización del Sistema Bancario Nacional se crea el BCC manteniéndose el primero con su carácter de banco estatal y comercial.

Entre las principales funciones, atribuciones y deberes del Banco Nacional de Cuba se encuentran: actuar como funcionario corresponsal de bancos extranjeros; obtener y otorgar créditos; mantener el registro, control, servicio y atención de la deuda externa de Cuba y realizar todo tipo de operaciones y negocios de intermediación financiera. Mantiene en sus registros contables la deuda externa del país, la oficial, la bancaria y la de proveedores e instrumenta las transacciones que se derivan de la renegociación de la deuda. **(2)**

Contabilidad bancaria

La contabilidad bancaria es la actividad de control de la información de todo el dinero que circula en una entidad bancaria. Básicamente se ocupa de la capacitación, la medición y la valoración de todos aquellos elementos financieros, que circulan internamente en un banco, con el propósito de suministrar a los gerentes la información necesaria para la toma decisiones. **(3)**

Asiento contable

El asiento contable es donde se refleja los hechos contables que son aquellos sucesos que hacen variar el balance o las cuentas de resultados, debiendo ser registrados para su posterior cómputo. Todas las variaciones contables han de ser medibles en unidades monetarias pues de lo contrario no se podrían contabilizar. **(4)**

Vencimiento

Un vencimiento es un asiento contable que representa un compromiso u obligación de pago o cobro que se hace exigible en una fecha futura. Se registra a través de transacciones en un libro contable llamado diario donde permanece pendiente de alguna operación.

Calendario

Un calendario engloba varias formas de pago que contendrán los vencimientos, representando un acuerdo entre las partes para establecer cronogramas de compromisos de pagos. Los vencimientos que contiene pueden ser de principal o de intereses ordinarios; el principal representa el monto inicial concedido o negociado; y por los intereses ordinarios debe entenderse “el rédito que produce o debe producir el principal; es decir, es el precio pagado por su uso durante un tiempo determinado de manera que su naturaleza jurídica consiste en la obtención de una cantidad como ganancia”. **(5)**

Los montos de principal y de interés ordinario se pueden pagar o cobrar parcialmente, trayendo consigo que se pueda contabilizar en varios vencimientos en fechas establecidas en el convenio.

Interés moratorio

El interés moratorio consiste en la sanción que debe imponerse por la entrega tardía del dinero de acuerdo con lo pactado en el acto contractual en el que quedó plasmado el préstamo o negociación. Es decir, “es la sanción que se le aplica al deudor por su incumplimiento imponiéndole una carga por su mora, que generalmente es una cantidad en numerario.” **(5)**

Interés acumulado

El interés acumulado surge a partir de un convenio entre las partes involucradas, donde se llega al acuerdo de acumular en una cuenta que pondrá el deudor a disposición del acreedor los intereses ordinarios fijados como parte del préstamo o negociación. De esta manera el interés moratorio no cuenta.

Capitalización de intereses

La capitalización de intereses es el proceso mediante el cual el interés no cobrado se agrega al principal no reembolsado en la fecha de pago o de vencimiento de un préstamo o negociación. Incluye además los intereses no pagados que son refinanciados o renovados mediante un nuevo crédito bancario. **(6)**

Pago por exportación

Representa un tipo de pago de vencimiento en el que las partes llegan al acuerdo de compensar el monto a pagar a través de exportaciones de producto.

1.3. Informatización de los procesos contables.

Los sistemas informáticos en la contabilidad posibilitan un manejo eficiente y rápido de grandes volúmenes de datos y reducen en gran medida la probabilidad de ocurrencia de errores que pueden cometer los operarios. A continuación se muestra una caracterización de algunos de los sistemas informáticos contables existentes que incluyen operaciones relacionadas con el proceso de gestión de los vencimientos.

1.3.1. Contaplus

Es uno de los programas de software de gestión más usados en España. La primera versión fue desarrollada a finales de 1985 por el Grupo SP (conocido como Sage SP). Constituye la gama de soluciones de gestión contable y financiera más utilizada por la Pequeña y Mediana empresa.

Este sistema permite la contabilización de las operaciones llevadas a cabo por una empresa a lo largo de varios ejercicios económicos, la obtención de resultados, la predefinición de asientos, la gestión informatizada del impuesto, la impresión de libros y documentos contables, de cuentas anuales y de modelos oficiales de impuesto y finalmente la gestión de vencimientos. **(7)**

Para realizar esta última funcionalidad posee un menú donde aparecen las opciones financieras más comunes en la gestión contable de la empresa. En él se encuentra la opción Vencimientos que permite la visualización de los vencimientos registrados atendiendo a criterios de búsqueda especificados por el usuario, la adición de uno nuevo y la impresión de información sobre los mismos, así como el control de cobros y pagos que se realizan sobre ellos en una fecha predeterminada, logrando obtenerse además unos listados de los recursos financieros que pueden resultar de gran interés para la toma de decisiones a corto y medio plazo.

1.3.2. Gestión MGD

Gestión MGD es un programa de contabilidad adaptado a la fiscalidad española en base a los requisitos establecidos por M.G.D. Su objetivo principal es llevar la contabilidad de la forma más eficaz posible intentando reducir el trabajo de entrada de datos, proporcionar la máxima información y organizar de forma sencilla los datos introducidos. Tiene la característica de ser un programa abierto a las aportaciones de los usuarios en cuanto al desarrollo del sistema, no limita el número de empresas a gestionar y es multiejercicio, sin embargo esta misma característica impide que sea un sistema seguro. Entre sus funcionalidades se encuentra la automatización de los procesos de cierre y apertura de ejercicio contable, dispone de un pequeño módulo de facturación y permite gestionar los vencimientos al introducirlos en las facturas, así como en los movimientos extra, y posteriormente confrontarlos con los pagos a través de tesorería. Este proceso incluye el acceso a los vencimientos existentes, así como la adición y procesamiento de los mismos.

Ambos sistemas ofrecen servicios para la contabilización de los vencimientos principalmente para el procesamiento de cobros y pagos de los mismos. Pero aun así los inconvenientes de costo en el caso del Contaplus y de seguridad en el caso del Gestión MGD, así como el hecho de que no fueron creados específicamente para bancos, además del carácter atípico de los procesos de negocio que se llevan a cabo en el BNC, son elementos determinantes para descartarlos como opciones viables que puedan dar solución al problema planteado. La realidad económica y política de Cuba generalmente exige la búsqueda de soluciones internas a problemas similares. Por esta razón el desarrollo del software cubano para automatizar los procesos en las instituciones bancarias aumenta gradualmente.

1.3.3. SABIC

Como sistema informático contable desarrollado en Cuba se encuentra el SABIC, el cual gestiona la actividad bancaria del BNC y otras instituciones. Dicho sistema es modular, transaccional y permite la actualización inmediata de los ficheros contables y la contabilización multimoneda. No obstante presenta algunos problemas que conllevan a la necesidad del desarrollo de un nuevo software, tales como:

- ✓ Utiliza MS-DOS como sistema de explotación. Este es un sistema operativo antiguo y monotarea donde solo se puede ejecutar un programa a la vez, ralentizando y entorpeciendo el trabajo del operador.
- ✓ No tiene integración con los sistemas SLBTR[3] y SISCOM[4]: usados para la comunicación entre los bancos nacionales y con los bancos internacionales respectivamente, a través de servicios de mensajería.
- ✓ No ofrece el tratamiento requerido a la validación de los datos de entrada impidiendo que sean detectadas de forma temprana ciertas irregularidades.
- ✓ Mucha de la información necesaria para realizar las operaciones dentro de la entidad se llevan a cabo a través de tablas Excel.
- ✓ No genera reportes personalizados que puedan ser configurados por el usuario y que muestren información específica sobre la actividad de la entidad.
- ✓ Existe el inconveniente de que los informes deben confeccionarse manualmente por el usuario a través de programas ofimáticos para su posterior impresión, además de que la información almacenada por el sistema no es suficiente para la confección de los mismos.
- ✓ Existe gran probabilidad de que los usuarios cometan errores que disminuyan la productividad y aumente el consumo de recursos materiales. **(8)**

[3] Sistema de Liquidación Bruta en Tiempo Real. Permite la realización en tiempo real de transacciones u operaciones de pago entre las diferentes entidades, utilizando medios electrónicos de comunicación y teniendo como centro al BCC para monitorear el flujo de las transferencias interbancarias que se realicen.
[4] Servicio de transmisión de mensajes financieros.

Además de estas dificultades es importante destacar que no presenta la flexibilidad necesaria para responder eficientemente a la complejidad actual de los procesos que se llevan a cabo en el BNC. Muestra de ello lo constituye la manera en que hoy se procesan los vencimientos.

El SABIC trabaja con tres tipos de ficheros: Maestros Contables, Maestros de Control y Maestros de Referencia. Los maestros contables lo constituyen: el Histórico, el Mayor y el Diario. En el Diario es donde se encuentran todos los vencimientos esperando ser procesados a través de un pago, cobro, renegociación, capitalización, acumulación de intereses o recalendarización.

Actualmente dichas operaciones no se hacen de forma centralizada. Los operarios deben entrar los datos para crear uno a uno los asientos a contabilizar a través de una funcionalidad llamada General. Dicho proceso resulta lento y engorroso; y en él existe alta probabilidad de ocurrencia de errores. En el caso del pago, existe una funcionalidad que inicialmente brindaba facilidad en la automatización de la contabilización de los vencimientos, pero dejó de ser una opción fiable para realizar dicha transacción, como consecuencia del aumento del dinamismo y complejidad de los procesos internos. El hecho de que el sistema no esté integrado con SISCOM, hace que el operario deba realizar la contabilización por el sistema operativo MS-DOS y luego reiniciar la máquina e iniciarla por el sistema operativo Windows, para enviar o recibir mensajes SWIFT. Esto trae como consecuencia el deterioro gradual del funcionamiento de las máquinas, así como la disminución del rendimiento de los operadores.

De esta manera se evidencia la necesidad del desarrollo de un nuevo sistema que mejore la gestión de los vencimientos en cuanto a tiempo, cantidad de operarios necesarios para realizar las operaciones, probabilidad de errores, facilidad de uso y flexibilidad en cuanto a futuros cambios en el proceso. Con este propósito se crea el sistema Quarxo reutilizando los procedimientos almacenados del núcleo central del SABIC sobre Windows, integrado con los sistemas SLBTR y SISCOM.

1.4. Metodología de Desarrollo RUP[5]

"Una metodología impone un proceso disciplinado sobre el desarrollo de software con el objetivo de hacerlo predecible y eficiente" **(9)**. RUP es la metodología empleada para el desarrollo del sistema

[5] ***R**ational **U**nified **P**rocess, en español Proceso Unificado de Rational, es un proceso de desarrollo de software.*

Quarxo. Entre sus principales características se encuentran las siguientes: soporta un enfoque iterativo permitiendo detectar tempranamente desajustes e inconsistencias entre los requerimientos, el diseño e implementación del sistema, con la opinión del usuario de manera que el equipo de desarrollo se mantiene enfocado en producir resultados que satisfagan las necesidades del mismo; permite capturar los requerimientos funcionales y asegura que direccionan el diseño, la implementación y las pruebas del sistema; y describe como diseñar una arquitectura flexible, robusta y con gran poder de adaptación a los cambios. RUP divide el proceso de desarrollo de software en cuatro fases dentro de las cuales se realizan tantas iteraciones como sean necesarias en dependencia de las características del proyecto. Para el desarrollo del presente trabajo es necesario apoyarse en las disciplinas de Análisis y Diseño, e Implementación.

Disciplina Análisis y Diseño

Genera la especificación de características operacionales que debe tener el software, transformando los requisitos en el diseño del futuro sistema. Además se desarrolla una arquitectura y se adapta el diseño de forma tal que sea consistente con el entorno de implementación.

Implementación

En esta disciplina se implementan las clases y objetos en ficheros fuente, binarios y ejecutables organizados en paquetes de implementación. Su principal objetivo es dar respuesta a los requisitos funcionales expuestos en el diseño, obteniéndose como resultado final el propio software.

1.5. Patrones arquitectónicos

Un patrón arquitectónico es un patrón de alto nivel que determina la arquitectura global de una aplicación, proporcionando un esquema genérico probado para solucionar un problema en específico. Proveen un vocabulario común y comprensible, facilitan la construcción de software con propiedades particulares y ayudan a construir arquitecturas heterogéneas y complejas **(10)**. Con este objetivo se utiliza para la definición de la arquitectura de la aplicación Quarxo el patrón ***MVC,*** entre los usados para sistemas interactivos.

MVC (Modelo Vista Controlador): Este patrón propone dividir la aplicación en tres capas: el Modelo, la Vista y el Controlador; el modelo es la representación de los datos del sistema; la vista se encarga de todo

lo relacionado con presentar la interfaz al usuario y el controlador es el encargado de escuchar los cambios en la vista y enviarlos al modelo, el cual remite a su vez los datos a la vista como un ciclo. El controlador es el que decide la vista y la información que se debe presentar. Al aplicarse este modelo se desacopla el modelo de las vistas y se hace a los sistemas flexibles y adaptables. **(11)**

1.6. Patrones de diseño

Posterior a la definición de la arquitectura de la aplicación, el diseño hace uso de varios patrones que constituyen la base para la búsqueda de soluciones a problemas que se presentan en situaciones comunes del diseño. Proponen una solución efectiva y probada para un problema general dentro del marco del diseño de software.

A continuación se describen un conjunto de patrones de diseño que son empleados en la aplicación, contribuyendo a su implementación organizada y eficiente.

Patrón de Acceso a Datos (DAO): Engloba todo el acceso a datos en una capa independiente, desacoplando la lógica de negocio de la lógica de acceso a datos de manera que se pueda cambiar la fuente de datos fácilmente. Reduce la complejidad de los objetos de negocio al abstraerlos de la implementación real de la comunicación con la fuente de datos.

Patrones de asignación de responsabilidades (GRAPS)

Patrón Experto: Se asigna una responsabilidad al experto en la información, o sea, se definen las clases y sus funcionalidades atendiendo a la información concreta que deben manejar. De este modo se obtiene un diseño con mayor cohesión y la información se mantiene encapsulada (disminución del acoplamiento).

Patrón Creador: Guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento.

Patrón Controller: Se asigna la responsabilidad de controlar el flujo de eventos del sistema a clases específicas que tengan una posición intermediaria entre la interfaz de usuario y las clases donde reside la lógica de la aplicación, facilitando la centralización de actividades.

Patrón Bajo Acoplamiento: Propone mantener pocas dependencias entre las clases, logrando que el código sea más fácil de entender, mantener y reutilizar.

Patrón Alta Cohesión: Propone que cada elemento del diseño debe realizar una labor única dentro del sistema, logrando un alto grado de funcionalidad combinada con una reducida cantidad de operaciones, trayendo como ventaja que se simplifique el mantenimiento y que aumente la capacidad de reutilización.

Patrones estructurales (GoF)

Facade (Fachada): El objetivo de este patrón es proveer un intermediario entre dos grupos de objetos, disminuyendo el número de objetos con los que trabaja un cliente y simplificando el acceso de éste a un conjunto de objetos relacionados. Se promueve un acoplamiento débil entre el subsistema y sus clientes, eliminándose o reduciéndose las dependencias.

Composite View (Vistas Compuestas): Un objeto vista que está compuesto de otros objetos vista. Por ejemplo una página JSP que incluye otras páginas JSP y HTML usando la directiva include. La aplicación de este patrón hace que la representación de vistas sea más manejable gestionando los diferentes elementos de una página por medio de una plantilla.

1.7. Ambiente de desarrollo de Quarxo

Para lograr reducir al mínimo el tiempo de desarrollo y lograr un producto con la calidad requerida, se hace necesario seleccionar correctamente las herramientas y tecnologías a utilizar teniendo en cuenta el contexto donde se aplicará el software.

Para el desarrollo del sistema Quarxo se definen un conjunto de tecnologías, herramientas, lenguajes y patrones a utilizar, en base a la disponibilidad e idoneidad de los mismos considerando las características del proyecto y el entorno de despliegue. A continuación se expone una breve caracterización de cada uno de estos elementos.

1.7.1. Lenguajes y herramientas

UML

Se utiliza Unified Modeling Language (UML) para el modelado de los procesos de negocio y funciones del sistema. Incluye las clases, esquemas de base de datos y componentes reutilizables de software en un lenguaje determinado y soporta el paradigma orientado a objetos.

Java

Para el desarrollo del sistema Quarxo se utiliza como lenguaje de programación Java, caracterizado por ser orientado a objetos, multiplataforma y de código abierto.

Javascript

Es un lenguaje que no requiere compilación y se utiliza comúnmente para la construcción de páginas web en combinación con el XHTML. Se caracteriza por ser un lenguaje que responde a eventos generados por el usuario o por el navegador, es independiente de la plataforma necesitando solo de un navegador para ejecutar el código, permite un desarrollo rápido y es relativamente fácil de aprender. Es soportado por la mayoría de los navegadores como Internet Explorer, Netscape y Mozilla Firefox. **(12)**

Plataforma JEE

Como plataforma de desarrollo se decide utilizar Java Enterprise Edition (JEE). La misma define un estándar para el desarrollo de aplicaciones empresariales basándolas en componentes modulares y estandarizados, proporcionando un conjunto completo de servicios a estos componentes, y manejando muchas de las funciones de la aplicación de forma automática sin necesidad de una programación compleja. **(13)**

Contenedor Web (Tomcat 6.0)

Como servidor de aplicaciones se determina utilizar el Apache Tomcat en su versión 6.0, el cual es un contenedor de servlets que implementa las especificaciones de JavaServer Page (JSP) y funciona como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. **(14)**

IDE de desarrollo (Eclipse 3.3)

Se selecciona como entorno de desarrollo integrado Eclipse en su versión 3.3. Esta herramienta es multiplataforma, de código abierto y presenta una arquitectura de plugins que lo hace ser bastante flexible y configurable.

Visual Paradigm 3.4

Como herramienta de modelado se decide emplear el Visual Paradigm en su versión 3.4. Esta herramienta soporta el ciclo de vida completo de RUP, permite generar casi cualquier tipo de documento o diagrama. Para su selección se tuvo en cuenta que su licencia fue comprada por la universidad.

Erwin

Para el modelado de la base de datos se utiliza el Erwin en su su versión 7.5 que permite diseñar, generar y mantener aplicaciones de bases de datos, desde un modelo lógico hasta un modelo físico.

Control de versiones (SubVersion)

Para el control de las versiones del sistema se utiliza el SubVersion 1.6.6, que está desarrollado sobre tecnología Open Source y se integra con Eclipse mediante el plugin SubEclipse. Permite recuperar versiones antiguas de los datos registrados y examinar el historial de los mismos, manejando la información de ficheros y directorios a través del tiempo, la cual radica en un árbol de ficheros en un repositorio central. **(15)**

Base de Datos (SQL)

SQL Server es un servidor de base de datos basado en el modelo relacional. Se caracteriza por brindar soporte para transacciones y procedimientos almacenados y posee como lenguajes de consulta al SQL y al T-SQL (en inglés: Transact-SQL). Requiere para su funcionamiento del sistema operativo Microsoft Windows, atentando contra la premisa de desarrollar las aplicaciones cubanas sobre software libre. No obstante, se utiliza el SQLServer 2005 para la gestión de la base de datos por petición del BNC y el BCC.

Esta decisión tuvo el propósito de reducir el tiempo de desarrollo de la aplicación a partir de la reutilización del núcleo del SABIC, que incluye una estructura base y un conjunto de procedimientos almacenados.

1.7.2. Frameworks

1.7.2.1. Spring

Se emplea para el desarrollo de la aplicación el Spring Framework, el cual contiene un conjunto de librerías de clases que brindan una estructura conceptual y tecnológica para el desarrollo de aplicaciones de la plataforma JEE. Permite crear soluciones bien documentadas, seguras y robustas y ofrece gran libertad a los desarrolladores en java. Además en su implementación tiene en cuenta las tres capas arquitectónicas: acceso a datos, negocio y presentación. **(16)**

Spring MVC

Dentro del Spring Framework se encuentra el Spring MVC, utilizado para atender las peticiones web simples con navegación lineal a través de clases "Controller". Entre las que se encuentran las clases ***SimpleFormController*** y ***MultiActionController*** como las más usadas en el desarrollo de la aplicación Quarxo. La primera se utiliza cuando se desea trabajar con un conjunto de parámetros recibidos desde una petición que se pueden mapear a un objeto específico. Permite que el formulario que contiene dichos parámetros se cargue inicialmente y genera la página donde serán procesados los datos. La segunda atiende tantas peticiones se necesitan, generalmente de un negocio determinado.

Spring WebFlow

Spring WebFlow es un módulo de Spring que permite operar la navegación de una aplicación Web, para lo cual proporciona la definición de un lenguaje declarativo de flujos. Surge como solución a las limitaciones del flujo de página de los frameworks MVC clásicos. Se utiliza cuando el caso de uso que se desarrolla presenta un flujo complejo y/o no lineal. La configuración del desarrollo del flujo es especificada a través de un archivo de configuración XML, permitiendo establecer reglas de navegación complejas de una manera sencilla. Un flujo en Spring WebFlow maneja la conversación (vacío entre Sesión y Petición) completa, de inicio a fin, punto en el que limpia la memoria automáticamente liberando al usuario de la responsabilidad de borrar los recursos en memoria que no estén siendo utilizados. **(16)**

Spring DAO

Spring DAO es un módulo perteneciente al framework Spring. Posee algunas librerías de clases para trabajar con base de datos a través del API[6] de Java JDBC[7] y brinda soporte para invocar procedimientos y funciones almacenadas. Ofrece además una capa de excepciones para el manejo de los errores emitidos por los servidores de base de datos. Ayuda a mantener el código simple y evita que sea muy repetitivo, además minimiza los errores al intentar cerrar la conexión con algunas bases de datos. **(16)**

Spring ORM

Spring ORM (Object Relation Mapping) es un módulo de Spring que permite su integración con los frameworks ORM más populares en el mercado. Dicha integración proporciona mayor seguridad, facilidad y eficiencia en el manejo de sesiones, recursos, transacciones ORM, excepciones y pruebas. Es utilizada en el proyecto para lograr la integración de Spring con el framework ORM Hibernate. **(16)**

Spring Transaction

Para asegurar que las funcionalidades de la aplicación se ejecuten de forma transaccional se decide utilizar Spring Transaction. Además de soportar el manejo de las transacciones para varios gestores de base de datos y recursos compartidos a través del API JTA[8], Spring Transaction permite definirlas en forma de anotaciones o basándose en declaraciones AOP[9] en el código de la aplicación, y exponerlas en los ficheros declarativos. Esta última forma es la más recomendaba en la mayoría de los casos, debido a que la administración de las transacciones requiere menos código y por tanto no depende de ninguna API. Por esta razón fue la utilizada para la implementación del sistema Quarxo.

Spring Security

Spring Security es un sub-proyecto de Spring que permite gestionar completamente la seguridad de aplicaciones Java. Ofrece servicios de identificación de usuarios y acceso a los recursos sin necesidad de

[6] Application Programming Interface.
[7] Java Data Base Connectivity
[8] Java Transaction API.
[9] Aspect Oriented Programming.

añadir código; proporciona asignación de permisos a usuarios o grupos de usuarios; y facilita, al separar claramente la seguridad de un sistema de su lógica de negocio, la modificación o adición de seguridad a aplicaciones existentes. Se utiliza con el propósito de garantizar la seguridad del sistema Quarxo.

1.7.2.2. Hibernate

Para el manejo del acceso a datos se decidió usar el Hibernate framework. Una de sus características principales es que abstrae el trabajo con la base de datos, soportando la manipulación de los datos persistentes objetualmente a través de ficheros que mapean clases Java contra tablas. Permite transacciones, asociaciones, polimorfismo, herencia, carga mediante referencia (carga *lazy,* consiste en cargar sólo la referencia de donde se encuentran los datos), persistencia transitiva y estrategias de fetching. Permite la ejecución de consultas SQL (Standard Query Language) y además posee un potente lenguaje de consulta de datos llamado HQL (Hibernate Query Language), que permite realizar consultas basándose en los objetos del negocio y no en las tablas de la base de datos. Se hace uso de Hibernate Framework en su versión 3.5.

1.7.2.3. Dojo toolkit

Se determinó usar las librerías de Dojo toolkit en su versión 1.3 para facilitar el trabajo en la capa de presentación por la parte del cliente. Contiene APIs y widgets (controles), incluye un sistema de empaquetado, efectos visuales de la interfaz de usuario y abstracción de eventos, además de la integración con tecnologías AJAX[10] permitiendo realizar peticiones asincrónicas al servidor. Dojo unifica estándares de codificación resolviendo los problemas de compatibilidad entre los navegadores. Incluye un gran cúmulo de componentes visuales basados en XHTML[11], CSS y Javascript para enriquecer la interfaz de usuario. **(12)**

[10] Ajax (Asynchronous JavaScript And XML) es una técnica de desarrollo web para crear aplicaciones interactivas.
[11] XHTML (Extensible Hypertext Markup Language) es una reformulación del HTML que es compatible con XML.

1.8. Conclusiones Parciales

En el Banco Nacional de Cuba existe la necesidad de un sistema automatizado que mejore la gestión de los vencimientos, de acuerdo al funcionamiento y dinamismo actual de dicho proceso, propósito que no es logrado con eficiencia por el sistema con el que se ha trabajado hasta ahora (SABIC).

Para la búsqueda de una solución a los problemas actuales del BNC, fue necesario realizar un estudio de los sistemas contables existentes tanto a nivel nacional como internacional que incluyen el manejo de los vencimientos dentro de sus funcionalidades, concluyéndose que los mismos no permiten solucionar el problema a resolver, determinándose el desarrollo de un nuevo sistema como la mejor alternativa. Además se realizó una caracterización de las herramientas, metodología, patrones de diseño a utilizar y lenguajes definidos por la dirección del proyecto para la implementación del nuevo sistema, sobre la premisa del desarrollo de aplicaciones cubanas utilizando software libre y teniendo en cuenta las peticiones del cliente y el éxito de su utilización en aplicaciones de alcance y complejidad similar a Quarxo, sentando las bases teóricas necesarias para comenzar el diseño de la solución.

CAPÍTULO 2: ARQUITECTURA Y DISEÑO DEL SUBSISTEMA VENCIMIENTO.

2.1. Introducción

Este capítulo incluye una caracterización de la arquitectura definida en el proyecto y los principales artefactos del Diseño del subsistema Vencimiento: diagrama de clases, diagramas de secuencia, diagrama de paquetes y el modelo de datos; generados a partir de las funcionalidades identificadas.

2.2. Arquitectura base de Quarxo

La arquitectura es el resultado de acoplar elementos arquitectónicos de forma apropiada para satisfacer los requerimientos funcionales y no funcionales de un sistema. Para desarrollar un sistema de información no hace falta inventar una nueva arquitectura de software, por lo general se adopta una conocida en función de sus características, ventajas y desventajas. Entre las más universales se encuentra la Arquitectura de tres niveles, en la cual, como lo indica su nombre, la carga se divide en tres partes (o capas): una capa para la presentación (interfaz de usuario), otra para el negocio y otra para el almacenamiento de información (acceso a datos).

Para el desarrollo del sistema Quarxo del proyecto SAGEB se utiliza esta arquitectura, atendiendo al funcionamiento y complejidad de los procesos de negocio, incluyéndose además una capa transversal a las otras, que contiene los objetos del dominio. Con el objetivo de lograr flexibilidad y adaptabilidad en el sistema se realiza una estructuración en subsistemas, componentes y módulos, los cuales se definieron atendiendo a las funcionalidades identificadas en el levantamiento de requisitos.

Los módulos agrupan un conjunto de casos de uso relacionados con uno o más procesos bancarios y presentan la arquitectura por capas explicada anteriormente; los subsistemas contienen uno o varios módulos relacionados entre sí por los procesos que ejecutan, y/o componentes; éstos representan un grupo de funcionalidades comunes que serán utilizados por otros módulos del sistema y pueden comportarse como módulos, subsistemas o contenedores de funcionalidades de negocio, reutilizables e independientes **(17)**. A continuación se muestran unas figuras que evidencian la arquitectura que tienen los subsistemas y módulos en el sistema Quarxo, y seguidamente una explicación de la función y

estructura de las capas de la arquitectura del sistema, así como la composición de paquetes por cada una de ellas.

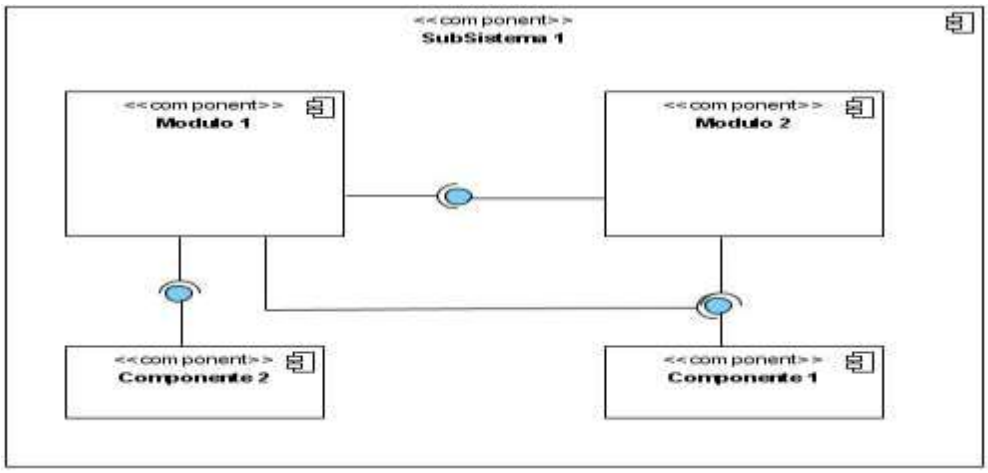

Ilustración 1: Arquitectura de un subsistema de Quarxo.

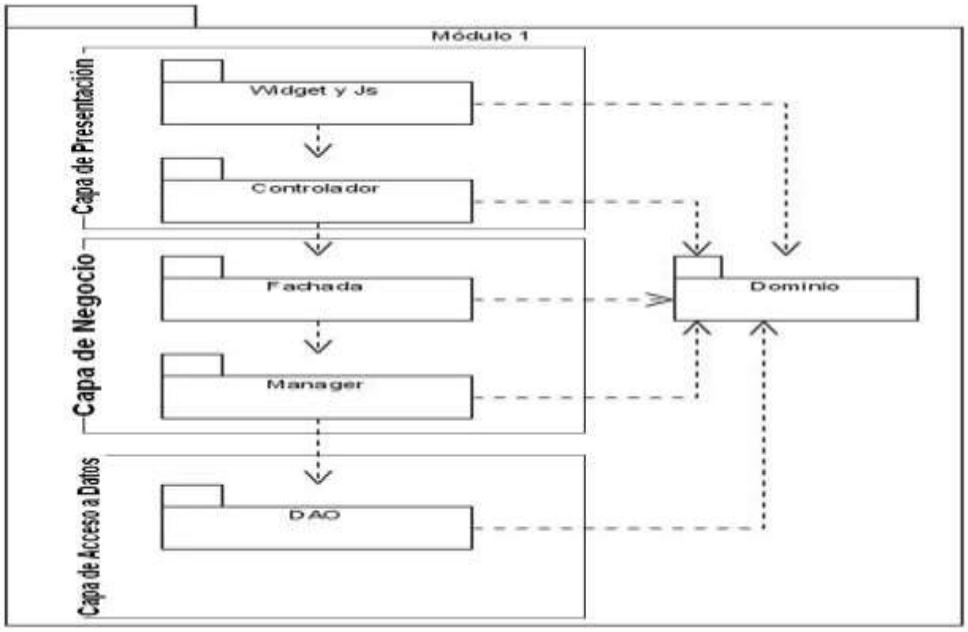

Ilustración 2: Arquitectura de un módulo de Quarxo.

Un paquete representa una agrupación de clases y otros paquetes. Las relaciones y organización entre ellos se evidencian a través de un diagrama de paquetes. Este diagrama forma parte del modelo de

diseño de la solución, pero teniendo en cuenta que dicha estructura fue definida como parte de la arquitectura y se comporta de igual manera para todos los módulos de la aplicación, se mostrará en este epígrafe la composición de paquetes de un módulo por capas.

2.2.1. Capa de acceso a datos

Esta capa contiene todas las operaciones relacionadas con la persistencia y acceso a información de la base de datos, a través de llamadas a funciones o procedimientos almacenados mediante Spring JDBC y de instrucciones básicas utilizando los frameworks Spring con Hibernate, cuya integración es posible mediante el uso de Spring ORM **(17)**.

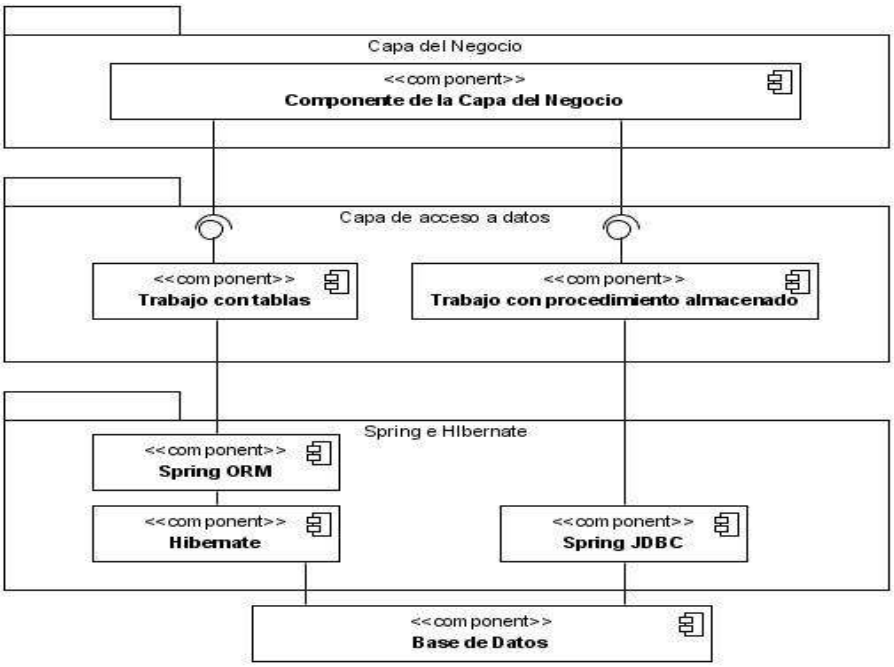

Ilustración 3: Arquitectura de la capa de acceso a datos.

Estructura de la capa de acceso a datos

La Capa de Acceso a Datos está compuesta por tres paquetes (ver **Ilustración 4**): ***dao***, ***dao.impl*** y ***dao.impl.map***. En el ***dao*** se encuentran las interfaces DAO y StoredProceduredDAO. Las primeras son utilizadas para el trabajo con la base de datos a través de las operaciones básicas y las segundas como vía para invocar a los procedimientos almacenados. En el paquete **dao.impl** están las clases que

implementan las interfaces del paquete dao, y en el paquete ***dao.impl.map*** se encuentran los ficheros de Hibernate que mapean los objetos de dominio con las tablas de la base de datos **(17)**.

Esta capa no depende de la de negocio pero si interactúa con ella a través de las interfaces que contiene.

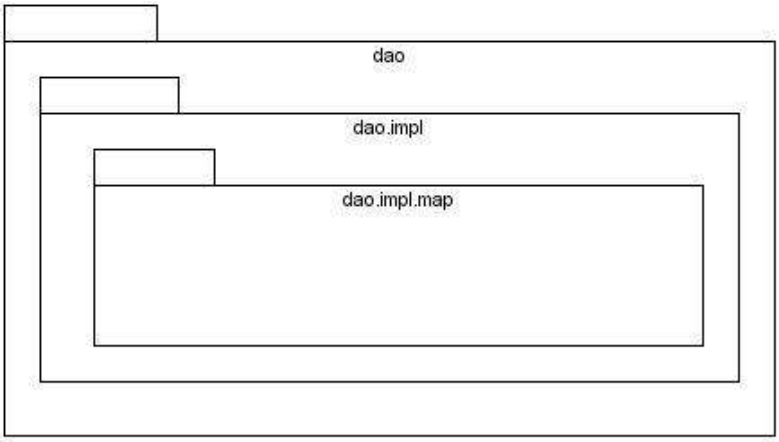

Ilustración 4: Estructura de paquetes de la capa de acceso datos.

2.2.2. Capa de negocio

En esta capa se encuentran todas las funcionalidades del sistema en diferentes niveles de transacción garantizando la consistencia en los datos persistentes. Son utilizados Spring MVC para la representación de las clases y sus relaciones, Spring AOP para ejecutar las mismas y Spring Security para aplicar seguridad a nivel de métodos.

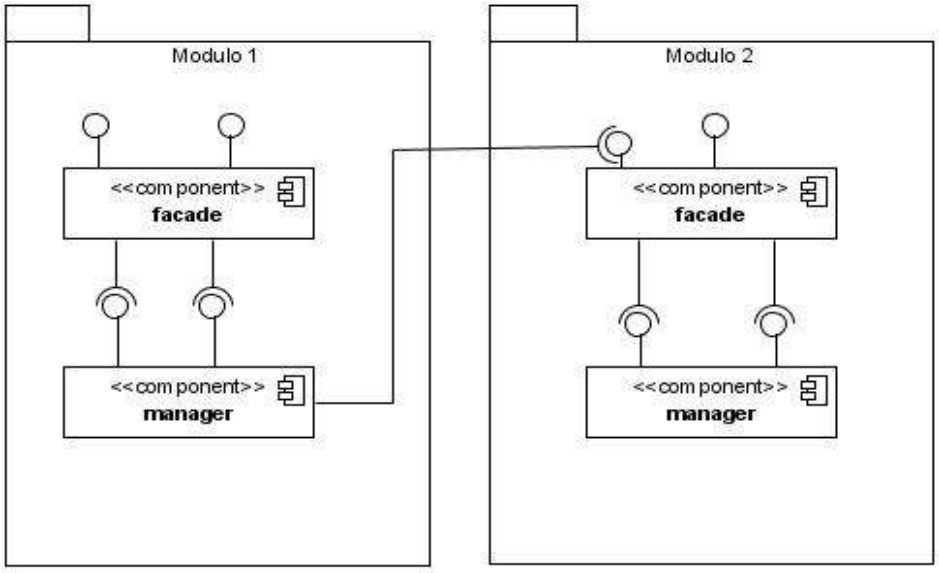

Ilustración 5: Arquitectura de la capa de negocio.

Estructura de la capa de negocio

La Capa del Negocio está dividida en dos subcapas principales: manager y facade. En la primera se encuentra la implementación del negocio del módulo correspondiente, accediendo en caso necesario a la Capa de Acceso a Datos para obtener o persistir información, a la de Dominio para generar los objetos correspondientes y a la subcapa facade de otros módulos para utilizar las funcionalidades que sean necesarias. Está representada por un paquete ***manager*** que contiene las interfaces con la declaración de las funcionalidades que representan el negocio del módulo correspondiente, y un paquete ***manager.impl*** con las clases que implementan dichas interfaces (ver **Ilustración 6**). La segunda constituye el punto de intercambio entre la capa de presentación y la de negocio, agrupando funcionalidades según su naturaleza que invocan los métodos de la subcapa manager del mismo módulo y que son llamadas desde la capa de presentación. Está representada por un paquete ***facade*** con las interfaces que declaran las funcionalidades y un paquete ***facade.impl*** con sus implementaciones (ver **Ilustración 6**).

Esta estructura permite un mayor control de cambios en la lógica de negocio, permitiendo que la necesidad de los mismos no conlleve a la realización de grandes y complejas variabilidades en la presentación **(17)**.

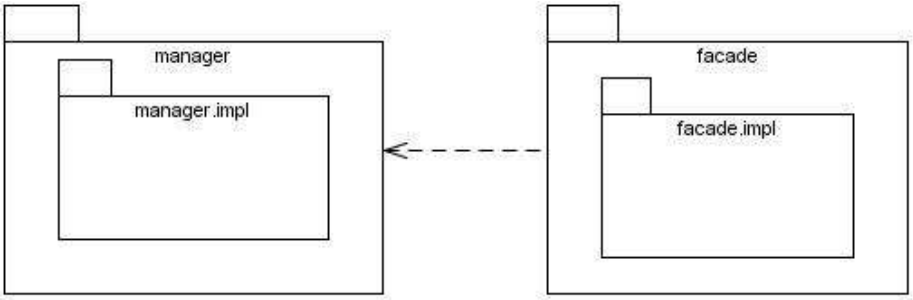

Ilustración 6: Estructura de paquetes de la capa de negocio.

2.2.3. Capa de presentación

En esta capa se desarrolla la lógica de presentación, se reciben y controlan los pedidos de la interfaz de usuario y se envían las respuestas procesadas a dichas peticiones, a través de la utilización de las funcionalidades brindadas por la fachada. Para representar y controlar los flujos de presentación complejos se utilizan los controladores de Spring Web Flow, y los de Spring MVC para responder a peticiones simples y lineales. En la parte del cliente se hace uso de la librería Dojo para generar y diseñar los componentes visuales con los que interactúa el usuario.

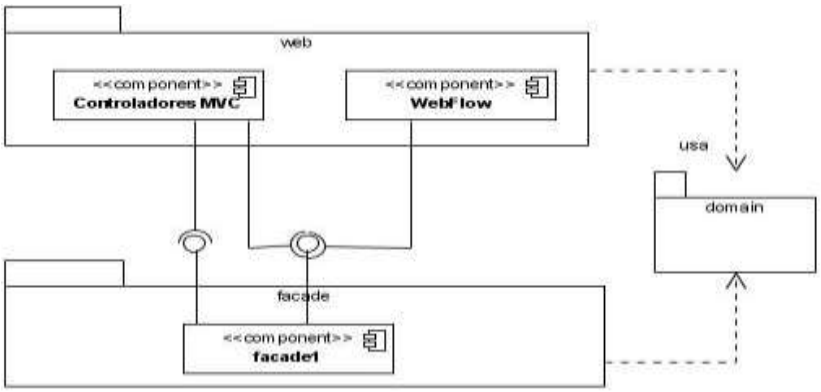

Ilustración 7: Arquitectura de la capa de presentación en el lado del servidor.

Estructura de la capa de presentación

En la parte del servidor todo el desarrollo de esta capa se encuentra en un paquete ***web*** (ver **Ilustración 8**) que contiene dos sub-paquetes ***webFlow*** y ***mvc***. El primero se encarga de la lógica de presentación

cuando se hace uso de Spring WebFlow. Para ello contiene los paquetes que engloban clases con una función específica:

command: representación de los objetos a manipular en los formularios de las páginas cliente.
flowHandler: personalización del trabajo con WebFlow.
serviceFlow: comunicación del flujo y la fachada.
validator: validación de los datos en el lado del servidor.
propertyEditor: conversión de los datos de la interfaz de usuario en objetos y viceversa.

El paquete ***mvc*** por su parte, está formado por un conjunto de sub-paquetes que contienen las clases encargadas de manejar la lógica de presentación que hace uso de Spring MVC, entre los que encuentran algunos de los ya descritos: ***validator, propertyEditor*** y ***command,*** además del paquete ***controller:*** contenedor de las clases que heredan de los controladores propuestos por este framework, para recibir y procesar las peticiones desde el cliente menos complejas en cuanto a flujo de presentación y generar las respuestas correspondientes a las mismas.

En la parte del cliente (ver **Ilustración 9**) todo el código javascript que contendrá las funciones, la creación de los componentes visuales y las validaciones, se almacena en ficheros .js que se encuentran en una carpeta con ese mismo nombre organizados por subsistemas y módulos dentro de la carpeta ***WebContent***. La misma contiene en la carpeta ***WEB-INF*** otro archivador llamado ***jsp,*** que engloba las páginas HTML que representan las interfaces de usuarios agrupadas por subsistema y módulos.

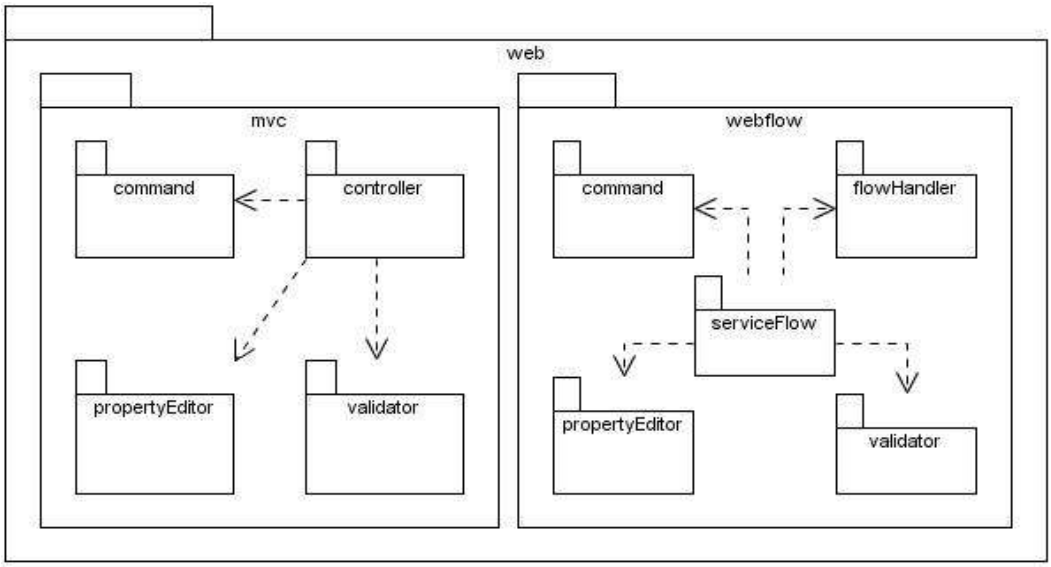

Ilustración 8: Estructura de paquetes de la capa de presentación en el lado del servidor.

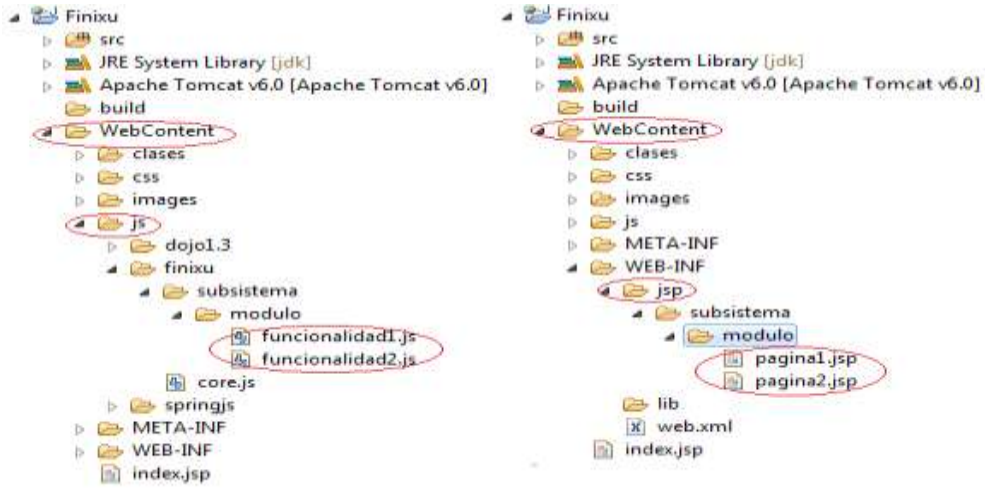

Ilustración 9: Estructura de paquetes de la capa de presentación en el lado del cliente.

2.3. Diseño del subsistema Vencimiento

El diseño consiste en la modelación del sistema de manera que soporte las restricciones y todos los requisitos, tanto funcionales como no funcionales. Entre las actividades fundamentales de la disciplina de Diseño se encuentran: identificar los elementos de diseño (Clases, Paquetes, Subsistemas de Diseño e Interfaces), diseñar componentes (Diseñar Casos de Uso y Crear Realizaciones de Casos de Uso), refinar

la arquitectura (Identificar Mecanismos de Diseño) y diseñar la Base de Datos. Los artefactos de salida resultantes de la realización de dichas actividades son el modelo de diseño, el modelo de despliegue y el modelo de datos: elementos fundamentales para el correcto desarrollo de las etapas posteriores. Teniendo en cuenta que en el flujo de implementación se refina la vista de arquitectura del modelo de despliegue asignando los componentes ejecutables a los nodos específicos, se muestra dicho artefacto en el próximo capítulo.

2.3.1. Modelo de diseño

Para realizar el modelo de diseño se toma como punto de partida la especificación de requerimientos realizada durante el flujo de trabajo de Requisitos, elemento clave para el diseño y la posterior implementación de la solución. Los requisitos funcionales definidos poseen una adecuada organización y documentación, no presentan ambigüedades y se encuentran correctamente descritos. Ofrecen prototipos de interfaz de usuario y glosario de términos completos que han sido esenciales para el entendimiento por parte del equipo de desarrollo. En sentido general, dicho artefacto de entrada cumple con la calidad y el nivel de detalle requerido para el diseño del subsistema Vencimiento. A partir de los requisitos se definen los casos de uso y se genera el modelo de diseño que incluye los artefactos: diagrama de paquetes, diagrama de clases del diseño y diagramas de interacción. Teniendo en cuenta las relaciones entre los casos de uso definidos se decidió estructurar el subsistema en los módulos: Common, Configuración y Gestionar Vencimiento.

El módulo Gestionar Vencimiento tiene la tarea de procesar todos los vencimientos que se contabilicen desde el resto de los módulos de la aplicación Quarxo. Su punto central de acceso lo constituye un buscador de vencimientos que lista los registrados en la Tabla “**M_Diario**” del núcleo central, a partir de los criterios de búsqueda que el usuario especifique. A partir de los resultados de la búsqueda se pueden seleccionar los vencimientos a procesar mediante las funcionalidades identificadas. De acuerdo a éstas se definen como ***Casos de Uso*** del módulo:

- ✓ **Pagar**: Permite el pago de uno o varios vencimientos de principal e intereses con sus moras en caso de que se indique. Incluye el envío de mensajes SLBTR o SWIFT.

- ✓ **Renegociar**: Permite renegociar varios vencimientos de principal e intereses, con sus moras en caso de que se indique. Brinda la posibilidad de realizar un pago parcial o total e incluye el envío de mensajes SLBTR o SWIFT correspondiente a dicha operación.
- ✓ **Cobrar**: Permite cobrar varios vencimientos vencidos de principal e intereses con sus moras.
- ✓ **Pasar a vencido**: Cambia las cuentas de los vencimientos que estén vencidos.
- ✓ **Acumular Intereses**: Acumula en una cuenta única los vencimientos vencidos de intereses contractuales.
- ✓ **Capitalizar**: Capitaliza en un calendario existente varios vencimientos vencidos de intereses contractuales, acumulados o moratorios.
- ✓ **Calendarizar**: Muestra el calendario al que pertenece un vencimiento y permite modificarlo.
- ✓ **Rebaja por exportación:** Constituye un tipo especial de pago donde se rebaja del principal por especie o producto.

Con el objetivo de automatizar estas operaciones fue necesario incluir una sección de configuración donde se listan todas las configuraciones registradas hasta el momento. Éstas definen los comportamientos y reglas que el sistema controla sobre cada cuenta a cuatros dígitos de un vencimiento, por ejemplo: los parámetros necesarios para las transacciones que pueda admitir dicha cuenta y las contrapartidas de las mismas; el término “contrapartidas” se refiere a los asientos que figuran en el haber de una cuenta y que tienen su compensación en una partida del debe de otra, o viceversa. El módulo responde a tres casos de uso: **registrar**, **actualizar** y **eliminar** una configuración. Por último se definió el módulo common para agrupar las funcionalidades y entidades comunes para todo el subsistema. Esta estructura por módulos se encuentra tanto en la parte del servidor como en la parte del cliente y se evidencia a través del modelo de paquetes.

2.3.1.1. Modelo de paquetes

Los paquetes permiten separar un modelo en partes manejables mediante la agrupación de clases y otros paquetes. Un diagrama de paquetes permite dividir al sistema orientado a objetos organizándolos en subsistemas y detallando sus relaciones. A continuación se figura el modelo de paquetes del subsistema Vencimiento.

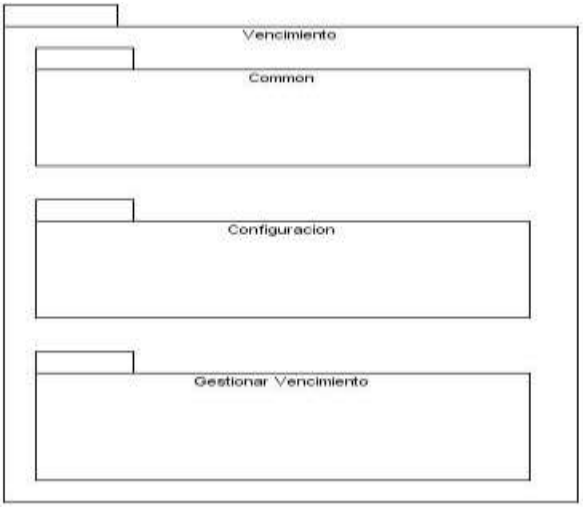

Ilustración 10: Modelo de Paquetes del subsistema Vencimiento.

Como objeto de estudio para presentar los elementos del diseño en el capítulo se selecciona el módulo Gestionar vencimiento, por contener un gran número de casos de uso críticos y por tanto representar el mayor por ciento de solución al problema actual del BNC en este proceso de negocio.

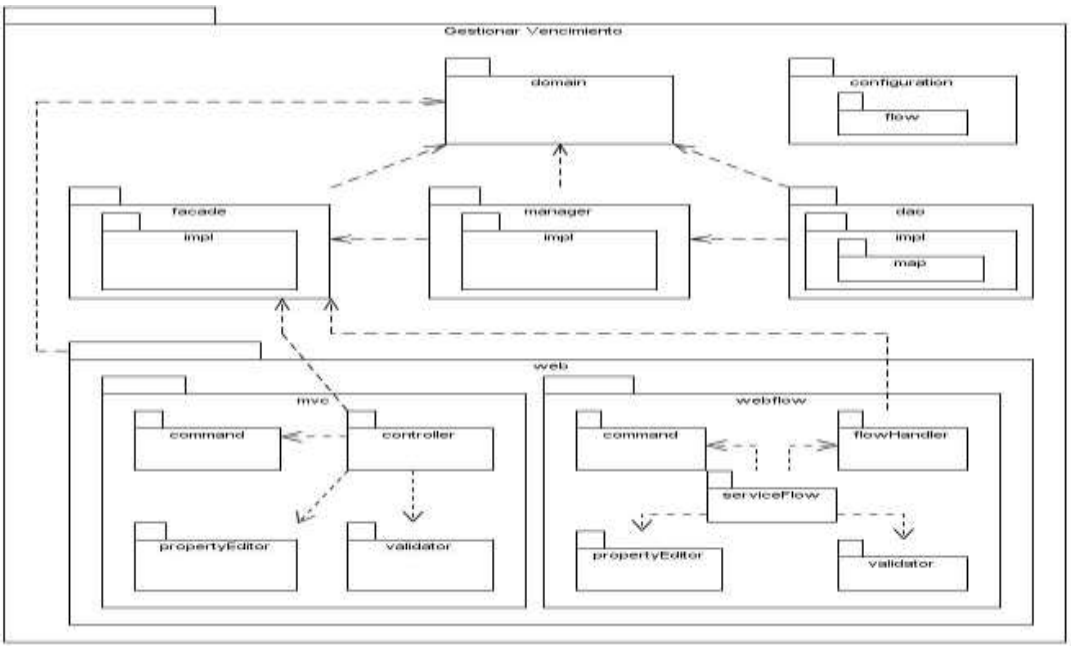

Ilustración 11: Modelo de Paquetes del módulo Gestionar vencimiento.

En el epígrafe anterior como parte de la descripción de la arquitectura de Quarxo se mostró la estructura de paquetes por capas y se detalló sus funciones. Para el completo entendimiento del diagrama sólo se describen los que no fueron vistos anteriormente:

domain: Se localizan las clases relacionadas que representan las entidades persistentes del módulo.

configuration: Se encuentran los ficheros de configuración en formato XML de los diferentes contextos de Spring para el módulo:

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

flow: En este paquete están presentes los archivos XML que definen los flujos de Spring WebFlow.

2.3.1.2. Diagramas de clases del diseño

El Diagrama de Clases permite conocer la estructura de las clases y sus relaciones para satisfacer los detalles de las implementaciones. Debido al gran volumen de entidades involucradas en el diagrama de clases del módulo Gestionar vencimiento, se muestran las ilustraciones separadas por las tres capas de la arquitectura del sistema.

Capa de presentación

La mayoría de los casos de uso presentes en el módulo contemplan la contabilización y presentan flujos complejos en sus procesos, haciendo imprescindible el uso de un controlador MultiAction de SpringWebFlow por cada uno de ellos. Para una mejor comprensión de las ilustraciones mostradas se propone una breve descripción de las clases involucradas en los diagramas:

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

SpringWebFlow: Paquete de clases que representa el mecanismo interno con el que SpringWebFlow gestiona las peticiones realizadas desde el cliente y mediante el cual se controlan los flujos por los que son guiados los usuarios en el sistema.

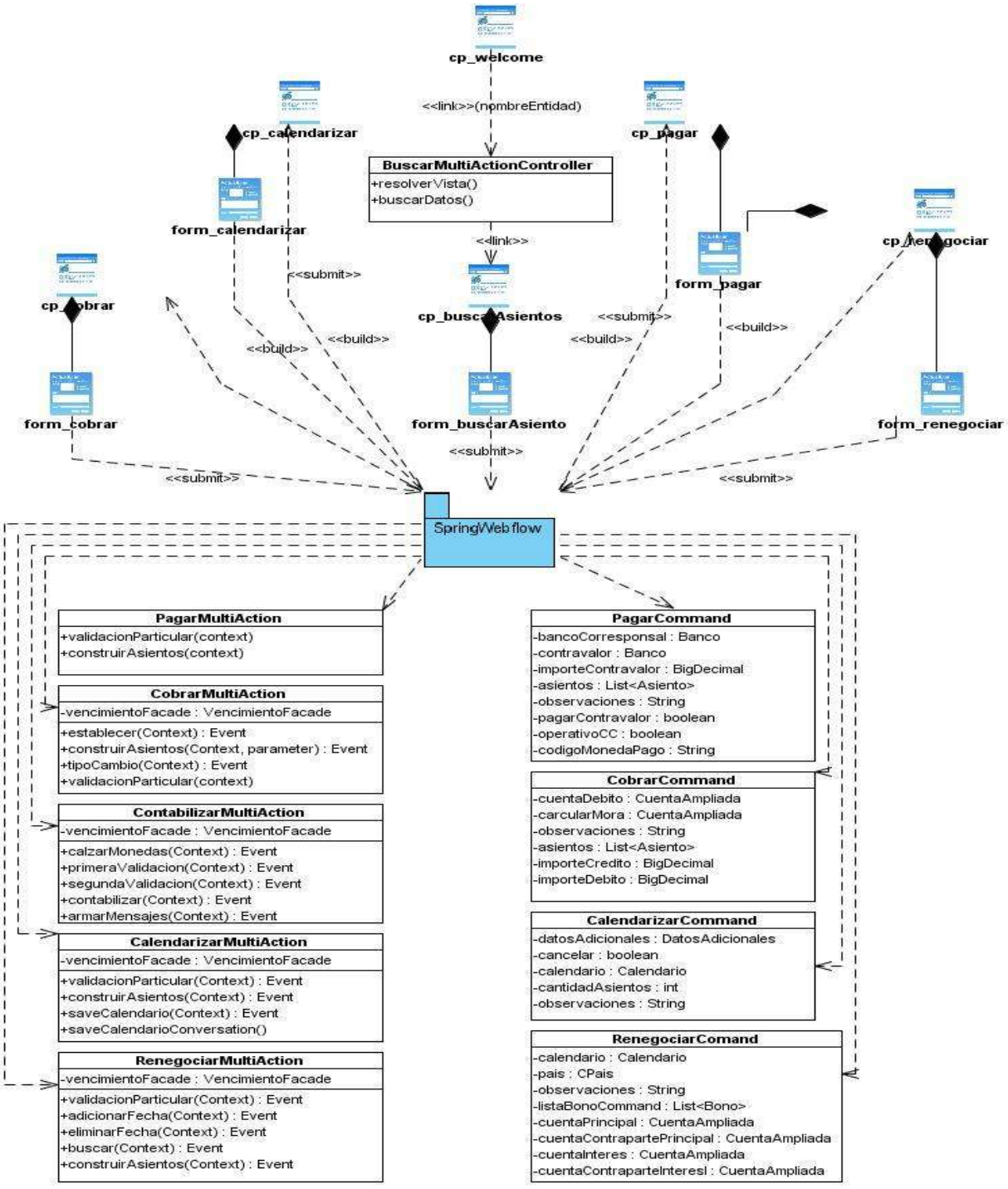

Ilustración 12.a: Diseño de la capa de presentación del módulo Gestionar vencimiento.

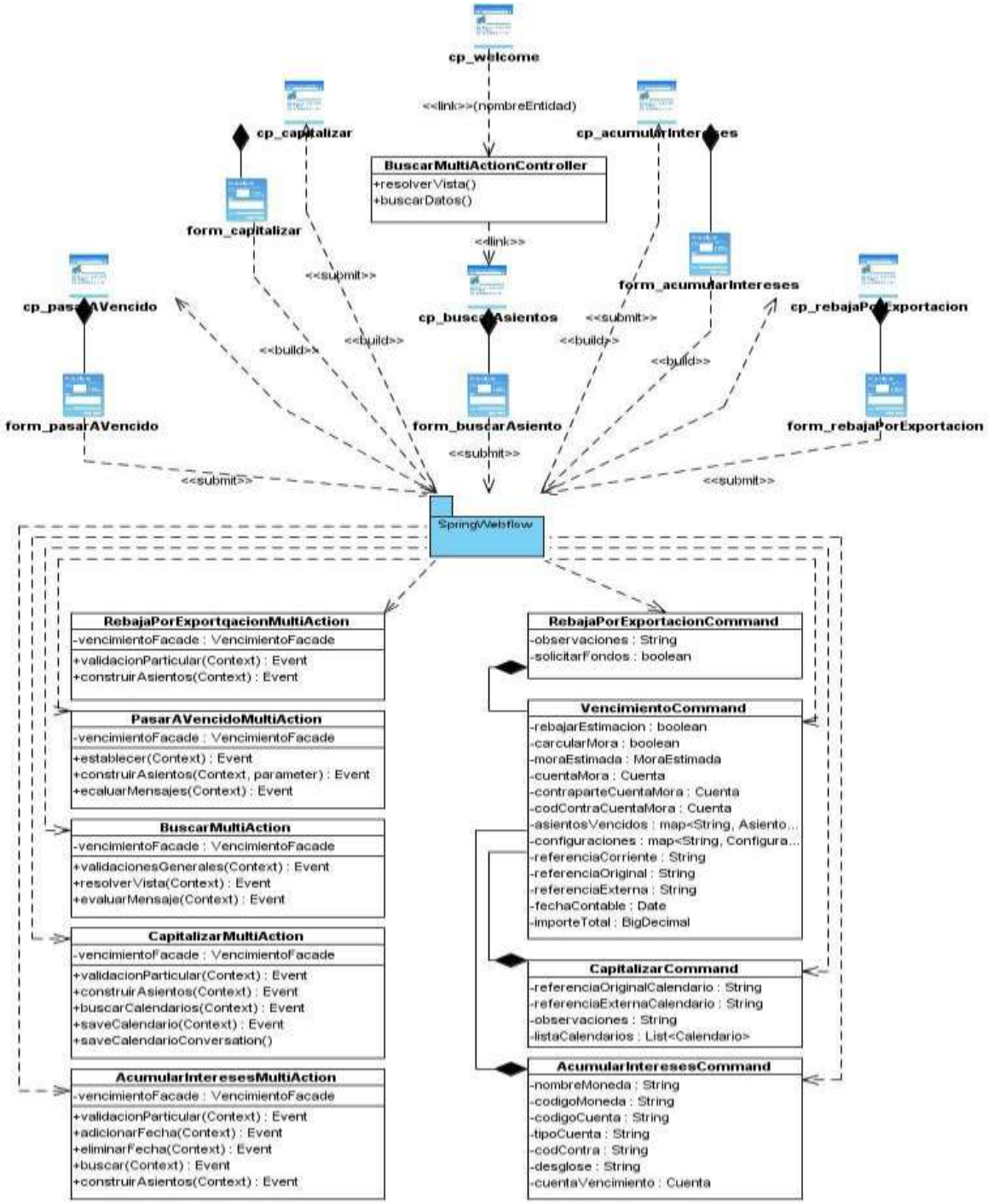

Ilustración 12.b: Diseño de la capa de presentación del módulo Gestionar vencimiento.

Las clases que heredan del controlador MultiAction de SpringWebFlow poseen una nomenclatura sugerente (***CasoDeUso***MultiAction); son las encargadas de procesar las peticiones referentes al caso de uso al que responden. En cada una de ellas excepto en el **BuscadorMultiAction** y en **ContabilizarMultiAction,** se encuentra una funcionalidad para validar los datos especificados desde el buscador de vencimientos y preparar la vista del caso de uso en cuestión, y otra para construir los asientos a contabilizar. La calendarización, capitalización y renegociación de vencimientos incluye el procesamiento de los calendarios, por tanto las clases que responden a estas operaciones poseen las funcionalidades que permiten el manejo de los mismos en el flujo. Por su parte la clase **BuscadorMultiAction** contiene las validaciones generales de los datos de los vencimientos seleccionados en la interfaz del buscador, para que puedan ser procesados con cualquiera de las funcionalidades del subsistema; además es la encargada de manipular el objeto Vencimiento en la sesión. Por último la clase **ContabilizarMultiAction** maneja todo el proceso de contabilización, que incluye tres validaciones previas, la acción de contabilizar en sí, la construcción de los mensajes en caso necesario y el calce de moneda que resuelve la diferencia de la misma entre los asientos a procesar, así como la gestión de las diferentes contrapartidas que pueden tener las diversas cuentas seleccionadas.

Capa de negocio

Como ya se había mencionado la capa de negocio cuenta con dos paquetes de clases: ***manager*** y ***facade.*** En el sistema Quarxo se definió utilizar una clase por cada módulo tanto en el ***manager*** como en el ***facade***. A través de la ilustración 13 se evidencian dichas clases y sus dependencias a gran escala.

La clase **VencimientoFacade** es la interfaz que contiene las declaraciones de todas las funcionalidades que necesitará el módulo en la capa de presentación, y la clase **VencimientoFacadeImpl** es la encargada de implementar dicha interfaz invocando las funcionalidades declaradas en la interfaz **VencimientoManager**, que dan solución al proceso de negocio del módulo en cuestión y son implementadas por la clase **VencimientoManagerImpl**. Ambas implementaciones contienen instancias de los objetos del dominio del módulo Gestionar vencimiento.

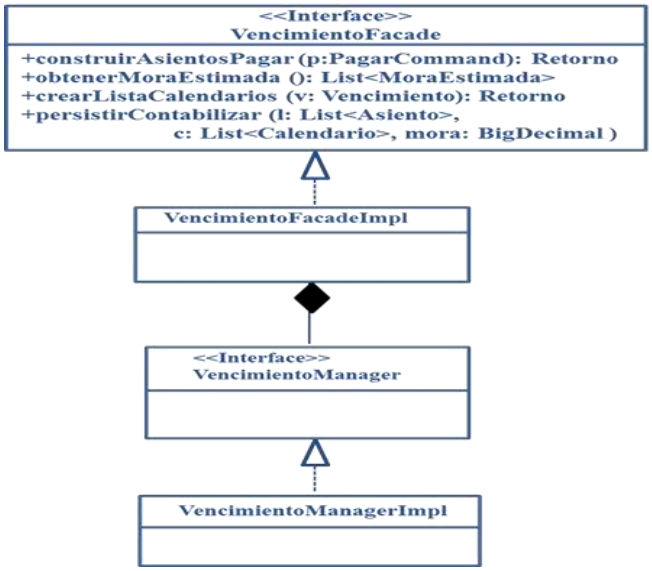

Ilustración 13: Diseño de la capa de negocio del módulo Gestionar vencimiento.

Capa de acceso a datos

Las clases **BaseDAO** y **AbstractBaseDAO**, interfaz y clase abstracta respectivamente, brindan un conjunto de funcionalidades básicas relacionadas con la persistencia y acceso de información de la base de datos. De la primera hereda la interfaz **FinixuBaseDAO**, la cual contiene además otras funcionalidades relacionadas con el paginado de listas de datos. Ésta se encuentra declarada e implementada en el paquete ***common*** de la aplicación. De la segunda extiende **FinixuAbstractBaseDAO** para implementar las funcionalidades de la clase **FinixuBaseDAO**. De esta manera las interfaces del acceso a datos del módulo en cuestión solo necesitan heredar de **FinixuBaseDAO** y sus implementaciones de **FinixuAbstractBaseDAO**. El diagrama que se muestra a continuación contiene, además de estas clases, las interfaces con las declaraciones de las funcionalidades del módulo y las clases que las implementan, que manejan la información persistente en la base de datos del objeto de dominio correspondiente.

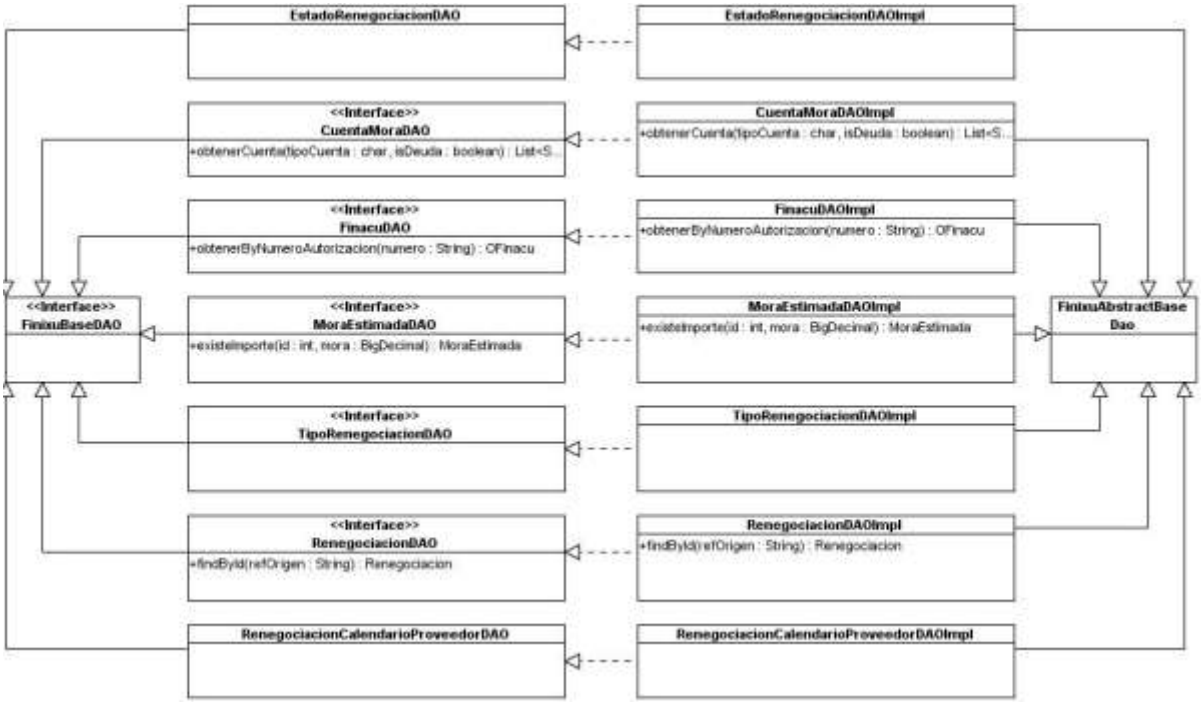

Ilustración 14: Diseño de la capa de acceso a datos del módulo Gestionar vencimiento.

2.3.1.3. Diagramas de interacción

Los Diagramas de Interacción capturan el comportamiento de los casos de uso mostrando la manera en que interactúan un grupo de objetos entre sí. Existen dos tipos de diagramas de interacción: el Diagrama de Secuencia y el Diagrama de Colaboración. El primero muestra una interacción ordenada según la secuencia temporal de eventos, mientras que el segundo muestra una interacción organizada basándose en los objetos que toman parte en la interacción y los enlaces entre los mismos. Aunque ambos enfatizan aspectos particulares trabajan sobre la misma información, de manera que se contemplan solamente los diagramas de secuencia.

Como objeto de estudio para mostrar en el capítulo se selecciona del módulo el escenario de **Pago**. Como todas las funcionalidades parten del flujo del buscador de vencimientos, se muestra el diagrama de secuencia del mismo y seguidamente el de la transacción del pago.

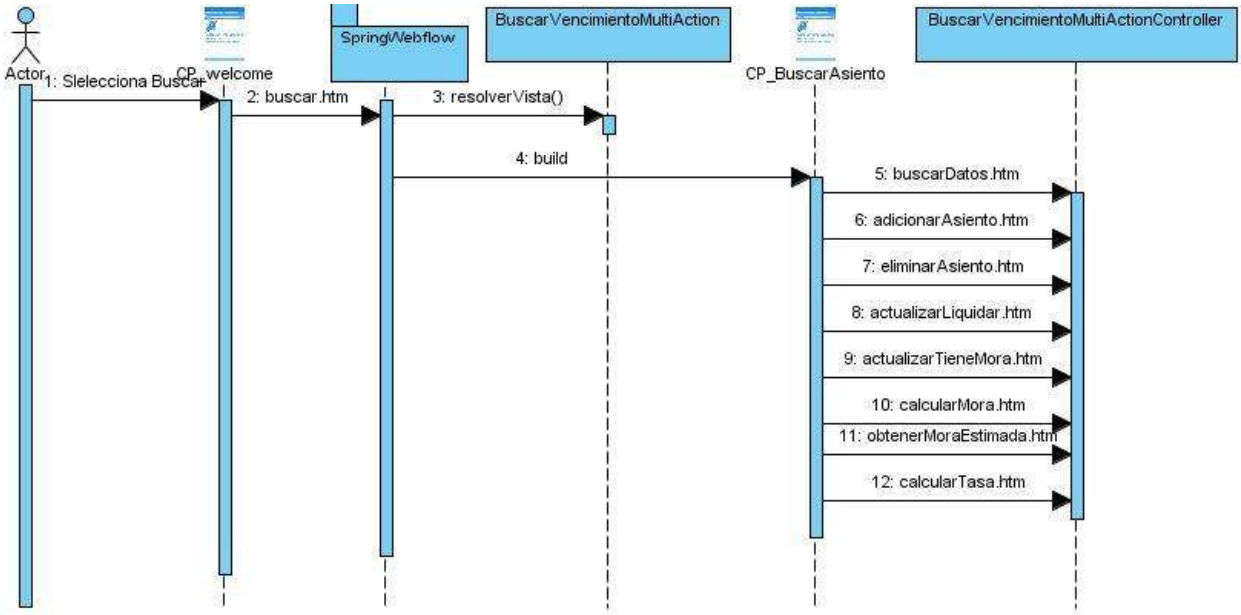

Ilustración 15: Diagrama de secuencia del buscador de vencimiento.

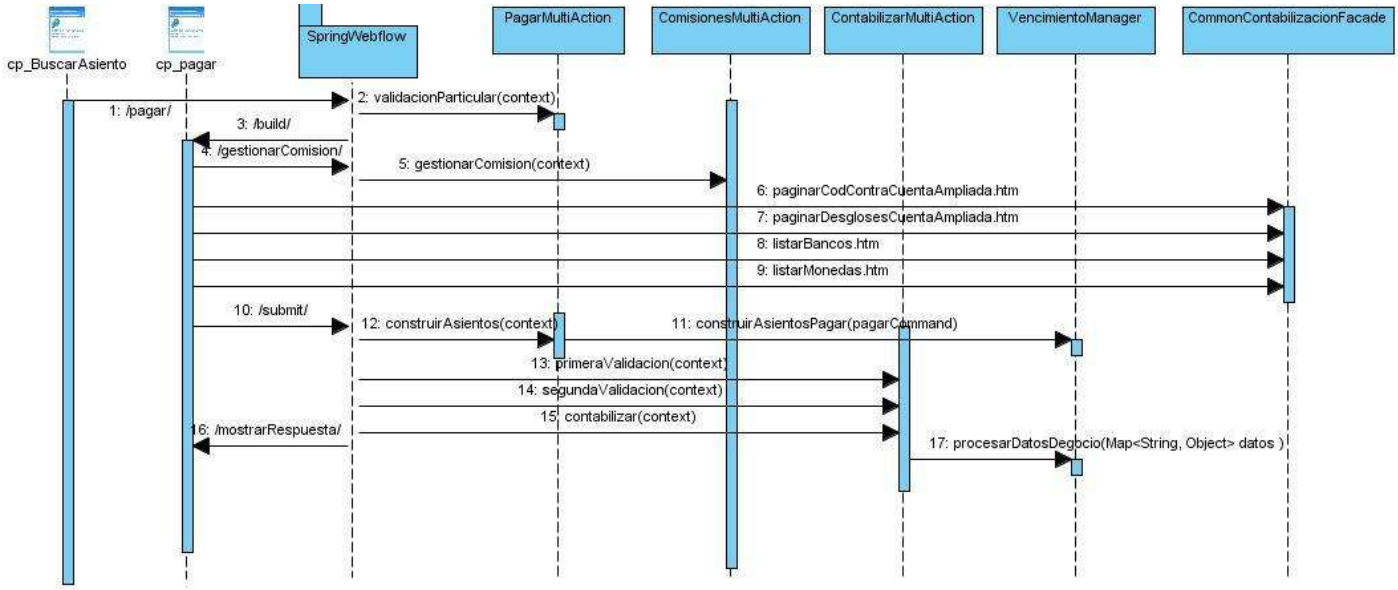

Ilustración 16: Diagrama de secuencia del caso de uso Pagar.

Descripción del flujo de sucesos

En el diagrama correspondiente al escenario **Pagar vencimiento**, el operario accede al buscador del subsistema seleccionando la opción **Buscar** dentro del menú **Gestionar vencimientos** en la página

cliente *welcome.jsp.* Ésta realiza una petición mediante un vínculo al motor de Spring Webflow, el cual construye la página cliente buscarAsiento.jsp, donde el usuario selecciona los criterios de búsqueda que desee y realiza una petición al hacer clic en el botón Buscar del formulario, que procesará la clase **BuscadorMultiActionController** que hereda del controlador MultiAction de Spring MVC. La misma envía al formulario, la lista de vencimientos que cumplen con los criterios especificados. Desde éste, se podrán realizar otras acciones que atenderá dicho controlador, tales como: marcar o desmarcar un asiento determinado de la tabla de asientos, que significa adicionarlo o eliminarlo de la lista de asientos a procesar; realizar el cálculo del importe de mora teniendo en cuenta datos introducidos por el usuario en caso necesario; y definir el importe a liquidar en el procesamiento del vencimiento seleccionado. Una vez que se seleccionan los asientos a procesar a través de una de las funcionalidades del subsistema, se elige la transacción que se desee realizar. En el caso mostrado se escoge la opción para pagar los vencimientos señalados. Esto supone una petición al motor de SpringWebFlow, el cual después de realizar una serie de validaciones generales a través de la clase **BuscadorMultiAction** y validaciones particulares del caso de uso Pagar a través de la clase **PagarMultiAction,** construye la página cliente *pagar.jsp.* La misma contiene un formulario con campos de datos correspondientes a los asientos seleccionados en el buscador, tanto de información de solo lectura como de modificación, y otros que permiten la especificación de información necesaria para la contabilidad. Con la creación de dicha página, se realizan una serie de peticiones, por lo general de cargas de datos, a las cuales responderá la clase **ContabilizacionMultiActionController** contenida en el paquete common de la aplicación. El formulario también incluye la gestión de las comisiones en caso de que se desee cobrar las asignadas a la transacción de pago. Para ello se realiza a través del componente comisiones, una petición al contenedor de Spring WebFlow que atenderá la clase **ComisionesMultiAction,** construyendo la vista correspondiente a dicho proceso. Una vez introducidos todos los datos necesarios se pueden realizar cualquiera de las acciones siguientes: cancelar la transacción a través del botón Cancelar, que interpreta Spring WebFlow como una petición de anulación de los cambios efectuados dirigiendo al usuario a la vista del buscador; visualizar los asientos antes de contabilizarlos a través de una pequeña imagen ubicada en la derecha superior de la página, donde se envía una petición que es atendida por la clase **ContabilizarMultiAction** de WebFlow, encargada de construir los asientos y mostrarlos en un *popup* de asientos contables; y finalmente contabilizar que constituye el proceso crítico de la transacción. Esta petición también es atendida por la clase anteriormente mencionada, la cual haciendo uso de la fachada **VencimientoFacade** realiza las validaciones previas a las

contabilización, la construcción de los asientos del pago y el calce de moneda, muestra al usuario las advertencias en caso de que existan y la opción de enviar un mensaje SWIFT o no. En dependencia de la selección del usuario se accede al componente de mensajería y se envían los datos necesarios al mismo para conformar el mensaje.

2.3.2. Modelo de datos

Las Bases de datos necesitan de una estructura que les permita almacenar datos, reconocer el contenido y recuperar la información. Tiene que conformarse en función de los requisitos del proceso del negocio que es la primera abstracción de la vista de la base de datos. Con este objetivo se crea un modelo de datos como lenguaje utilizado para la descripción de los tipos de datos, sus restricciones, relaciones y operaciones de manipulación.

El sistema Quarxo se desarrolla reutilizando el núcleo del SABIC, de manera que se usa la base de datos con la que dicha aplicación trabaja. Sin embargo, fue necesario agregar algunas tablas para garantizar la persistencia de todas las entidades implicadas en el proceso de gestión de vencimientos. A continuación se muestran las tablas principales del modelo de datos diseñado.

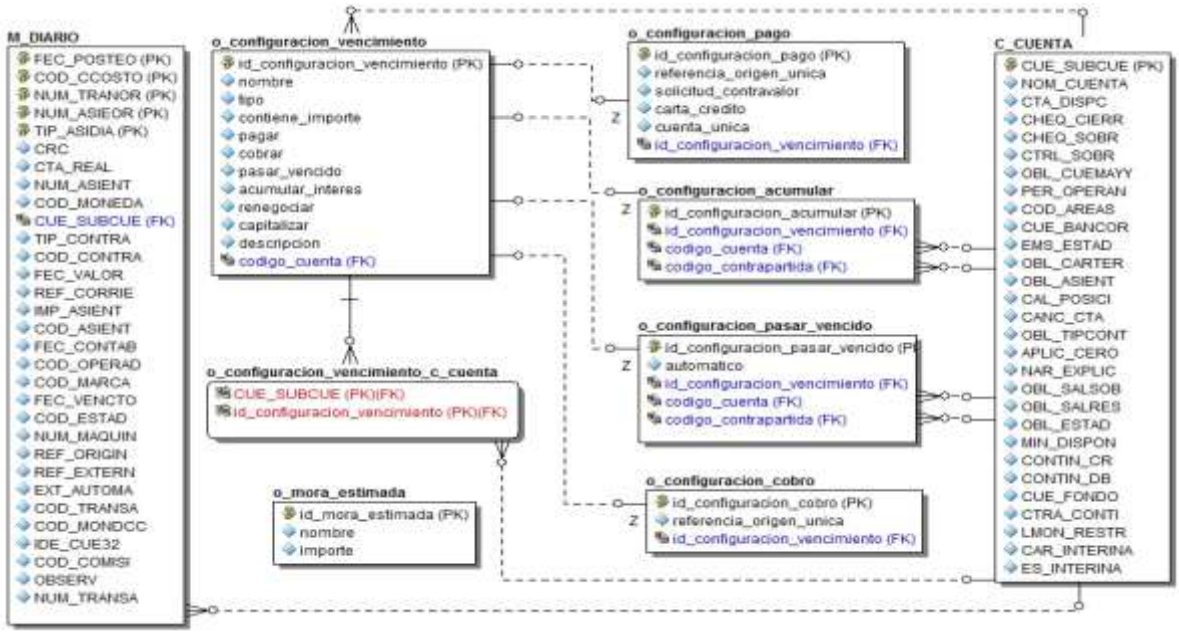

Ilustración 17: Modelo de Datos del subsistema Vencimiento.

Las tablas **M_DIARO** y **C_CUENTA** pertenecen al núcleo del SABIC, sin embargo, se presentan en el modelo de datos para una mejor comprensión de la estructura del mismo. El procesamiento de un vencimiento a través de cualquier operación puede ser parcial o total; el primer caso conlleva a una extracción completa del asiento de la tabla **M_DIARIO** y el segundo incluye además la inserción de un vencimiento, cuyo importe representa la diferencia no procesada. Teniendo en cuenta las funcionalidades del módulo configuración, se crea la tabla **o_configuracion_vencimiento** para almacenar las especificaciones de las cuentas de 4 dígitos de los vencimientos a procesar.

Para personalizar la forma en que serán realizadas sobre cada cuenta las transacciones, atendiendo a sus niveles de complejidad y variabilidad, se crean las tablas **o_configuracion_pago, o_configuracion_acumular**, **o_configuracion_pasar_vencido** y **o_configuracion_cobro**, que contienen las especificaciones de la operación correspondiente. Se elabora la tabla intermedia **o_configuracion_vencimiento_c_cuenta** para representar la relación entre las cuentas y sus contrapartidas y la tabla **o_mora_estimada** para almacenar las proyecciones de las moras para los pagos.

El modelo de datos del subsistema Vencimiento se diseña en base a lograr la fidelidad a las necesidades del problema y la reducción a la mínima expresión de los campos nulos, de los riesgos de inconsistencia y redundancia en los datos.

2.3.3. Patrones empleados

Una correcta aplicación de patrones contribuye a que el diseño tenga la calidad y flexibilidad requerida para la implementación de la solución. En el capítulo anterior se caracterizaron los patrones arquitectónicos y de diseño aplicados en el desarrollo del sistema Quarxo de manera general. En este epígrafe se ejemplifica el empleo de algunos de los patrones descritos en el subsistema Vencimiento.

Controlador: La clase controladora **ContabilizarMultiAction** constituye un ejemplo de la aplicación de este patrón, centralizando las actividades que responden a las peticiones del usuario referentes a la contabilización y funcionando como intermediaria entre la lógica del proceso y la presentación.

Experto: Dicho patrón es evidenciado en la definición de las interfaces DAO y sus implementaciones, las cuales se especializan en realizar las operaciones de acceso a datos referente a una determinada entidad.

Alta cohesión: Este patrón es utilizado en el diseño de los módulos del subsistema de manera general; evidenciándose en la agrupación de las clases en dependencia de los requerimientos.

Bajo acoplamiento: Este patrón es empleado en el diseño de las clases del subsistema de manera general, para disminuir la dependencia entre los elementos del diseño de cada módulo haciendo que cada uno de ellos, para realizar sus funciones, acceda solo a las clases de nivel inferior correspondiente y se abstraiga de la implementación de éstas, logrando que el código sea más fácil de entender, mantener y reutilizar.

Fachada: Este patrón se evidencia con la definición de la clase **VencimientoFacade** que provee a los controladores las funcionalidades del negocio del módulo, permitiendo la disminución del número de objetos con que deben trabajar y reduciendo el impacto de cambios posteriores en la lógica de negocio.

Patrón de Acceso a Datos (DAO): Se aplica con la definición de las interfaces DAO que disminuyen la complejidad de los objetos del dominio, al librarlos de la responsabilidad de manejar la implementación de sus fuentes de datos.

MVC (Modelo Vista Controlador): Se evidencia su aplicación en la arquitectura dividida en tres capas, utilizada en el proyecto SAGEB. La capa de negocio (que incluye las clases **VencimientoFacade** y **VencimientoManager**) y de acceso a datos (que contiene las interfaces DAO por cada funcionalidad y sus implementaciones) constituyen el Modelo, la capa de presentación en la parte del cliente que contiene las páginas *.jsp* constituyen la Vista, y la capa de presentación en el lado del servidor, que engloba el conjunto de clases que hacen uso de los controladores tanto de Spring MVC como de Spring WebFlow, constituye el Controlador.

Composite View (Vistas Compuestas): Como ejemplo se encuentra la página pagar.jsp, la cual utiliza en su formulario los componentes comisiones y contraparte que representan otras páginas.

```
<div class="FilaCompleta">
    <c:if test="${pagarCommand.vencimiento.tieneContraparte}">
    <%@ include file="/WEB-INF/jsps/vencimiento/gestionarvencimiento/contraparte.jsp"%>
    </c:if>
</div>
<br>
<div class="SpaceCH"></div>     <br>

<div><%@ include file="/WEB-INF/jsps/common/global/comisiones.jsp"%></div>
```

2.3.4. Validación del diseño

El diseño está enfocado a convertir los requisitos del cliente en un modelo que al ser implementado, se obtenga el producto deseado. Generalmente constituye el punto de partida para el desarrollo de software una vez especificados los requerimientos del sistema. Evidentemente realizar una validación del mismo para verificar su calidad y flexibilidad, garantiza una buena base para la implementación. Con este objetivo se utilizan un conjunto de métricas de software orientadas a determinar, entre otros aspectos, qué características del modelo de diseño se pueden estimar para comprobar que el sistema será fácil de implementar en cuanto a organización. Entre ellas se seleccionan las métricas **Tamaño Operacional de Clase** y **Relaciones entre Clases** para validar el diseño del subsistema Vencimiento.

Tamaño Operacional de Clase

Esta métrica se determina por el número total de operaciones que están encapsuladas dentro de la clase. Grandes valores de esta medida muestran que la clase puede tener demasiada responsabilidad, lo cual reducirá la reusabilidad de la misma y complicará su implementación. Por otro lado, en cuanto menor sea el valor medio para el tamaño más probable es que las clases tengan menos responsabilidad y complejidad y más nivel de reutilización. La aplicación de esta métrica demuestra que el diseño presenta 100% de aceptabilidad en cuanto a responsabilidad y complejidad de implementación de las clases y un 89% con respecto a la reutilización de las mismas, representando un resultado positivo.

Relaciones entre Clases

Esta métrica se determina por la cantidad de relaciones existentes entre las clases contenidas en el diseño. El número de dependencias es directamente proporcional al nivel de acoplamiento, a la complejidad del mantenimiento y a la cantidad de pruebas a realizar sobre las clases, y es inversamente proporcional al grado de reutilización de las mismas. La aplicación de esta métrica evidencia que el diseño del subsistema posee el 88% de aceptabilidad con respecto a todos los atributos de calidad afectados, arrojando resultados positivos.

Matriz de inferencia de indicadores de calidad

La matriz de inferencia permite evaluar positiva o negativamente los resultados obtenidos de las relaciones atributo/métrica en una escala numérica, donde el valor 1 representa un resultado positivo y el valor 0 negativo. Si la métrica no influye en el atributo de calidad la relación es considerada como nula y es representada con un guión simple (-). Al promediar por cada atributo las relaciones no nulas, se obtiene un resultado general que al ubicarse en el rango de evaluación (valor <=0.4: “Mala”, (valor >0.4 y valor<0.7: “Regular”, valor>=0.7: “Buena”) permite arribar a conclusiones sobre la calidad del diseño propuesto. Teniendo en cuenta que los resultados arrojados con la aplicación de las métricas fueron positivos para todos los atributos, la matriz de la inferencia queda elaborada de la siguiente manera:

Atributos\Métricas	TOC	RC	Promedio
Responsabilidad	1	(-)	1
Complejidad de Implementación	1	(-)	1
Reutilización	1	1	1
Acoplamiento	(-)	1	1
Complejidad de Mantenimiento	(-)	1	1
Cantidad de pruebas	(-)	1	1

Tabla 1: Resultados de la evaluación de la relación atributo/relación.

El promedio general es 1 y teniendo en cuenta el rango de evaluación se concluye que el subsistema Vencimiento presenta un diseño con buena calidad.

2.4. Conclusiones Parciales

La especificación de los requisitos capturados sobre el proceso de gestión de los vencimientos, constituyó el punto de partida para la definición del diseño de Vencimiento, siguiendo la arquitectura definida para el sistema Quarxo. Un modelo de diseño con calidad debe ser resistente a cambios en el entorno de implementación y fácil de mantener en relación a otros posibles modelos de objetos. La correcta aplicación de los patrones descritos contribuyó a la buena calidad del diseño propuesto, comprobada finalmente a través de la aplicación de dos de las métricas de validación de diseño, sentando de manera eficiente las bases para la implementación de las funcionalidades del subsistema.

CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DEL SUBSISTEMA VENCIMIENTO.

3.1. Introducción

En el flujo de trabajo de Implementación se realizan un conjunto de actividades encaminadas a elaborar la solución a partir del diseño definido. Dicha implementación produce versiones operacionales que deben ser validadas mediante las pruebas, las cuales son un elemento crítico de garantía de la calidad del software y representan una revisión final de las especificaciones de diseño y codificación. El Flujo de Trabajo Implementación describe cómo los elementos del Modelo de Diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el Modelo de Despliegue. En el último capítulo del presente trabajo de diploma se explican los principales elementos de la implementación del subsistema Vencimiento y de la validación de la misma, tales como: la integración de los componentes del sistema con los módulos del subsistema en cuestión, los estándares de codificación utilizados, la descripción de las clases, sus funcionalidades y la utilización de Spring WebFlow Framework a través de un caso de estudio, el refinamiento de la vista de arquitectura del modelo de despliegue y los elementos de validación de la solución.

3.2. Elementos fundamentales de la implementación

3.2.1. Utilización de Spring WebFlow Framework

Teniendo en cuenta la complejidad de los casos de uso del módulo Gestionar vencimiento, así como la relación existente entre ellos, se utiliza para su implementación Spring WebFlow Framework lográndose una mayor reutilización de código y una mejor navegabilidad entre las vistas involucradas. Se crea un flujo común que permite seleccionar y preparar los vencimientos a procesar como punto de partida de todos los casos de uso. Valorando las funcionalidades que presentan flujos complejos y que son utilizados prácticamente por todos los casos de uso, se crearon flujos independientes con el propósito de que pudieran ser reutilizados.

La entrada al módulo Gestionar vencimiento es la vista de buscar asientos, donde se filtran por criterios de búsqueda especificados por el usuario, los asientos que cumplen con dichas restricciones. Brinda la posibilidad de seleccionarlos, definir el importe a liquidar y el monto de mora a procesar por un pago,

cobro o renegociación en caso necesario, así como visualizar la información elemental antes de realizar la operación deseada. Presenta un flujo que contiene el objeto Vencimiento, que engloba toda la información obtenida del buscador para el procesamiento del mismo, que será inicializado a través de la funcionalidad ***resetCommand*** siempre antes de realizar cualquier operación y en ocasiones donde se necesite también. Dicho flujo es el encargado de generar la vista antes mencionada y de manejar la transición al sub-flujo correspondiente a cada caso de uso.

```
<var name="vencimiento"
    class="cu.uci.finixubnc.vencimiento.gestionarvencimiento.domain.Vencimiento" />
<action-state id="resetCommand">
    <evaluate expression="buscadorMultiAction.resetCommand"/>
    <transition to="buscador"/>
</action-state>
<view-state id="buscador" view="buscarAsiento" model="vencimiento">
    <on-entry>
        <evaluate expression="buscadorMultiAction.resolverVista" />
    </on-entry>
    <transition on="t0pagar" to="validar" />
    <transition on="t1renegociar" to="validar" />
    <transition on="t2cobrar" to="validar" />
    <transition on="t3pasaravencido" to="validar" />
    <transition on="t4acumularintereses" to="validar" />
    <transition on="t5capitalizar" to="validar" />
    <transition on="t6calendarizar" to="validar" />
    <transition on="t7rebajar" to="validar" />
    <transition on="okEnd" to="buscador" />
    <transition on="resetCommand" to="resetCommand"/>
    <transition on="cancelar" to="buscador">
        <evaluate expression="buscadorMultiAction.evaluarMensajeError"/>
    </transition>
</view-state>
```

Además es el responsable de validar los datos seleccionados a través de la funcionalidad ***validacionesGeneralesVencimiento*** e impedir la ejecución de las operaciones en caso de existir algún error, así como almacenar el mensaje del error ocurrido para que pueda ser notificado al usuario desde la vista del buscador.

```
<action-state id="validar">
    <evaluate expression="buscadorMultiAction.validacionesGeneralesVencimiento"/>
    <transition on="error" to="buscador" />
    <transition to="${currentEvent.id}" />
</action-state>
```

Atendiendo a que los flujos que responden a los casos del uso del módulo en cuestión tienen una estructura y funcionamiento similar, para su explicación, dando seguimiento al caso de estudio utilizado para ilustrar algunos de los diagramas del diseño generados, se muestran como ejemplo sentencias de código del archivo XML que define el flujo del *pago de vencimiento*, el cual responde a la funcionalidad más reutilizable y compleja del subsistema en general. Cada sub-flujo tendrá como entrada el objeto

Vencimiento y como salida el mensaje resultante de la operación, que puede ser una notificación de error, de advertencia o de realización satisfactoria.

```
<subflow-state id="tOpagar" subflow="pagarvencimiento-flow">
    <input name="vencimiento" value="flowScope.vencimiento"
        type="cu.uci.finixubnc.vencimiento.gestionarvencimiento.domain.Vencimiento"/>
    <output name="mensajeError" />
    <transition to="resetCommand" />
</subflow-state>
```

Al entrar a cada sub-flujo que responde a un caso de uso se ejecuta una validación teniendo en cuenta aspectos particulares del mismo, la cual garantiza un tratamiento de error análogo al que se realiza en la validación general del buscador. Con excepción del caso de uso *pasar a vencido,* se hace necesario efectuar la carga de la lista de contrapartes que contiene las cuentas previamente configuradas con la información de sus respectivas contrapartidas.

```
<input name="vencimiento"
    type="cu.uci.finixubnc.vencimiento.gestionarvencimiento.domain.Vencimiento"
    required="true" />

<action-state id="validacionParticular">
    <evaluate expression="pagarMultiAction.validacionParticular"></evaluate>
    <transition on="error" to="errorEndState" />
    <transition on="success" to="listarContrapartes" />
</action-state>

<action-state id="listarContrapartes">
    <evaluate expression="contabilizarMultiActionVencimiento.listarContrapartes"/>
    <transition to="registrarPago"/>
</action-state>
```

Cada sub-flujo genera la vista correspondiente a su caso de uso, específica el objeto *command* que será manipulado en la misma y maneja el conjunto de transiciones que responden a una funcionalidad determinada.

```
<view-state id="registrarPago" view="pagar" model="pagarCommand">
    <transition on="mostrarMultiplesCuentas" to="mostrarMultiplesCuentas" />
    <transition on="actualizarContrapartes" to="contrapartesSubFlow" />
    <transition on="contabilizar" to="construirAsientos" />
    <transition on="cancelar" bind="false" validate="false" to="endState" />
    <transition on="okEnd" to="endState" />
    <transition on="actualizarDatosComisiones" validate="false" bind="false" to="registrarPago">
        <evaluate expression="comisionesMultiAction.actualizarDatosComisiones" />
    </transition>
    <transition on="gestionarComision" validate="false" to="gestionarComisionSubflow">
        <evaluate expression="comisionesMultiAction.registrarCallMetodo" />
    </transition>
</view-state>
```

Para realizar un crédito o débito a múltiples cuentas, genera una vista con la información de cada una de ellas.

```
<view-state id="mostrarMultiplesCuentas" popup="true" view="mostrarListaCuentas">
    <on-render>
        <evaluate expression="pagarMultiAction.listarCuentasAmpliadas" />
    </on-render>
</view-state>
```

Para actualizar las contrapartes de las cuentas involucradas en la operación se permite la entrada al sub-flujo *contrapartesSubFlow*, que genera una vista mostrando la información y controlando la actualización de la misma.

```
<subflow-state id="contrapartesSubFlow" subflow="contrapartes-flow">
     <transition to="listarContrapartes" />
</subflow-state>
```

Para el cobro de las comisiones correspondientes al caso de uso se posibilita la entrada al sub-flujo *gestionarComisionSubFlow* que origina la vista que permite adicionar, eliminar o modificar las comisiones.

```
<subflow-state id="gestionarComisionSubflow" subflow="gestionarComision-flow">
     <transition on="end" to="registrarPago" />
</subflow-state>
```

Para la realización de la contabilización se construyen los asientos y luego se remite al sub-flujo *contabilizar-flow,* proporcionándole al mismo la información necesaria previa al proceso a través de un conjunto de variables de entrada. De esta manera se controlan aspectos como: el envío opcional de mensajes SWIFT, los importes de mora a descontar y la persistencia de los calendarios, así como la contabilización o la visualización de los asientos construidos.

```
<action-state id="construirAsientos">
    <evaluate expression="pagarMultiAction.construirAsientos"></evaluate>
    <transition on="error" to="registrarPago" />
    <transition to="contabilizarSubFlow" />
</action-state>

<subflow-state id="contabilizarSubFlow" subflow="contabilizar-flow">
 <input name="hayMensajeria" type="java.lang.Boolean" value="true" />
 <input name="visualizar" value="flashScope.visualizar" type="java.lang.Boolean" />
 <input name="asientos" value="flashScope.asientos" type="java.util.ArrayList" />
 <input name="calendarios" value="flashScope.calendarios" type="java.util.ArrayList" />
 <input name="vencimientoMensaje" value="flashScope.vencimientoMensaje"
    type="VencimientoMensaje" />
 <input name="idMoraEstimada" type="java.lang.Integer" value="flashScope.idMoraEstimada" />
 <input name="moraADescontar" type="java.math.BigDecimal" value="flashScope.moraADescontar"/>
 <output name="mensajeError" />

 <transition to="registrarPago" />
</subflow-state>
```

Considerando la importancia del proceso de contabilización se muestra en detalles el funcionamiento del sub-flujo antes mencionado. Inicialmente se generan los asientos necesarios para resolver la diferencia de

monedas a través de la funcionalidad ***calzarMonedas***. Luego se analiza si se debe generar la vista que muestra la información de los asientos antes de contabilizarlos, o si se debe iniciar el procedimiento contable. Una vez seleccionada la segunda opción se ejecuta una primera validación en la que se cancela la operación si se detecta algún error y se brinda al usuario la información del mismo, o de lo contrario se procede a una segunda validación que puede generar errores o advertencias; el tratamiento de errores es similar al efectuado en la primera validación; las advertencias se notifican al usuario quien decide si proseguir con la operación o cancelarla.

```
<action-state id="init">
    <evaluate expression="contabilizarMultiActionVencimiento.calzarMoneda" />
    <transition on="error" to="errorEndState" />
    <transition to="isVisualizar" />
</action-state>

<decision-state id="isVisualizar">
    <if test="flowScope.visualizar" then="verAsientosState" else="primeraValidacion" />
</decision-state>

<view-state id="verAsientosState" popup="true" view="../../common/global/asientosContables">
    <on-render>
        <evaluate expression="contabilidadMultiAction.prepararAsientosTransaccion" />
    </on-render>
</view-state>

<action-state id="primeraValidacion">
    <evaluate expression="contabilizarMultiActionVencimiento.primeraValidacion" />
    <transition on="error" to="errorEndState" />
    <transition to="segundaValidacion" />
</action-state>

<action-state id="segundaValidacion">
    <evaluate expression="contabilizarMultiActionVencimiento.segundaValidacion" />
    <transition on="error" to="mostrarAdvertenciasContabilizar" />
    <transition on="success" to="hayMensajeria" />
</action-state>

 <view-state id="mostrarAdvertenciasContabilizar" view="mostrarAdvertenciasContabilizar">
     <transition on="contabilizar" to="hayMensajeria" />
     <transition on="terminar" to="errorEndState">
         <evaluate expression="contabilizarMultiActionVencimiento.anularMensajeError" />
     </transition>
 </view-state>
```

Posteriormente se analiza si se decide o no enviar un mensaje SWIFT para proceder con la contabilización en caso negativo o contrariamente conformar el mensaje y enviar al sub-flujo de mensajería, la información necesaria para realizar sus operaciones y la contabilización. Para ambas opciones se muestra como resultado una notificación al usuario, con el error ocurrido brindando la

posibilidad de rectificarlo en la misma vista, o con un mensaje de realización satisfactoria en caso contrario.

```
<decision-state id="hayMensajeria">
    <if test="flowScope.hayMensajeria" then="mensajeriaSubFlow" else="contabilizar" />
</decision-state>

<action-state id="contabilizar">
    <evaluate expression="contabilizarMultiActionVencimiento.contabilizar" />
    <transition to="errorEndState" />
</action-state>

<subflow-state id="mensajeriaSubFlow" subflow="conformarMensaje-flow">
    <on-entry>
        <evaluate expression="contabilizarMultiActionVencimiento.armarMensajes" />
    </on-entry>
    <input name="idOperacion" value="flashScope.idOperacion" />
    <input name="xmlEntidad" value="flashScope.xmlEntidad" />
    <input name="datosNegocio" value="flashScope.datosNegocio" />

    <transition on="end" to="errorEndState">
        <evaluate
            expression="contabilizarMultiActionVencimiento.verificarError(flowRequestContext,currentEvent.attributes.exception)"></evaluate>
    </transition>
</subflow-state>
```

3.2.2. Estándares de Codificación

Los estándares de codificación son normas de programación enfocadas a la estructura del programa para facilitar la lectura, comprensión y mantenimiento del código. El establecimiento de un estándar de codificación sólido y la utilización de buenas prácticas de programación en función de generar un código de alta calidad, son factores necesarios para lograr un buen rendimiento y eficiencia en el desarrollo del software.

Con este objetivo se utiliza en la implementación del sistema Quarxo, la notación ***PascalCasing*** para la nomenclatura de las clases y la notación ***CamelCasing*** para nombrar las variables y métodos presentes en cada una de ellas. En la primera se obvia el uso de artículos y se define que los nombres de las clases deben comenzar con letra mayúscula, y en caso de presentar varias palabras la letra inicial de cada una también debe serlo, por ejemplo: VencimientoMensaje. La segunda determina que los nombres de los métodos y variables deben comenzar con letra minúscula y en caso de estar compuesto por varias palabras, a partir de la segunda la letra inicial debe ser mayúscula, ejemplo: validacionParticular(). Se aplican respectivamente para la nomenclatura de los ficheros de código javascript y para la de sus

funciones y variables internas. Los nombres de los paquetes no pueden contener letras mayúsculas y deben ser sustantivos que sugieran el contenido del mismo. Ejemplo: jsp. Siguiendo estos estándares se estableció una nomenclatura en las clases por cada una de las capas de la arquitectura.

En la capa de presentación

Las clases pertenecientes a los sub-paquetes mvc y webflow del paquete web tendrán la siguiente estructura en sus nombres:

Paquete controller: [Nombre de la funcionalidad a la que responde] + [Nombre del controlador de Spring MVC que se hereda]. Ejemplo: *CargarDatosMultiActionController*

Paquete serviceFlow: [Nombre de la funcionalidad a la que responde] + [Nombre del controlador de Spring WebFlow que se hereda]. Ejemplo: *PagarMultiAction*.

Las clases contenidas en el resto de los paquetes tendrán la siguiente nomenclatura:

[Nombre de la funcionalidad a la que responde] + [Nombre del paquete]. Ejemplo: *PagarCommand*.

En la capa de negocio

Las clases contenidas a los paquetes facade y manager y sus sub-paquetes impl respectivamente se nombrarán de la manera siguiente:

Para las interfaces: [Nombre del módulo] + [Nombre del paquete]. Ejemplo: *VencimientoFacade*

Para sus implementaciones: [Nombre del módulo] + [Nombre del paquete] + Impl. Ejemplo: *VencimientoFacadeImpl.*

En la capa de acceso a datos

Las clases englobadas en el paquete dao y su sub-paquete impl tendrán esta nomenclatura:

Paquete dao: [Nombre de la entidad] + DAO .Ejemplo: MoraEstimadaDAO.

Paquete dao.impl: [Nombre de la entidad] + DAOImpl .Ejemplo: MoraEstimadaDAOImpl.

3.1.1. Descripción de las clases y funcionalidades del subsistema

Atendiendo al caso de estudio seleccionado, se describirán las clases más importantes: **PagarMultiAction**, encargada de manejar el flujo del pago de los vencimientos; la clase **PagarCommand**, usada para recoger la información necesaria del formulario de la vista del pago para formar los asientos a procesar; y la clase **ContabilizarMultiAction**, responsable de manejar el proceso de contabilización.

Nombre: PagarMultiAction	
Tipo de clase: Controladora	
Atributo	**Tipo**
vencimientoFacade	VencimientoFacade
Para cada responsabilidad:	
Nombre:	**Descripción:**
validacionParticular(RequestContext contex)	Valida la lista de asientos proveniente del buscador atendiendo a las especificaciones propias del Pago.
listarCuentasAmpliadas(RequestContext contex)	Se encarga de listar todas las contrapartidas que serán mostradas al usuario en un popup.
setearRadioButton(RequestContext contex)	Actualiza los datos a mostrar en el componente de cuenta única.
listarTiposDeCambio(RequestContext contex)	Lista los tipos de cambio existentes en la base de datos.
construirAsientos(RequestContext contex)	Se encarga de construir los asientos a contabilizar partiendo de los datos contenidos en el objeto de tipo PagarCommand.

Tabla 2: Descripción de la clase PagarMultiAction del módulo Gestionar vencimiento

Nombre: ContabilizarMultiAction	
Tipo de clase: Controladora	
Atributo	**Tipo**
vencimientoFacade	VencimientoFacade
Para cada responsabilidad:	
Nombre:	**Descripción:**
calzarMonedas(RequestContext contex)	Genera los asientos para resolver la diferencia de monedas.
armarMensajes(RequestContext contex)	Prepara todos los datos del mensaje SWIFT a conformar.
primeraValidacion(RequestContext contex)	Permite invocar la primera validación de los asientos a contabilizar y en caso de obtener algún mensaje de error o alerta indicárselo al flujo.
segundaValidacion(RequestContext contex)	Permite invocar la segunda validación sobre la lista de asientos y en caso de obtener algún mensaje de error o alerta indicárselo al flujo.
Contabilizar(RequestContext contex)	Permite contabilizar la lista de asientos y persistir las entidades necesarias de forma transaccional. En caso de obtenerse algún error se le indica al flujo.

Tabla 3: Descripción de la clase ContabilizarMultiAction del módulo Gestionar vencimiento.

Nombre: PagarCommand	
Tipo de clase: Modelo	
Atributo	**Tipo**
Vencimiento	Vencimiento
cuentaDebito	CuentaAmpliada

cuentaCredito	CuentaAmpliada
importeDebito	BigDecimal
importeCredito	BigDecimal
Observaciones	String
Comisiones	List<CobroComision>
tipoCambioDb	NomTipTip
clienteCredito	String
clienteDebito	String
desgloseDebito	String
desgloseCredito	String
fechaCambioDb	Date
tasaDb	BigDecimal
listaIDAsientosSeleccionados	String
Asientos	List<Asiento>
calcularTipoCambio	Boolean
bancoCorresponsal	Banco
Contravalor	Banco
pagarContravalor	Boolean
operativoCartaCredito	Boolean
contieneImporte	Boolean
cuentaUnicaCapaciodadFinanciera	Boolean
cuentaUnicaGarantias	Boolean
cuentaUnicaProy	Boolean
cuentaUnicaNA	Boolean
importeContravalor	BigDecimal
tasaCr	BigDecimal
fechaCambioCr	*Date*
nomTipCambioCr	*NomTipTip*
referenciaOrigen	*String*
datosAdicionales	Map
Proy	String
observacionesSLBTR	String
codigoMonedaPago	String
siglaMonedaPago	String
tipoPago	TipoPago
disponibilidadDB	BigDecimal
isContabilizar	Boolean
cuentaSolicitudFondo	String

Tabla 4: Descripción de la clase PagarCommand del módulo Gestionar vencimiento.

3.1.2. Integración de componentes

Los componentes son activos reutilizables que representan elementos genéricos, independientes y que responden a un problema determinado. Las reglas de diseño que deben obedecer son especificadas por el modelo de componentes. Entre los estándares y convenciones que impone dicho modelo se encuentran los esquemas de interacción a partir de los cuales se describe cómo interactúan los componentes entre sí. Para el desarrollo del sistema Quarxo se crean un grupo de componentes de los que consumirán servicios los subsistemas y módulos de la aplicación. Se muestra a continuación el diagrama que evidencia la interacción del subsistema Vencimiento con los componentes definidos y una descripción de la función que realiza cada uno de ellos.

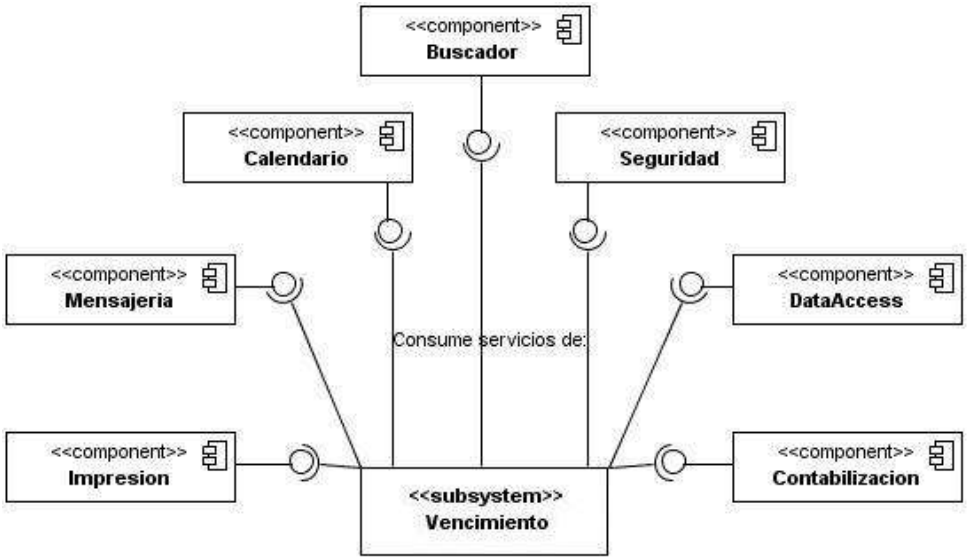

Ilustración 18: Diagrama de interacción de componentes.

El componente ***Mensajería*** contiene las funcionalidades que garantizan la recepción y el envío automático de los mensajes resultantes de la contabilidad, disminuyéndose el esfuerzo de los usuarios del sistema para enviar o recibir dichos mensajes. Como se había mencionado anteriormente se utiliza en el pago de vencimiento. Por su parte, el componente ***Seguridad*** gestiona la información de los usuarios, roles y permisos necesarios para lograr la confiabilidad requerida en el sistema. Es utilizado para controlar el acceso del usuario a las posibles operaciones a realizar sobre los vencimientos. El componente ***Contabilización*** engloba el conjunto de funcionalidades que permiten la realización de dicho proceso; constituye el de mayor consumo por parte del subsistema. En el caso del ***dataAccess*** su uso es

imprescindible para el funcionamiento de todo el sistema, porque se encarga de proporcionar los elementos necesarios para la conexión a la base de datos. La búsqueda de los asientos a contabilizar o de las cuentas a configurar a partir de criterios especificados por el usuario, son el punto de partida para ejecutar las funcionalidades en los módulos Configuración y Gestionar vencimiento respectivamente, función posibilitada por la utilización del componente ***Buscador***. Para la impresión de información referente a la contabilidad se consumen los servicios de ***Impresión*** y se hace uso del componente ***Calendario*** para la gestión de las formas de pago que contienen los vencimientos.

3.1.3. Modelo de despliegue

El diagrama de despliegue describe la arquitectura física del sistema durante la ejecución en términos de procesadores, dispositivos y componentes de software. Describen la estructura de los elementos de hardware y el software que ejecuta cada uno de ellos. Es utilizado para capturar los elementos de configuración del procesamiento y las conexiones entre esos elementos. También se utiliza para visualizar la distribución de los componentes de software en los nodos físicos. La implementación produce un refinamiento de la vista de arquitectura del modelo de despliegue, donde los componentes ejecutables son asignados a nodos. El entorno de despliegue de la aplicación queda estructurado de la siguiente manera: se cuenta con aproximadamente 150 máquinas cliente con los programas instalados Mozilla Firefox 3.5 para acceder a la aplicación Quarxo, Adobe Reader, Microsoft Office y la Máquina virtual jdk 6.20; cada una está conectada a una impresora. Los sistemas Quarxo, SLBTR y SISCOM se encuentran en el servidor de aplicaciones del BNC, que al igual que el servidor de Base de Datos posee como sistema operativo Windows Server 2003; requiere del contenedor web Apache Tomcat 6.0, la Máquina Virtual de java y el Microsoft Office. El servidor de base de datos donde se encuentran el núcleo del SABIC y el de SISCOM requiere del SQL Server 2005.

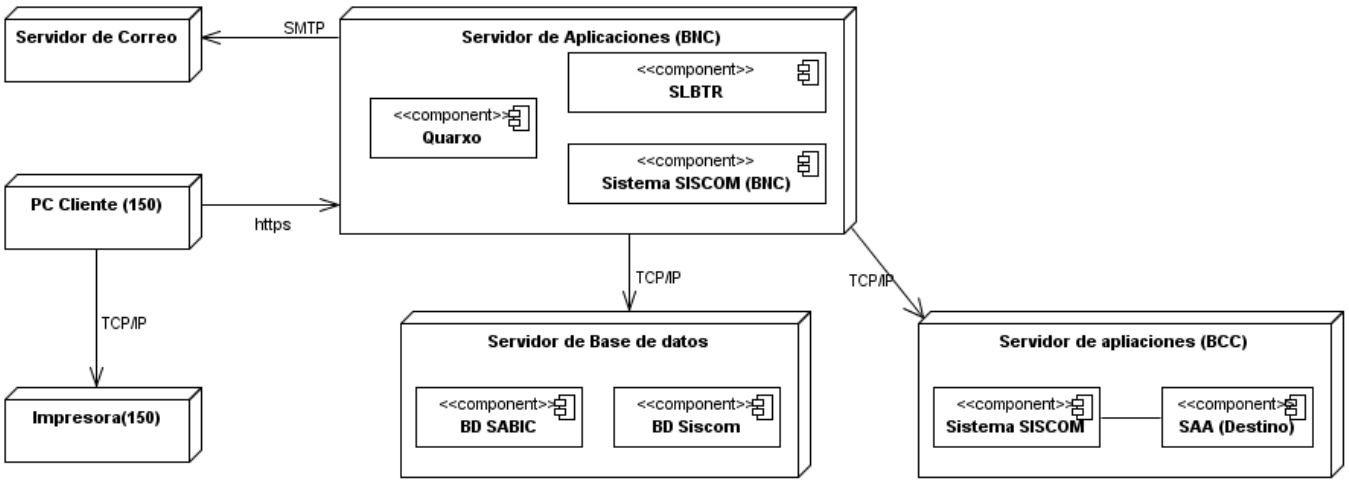

Ilustración 19: Diagrama de despliegue de Quarxo.

3.2. Validación del subsistema Vencimiento

En el desarrollo del software el proceso de prueba es la clave para detectar errores o fallas. Una selección cuidadosa de los datos de prueba puede ofrecer mucha confianza en cuanto al desempeño del sistema. La estrategia de prueba seguida para validar el subsistema Vencimiento consiste en la realización de pruebas en los niveles de unidad, de integración, de sistema y de aceptación consecutivamente.

Se comienza por las pruebas funcionales del nivel de Unidad a través del método de caja negra, enfocadas a la validación de las funciones y casos de uso. El método consiste en probar el sistema sobre sus interfaces de usuario, es decir, verificar que la aplicación hace lo que debe hacer independientemente de la forma en que fueron implementadas las funcionalidades. Con este objetivo se definen un conjunto de casos de prueba que contienen las condiciones de entrada suficientes para ejercitar completamente los requisitos funcionales de ambos módulos de Vencimiento por separados. Para confeccionar los casos de prueba de Caja Negra existen distintos criterios: *Particiones de Equivalencia, .Análisis de Valores Límite, Métodos Basados en Grafos, Pruebas de Comparación* y *Análisis Causa-Efecto.* La técnica empleada fue la de Partición de Equivalencia, que representa una de las más efectivas al examinar los valores válidos e inválidos de las entradas existentes en el subsistema, permitiendo descubrir de forma inmediata una clase de errores que de otro modo requerirían la ejecución de muchos casos antes de detectar el error genérico.

La realización de estas pruebas en cuatro iteraciones arrojó resultados positivos sobre el funcionamiento del subsistema.

En el nivel de Integración se combinan y prueban a la vez múltiples componentes. El objetivo es tomar los que fueron probados en el nivel de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño. Se comprueba la correcta integración de ambos módulos del subsistema y del mismo con el resto de los componentes de la aplicación.

En el nivel de Sistema las pruebas se hacen cuando el software está funcionando como un todo después que los componentes de software y hardware han sido integrados. Los tipos de pruebas realizadas sobre Vencimiento pertenecientes a este nivel son: la de seguridad, enfocada a verificar que los datos o el sistema solamente es accedido por los actores deseados; y la de rendimiento, diseñada para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado. La impresora es el hardware externo que debe integrarse para la impresión de la información referente a la contabilidad de los vencimientos procesados. La comprobación del buen funcionamiento del subsistema con dicha integración y de la seguridad también brinda resultados positivos.

Una vez probado el subsistema por el equipo de desarrollo del proyecto, el Departamento de Calisoft de la Universidad procede a liberar los módulos de toda la aplicación a través de la realización de pruebas funcionales, de carga y de estrés. La prueba de carga permite validar y valorar la aceptabilidad de los límites operacionales de un sistema bajo carga de trabajo variable, mientras el sistema bajo prueba permanece constante; y la de estrés se enfoca a evaluar cómo el sistema responde bajo condiciones anormales (extrema sobrecarga, insuficiente memoria, entre otros). Como resultado de las pruebas realizadas por Calisoft en tres iteraciones, se manifiesta su aprobación en el acta de liberación del sistema. Actualmente se está sometiendo la aplicación a las pruebas de aceptación del cliente en el Banco Nacional de Cuba, siendo la etapa final de validación antes del despliegue del sistema, cuyo objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar las funciones para las cuales fue construido.

El documento emitido por Calisoft liberando el producto, se puede consultar en el Anexo # 1 del documento.

3.3. Conclusiones parciales

Con la implementación del subsistema Vencimiento se obtiene una aplicación funcional para el sistema Quarxo, que gestiona eficientemente el procesamiento de los vencimientos. Esta etapa del desarrollo se rige por el diseño definido previamente, permitiendo que se realice de forma organizada y ágil. Las pruebas realizadas tanto por el grupo de calidad del proyecto como por el equipo de Calisoft validan la implementación realizada, demostrando que cumple con los requerimientos necesarios para satisfacer las necesidades del cliente.

CONCLUSIONES

Con el desarrollo del presente trabajo de diploma, se cumplieron los objetivos específicos propuestos, permitiendo arribar a las siguientes conclusiones:

- ✓ La caracterización de la metodología, lenguajes, tecnologías y herramientas definidas para el desarrollo del sistema Quarxo y la valoración de los sistemas informáticos contables nacionales e internacionales que involucran el manejo de los vencimientos, constituyeron los elementos necesarios para fundamentar las bases teóricas para darle solución al problema planteado.
- ✓ La generación de los artefactos durante el desarrollo de los flujos de Diseño e Implementación proporcionó la obtención del subsistema Vencimiento y la integración satisfactoria del mismo al sistema Quarxo.
- ✓ La validación del diseño mediante la aplicación de las métricas, las pruebas realizadas por el equipo de desarrollo del proyecto SAGEB y la liberación del mismo por Calisoft, demostró que la solución del subsistema Vencimiento cumple con los requerimientos, la eficiencia y la estabilidad necesaria para ser desplegado en el Banco Nacional de Cuba.

De manera general se solucionó el problema existente a través de la realización de todas las tareas definidas, obteniéndose como resultado el subsistema Vencimiento, que permitirá, a través de sus funcionalidades mejorar la gestión de los vencimientos en el Banco Nacional de Cuba en cuanto a tiempo, cantidad de operarios involucrados y disminución de errores en el proceso.

RECOMENDACIONES

Atendiendo a las conclusiones a las que se arriban con el desarrollo del trabajo, se recomienda:

- ✓ Adaptar el subsistema obtenido para que pueda ser utilizado en un futuro por otras entidades bancarias que incluyan la gestión de los vencimientos.
- ✓ Utilizar las tecnologías y herramientas estudiadas en el desarrollo de otros sistemas informáticos de gestión bancaria.

BIBLIOGRAFÍA

1. **Academia Universitaria, S.L.** Platon. [En línea] [Citado el: 10 de Diciembre de 2010.] http://www.aplaton.net/gestion.htm.

2. **Bazo, Ursula.** CIO Perú. *CIO Perú.* [En línea] Saya Comunicaciones S.A.C. [Citado el: 20 de Noviembre de 2010.] http://cioperu.pe/articulo/4713/las-ventajas-de-la-automatizacion.aspx.

3. Contaplus élite. [En línea] [Citado el: 17 de Diciembre de 2010.] http://ciberconta.unizar.es/LECCION/contaplus/200.HTM.

4. **Cooper, J. W. 2000.** *java Design Patterns .* s.l. : Addison Wesley, 2000.

5. Cuba. Sitio del Gobierno de la Republica de Cuba. [En línea] [Citado el: 21 de Noviembre de 2010.] http://www.cubagob.cu/.

6. Definición de. *Definición de modelo de datos.* [En línea] [Citado el: 27 de Enero de 2011.] http://definicion.de/modelo-de-datos/.

7. **E. Gamma, R. Helm, R. Johnson, J. Vlissides. 1994.** *Desing Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Addison Wesley, 1994.

8. **Hista Internacional . 2007.** Hista Internacional. [En línea] 27 de Febrero de 2007. [Citado el: 11 de Diciembre de 2010.] http://www.histaintl.com/servicios/consulting/rup.php.

9. **J. Crupi, D. Alur, D. Malks. 2003.** *Core J2EE Patterns, 2da edition.* s.l. : Prentice Hall, 2003.

10. **Javier, Enrique &.** Buenas Tareas. [En línea] [Citado el: 15 de Diciembre de 2010.] http://www.buenastareas.com/ensayos/Interes-Legal-e-Interes-Convencional/819536.html.

11. **Pablo, Juan. 2009.** Zona-Net. [En línea] 26 de Febrero de 2009. [Citado el: 17 de Diciembre de 2010.] http://www.zona-net.com/search/Gesti%C3%B3n+MGD%3A+Programa+de+contabilidad+gratuito.

12. **Pantaleón, Ana. 2002.** El Pais. *Las tarjetas seguras de pago se estrenan en España para atajar el fraude electrónico.* [En línea] 25 de Abril de 2002. [Citado el: 17 de Diciembre de 2010.]

13. http://www.elpais.es/suple/ciberpais/articulo.html?d_date=20020425&xref=20020425elpcibenr_1&type=Tes&anchor=elpcibred.

14. **Patricio Salinas Caro, Nancy Histchfeld K.** DCC Depar. [En línea] [Citado el: 16 de Diciembre de 2010.] http://www.dcc.uchile.cl/~psalinas/uml/introduccion.html.

15. **Quintero, Mgtr. Irma Y. García De. 2010.** Buenas Tareas. *19 de Diciembre de 2010.* [En línea] 19 de Diciembre de 2010. [Citado el: 15 de Diciembre de 2010.] http://www.buenastareas.com/ensayos/Contabilidad-Financiera/2092666.html.

16. **Romero, Marielisa. 2011.** Contabilidad y auditoria. [En línea] 20 de Marzo de 2011. [Citado el: 22 de Marzo de 2011.] http://trabajomariaelisaromero.blogspot.com/2011/03/contabilidad-bancaria.html.

17. **Serendipity, Cooperativa. 2006.** RENa. *RENa.* [En línea] CENIT, 2006. [Citado el: 13 de Noviembre de 2010.] http://www.rena.edu.ve/cuartaEtapa/metodologia/fundamentacionTeorica.html.

18. **Sinibaldi, Fabian. 2008.** Todo Productos Financieros. *sitio Web de Todo Productos Financieros.* [En línea] 16 de Noviembre de 2008. [Citado el: 13 de Noviembre de 2010.] http://todoproductosfinancieros.com.

19. **Tamargo, Lourdes Cerezal. 2001.** BETSIME. [En línea] 2001. [Citado el: 21 de Noviembre de 2010.] http://www.betsime.disaic.cu/secciones/tec_feb_02.htm.

20. **V, Srolamtj. 1999.** Proceedings of the IEEE/IAFE 1999 Conference on Computational Intelligence for Financial engineering (CIFEr). *Intelligent Trading Systems: a Multi-Agent Hybrid Architecture.* [En línea] 28 de March de 1999. [Citado el: 17 de December de 2010.] http://biblioteca.itesm.mx/cgi-bin/nav/salta?cual=ieeexplore:6229.

21. **2000.** vLex. [En línea] 1 de Noviembre de 2000. [Citado el: 10 de Diciembre de 2010.] http://vlex.com.mx/vid/sentencia-ejecutoria-contradiccion-26827431.

22. **Pablo, Juan. 2009.** Zona-Net. [En línea] 26 de Febrero de 2009. [Citado el: 17 de Diciembre de 2010.] http://www.zona-net.com/search/Gesti%C3%B3n+MGD%3A+Programa+de+contabilidad+gratuito.

23. **Hista Internacional . 2007.** Hista Internacional. [En línea] 27 de Febrero de 2007. [Citado el: 11 de Diciembre de 2010.] http://www.histaintl.com/servicios/consulting/rup.php.

24. **Patricio Salinas Caro, Nancy Histchfeld K.** DCC Depar. [En línea] [Citado el: 16 de Diciembre de 2010.] http://www.dcc.uchile.cl/~psalinas/uml/introduccion.html.

REFERENCIAS BIBLIOGRÁFICAS

(1) Carolina Peña Zepeda, Belen Allende Sanchez, Mishel Ulloa Cuevas. 2010. Historia de Mexico II. *Creacion de instituciones bancarias.* [Online] Octubre 27, 2010. [Cited: Febrero 12, 2011.] http://mexicombc.blogspot.com/2010/10/gobierno-de-plutarco-calles.html.

(2) BNC. 2007. *MANUAL DE INSTRUCCIONES Y PROCEDIMIENTOS. ANTECEDENTES HISTORICOS.* Habana : s.n., 2007.

(2) BNC. 2007. *MISIÓN Y OBJETIVOS FUNDAMENTALES DEL BNC.* Habana : s.n., 2007.

(3) Romero, Marielisa. 2011. Contabilidad y auditoria. [Online] Marzo 20, 2011. [Cited: Marzo 22, 2011.] http://trabajomariaelisaromero.blogspot.com.

(4) Lara, Luis Eugenio Dávila. 2003. Resumen del tema II "Conceptos de Contabilidad y Principios de Contabilidad". [En línea] Agosto de 2003. [Citado el: 13 de Febrero de 2011.] http://www.elcontadorpublico.com.ve/UnidadII.htm.

(5) 2000. vLex. [Online] Noviembre 1, 2000. [Cited: Diciembre 10, 2010.] http://vlex.com.mx/vid/sentencia-ejecutoria-contradiccion-26827431.

(6) Valdez, Francisco Soberón. 1997. Capital mínimo para comenzar a operar. [En línea] 17 de Diciembre de 1997. [Citado el: 14 de Febrero de 2011.] http://www.bc.gov.cu/Espanol/Leyes/Supervision/Capitulo%20I/capitulo%201_1-1.pdf.

(7) Fernando Llena Macarulla, Alfonso López Viñegla. Contaplus élite. [En línea] 2000 [Citado el: 17 de Diciembre de 2010.] http://ciberconta.unizar.es/LECCION/contaplus/200.HTM.

(8) Díaz Mesa, Lissett 2009. *Proyecto Técnico SAGEB*. Habana : s.n., 20 de Junio de 2009.

(9) Sierra, Alejandro. 2003. Programacion Extrema. *La nueva Metodologia.* [En línea] Marzo/Abril de 2003. [Citado el: 3 de Marzo de 2011.] http://www.programacionextrema.org/articulos/newMethodology.es.html.

(10) Alexander Cristopher., Ishikawa S., Silverstein M. 1977. *Un lenguaje de patrones.* Barcelona : Gustavo Gili, 1977.

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

(12) Draper, Drave. 2008. *Dojo Concepts For Java developers.* Estados Unidos : IBM CORPORATION, 2008.

(13) Juan Manuel Barrios N. 2003. *Investigación de la plataforma JEE y su aplicación práctica*. [En línea] 30 de Mayo de 2003. [Citado el: 15 de junio de 2011.] http://www.dcc.uchile.cl/~jbarrios/J2EE/MemoriaJ2EEWeb.html

(14) Sun 1999. *Apache Tomcat*. [En línea] 1999. [Citado el: 15 de junio de 2011.] http://tomcat.apache.org/.

(15) Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato 2004. *Control de versiones con SubVersion*. [En línea] 2004. [Citado el: 15 de junio de 2011.] http://svnbook.red-bean.com/nightly/es/svn-ch-1-sect-1.html

(16) Craig Walls, Ryan Breidenbach. 2007. *Spring in Action Second Edition.* Estados Unidos : Manning Publications, 2007

(17) Chaviano, Ing. Adolfo Miguel Iglesias. 2008. *Documento de Arquitectura para el proyecto SAGEB.* Habana : s.n., 2008.

Glosario de términos

API

Es un conjunto de funciones y métodos que ofrece cierta biblioteca (conjunto de subprogramas) para ser utilizado por otro software como una capa de abstracción.

Calendario

Conjunto de formas de pago que contienen vencimientos de principal y de intereses ordinarios.

Capitalización de intereses

Se adiciona el interés no liquidado, al principal no reembolsado en una fecha de pago o vencimiento establecida y se fija un nuevo interés a cobrar.

Estrategia de captura (fetch)

Estrategia que Hibernate usa para obtener los objetos asociados, cuando se necesite navegar por una asociación.

Interés ordinario

Impuesto monetario por el uso del principal por un tiempo determinado.

JTA

Establece una serie de Interfaces java entre el manejador de transacciones y el servidor de aplicaciones, el manejador de recursos y las aplicaciones transaccionales.

Pago por exportación

Tipo especial de pago de vencimiento, donde se compensa el monto a pagar a través de exportaciones de producto.

Persistencia transitiva

Permite insertar, actualizar o borrar un objeto de una asociación sin necesidad de hacerlo explícitamente. Por defecto Hibernate no navega por las asociaciones, para ello permite la configuración de cada una de las asociaciones a través del atributo cascade (en asociaciones many-to-one, one-to-many y many-to-many).

Plugins

Un **plug-in** es un módulo de hardware o software que adiciona una característica o un servicio específico a un sistema existente.

Principal

Monto que se concede inicialmente en un préstamo o negociación.

Programas ofimáticos

Los programas ofimáticos son los que se utilizan en oficinas y sirven para diferentes funciones como: crear, modificar, organizar, escanear, imprimir, etc. archivos y documentos.

Rédito

Renta, utilidad o beneficio que rinde un capital.

Servlet

Su función principal es proveer páginas web dinámicas y personalizadas, utilizando para este objetivo el acceso a bases de datos, flujos de trabajo y otros recursos.

Sistema Bancario

Conjunto de instituciones que permiten el desarrollo de todas aquellas transacciones (entre personas, empresas y organizaciones) que impliquen el uso de recursos financieros.

Vencimiento

Asiento contable pendiente de alguna operación almacenado en el diario.

Anexos

Anexo # 1. Acta de liberación de Calisoft

UCI Universidad de las Ciencias Informáticas

Acta de Liberación de Productos Software

Fecha de liberación: 20 de diciembre de 2010.

Emitida a favor de: SAGEB.

1. **Datos del producto**

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

1

Ilustración 20a: Acta de liberación de los módulos y subsistemas de Quarxo emitida por Calisoft

UCI Universidad de las Ciencias Informáticas

Acta de Liberación de Productos Software

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

Delvis Echeverría Perez
Nombre y Apellidos
Responsable Calisoft

Lissett Díaz Mesa
Nombre y Apellidos
Responsable Proyecto

Ilustración 20b: Acta de liberación de los módulos y subsistemas de Quarxo emitida por Calisoft

Anexo # 1. Prototipos de Interfaz de Usuario

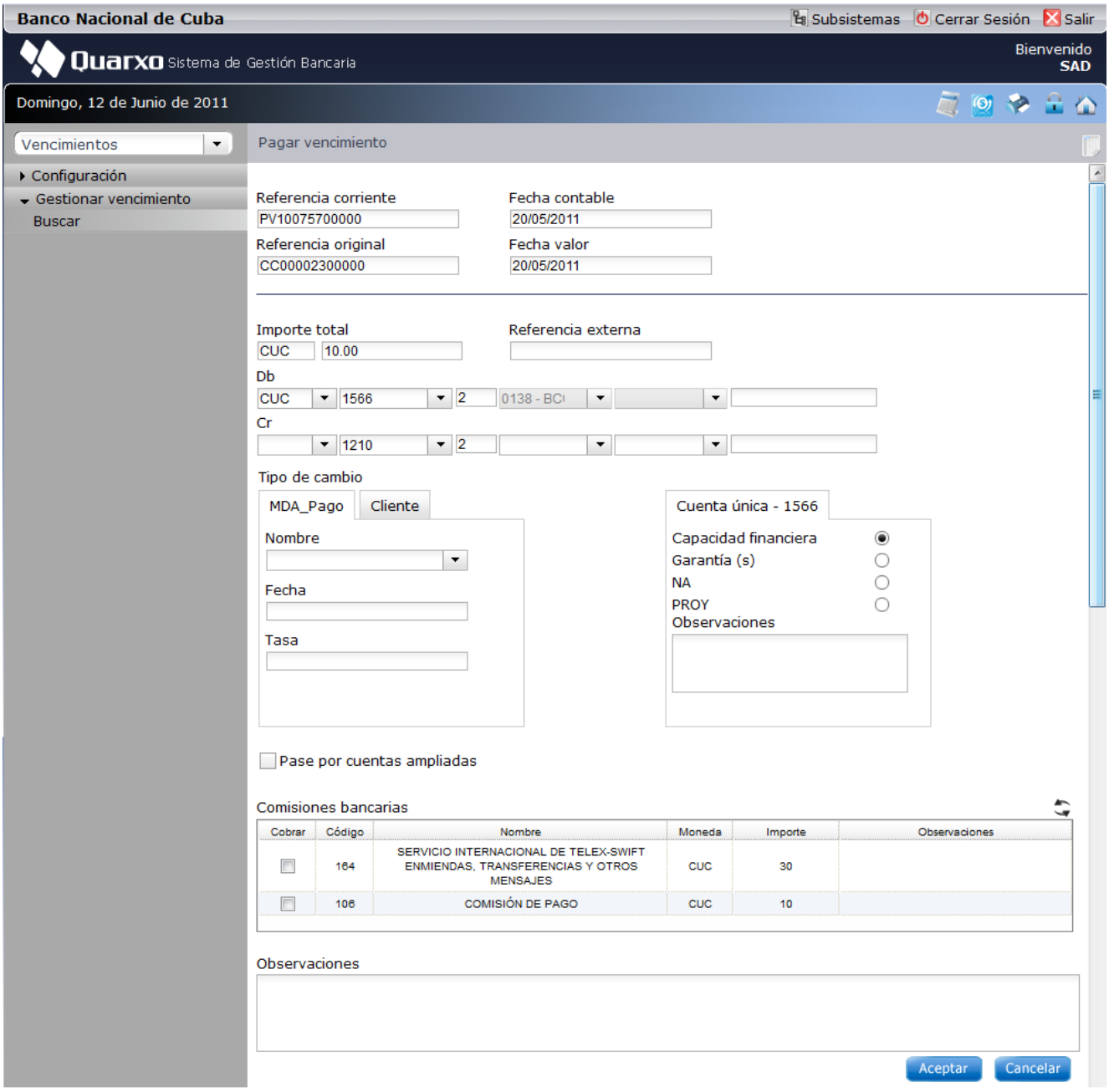

Ilustración 21: Prototipo de interfaz de usuario del caso de uso Pagar vencimiento