UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 3

DESARROLLO DEL SUBSISTEMA DEUDAS PARA EL SISTEMA DE GESTIÓN BANCARIA QUARXO.

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

Autores:

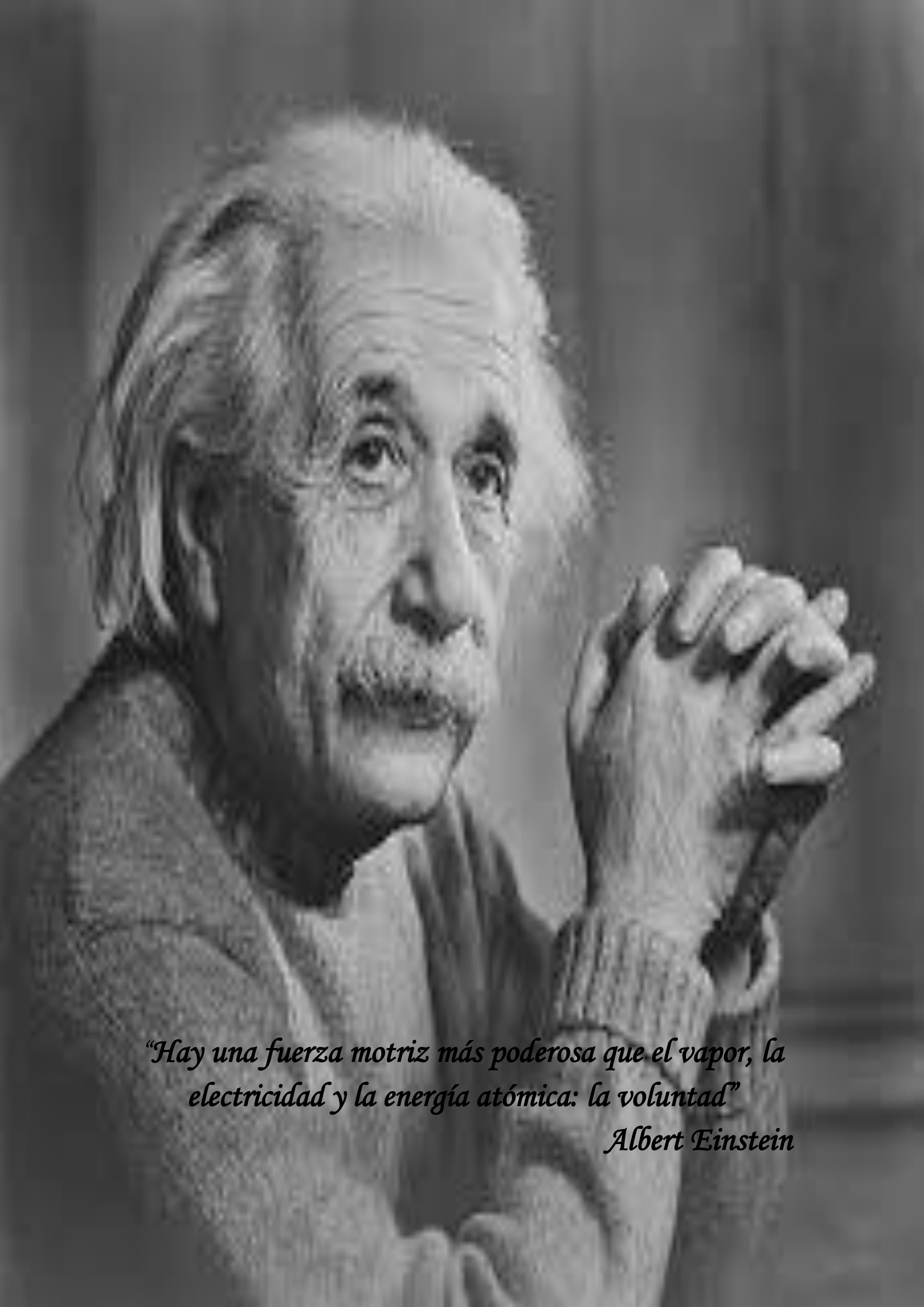
Iraidis Marina Hechavarria Felipich

Ornoldo Acebedo Arzuaga

Tutor:

Ing. Eduardo Rojas Escobar

Ciudad de la Habana, junio de 2014
"Año 56 de la Revolución"



"Hay una fuerza motriz más poderosa que el vapor, la electricidad y la energía atómica: la voluntad"

Albert Einstein

DECLARACIÓN DE AUTORÍA

Declaramos ser autores del presente trabajo y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.
Para que así conste firmamos a los ____ días del mes de _____ del año ____.

Iraidis Marina Hechavarria Felipich	Ornoldo Acebedo Arzuaga
Firma del Autor	Firma del Autor

Ing. Eduardo Rojas Escobar
Firma del Tutor.

Iraidis Marina Hechavarria Felipich:

A:

Hay personas que tocan tu vida y dejan huellas imborrables, dedicar esta tesis no es solo plasmar ideas, es poner y reconocer a todos aquellos que me impulsaron a seguir cuando el camino era incierto, es recordar a personas importantes como mis ***abuelos*** *y mis* ***padres****. La* ***familia*** *que uno escoge a medida que transita por el largo camino de la vida es especial, pues te puedes encontrar con personas lo suficientemente fuertes para mover montañas cuando piensas que todo se te viene encima. Sobre todo quisiera compartir este momento con la persona más especial de mi vida Dunia Iraidis Felipich, porque a esa maravillosa mujer le debo todo. Gracias mamá.*

Ornoldo Acebedo Arzuaga:

A:

Mis ***padres*** *por siempre creer en mí y apoyarme en todo momento.*

Toda mi ***familia*** *que se mantuvo siempre al tanto de mí en todos estos años.*

Todos los ***amigos*** *que he hecho durante estos años.*

Iraidis Marina Hechavarria Felipich:

Agradezco a mis padres Dunia y Emilio por estar presentes en mi vida, por apoyarme en cada segundo, sin ustedes no hubiera sido posible alcanzar la meta.

A Ornoldo, mi compañero de tesis y a Eduardo mi tutor por su dedicación y paciencia.

A Leo, Manuel, Evelio, Bello, Matias, Dariel y demás miembros del proyecto que tanto me han ayudado en mi preparación y superación.
A todos los que de una forma u otra contribuyeron en mi formación, en especial a los profesores que me han impartido clases a lo largo de mi trayectoria como estudiante.

A mis compañeros de estudio, en especial a Dailenis, por lograr arrastrar con esta carga por tanto tiempo.
A Aris por cambiarme la vida y creer en mí.
A Yda por ser mi amiga incondicional y estar siempre que te necesito.
A Raylith, Ivón y Alain por tantos buenos y malos momentos que hemos pasado juntos.

A Fidel y la revolución por la oportunidad que nos ofrece. Gracias a todos los que de una forma u otra han hecho posible la realización de esta tesis.

Ornoldo Acebedo Arzuaga:

A esta Revolución y a Fidel Castro Ruz por hacer de esta escuela una realidad.

A mis padres por todo apoyo que siempre me han dado a ustedes se lo debo todo y lo que soy hoy es gracias a ustedes.

A mi hermana por todo apoyo y cariño que siempre me han dado.

A mi tutor Eduardo por su gran ayuda que no olvidaré, por su amistad y por lo que he aprendido.

A mi compañera de tesis Iraidis.

A Yendrie por ser un hermano durante estos años gracias por tu amistad y preocupación, y por hacer más fácil mi paso por la escuela.

A mi segunda hermana y amiga por más de una década que siempre me ha apoyado en todo momento, sin importar el problema gracias por tus consejos aunque no estés aquí en estos momentos.

A todos las amistades que he hecho durante toda la carrera que han sido mi segunda familia de todos me llevo algo, en especial a todos los que han sido y son parte del grupo, de los cuales he aprendido muchas cosas, muchas gracias por ser mis amigos. A los amigos que conozco desde antes de entrar en la UCI como Yordan y a mis amigos de Las Tunas.

A los que me ayudaron y brindaron su ayuda Leo, Reinier, la gente del proyecto Banco y todos los profes que me han dado clases.

A este tribunal, por su paciencia, comprensión y exigencia que permitió realizar un trabajo con mayor calidad.

RESUMEN

Debido al perfeccionamiento tecnológico que se realiza en Cuba, el Sistema Bancario Cubano, particularmente el Banco Nacional de Cuba (BNC) en convenio con la Universidad de las Ciencias Informáticas (UCI), deciden desarrollar un sistema para compensar sus propósitos de responder de forma rápida y certera a la actividad contable que se realiza en la entidad. Actualmente se encuentra en ejecución la primera fase del Sistema de Gestión Bancaria (Quarxo) en el BNC. Este posee varios subsistemas que abarcan gran parte de las operaciones realizadas en el BNC, sin embargo no comprende todas las áreas del BNC, por lo cual se definió una segunda fase, encaminada al perfeccionamiento del sistema a través de la implementación de nuevas funcionalidades.

El presente trabajo a través de disciplinas como Modelado de Negocio, Análisis y Diseño, Implementación y Validación, desarrolla el subsistema Deudas incorporado al Sistema Quarxo, en aras de satisfacer al cliente. El contenido del trabajo en base a lo planteado incluye la caracterización de patrones de diseño, arquitectura, lenguajes, herramientas y tecnologías utilizadas, así como las pruebas realizadas para la validación de la solución propuesta y del modelo de diseño, obteniendo como resultado el subsistema Deudas que permite agilizar los procesos dentro de la gestión de deudas que tienen lugar en el BNC.

Palabras clave: banco, deudas, gestión, subsistema.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El Banco Nacional de Cuba es la institución facultada para dirigir todas las negociaciones bancarias en representación del estado cubano. Dentro de las funciones que realiza se encuentra la gestión de las deudas externas del país, actividad que resulta de gran importancia. (1)

El BNC realizaba sus operaciones bancarias a través del Sistema Automatizado para la Banca Internacional de Comercio (SABIC), pero la necesidad de modernización de las aplicaciones informáticas producto a cambios ocurridos en la esfera bancaria en los últimos años, así como a transformaciones específicas en sus procesos, posibilitó el desarrollo del Sistema de Gestión Bancaria Quarxo.

El sistema Quarxo fue desarrollado en la UCI por el proyecto Sistema Automatizado para la Gestión Bancaria (SAGEB) perteneciente al Centro de Informatización de Entidades (CEIGE) como parte de la búsqueda de soluciones internas factibles a problemas sociales y económicos. Además satisface las actuales necesidades operacionales del BNC con mayor rapidez, seguridad y mínimo costo, entregado a mediados del 2012.

Para el desarrollo del Sistema Quarxo, se reutilizó el núcleo de SABIC (un conjunto de tablas, triggers y procedimientos almacenados). El sistema está dividido en subsistemas que responden a los principales procesos del negocio del BNC, tales como Cartas de Créditos, Títulos Valores, Depósitos, Créditos, mensajería SLBTR (Sistema de Liquidación Bruta en Tiempo Real) y Contabilidad. (2)

La primera versión del sistema no abarca completamente las áreas del BNC, lo cual constituye una necesidad final del cliente, esto provoca demora en algunos de los procesos que se realizan en la institución. Para disminuir el tiempo de ejecución de sus procesos, se decide realizar una segunda iteración con el objetivo de añadir nuevas funcionalidades que permitan agilizar las áreas del BNC.

Una de las principales funciones del BNC es llevar el control de las deudas, tanto activas como inactivas, que contrae Cuba con otros países. El soporte que brinda Quarxo a este tipo de operaciones no es el adecuado, a continuación se muestran las causas:

- Los operadores del BNC deben registrar información que no es validada por el sistema, esto propicia la introducción de errores en los datos.

- Las funcionalidades que permiten a los operadores llevar el control de las deudas están distribuidas por varios subsistemas, las cuales no brindan soporte a todas las operaciones necesarias para gestionar las deudas en su totalidad.
- La información asociada a las deudas activas e inactivas no está debidamente estructurada, esto dificulta y demora el trabajo de los operadores, quienes en ocasiones deben solicitar la realización de cambios o la introducción de datos directamente en la base de datos.
- La información necesaria para generar los reportes que se envían al Banco Central de Cuba (BCC) está dispersa dentro de la base de datos, elevando la complejidad de las consultas SQL empleadas para la obtención de los datos en la creación de los reportes.

Luego de realizar un análisis de la problemática existente se define el siguiente **problema a resolver**: ¿Cómo agilizar los procesos de gestión de deudas dentro del Sistema de Gestión Bancaria Quarxo?, teniendo como **objeto de estudio:** la gestión de deudas en los procesos contables, enmarcados en el **campo de acción:** la gestión de deudas en el sistema Quarxo.

El **objetivo general** que se persigue con la investigación es desarrollar el subsistema Deudas para el sistema Quarxo que permita agilizar los procesos de gestión de deudas dentro del Banco Nacional de Cuba.

Para darle cumplimiento al objetivo general se trazaron los siguientes **objetivos específicos:**

- Elaborar el marco teórico de la investigación respondiendo al estado del arte del tema a investigar para lograr las bases de la investigación.
- Modelar el negocio asociado al proceso de gestión de deudas en el BNC.
- Realizar la especificación de los requisitos funcionales.
- Realizar el análisis y diseño de los módulos del subsistema Deudas.
- Realizar la implementación de las funcionalidades de los módulos del subsistema Deudas.
- Validar las funcionalidades del subsistema Deudas mediante pruebas de caja negra y pruebas de caja blanca.

A partir del problema a resolver y el objetivo general se plantea la siguiente **idea a defender:** Si se desarrolla el subsistema Deudas se permitirá agilizar los procesos de gestión de deudas del Sistema de Gestión Bancaria Quarxo.

Durante el desarrollo del trabajo es necesario el uso de algunos métodos científicos de investigación como:

Métodos Lógicos

- **Modelación:** utilizado para representar de forma visual los elementos fundamentales, así como el flujo de los principales escenarios de la propuesta de solución.
- **Analítico-Sintético:** utilizado para procesar la documentación existente sobre los principales sistemas que permiten la gestión de Deudas, con el objetivo de comprender las características fundamentales necesarias para este proceso.
- **Sistémico:** utilizado para estudiar el comportamiento del sistema en su totalidad, sus componentes, la relación entre ellos, complejidad y dinámica.
- **Análisis histórico lógico:** utilizado para realizar un estudio sobre la trayectoria del sistema, que permite conocer su desarrollo, funcionamiento, sus características y evolución.

Método Empírico:

- **Observación:** utilizado para obtener información de forma prolongada a medida que se desarrollaba el proceso, con el objetivo de analizar el comportamiento del sistema.

El presente trabajo está estructurado en tres capítulos:

Capítulo 1: Fundamentación teórica

El capítulo expone los conceptos que servirán de soporte teórico para el desarrollo de la investigación. Se realiza un análisis de los procesos relacionados con la gestión bancaria en la actualidad, tanto a nivel nacional como internacional. Se fundamenta sobre la metodología a emplear, además de los lenguajes y herramientas que contribuirán al desarrollo de la solución que se propone en el presente trabajo.

Capítulo 2: Modelado del Negocio, Requisitos, Análisis y Diseño

El capítulo se centra en las disciplinas Modelado de Negocio, Requisitos, Análisis y Diseño, describiendo los resultados obtenidos en cada disciplina. Contiene además la descripción de los patrones de diseño utilizados, así como la validación del diseño propuesto.

Capítulo 3: Implementación y Pruebas

El capítulo está destinado a la implementación de la solución propuesta a través de artefactos generados en las fases anteriores. Se realiza la descripción de los estándares de codificación utilizados. Se precisan las pruebas de software que fueron realizadas al código y a la interfaz de las funcionalidades desarrolladas. Se muestran además los resultados de la validación de la investigación.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se definen los conceptos principales empleados por los trabajadores del área de Deudas del BNC. Se muestra un estudio de varios sistemas informáticos que realizan la gestión de Deudas. Además describe el modelo de desarrollo utilizado, así como las tecnologías y herramientas a utilizar para lograr el objetivo general del trabajo.

1.1 Conceptos fundamentales del dominio del problema

En el área de Deudas del BNC el personal encargado emplea varios términos y conceptos relacionados con la investigación y que se utilizarán a lo largo del proceso, siendo elementos clave para el entendimiento del problema y del negocio.

Deuda: obligación jurídica de pagar, reintegrar o satisfacer en su totalidad un préstamo realizado ya sea en especie o en dinero, contraída entre el acreedor y el deudor, que pueden ser personas físicas o jurídicas. (3)

Deuda Activa: cualquier importe adeudado a un negocio como resultado de una adquisición de bienes o servicios en términos de crédito. (3)

Deuda Inactiva: cualquier importe adeudado a un negocio como resultado de una adquisición de bienes o servicios, pero a diferencia de la deuda activa el importe de esta no se paga, pero sus intereses continúan creciendo a medida que pasa el tiempo. (3)

Deuda Oficial: deuda contraída por la parte cubana a nombre del Estado Cubano o garantizada por él. También es conocida como deuda pública y normalmente la entidad que se endeuda bajo estas condiciones, tiene documentos legales que la facultan para actuar a nombre del Estado Cubano. (3)

Deuda Oficial Bilateral: abarca los préstamos intergubernamentales y créditos económicos obtenidos para cubrir importaciones, pagaderas y garantizados con seguros de gobierno, así como los convenios bilaterales de cobros y pagos entre Cuba y otros países, a través de sus bancos centrales. (3)

Deuda Oficial Multilateral: abarca todos los préstamos y los depósitos a corto, medio y largo plazo recibidos de instituciones internacionales y/o bancarias, que tengan un carácter multilateral, como por ejemplo: Fondo Internacional de Desarrollo Agrícola (FIDA),Fondo Común, entre otros posibles. (3)

Deudas con Instituciones Financieras: abarca los préstamos, depósitos y financiamientos bancarios no asegurados de corto, mediano y largo plazo, recibidos de instituciones bancarias financieras que no tenga carácter multilateral. (3)

Deuda con Proveedores: abarca todos los cobros anticipados y financiamientos comerciales no asegurados recibidos de proveedores no resistentes a mediano y largo plazo. El indicador ¨Cobro anticipado¨ representa el saldo pendiente de aplicación de los cobros anticipados recibidos de compradores extranjeros a favor de entidades vendedoras cubanas a cuenta de futuras exportaciones pagaderas a la visa. (3)

Calendario de pago: 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 tipos de vencimientos que contiene pueden ser de principal o de intereses ordinarios, los cuales se pueden pagar o cobrar parcialmente, trayendo consigo que se puedan contabilizar en varios vencimientos en fechas establecidas en el convenio. (4)

Vencimiento: cumplimiento del plazo o fin de un período fijado para una deuda, una obligación o un contrato. (5)

Condonar: renuncia del acreedor a sus derechos, liberando al deudor de su obligación o deuda. (6)

Mora: interés que se cobra adicionalmente al deudor en un principio para compensar el atraso en el pago o el no cumplimiento de una deuda. (7)

Renegociar: acuerdo entre el deudor y el acreedor que modifican las condiciones de pago de una deuda. (8)

Enmendar: corregir un error o modificar determinado dato. (9)

Pago: entrega de un dinero o especie que se debe. Satisfacción, premio o recompensa. (5)

Bonos: título de deuda emitido por el Estado, una empresa privada o un banco, por el cual la persona que lo compra recibe periódicamente un interés fijo; generalmente tienen un vencimiento a corto o medio plazo. (5)

1.2 Estado del Arte

Los sistemas bancarios han evolucionado al trascurrir el tiempo, implementándose sistemas para la mejora de sus procesos. A continuación se muestran los resultados

del estudio realizado de sistemas existentes para la gestión de las deudas en entidades bancarias.

1.2.1 Sistemas de Gestión Bancaria

Los sistemas de gestión bancaria permiten el desarrollo de transacciones entre personas y/o organizaciones que impliquen el empleo de dinero o la prestación de los servicios a una entidad. Estos sistemas gestionan los procesos generales y los flujos de información existentes dentro de estas instancias; facilita el trabajo de sus operadores dentro de las entidades financieras, además proporciona una mejora continua de las políticas, los procedimientos y procesos de la organización. (3)

En la investigación correspondiente a la elaboración del subsistema para la gestión de Deudas, se realizó una búsqueda bibliográfica mediante la cual se encontraron tres herramientas que realizan este proceso. De estas herramientas dos son soluciones internacionales y una solución es de factura nacional.

Soluciones Internacionales

Sistema de Gestión y Análisis de la Deuda **SIGADE versión 6:** Software privativo que está concebido para ayudar a la gestión de deuda públicas externas e internas. El SIGADE gestiona las obligaciones de deuda como las deudas contraídas o garantizadas por el Estado o las deudas reasignadas, así como las donaciones y las reorganizaciones de deuda. También puede utilizarse para realizar el seguimiento de la deuda externa privada sin garantía. El SIGADE llena plenamente las amplias necesidades de una oficina de gestión de la deuda, ya sea como *front office* (emisión de títulos de deuda), *middle office* (análisis) o *back office* (operaciones de registro y gestión). Proporciona información precisa y oportuna para la gestión de la deuda. El SIGADE permite a los administradores de la deuda realizar las siguientes operaciones: (10)

- Registrar toda la información relativa a los préstamos, las donaciones y los títulos de deuda, incluida su posible relación con proyectos y con las distintas cuentas presupuestarias nacionales.
- Crear y actualizar automáticamente los desembolsos estimados.
- Calcular automáticamente todas las tablas de amortización.
- Registrar los giros reales, las colocaciones reales y las operaciones del servicio de la deuda.

- Identificar los préstamos con atraso en el servicio de la deuda y calcular los intereses por mora.
- Elaborar una amplia gama de informes estándares y personalizados, incluidos informes para boletines estadísticos e informes con fines de validación y control.
- Analizar su cartera de deuda y formular estrategias en materia de deuda.

Tallyman: Es un software de gestión de recobro alrededor del cual gira la solución de gestión de deudas. Este sistema racionaliza el proceso de recobro, reduce las deudas, mejora el flujo de caja, reduce el aprovisionamiento y posibilita la regularización de un mayor número de clientes, lo que permite mejorar las relaciones y permite aprovechar futuras oportunidades de venta. (11)

Ventajas:

- **Implantación rápida:** proceso que permite cobrar las deudas pendientes en menos tiempo y de forma más agilizada. (11)
- **Automatización de impagos**: automatiza muchos de los procesos estándar de cobro y recupera el máximo nivel de ingresos con una intervención manual mínima, por lo que el recobro se realizan con mayor eficiencia y con menores costes. (11)
- **Control de los agentes de recobro**: la funcionalidad probada para los agentes de recobro (*Debt Collection Agency*, DCA) garantiza que el tipo de deuda que se asigna a cada uno de ellos es la que mejor se ajusta a su capacidad para que el recobro se realice correctamente, lo que supone un aumento del dinero recaudado. (11)
- **Flexibilidad**: el sistema permite aplicar tratamientos especiales a las cuentas en función del comportamiento específico de los clientes, ofreciendo también la oportunidad de emplear nuevos métodos para abordar nuevas situaciones.

Solución Nacional

Quarxo es una aplicación web desarrollada en la UCI para el BNC. La misma sustituye al SABIC como solución a los procesos más críticos del BNC y garantiza la gestión de los procesos de una forma más sencilla, segura y eficiente, a través de un conjunto de subsistemas. Posibilita generar reportes por el usuario que muestran información específica sobre la actividad contable. Cumple con las normativas y políticas bancarias cubanas.

Las funcionalidades para llevar el control de las deudas están distribuidas por varios subsistemas cómo Contabilidad y Vencimientos. La gestión de las deudas se realiza a través de la gestión de nomencladores que se encuentra en el subsistema Contabilidad donde se registra una renegociación. En el subsistema Vencimiento se registran los vencimientos, se realiza posteriormente una búsqueda del vencimiento a renegociar y se ejecuta la renegociación.

1.2.2 Valoración de los sistemas de gestión bancaria

Los sistemas internacionales SIGADE y Tallyman no se pueden integrar al sistema Quarxo, motivo por el cual sería necesario montar toda la infraestructura tecnológica para dar soporte al nuevo sistema y mantener de igual forma al sistema existente. Esta es una de las causas por lo cual sería más costoso adquirir y adaptar unos de estos sistemas, que incorporarle al sistema Quarxo nuevas funcionalidades.

Otra de las limitantes para la utilización de los sistemas SIGADE y Tallyman es que poseen licencia privativa, rompiendo así con las nuevas proyecciones del país al uso y desarrollo de aplicaciones libres con el objetivo de lograr soberanía tecnológica. Desechando además la obligación de comprar las licencias de estos productos. Por las desventajas que le infieren al país estos sistemas y por no ajustarse a las necesidades particulares del BNC, se propone el desarrollo del subsistema Deudas para el sistema Quarxo.

Como resultado del estudio realizado se define que el proceso partirá del registro de un acuerdo, el cual contiene los datos comunes tanto para las deudas activas como inactivas, a este acuerdo se le adicionará al menos un objetivo. Los objetivos se le asignarán un calendario de vencimiento, al cual se le podrán renegociar uno o varios vencimientos mediante pago, recalendarización, condonación o pago por bonos.

1.3 Modelo de desarrollo

A continuación se describe el modelo de desarrollo de software a seguir en su versión 1.2, el que estableció para dirigir el desarrollo de los proyectos pertenecientes al CEIGE. Este modelo establece varias fases, compuestas por una o varias disciplinas en las que se definen una series de artefactos a realizar.

1.3.1 Modelo de ciclo de vida de los proyectos del CEIGE Versión 1.2

El modelo de ciclo de vida que se presenta en la Figura 1 define las fases por las que transitarán los proyectos de desarrollo de software del CEIGE. (12)

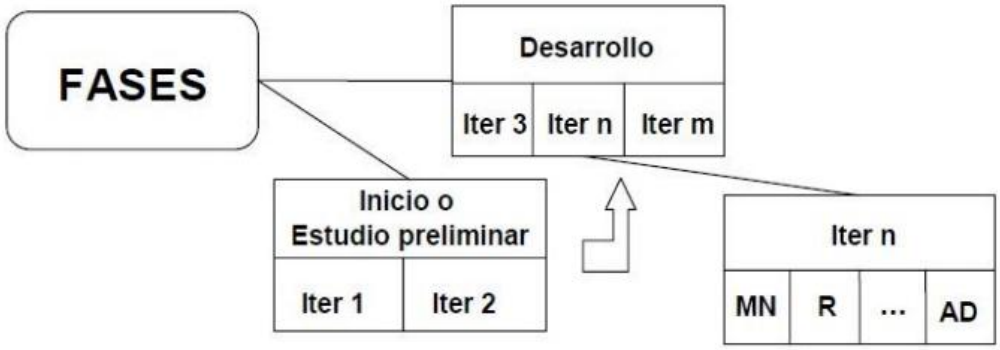


Figura 1: Fases del ciclo de vida de proyectos CEIGE

Descripción de las fases del ciclo de vida de los proyectos

Inicio o Estudio preliminar: durante el inicio del proyecto se realizan las actividades relacionadas con la planeación del proyecto a un alto nivel, la evaluación de la factibilidad del proyecto y el registro de este. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo, y decidir si se ejecuta o no el proyecto. (12)

Los objetivos de la fase son:

- Asegurar la factibilidad del proyecto.
- Establecer un plan para la ejecución del proyecto.

Hitos:

- Plan de desarrollo de software.
- Acta de inicio del proyecto firmada.

Durante la primera versión del sistema Quarxo realizó la fase de Inicio o estudio preliminar, por lo que en el presente trabajo solo se hará énfasis en la fase de Desarrollo.

Desarrollo: en esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se refinan los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto. (12)

El objetivo de esta fase es:

- Obtener un sistema que satisfaga las necesidades de los clientes y usuarios finales.

Hito:

- Producto liberado por entidad certificadora de calidad.

En esta fase se ejecutan las disciplinas Modelado de negocio, Requisitos, Análisis y diseño, Implementación y Pruebas internas.

Descripción de las disciplinas realizadas

Modelado del Negocio: En esta disciplina se identifican los procesos que se realizan en el área que se desea automatizar, se documentan todas las acciones a tener en cuenta en el análisis para el desarrollo de la solución propuesta y se generan los siguientes artefactos: Mapa conceptual, Modelo de Procesos de Negocio, Mapa de procesos de Negocio y Descripción de procesos de negocio.

Requisitos: Durante esta disciplina se identifican y describen los requisitos funcionales y no funcionales del sistema que son validados a través de prototipos y el acta de conformidad del cliente. Los artefactos que se generan son: Especificación de requisitos de software, Descripción de requisitos.

Análisis y Diseño: A partir del estudio y el análisis de los artefactos generados en el negocio y requisitos se diseña: el Modelo de Datos, Diagrama de clases del dominio, Diagramas de secuencia y el Diagrama de paquetes para posteriormente generar el Modelo de diseño, documento importante para poder tomar decisiones al desarrollar la propuesta de solución. Además se generan los casos de prueba de la aplicación, los cuales son utilizados para comprobar si la aplicación realiza lo que se espera.

Disciplina Implementación: Durante esta disciplina se tienen en cuenta los resultados del análisis y el diseño y se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, scripts y algoritmos de implementación. Al reutilizar componentes software ya implementados se realiza el desarrollo necesario para ajustar a los requisitos actuales y posteriormente realizar la integración de los componentes.

Pruebas internas: en esta disciplina se desarrollan las pruebas del grupo de calidad del centro verificando el resultado de la implementación. Permite identificar posibles errores en la documentación y el software, además el equipo de trabajo del proyecto realiza pruebas unitarias para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.

1.4 Ambiente de desarrollo de Quarxo

Para lograr un producto con la calidad requerida, se hace necesario seleccionar correctamente las herramientas y tecnologías a utilizar teniendo en cuenta el contexto en el que se aplicará la propuesta de solución.

Teniendo en cuenta las tendencias del mundo actual y lo antes mencionado se estudian un conjunto de tecnologías, herramientas, lenguajes y patrones que han sido seleccionados anteriormente por la dirección del proyecto SAGEB. A continuación se expone una breve caracterización de cada uno de estos elementos:

1.4.1 Lenguajes de Modelado y Desarrollo

Notación para la Modelación de Procesos de Negocio (BPMN[1]): estándar internacional de modelado de procesos de negocios, independiente de cualquier metodología. Permite modelar los procesos de una manera unificada y estandarizada permitiendo un entendimiento a todas las personas de una organización. Define la notación y semántica de un Diagrama de Procesos de Negocio. (13)

Lenguaje Unificado de Modelado (UML[2]): es un lenguaje unificado de modelado de procesos que está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Sirve para el modelado completo de sistemas complejos, tanto en el diseño de los sistemas de software, como para arquitectura de hardware donde se ejecute. (14)

Java: es un lenguaje de programación orientado a objetos, es sencillo, independiente de plataforma, de gran rendimiento, robusto, con capacidad multihilo. Se utiliza en un amplio abanico de posibilidades permitiendo desarrollar una gran cantidad de aplicaciones en este lenguaje. (15)

JavaScript: es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. Técnicamente es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. Los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. (16)

[1] Business Process Modeling Notation
[2] Unified Modeling Language

1.4.2 Herramientas de Desarrollo

Visual Paradigma para UML 8.0: es una herramienta CASE [3](Ingeniería de Software Asistida por Computadora) que soporta el ciclo de vida completo del desarrollo de software análisis y diseño, implementación y pruebas. Ayuda a una rápida construcción de aplicaciones de calidad y a un menor coste. Permite construir diagramas de diversos tipos, código inverso, generar código desde diagramas y generar documentación. (17)

Plataforma Java Enterprise Edition (J2EE): provee una arquitectura robusta para el desarrollo de aplicaciones empresariales en el lenguaje Java, utilizando un modelo multicapas. Esta plataforma incluye una serie de APIs (Programación de Aplicaciones de Interfaz), tecnologías, herramientas de desarrollo, especificaciones implementaciones de referencia de los servicios que brinda. Facilita el desarrollo de aplicaciones portables, escalables y seguras, ya que incluye soluciones para cuestiones comunes como transacciones, distribución de objetos remotos, soporte de servicios web, trabajo con XML. (2)

Eclipse Galileo 3.5: es un Entorno de Desarrollo Integrado (IDE[4]) para Java muy potente. Fue creado por la IBM bajo la filosofía de software libre. Se está convirtiendo en el estándar de puntera de los entornos de desarrollo para Java. Está compuesto por componentes (plugins) que se pueden o no incluir en dependencia de las necesidades del desarrollador. De hecho, existen complementos que permiten usarlo para programar en otros lenguajes aparte del Java como son PHP, Perl, Python, C/C++. (18)

Microsoft SQL Server 2005: es un servidor de base de datos basados en el modelo relacional que se caracteriza por brindar soporte para transacciones y procedimientos almacenados. Incluye un potente entorno gráfico para administración, permite además trabajar en modo cliente-servidor y promueve la escalabilidad y seguridad de la información. Requiere para su funcionamiento un sistema operativo Microsoft Windows. (19)

Apache Tomcat 6.0: es un servidor web que a su vez es una implementación de las tecnologías Java Servlet y Java Server Pages. Es desarrollado en un ambiente participativo y publicado bajo la licencia Apache versión 2. (20)

[3] Computer Aided Software Engineering
[4] Integrated Development Environment

Subversion 1.6.6: es un sistema de control de versiones libre y de código abierto conocido habitualmente como SVN que se ha expandido en gran medida dentro de la comunidad del desarrollo de software. Maneja ficheros y directorios a través del tiempo y la información radica en un árbol de ficheros en un repositorio central. Subversion puede acceder al repositorio a través de redes lo que le permite ser usado por personas en ordenadores distintos fomentando la colaboración que deriva en la reducción del tiempo de desarrollo. Existen diferentes clientes para el Subversion ya sean programas independientes como el TortoiseSVN y el Subclipse para integrarlo con Eclipse. (21)

1.4.3 Frameworks

Hibernate 3.5: es un framework objeto/relacional y un generador de sentencias SQL que permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones y un gran número de tipos de datos. Se integra en cualquier tipo de aplicación justo por encima del contenedor de datos. Permite generar las sentencias y su ejecución, manteniendo la portabilidad entre todas las bases de datos con un ligero incremento en el tiempo de ejecución y ofrece también un lenguaje de consulta de datos llamado HQL. (22)

Dojo 1.3: es una librería JavaScript de código abierto, la cual brinda una variedad de clases y widgets para facilitar el desarrollo de aplicaciones web. Dojo puede ser interpretado por diferentes navegadores web y posee un sistema de empaquetado muy parecido al del JDK (Java Development Kit) de Java. La gran variedad de clases componentes y widgets, es un punto muy importante en la elaboración en la capa presentación de la aplicación, posibilita a los programadores de interfaz una serie de funcionalidades y elementos dinámicos que facilita la programación en el cliente. Incluye varios APIs, efectos visuales, un gran cúmulo de componentes visuales basados en XHTML (Lenguaje Extensible de Marcado de Hipertexto), CSS (Hojas de Estilo en Cascada) y JavaScript que permiten el desarrollo de aplicaciones web enriquecidas. (23)

Spring 2.5: es un framework de código abierto creado para abordar la complejidad del desarrollo de aplicaciones empresariales. Cualquier aplicación Java se puede beneficiar de Spring en términos de simplicidad, capacidad de prueba, y acoplamiento. Proporciona una potente gestión de configuración basada en JavaBeans, además de una capa genérica de abstracción para la gestión de transacciones. (24)

Spring MVC 2.5.6: se utiliza para atender las peticiones web simples con navegación lineal a través de clases Controller. Entre las que se encuentran la clase SimpleFormController y MultiActionController las cuales son usadas para trabajar con un conjunto de parámetros recibidos desde una petición, permitiendo que el formulario que contiene los parámetros se cargue inicialmente generando la página donde serán procesados los datos y la atención a múltiples peticiones, permitiendo su procesamiento, estas generalmente son de un negocio determinado. (25)

Spring WebFlow 1.9.7: Permite operar la navegación de la aplicación web. Al ser limitado el flujo de páginas brindado por los frameworks MVC clásicos, surge el framework Spring WebFlow. Los flujos web en este framework son diseñados para ser auto-controlados, brindando la posibilidad de definir reglas de navegación múltiples y complejas. En Spring WebFlow un flujo controla la conversación completa (entorno nuevo que define el vacío entre Sesión y Petición) desde que inicia hasta que termina, limpiando automáticamente la memoria, siempre y cuando se termine el flujo. Permite la creación de flujos reutilizables en toda la aplicación. (25)

Conclusiones parciales

- El estudio de sistemas SIGADE y Tallyman permitió estructurar y agrupar los procesos e informaciones concernientes a las deudas de forma diferente a la que se realizaba.
- El estudio del sistema Quarxo permitió conocer problemas no identificados que afectan la gestión de Deudas en el BNC, así como algunas funcionalidades que pueden ser usadas durante la implementación del subsistema Deudas.
- El estudio del Modelo de desarrollo de software del CEIGE permitió la comprensión de sus principales características y sus fases con el objetivo de guiar el posterior diseño de la solución.
- El estudio de las tecnologías definidas por el proyecto para la obtención del sistema Quarxo demostró que estas permiten dar soporte al desarrollo del subsistema Deudas, además de encontrarse acordes con las exigencias del cliente y las políticas del centro.

CAPÍTULO 2: MODELADO DE NEGOCIO, REQUISITOS, ANÁLISIS Y DISEÑO

Introducción

En el presente capítulo se abordan las disciplinas de Modelado de Negocio, Requisitos, Análisis y Diseño para el desarrollo del subsistema Deudas mediante la transformación de los requisitos funcionales en el diseño del futuro subsistema. Se muestran además: la especificación de los requisitos; el modelo de datos que recogerá toda la información; la arquitectura base definida; el diagrama de componentes; el diagrama de paquetes para la organización de los módulos; y el diagrama de clases para el entendimiento de las clases y paquetes relacionados.

2.1 Modelado de negocio

Es la fase del modelo de desarrollo de software destinada a entender la estructura y dinámica de la organización. Se comprende cómo funciona el negocio que se desea informatizar para tener garantías de que el software desarrollado va a cumplir su propósito. Su objetivo fundamental es que tanto los clientes como los desarrolladores tengan un entendimiento común de la organización. (12)

2.1.1 Modelo conceptual

Un modelo conceptual es una representación visual de las clases conceptuales del dominio. En él se identifican los principales elementos físicos o lógicos del negocio que ayuden a entender el problema y a partir de este se obtiene el esquema conceptual de la base de datos. Su principal objetivo es establecer un vocabulario común para una mejor comprensión de los conceptos más importantes, dentro del contexto en el que se realice, facilitando el posterior levantamiento de requisitos. (12)

Para la elaboración del modelo conceptual relacionado al proceso de Renegociación se identificaron las clases conceptuales del negocio, los atributos y las relaciones existentes entre las clases. A continuación se muestra el modelo conceptual con las clases identificadas y los atributos correspondientes. (Figura 2)

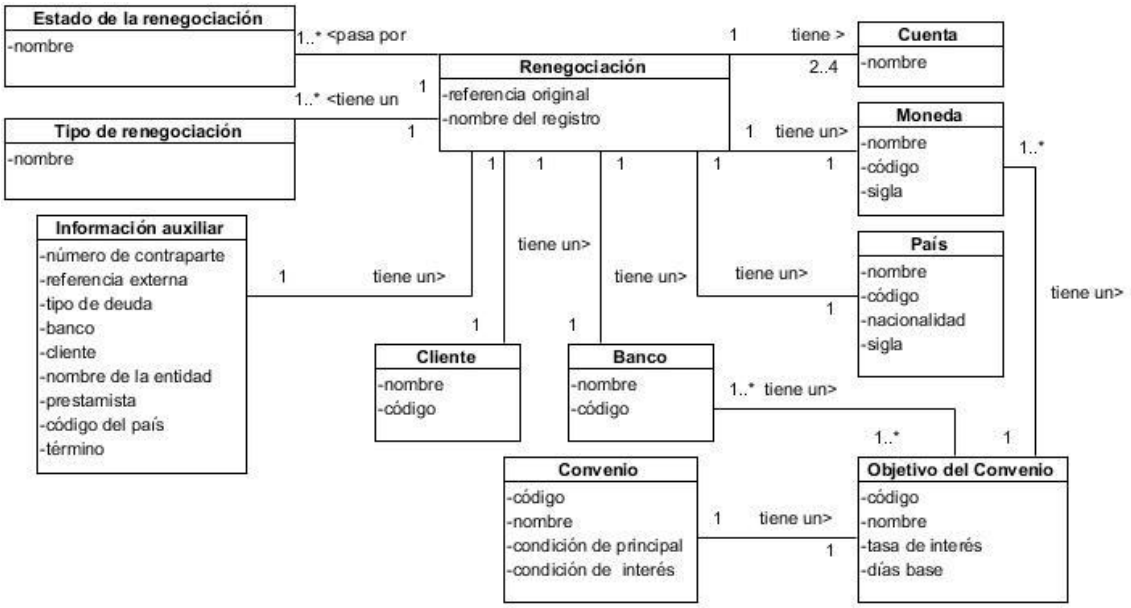


Figura 2: Modelo conceptual

2.1.2 Descripción del proceso de negocio

En el área de Deudas del BNC se encuentra el personal capacitado para llevar el control y gestión de todas las operaciones relacionadas con las deudas. El control y gestión de Deudas, sistema Quarxo, haciendo uso de las funcionalidades distribuidas en diferentes subsistemas. En esta área se tiene establecido que el acuerdo suscrito entre dos partes, tiene asociado un plazo o período pertinente para el pago de lo acordado, una vez incumplido este, ambas partes deben de renegociar las nuevas formas de pago de la deuda surgida por el incumplimiento del acuerdo.

A continuación se muestran los pasos para realizar una renegociación una vez incumplido el tiempo acordado para el pago. (**Error! Reference source not found.**)

Durante la ejecución de la presente disciplina se identificaron los procesos que se realizan en el área de Deudas, documentándose a través de los siguientes artefactos:

- CIG-QF2-N-SEG-i1301 Modelo conceptual.
- CIG-QF2-N-SEG-i1701 Reglas de negocio.

2.1.3 Reglas del negocio

Las reglas del negocio son restricciones de comportamiento y proporcionan soporte para la dirección de las actividades de negocio y se aplican a lo largo de los procesos y procedimientos. Una regla de negocio delimita un aspecto del negocio con el objetivo de establecer una estructura. Deben ser claras y escritas en un lenguaje natural,

expresadas en términos sencillos para su total comprensión. Existen diferentes tipos de reglas dentro de las que se encuentran: (26)

- Reglas Textuales
- Reglas del Modelo de Datos
- Reglas de Relación
- Reglas de Derivación

Las reglas de negocios se pueden consultar en el Anexo 1.

2.2 Requisitos

La ingeniería de requisitos del software es un proceso de descubrimiento, refinamiento, modelado y especificación. Se refinan en detalle los requisitos del sistema y el papel asignado al software inicialmente asignado por el ingeniero del sistema. Se crean modelos de los requisitos de datos, flujo de información y control, y del comportamiento operativo. Se analizan soluciones alternativas y se crea el modelo completo del análisis. (27)

2.2.1 Técnicas para la captura de los requisitos funcionales

Antes que los requisitos puedan ser analizados, modelados o especificados, deben ser recogidos a través de un proceso de obtención, por lo que fue necesario el empleo de técnicas que permitan establecer una buena comunicación con los interesados del producto, en este caso los clientes. Las técnicas que guiaron al analista durante el proceso de obtención de los requisitos de software fueron: (27)

- **Tormenta de ideas:** Consiste en la generación de ideas en un ambiente libre de críticas o juicios. Ayuda a generar una variedad de vistas del problema y a formularlo de diferentes formas. (28) Para su puesta en práctica fue necesario realizar una reunión en la cual los clientes y el equipo de desarrollo brindaron sus ideas en cuanto a la propuesta de solución, obteniéndose una visión general de las necesidades del sistema.
- **Introspección:** En esta técnica el analista se pone en el lugar del cliente e imagina cómo desearía este la aplicación de software. (28) Basado en esto el equipo de desarrollo entregó una serie de recomendaciones sobre las funcionalidades que debería tener la aplicación.

2.2.2 Descripción de los requisitos funcionales

En la especificación de los requisitos se expone una descripción completa de las funcionalidades del sistema que se va a desarrollar. Se describen todas las

interacciones que tendrán los usuarios con el software, así como la definición de los requisitos funcionales y no funcionales (o complementarios).

Los requisitos funcionales obtenidos se encuentran distribuidos en cada módulo de la siguiente forma:

Módulo gestionar Acuerdo: Este módulo tiene como objetivo principal guardar las configuraciones de los acuerdos para que estas puedan ser utilizadas en cualquier momento. En el mismo se integrarán los siguientes requisitos:

RF1: Registrar acuerdo	Consiste en registrar los datos de los acuerdos en el sistema por el usuario.
RF2: Registrar acuerdo utilizando maqueta	Consiste en registrar los datos de un nuevo acuerdo en el sistema usando como plantilla uno antes registrado.
RF3: Actualizar acuerdo	Consiste en actualizar los datos de un acuerdo creado anteriormente por el usuario.
RF4: Consultar acuerdo	Consiste en poder visualizar los datos de un acuerdo seleccionado.
RF5: Eliminar acuerdo	Consiste en eliminar el acuerdo seleccionado de la base de datos.

Tabla 1: Requisitos funcionales del módulo gestionar Acuerdo

Módulo gestionar Objetivo: Este módulo tiene como objetivo principal asignar vencimientos a los objetivos que contienes los acuerdos. En el mismo se integrarán los siguientes requisitos:

RF6: Enmendar objetivo	Corregir datos del calendario de vencimientos asociado a un objetivo.
RF7: Registrar vencimiento	Asignar un calendario de vencimientos a un objetivo.

Tabla 2: Requisitos funcionales del módulo gestionar Objetivo

Módulo Gestionar Renegociación: Este módulo tiene como objetivo principal permitir renegociar los vencimientos asociados a los objetivos. En el mismo se integrarán los siguientes requisitos:

RF8: Renegociar vencimiento	Consiste en renegociar uno o varios vencimientos de pago, asociados a un objetivo. Se puede renegociar mediante Pago, Recalendarización, Condonación o Pago por bonos.

Tabla 3: Requisitos funcionales del módulo gestionar Renegociación

A continuación se muestra la descripción del requisito funcional Renegociar vencimiento.

Precondiciones	El usuario se ha identificado y autenticado ante el sistema y tiene permisos para ejecutar esta acción.
Flujo de eventos	
Flujo básico Renegociar acuerdo.	
1	El usuario selecciona la opción Buscar.
2	El sistema muestra una lista de todos los vencimientos registrados en el
3	El usuario selecciona el acuerdo deseado y selecciona la opción renegociar
4	El sistema muestra el acuerdo seleccionado.
5	El usuario modifica los datos de la contabilización. Fecha contable. Fecha valor. Pagar. Recalendarizar. Condonar. Bonos.
6	El usuario selecciona la opción Pagar y define que porciento del importe que
7	El usuario selecciona la opción Registrar el pago.
8	El sistema valida los datos y muestra el componente de Pagos
9	El usuario selecciona las cuentas ampliadas de crédito y débito y selecciona la
10	El sistema valida los datos del pago y regresa a la vista principal.
11	El usuario selecciona la opción Aceptar.
12	El sistema valida que los datos de la renegociación sean correctos, registra los asientos de la contabilidad y muestra el mensaje: “Operación realizada
13	Concluye el requisito.
Pos-condiciones	
•	Se registran en el diario los asientos de la renegociación.
Flujos alternativos	
Flujo alternativo 6.a El usuario cancela la acción.	
1 El usuario selecciona la opción Cancelar.	
2 El sistema muestra el mensaje: ¿Está seguro que desea cancelar la operación?	
3 El usuario acepta el mensaje.	
4 Concluye el requisito.	
Pos-condiciones	
1 N/A	
Flujo alternativo 7.a El usuario selecciona la opción Pago futuro.	
1 El sistema oculta la opción Registrar el pago.	
2 Continúa en el paso 11 del flujo básico.	
Pos-condiciones	
1 N/A	
Flujo alternativo 11.a El usuario selecciona la opción Recalendarizar.	

1.	El usuario define que porciento del importe que desea recalendarizar.
2.	El usuario selecciona la opción Nuevo calendario.
3.	El sistema muestra una ventana Crear nuevo calendario
4.	El usuario introduce los datos. -Referencia externa -Importe -Desglose de cuenta -Desglose de contraparte
5.	El usuario selecciona la opción Aceptar.
6.	El sistema valida los datos y actualiza la tabla Calendarios de la vista principal.
7.	El usuario selecciona un calendario de la tabla y selecciona la opción
8.	El sistema valida que los datos de la renegociación sean válidos y muestra el componente Calendario.
9.	El usuario define los vencimientos de principal e interés.
10.	El usuario selecciona la opción Aceptar
11.	El sistema valida que el calendario definido es válido y regresa a la vista principal. Continúa en el paso 11 del flujo básico.

Pos-condiciones
1 N/A
Flujo alternativo 11.a.6 El usuario introduce datos inválidos.
1 El sistema señala los campos incorrectos.
2 El usuario corrige los datos.
3 Continúa en el paso 11 del flujo básico.
Pos-condiciones
1 N/A
Flujo alternativo 11.a.8 El usuario introduce datos inválidos.
1 El sistema señala los datos incorrectos de la renegociación.
2 El usuario corrige los datos.
3 Continua en el paso 7 del flujo alternativo 11.a
Pos-condiciones
1 N/A
Flujo alternativo 11.a.8 El usuario cancela la acción.
1 El usuario selecciona la opción cancelar.
2 El sistema regresa a la vista principal.
3 Continúa en el paso 11 del flujo básico.
Pos-condiciones
1 N/A
Flujo alternativo 11.a.11 El calendario contiene datos incorrectos.
1 El sistema muestra un mensaje indicando los errores del calendario.
2 El usuario corrige los datos.
3 Continua en el paso 10 del flujo alternativo 11.a
Pos-condiciones
1 N/A
Flujo alternativo 11.b El usuario selecciona la opción Condonar.

1.	El usuario define que porciento del importe que desea condonar.
2.	Continúa en el paso 11 del flujo básico.

Pos-condiciones	
1 N/A	
Flujo alternativo 11.c El usuario selecciona la opción Bonos.	
1.	El usuario define que porciento del importe que desea pagar mediante bonos.
2.	El usuario selecciona la opción Nuevo calendario.
3.	El sistema muestra una ventana Crear nuevo calendario
4.	El usuario introduce los datos -Referencia externa
5.	El usuario selecciona la opción Aceptar.
6.	El sistema valida los datos y actualiza la tabla Calendarios de la vista principal.
7.	El usuario selecciona un calendario de la tabla y selecciona la opción
8.	El sistema valida que los datos de la renegociación sean válidos y muestra el componente Calendario.
9.	El usuario define los vencimientos de principal e interés.
10.	El usuario selecciona la opción Aceptar
11.	El sistema valida que el calendario definido es válido y regresa a la vista principal. Continúa en el paso 11 del flujo básico.
Pos-condiciones	
1 N/A	
Flujo alternativo 11.c.3 El usuario introduce datos inválidos.	
1 El sistema señala los datos incorrectos de la renegociación.	
2 El usuario corrige los datos.	
3 Continua en el paso 2 del flujo alternativo 11.b	
Pos-condiciones	
1 N/A	
Flujo alternativo 11.c.5 El usuario cancela la acción.	
1 El usuario selecciona la opción cancelar.	
2 El sistema regresa a la vista principal.	
3 Continúa en el paso 11 del flujo básico.	
Pos-condiciones	
1 N/A	
Flujo alternativo 11.c.6 El calendario contiene datos incorrectos.	
1 El sistema muestra un mensaje indicando los errores del calendario.	
2 El usuario corrige los datos.	
3 Continua en el paso 5 del flujo alternativo 11.b	
Pos-condiciones	
1 N/A	
Flujo alternativo 11.d El usuario selecciona la Ver asientos contables.	
1.	El sistema valida que los datos de la renegociación sean correctos y muestra una ventana con los asientos de la contabilidad que genera la operación.
2.	El usuario selecciona la opción Aceptar
3.	El sistema oculta la ventana. Continúa en el paso 11 del flujo básico.
Pos-condiciones	
1 N/A	
Flujo alternativo 11.e El usuario cancela la acción.	
1 El usuario selecciona la opción Cancelar.	
2 El sistema muestra el mensaje: ¿Está seguro que desea cancelar la operación?	

3 El usuario acepta el mensaje.		
4 Concluye el requisito.		
Pos-condiciones		
1 No se registran los asientos de la renegociación.		
Flujo alternativo 12.a El usuario introduce datos incorrectos.		
1 El sistema señala los datos incorrectos de la renegociación.		
2 El usuario corrige los datos incorrectos.		
3 Continua el paso 11 del flujo básico.		
Pos-condiciones		
1 N/A		
Flujo alternativo 12.b No se realizó la configuración del pago.		
1. El sistema muestra el mensaje “Debe configurar las cuentas para el pago del porciento definido”.		
2 El usuario configura las cuentas del pago.		
3 Continúa en el paso 11 del flujo básico.		
Pos-condiciones		
1 N/A		
Flujo alternativo 12.b No se realizó la configuración del calendario para recalendarizar.		
1. El sistema muestra el mensaje “Debe configurar el calendario para la recalendarización del porciento definido”.		
2 El usuario configura el calendario para la recalendarización.		
3 Continúa en el paso 11 del flujo básico.		
Pos-condiciones		
1 N/A		
Flujo alternativo 12.b No se realizó la configuración del calendario para bonos.		
2. El sistema muestra el mensaje “Debe configurar el calendario para el pago con		
2 El usuario configura el calendario para el pago con bonos.		
3 Continúa en el paso 11 del flujo básico.		
Pos-condiciones		
1 N/A		
Validaciones		
1 N/A.		
Relaciones	**Requisitos Incluidos**	N/A.
	Extensiones	N/A.
Conceptos	N/A.	
Requisitos especiales	N/A.	
Asuntos pendientes	N/A.	

Tabla 4: Descripción del requisito funcional Renegociar vencimiento

Requisitos no funcionales:

Los requisitos no funcionales son exigencias de cualidades que se le imponen al software, los cuales especifican criterios que pueden usarse para calificar las funcionalidades de un sistema en lugar de sus comportamientos específicos. Definen

propiedades o características ya sea del sistema, del proyecto o del servicio de soporte, que no son requeridas junto con la especificación del sistema pero que no se satisface añadiendo código, sino cumpliendo con esta como una restricción del software. (29)

Los requisitos no funcionales del sistema a desarrollar son (**Error! Reference source not found.**):

Tipo de requisito	**Características**		
	PCs clientes	PCs servidores	
		Servidor 1	Servidor 2
Software	• Máquina virtual de Java 6.0 u 20 o superior. • Mozilla Firefox 3.6. • Windows XP o superior.	• Windows Server 2003. • Apache Tomcat 6.0 o superior. • Máquina virtual de Java 6.0 u 20 o superior.	• Windows Server 2003. • Microsoft SQL Server 2005 o superior.
Hardware	• Procesador Pentium IV o superior 2.0 GHZ o superior. • RAM: 256 MB (recomendado 512) Una tarjeta de red.	• Procesador: Core 2 Duo 2.0 GHZ o superior • RAM: 4 GB • Disco duro: 160 GB • UPS: 1 • Lector de CD: 1	
Funcionalidad	• El sistema debe mostrar los errores en forma de mensaje, incluyendo una descripción detallada del error.		
Seguridad	• El sistema permitirá la visualización de la información según el usuario autenticado. • El sistema implementará el uso de campos obligatorios y validaciones para garantizar la integridad de la información que se introduce por el usuario.		
Usabilidad	1. Los formularios serán estandarizados, por tanto: • Los campos de texto tendrán un tamaño estándar, de acuerdo con las dimensiones que se tengan en la página y en la medida que se llene. • No se utilizarán textos extensos para las etiquetas de la interfaz de usuario. 2. En caso de que los resultados de las consultas tengan muchas		

	coincidencias, estos se mostrarán de forma paginada en una tabla. • Se debe mostrar en la parte inferior de la tabla el total de elementos de la búsqueda encontrados. • Es necesario mostrar en la parte inferior de la tabla opciones de navegación: ir a la primera página, ir hacia atrás, ir hacia la siguiente e ir hacia la última página. 3. El menú de navegación estará disponible en todas las páginas.
Disponibilidad	• El sistema debe estar en uso durante toda la jornada laboral del BNC.

Tabla 5: Requisitos no funcionales

2.2.3 Validación de requisitos

La validación de requisitos examina las especificaciones para asegurar que todos los requisitos del sistema han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados hayan sido corregidos, y que el resultado del trabajo se ajusta a los estándares establecidos para el proceso, el proyecto y el producto. (27)

Técnicas utilizadas durante el desarrollo de la propuesta de solución:

Revisión Técnica Formal (RTF): El equipo de revisión examina el modelado de requisitos buscando errores en el contenido de la documentación, (información incompleta, requisitos contradictorios o requisitos imposibles o inalcanzables) con el objetivo de encontrar conflictos y poder trazar alternativas de solución. (27)

Prototipo: Es utilizado como modelo a escala de la solución final, para brindarle al cliente una visión más clara de cómo quedaría el producto que va a recibir. Mediante los prototipos se puede verificar si las especificaciones han sido construidas de acuerdo a los requisitos del sistema, teniendo como resultado un modelo para el desarrollo del producto atendiendo a las necesidades específicas del cliente. (27)

2.3 Análisis y Diseño

En esta disciplina se detallan los procesos del sistema, y se modelan para dar soporte a todos los requisitos, proporcionando solidez y estabilidad a la arquitectura.

2.3.1 Arquitectura del sistema

La arquitectura del sistema es la estructura global y jerárquica de los módulos que componen el sistema y la manera de interactuar entre sí de estos componentes. El sistema Quarxo basado en una arquitectura de n capas (Capa de Presentación, Capa

de Negocio, la Capa de Acceso a Datos y una capa Dominio transversal a las otras capas), tiene como objetivo separar responsabilidades, facilitar las invocaciones a funciones y un mejor desarrollo del sistema (Figura 3). (2)

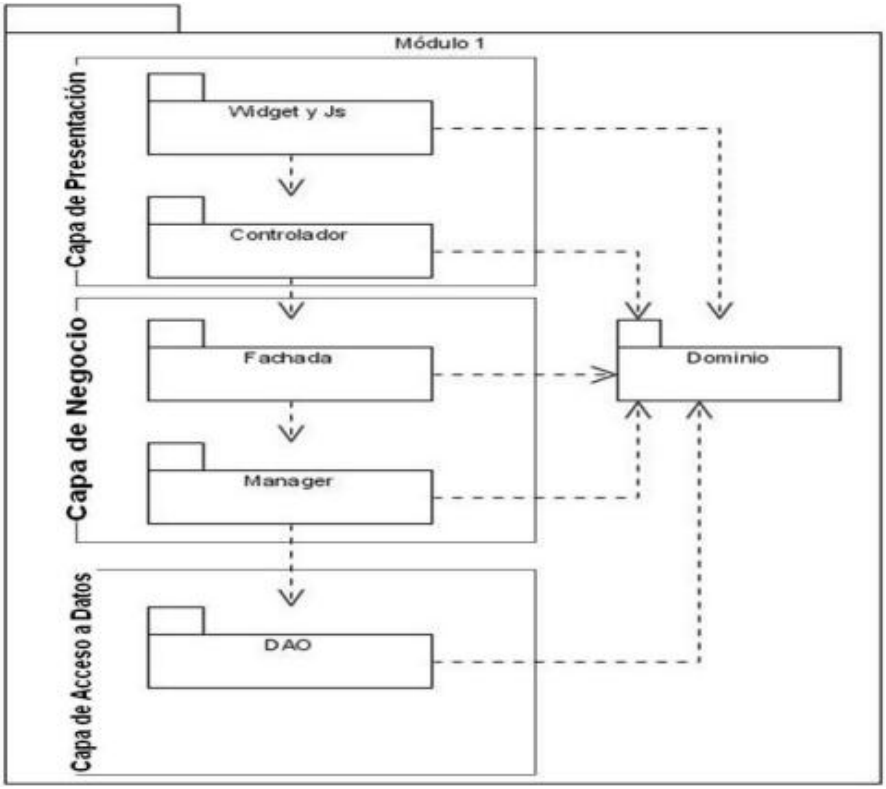


Figura 3: Arquitectura del sistema Quarxo

Capa de presentación

En esta capa se desarrolla la lógica de presentación. En el servidor se utiliza el framework Spring MVC para recibir, controlar y enviar una respuesta a las peticiones realizadas desde el cliente. Se utiliza también el framework Spring WebFlow para representar y controlar los flujos de presentación que sean complejos y reutilizables por varios módulos de la aplicación. En el cliente se utiliza la librería Dojo para generar las interfaces que interactuarán con el usuario. La capa de presentación se relaciona con la capa de negocios y la capa de dominio. (2)

Capa de negocio

Esta capa utiliza las subcapas **Fachada** y **Manager**.

- La **Fachada** es el punto de intercambio entre la capa de presentación y la capa de negocio. Esta subcapa no contiene lógica de negocio, sino que agrupa funcionalidades, según su naturaleza, para que puedan ser invocadas

desde la capa de presentación. Además delega a la subcapa Manager la realización de la lógica del negocio.

- La subcapa **Manager** tiene la jerarquía de clases suficiente para implementar el negocio de la aplicación, la cual utiliza la capa de acceso a datos para obtener datos que están persistidos, así como la capa de dominio para generar los objetos de dominios.

Desde la capa de negocio se envuelven todas las funcionalidades de la aplicación en diferentes niveles de transacción para evitar inconsistencia en los datos persistentes. Esta utiliza el contenedor de Spring framework para declarar y representar las relaciones de dependencia de cada una de las clases que existan. También usa las políticas de transacciones que propone Spring framework y Spring AOP para ejecutar las transacciones. (2)

Capa de acceso a datos

En esta capa se realizan todas las operaciones relacionadas con el gestor de base de datos. La capa de negocio interactúa con esta capa a través de sus interfaces. Para desarrollar esta capa se utiliza el patrón DAO (Objeto de Acceso a Datos). El diseño general de esta capa contiene una interface que responde a un grupo de operaciones de una entidad de dominio persistente y la correspondiente implementación de esa interface. Además se utiliza el framework de persistencia Hibernate y los módulos Spring ORM y Spring DAO. (2)

2.3.2 Modelo de Diseño

Un Modelo de Diseño es una abstracción del Modelo de Implementación y su código fuente, el cual fundamentalmente se emplea para representar y documentar su diseño. El objetivo del diseño es producir un modelo o representación de una entidad, usado como entrada esencial en las actividades relacionadas a implementación, además de especificar una solución que trabaje y pueda ser fácilmente convertida en código fuente y construir una arquitectura simple y fácilmente extensible. (30)

Modelo de datos

Los modelos de datos aportan la base conceptual para diseñar aplicaciones que hacen un uso intensivo de datos, así como la base formal para las herramientas y técnicas empleadas en el desarrollo y uso de sistemas de información. (31)

El modelo de datos se compone de tres piezas de información interrelacionadas: el objeto de datos, los atributos y la relación que conecta los objetos entre sí (Figura 4). (27)

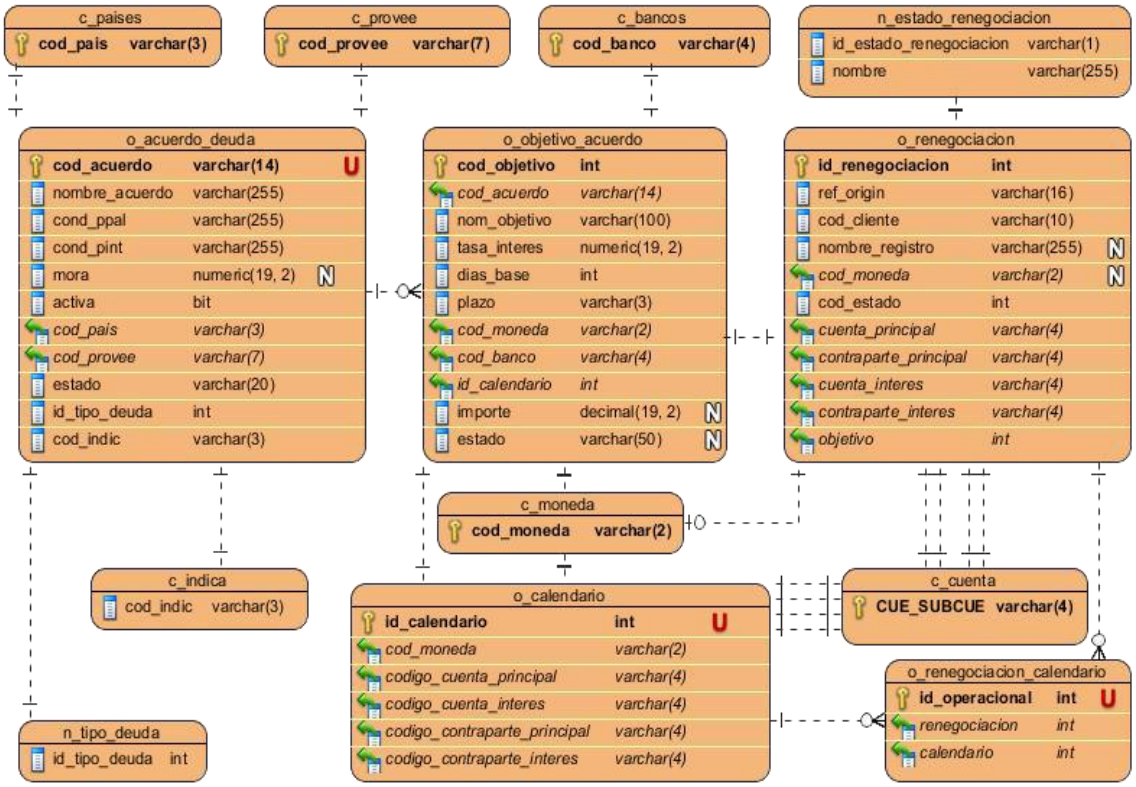


Figura 4: Modelo de datos

Descripción de las tablas:

El modelo de datos lo conforman 13 tablas, destacándose **o_acuerdo_deuda**, **o_objetivo_acuerdo** y **o_renegociación**. La tabla o_objetivo_acuerdo almacena los datos de los objetivos y la tabla o_acuerdo_deuda contiene los datos de los acuerdos, los cuales pueden tener uno o más objetivos asociados. En la tabla o_renegociación se registran los datos asociados a las formas de pago determinadas cuando se realiza una renegociación de un vencimiento.

Diagrama de paquetes

Un diagrama de paquetes es un mecanismo utilizado para agrupar elementos de UML facilitando de ésta forma el manejo de los modelos de un sistema complejo. Este muestra como está estructurado el sistema. Pueden ser simples estructuras conceptuales o estar reflejados en la implementación. Permiten dividir un modelo para agrupar y encapsular sus elementos en unidades lógicas individuales. En general, pueden tener una interfaz (métodos de clases e interfaces exportadas) y una realización de éstas interfaces (clases internas que implementan estas interfaces). Los paquetes pueden estar anidados unos dentro de otros, y unos paquetes pueden depender de otros paquetes. Se pueden utilizar para plantear la arquitectura del sistema a nivel macro. (32)

A continuación se muestra la estructura y dependencia de paquetes del módulo Renegociar del subsistema Deudas con su correspondiente explicación (Figura 5 El diagrama de paquetes que comprende al módulo Acuerdo y Objetivo se comportan de forma similar ver Tabla 18: Reglas del negocio

Anexo 2.

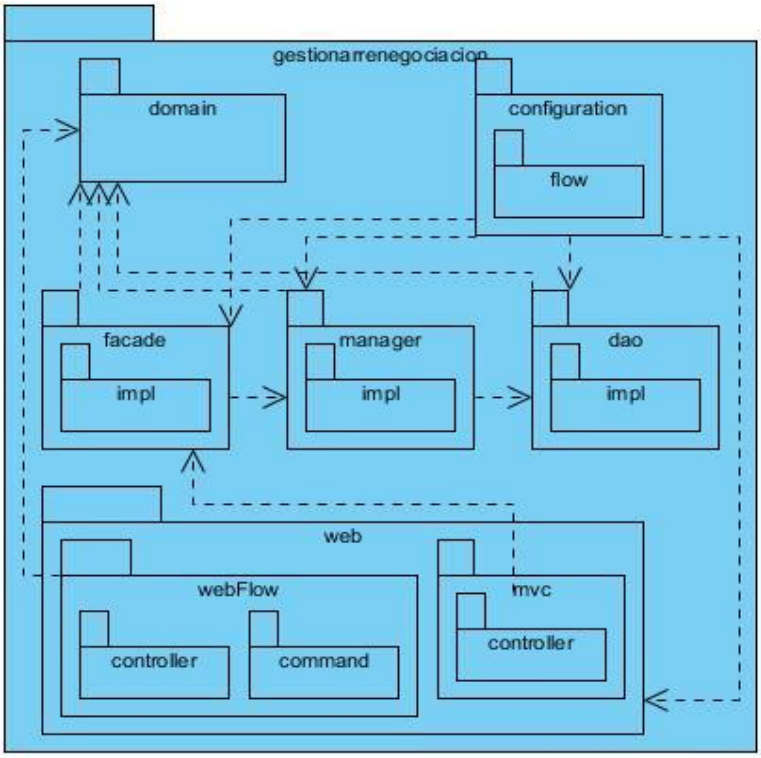


Figura 5: Diagrama de paquetes del módulo gestionar Renegociación

Descripción de los paquetes:

Paquete facade: Contiene la interfaz y la implementación que brindan las funcionalidades que se consumirán por otros módulos o por la capa de presentación.

Paquete manager: Contiene la interfaz y la implementación que brindan las funcionalidades del negocio del módulo correspondiente.

Paquete dao: Contiene las interfaces y las clases de implementación del acceso a datos del módulo.

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

Paquete domain: Paquete con las clases relacionadas con el dominio del módulo en cuestión.

Paquete mvc: Contiene los paquetes que manejan la lógica de presentación en el servidor para Spring MVC.

- **Sub-Paquete controller:** Contiene las clases que heredan de las clases Controller que brinda Spring MVC para responder a las peticiones realizadas por el cliente.

Paquete webflow: Se agrupa todo lo referente a la lógica de presentación cuando se hace uso de Spring WebFlow.

- **Sub-Paquete controller:** Contiene las clases que heredan de las clases Controller que brinda Spring webflow para responder a las peticiones realizadas por el cliente.
- **Sub-Paquete command:** Se encuentran las clases que representan los datos que son introducidos y mostrados en los formularios de la interfaz de usuario.

Paquete configuration: Contiene los ficheros de configuración del módulo en formatos XML de los diferentes contextos de Spring, en forma de mapeo de peticiones, controladores y vistas, a continuación se muestran los ficheros:

  - servlet.xml: Define el contexto de SpringMVC.
  - bussiness.xml: Define el contexto para el negocio.
  - webflow.xml: Define el contexto para Spring WebFlow.
  - dataaccess.xml: Define el contexto para acceso a datos.
- **Sub-Paquete flow:** Se encuentran presentes los archivos XML que definen los flujos para Spring WebFlow.

Diagrama de componentes

El diagrama de componentes está formado por interfaces, relaciones y componentes. Un componente de software es una parte física de un sistema que se encuentra en la computadora, puede tomarse por componente una tabla, archivo de datos, ejecutables, bibliotecas de vínculos dinámicos, documentos y cosas por el estilo (Figura 6). (14)

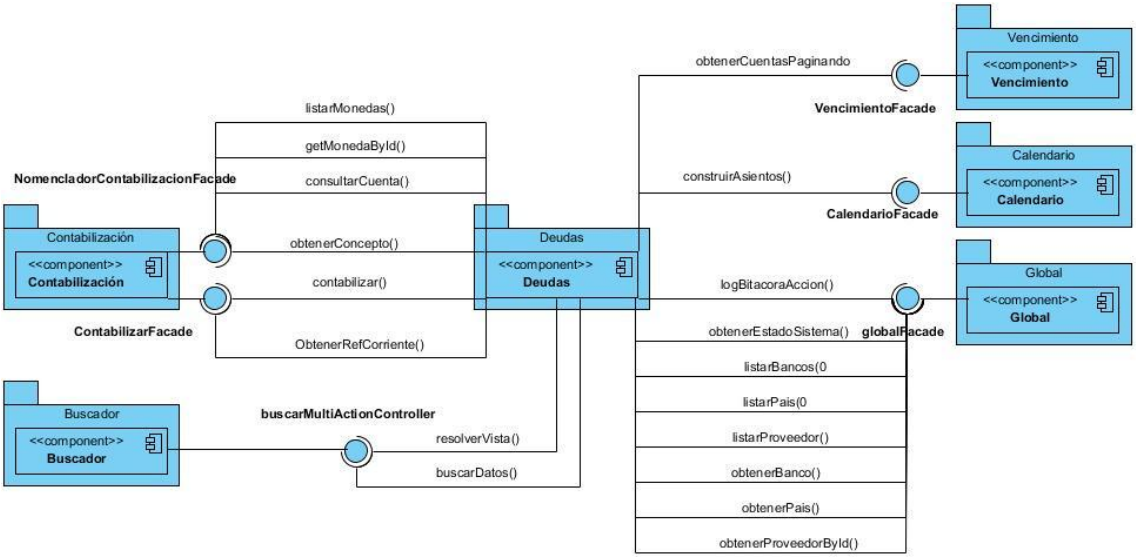


Figura 6: Diagrama de componentes

Buscador: Mediante este componente se ofrecen un grupo de funcionalidades y clases que forman en su conjunto el motor de búsqueda del sistema, que mediante una interfaz gráfica se solicitan los diferentes conceptos y criterios en dependencia del módulo donde se emplee.

Calendario: Brinda un conjunto de clases además de una interfaz gráfica, responsables de gestionar la creación de las formas de pago y los vencimientos tanto para los pagos principales como para los intereses, relacionada con una determinada operación bancaria.

Contabilización: Propone un grupo de clases necesarias para realizar la contabilización de determinadas operaciones que así lo requieran, otro grupo de clases y una interfaz gráfica para el cobro de comisiones que se asocien a determinada operación.

Global: Proporciona funcionalidades que permiten obtener datos de los nomencladores del sistema.

Vencimiento: Permite obtener información referente a las cuentas.

Diagrama de clases del diseño

Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucra el sistema, las cuales pueden ser asociativas, de herencia, de uso y de convencimiento. Soporta los requisitos funcionales del sistema en concreto y los

servicios que el sistema debería proporcionar a sus usuarios finales, es decir, muestra un conjunto de clases, interfaces, colaboraciones y sus relaciones. (33)

Un diagrama de clases del diseño está compuesto por los siguientes elementos:

- **Clase:** atributos, métodos y visibilidad.
- **Relaciones:** herencia, composición, agregación, asociación y uso.

A continuación se muestra el diagrama de clases que componen el módulo Renegociar y una breve descripción, para su mejor entendimiento. Los diagramas de clases de los módulos gestionar Acuerdo y gestionar Objetivos (ver Anexo 3).

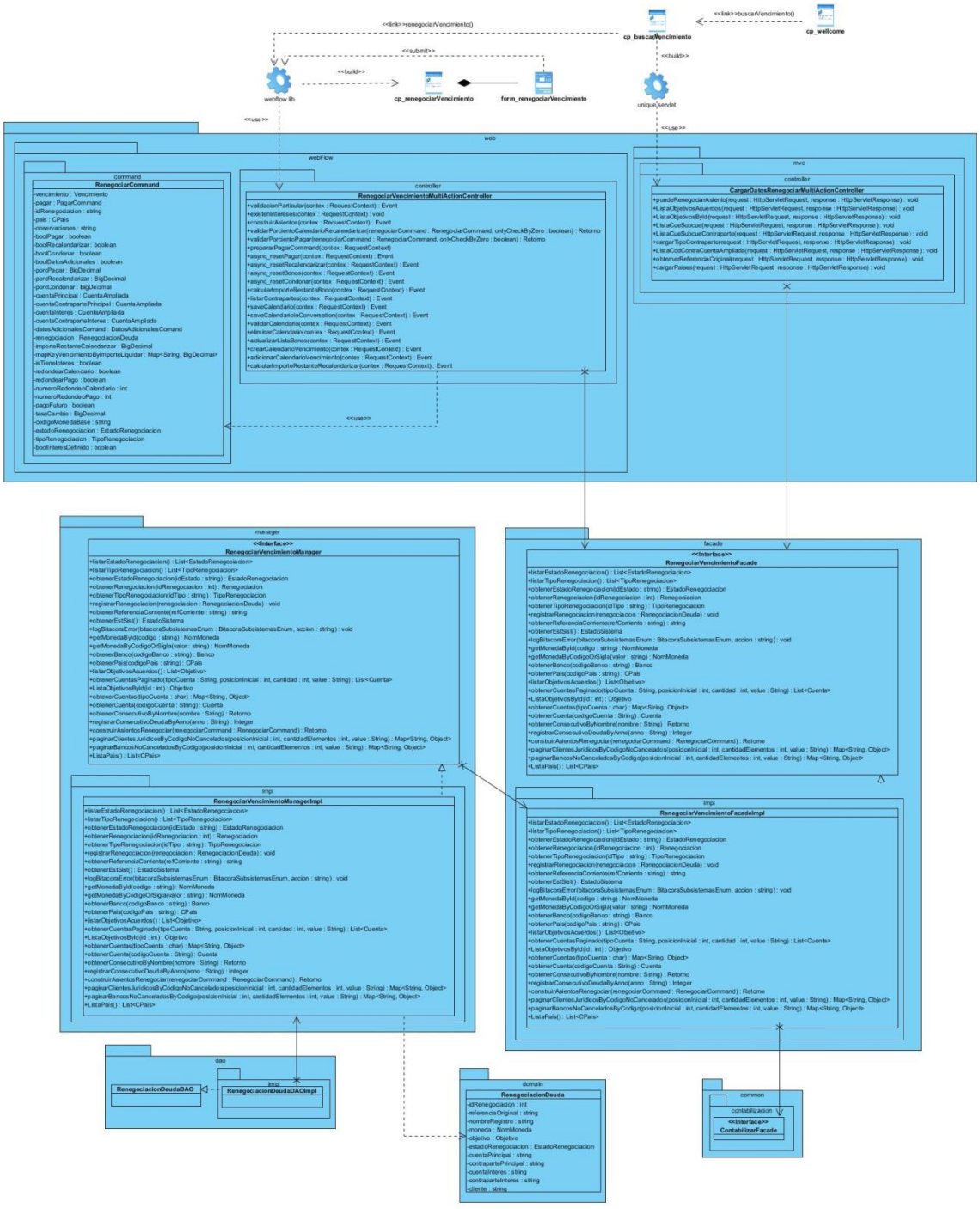


Figura 7: Diagrama de clases del diseño del módulo gestionar Renegociación

Descripción de las clases:

Client page (cp): Representan páginas web encargadas de mostrar los formularios (form) e información al usuario.

Form: Representa una colección de campos de entrada de datos, es parte de una cp.

RenegociarVencimientoMultiAction (Tabla 6)**:** Es la clase asociada al flujo de renegociar vencimiento.

CargarDatosRenegociarMultiActionController (Tabla 7)**:** Devuelve datos asociados a la renegociación.

RenegociarVencimientoFacade: A través de esta clase se puede hacer uso de todas las funcionalidades que brinda el módulo.

RenegociarVencimientoManager (Tabla 8)**:** Es la encargada de implementar todas las funcionalidades del módulo, así como hacer uso de funcionalidades existentes en otros módulos y/o subsistemas.

RenegociacionDeuda (Tabla 9)**:** Es la que permite representar en forma de objeto los datos contenidos dentro de la tabla o_renegociacion.

RenegociacionDeudaDAO: Es la encargada de gestionar la información asociada a la clase RenegociacionDeuda perteneciente al mismo módulo.

Descripción de los métodos de las clases:

RenegociarVencimientoMultiAction	
validacionParticular(RequestContext context): Event	Valida los datos de una renegociación.
ListarContrapartes(RequestContext context): Event	Lista las contrapartes de las cuentas.
calcularImporteRestanteRecalendarizar(RequestContext context): Event	Calcula el importe restante a recalendarizar.
validarPorcientoCalendarioRecalendarizar(RenegociarCommand renegociarCommand, Boolean onlyCheckByZero): Retorno	Valida el porciento definido para recalendarizar.
calcularImporteRestanteBonos(RequestContext context): Event	Calcula el importe restante a pagar mediante bonos.
eliminarCalendario(RequestContext context): Event	Elimina un calendario de vencimiento.
adicionarCalendarioVencimiento(RequestContext context): Event	Adiciona un nuevo calendario de vencimiento.
crearCalendarioVencimiento(RequestContext context): Event	Crea un calendario a un vencimiento.
async_resetPagar(RequestContext context): Event	Elimina los datos asociados al pago de la renegociación.

async_resetRecalendarizar(RequestContext context): Event	Elimina los datos asociados a una recalendarización.
async_resetBonos(RequestContext context): Event	Elimina los calendarios de bonos definidos.
async_resetCondonar(RequestContext context): Event	Elimina los datos asociados a la condonación.
construirAsientos(RequestContext context): Event	Construye los asientos de la contabilidad.
prepararPagarCommand(RequestContext context): Event	Prepara los datos para enviarlo al flujo del componente pago.
existenIntereses(RequestContext context): void	Comprueba si los calendarios asociados a la recalendarización generan intereses.
saveCalendarioInConversation(RequestContext context): Event	Envía el calendario al flujo del componente calendario.
saveCalendario(RequestContext context): Event	Asocia el calendario que devuelve el componente calendario a la renegociación.
validarPorcientoPagar(RenegociarCommand renegociarCommand,Boolean onlyCheckByZero): Retorno	Valida el porciento definido para el pago.

Tabla 6: Descripción de la clase RenegociarVencimientoMultiAction

CargarDatosRenegociarMultiActionController	
puedeRenegociarAsiento(HttpServletRequest request,HttpServletResponse response): void	Verifica si el vencimiento seleccionado puede ser renegociado.
ListaObjetivosAcuerdos(HttpServletRequest request,HttpServletResponse response): void	Lista todos los objetivos asociados a los acuerdos registrados.
ListaObjetivosById(HttpServletRequest request, HttpServletResponse response): void	Lista todos los objetivos dado el identificador del acuerdo.
ListaCueSubcue(HttpServletRequest request, HttpServletResponse response): void	Lista las cuentas.
ListaCueSubcueContraparte(HttpServletRequest request,HttpServletResponse response): void	Lista contrapartidas de cuenta.
cargarTipoContraparte(HttpServletRequest request, HttpServletResponse response): void	Devuelve el tipo de Contra Cuenta.
ListaCodContraCuentaAmpliada(HttpServletRequest request,HttpServletResponse response): void	Lista el código de la cuenta ampliada por tipo de contraparte.
obternerReferenciaOriginal(HttpServletRequest	Devuelve una referencia según su

request, HttpServletResponse response): void	inicial.
cargarPaises(HttpServletRequest request,HttpServletResponse response): void	Lista los países.

Tabla 7: Descripción de la clase CargarDatosRenegociarMultiActionController

RenegociarVencimientoManager	
ListaEstadoRenegociacion():List<EstadoRenegociacion>	Lista los estados de la renegociación.
ListaTipoRenegociacion():List<TipoRenegociacion>	Lista los tipos de renegociación.
obtenerRenegociacion(int idRenegociacion): RenegociacionDeuda	Devuelve una renegociación dado su identificador.
registrarRenegociacion(RenegociacionDeuda renegociacion): void	Registra una nueva renegociación.
obtenerReferenciaCorriente(String refCorrienete): String	Devuelve una referencia corriente.
obtenerEstSist():EstadoSistema	Devuelve el estado del sistema.
logBitacoraError(BitacoraSubsistemasEnum bitacoraSubsistemasEnum, String accion): void	Escribir una traza dentro de la bitácora de errores del sistema.
getMonedaById(String codigo): NomMoneda	Devuelve una moneda dado el id.
getMonedaByCodigoOrSigla(String valor): NomMoneda	Devuelve una moneda dado el código.
obtenerBanco(String codigoBanco): Banco	Devuelve un banco dado el código.
obtenerPais(String codigoPais): CPais	Devuelve un país dado el código.
ListaObjetivosAcuerdos():List<Objetivo>	Lista todos los objetivos.
ListaDesgloses(String moneda,String subCuenta, String tipContra, String codContra, int posicionInicial,int cantidadElementos, String value): Map<String, Object>	Lista los desgloses.
obtenerCuentasPaginado(String tipoCuenta, int posicionInicial,int cantidad,String value): List<Cuenta>	Lista las cuentas.
ListaObjetivosById(int id): Objetivo	Lista todos los objetivos dado el id del acuerdo.

obtenerCuentas(char tipoCuenta): Map<String, Object>	Lista las cuentas dado el tipo de cuenta.
obtenerCuenta(String codigoCuenta): Cuenta	Lista las cuentas dado el código de cuenta.
obtenerConsecutivoByNombre(String nombre): Retorno	Genera un consecutivo dado una cadena de caracteres.
registrarConsecutivoDeudaByAnno(String anno): Integer	Registra un consecutivo en la base de datos.
construirAsientosRenegociar(RenegociarCommand renegociarCommand): Retorno	Crea un asiento para una renegociación.
paginarClientesJuridicosByCodigoNoCancelados(int posicionInicial, int cantidadElementos, String value): Map<String, Object>	Lista los clientes jurídicos dado el código.
paginarBancosNoCanceladosByCodigo(int posicionInicial, int cantidadElementos, String value): Map<String, Object>	Lista los clientes jurídicos.

Tabla 8: Descripción de la clase RenegociarVencimientoManager

Descripción de los atributos de la clase:

RenegociacionDeuda	
Atributos	**Descripción**
idRenegociacion: int	Identificador de la renegociación.
referenciaOriginal: String	Referencia original.
cliente: String	Código de cliente
nombreRegistro: String	Nombre del registro
objetivo: Objetivo	Objetivo
moneda: NomMoneda	Código de moneda
estadoRenegociacion: EstadoRenegociacion	El estado de la renegociación.
cuentaPrincipal: Cuenta	Cuenta del principal.
contrapartePrincipal: Cuenta	Contraparte del principal.
cuentaInteres: Cuenta	Cuenta de interés.
contraparteInteres: Cuenta	Contraparte de la cuenta interés.

Tabla 9: Descripción de la clase RenegociacionDeuda

Diagrama de Secuencia

Los diagramas de secuencia son utilizados para modelar los aspectos dinámicos de los sistemas, muestran la realización de un flujo o escenario concreto de un requisito

en términos de interacción entre objetos del diseño. Se les atribuye una gran importancia debido a la compresión del sistema a la que puede llegarse a través de ellos. (34)

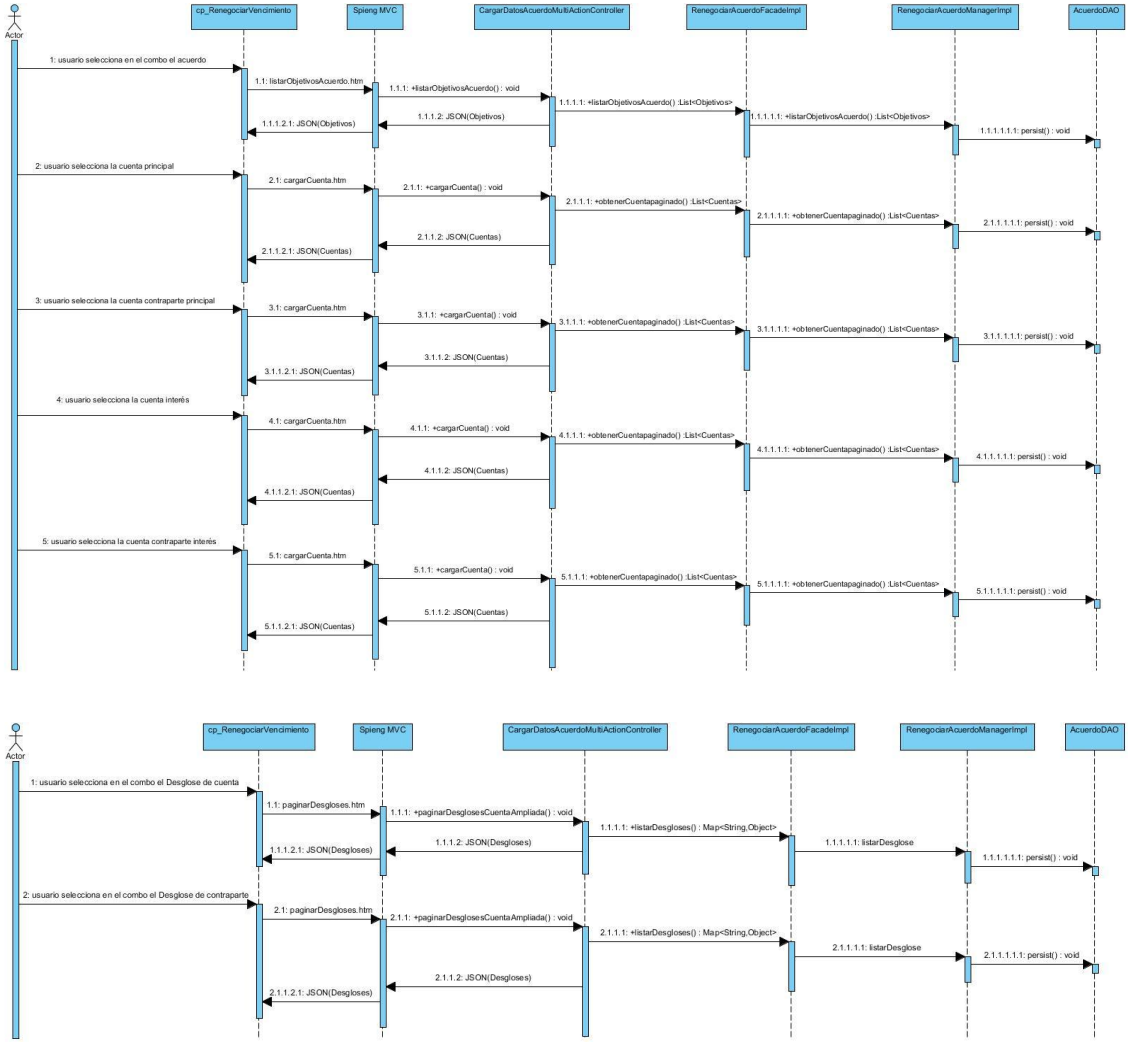


Figura 8: Diagrama de secuencia Renegociar vencimiento

Para consultar los diagramas de secuencia ver Anexo 4.

2.3.3 Patrones de diseño empleados

Son un buen método para resolver pequeños problemas que pueden adaptarse a una variedad mucho más amplia de problemas específicos facilitando la reutilización del código. (27)

Patrones Generales para Asignar Responsabilidades (GRASP)

GRASP (Patrones Generales de Software para Asignación de Responsabilidades) es un conjunto de patrones que permite la asignación de responsabilidad en parámetros

útiles para el diseño del producto. Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos expresados en forma de patrones. (35)

GRASP destaca 5 patrones principales: Experto, Creador, Bajo acoplamiento, Alta cohesión, Controlador.

Patrón Experto: se asigna una responsabilidad al experto en la información, evidenciado este patrón en la definición de las interfaces DAO y sus implementaciones, que se especializan en realizar las operaciones de acceso a datos referente a una determinada entidad. (35)

Patrón Creador: Guía la asignación de responsabilidades relacionadas con la creación de objeto, evidenciándose el mismo en la clase interna de Spring, Abstract Factory que tiene la función de construir e instanciar un conjunto de objetos relacionados, permitiendo definir clases abstractas para crear familias de objetos. (35)

Patrón Bajo Acoplamiento: Este patrón se manifiesta con la definición de interfaces e implementaciones, como por ejemplo en la interfaz RenegociarVencimientoFacade y su implementación RenegociarVencimientoFacadeImpl permitiendo que clases como RenegociarVencimientoMultiAction y CargarDatosRenegociarMultiActionController se relacionen únicamente con ellas para realizar sus operaciones, reduciendo el impacto de cambios posteriores en el negocio del sistema. (35)

Patrón Alta Cohesión: este patrón es utilizado en el diseño de cada uno de los módulos del subsistema, manejando la menor cantidad de entidades posibles, de manera tal que a las clases pertenecientes a las diferentes capas se les asignan las responsabilidades necesarias y bien delimitadas. (35)

Patrón Controlador: las clases controladoras RenegociarVencimientoMultiAction y CargarDatosRenegociarMultiActionController constituyen un ejemplo de la aplicación de este patrón, centralizando las actividades que reciben y respondiendo a las peticiones de la interfaz de usuario, funcionando como intermediaria entre la lógica del proceso y la presentación. (35)

Patrones Estructurales de Gang of Four (GoF)

Los patrones Estructurales de Gang of Four (GoF) se dividen en los tres siguientes grupos:

Los patrones **creacionales** permiten la inicialización y configuración de objetos. (36)

- **Patrón Singleton:** para garantizar que una clase sólo tiene una única instancia y que proporcione así un punto de acceso global a la misma se declaran en los archivos de configuración de Spring los beans evidenciándose el use de este patrón.

Los patrones **estructurales** describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. (36)

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

Los patrones de **comportamiento** más que describir objetos o clases, describen la comunicación entre ellos. (36)

- **Patrón Cadena de Responsabilidad:** cuando se solicita información a través de una petición realizada desde la vista que se encuentra en la base de datos, este patrón establece una cadena dentro del subsistema, de tal forma que una petición desde el cliente es manejada por una clase controladora, luego por la clase fachada, el manager y finalmente la clase DAO. Este flujo llega siempre hasta clase que le puede dar respuesta por sí misma.
- **Patrón de Objeto de Acceso a Datos (DAO):** es evidenciado en la definición de las clases DAO como AcuerdoDAO, TipoDeudaDAO y ObjetivoDAO las cuales facilitan las funcionalidades específicas a realizar sobre la base de datos.

2.4 Validación del diseño

Los objetivos primarios para las métricas orientadas a objetos no son diferentes de aquellos de las métricas desarrolladas para el software convencional: (27)

- Para entender mejor la calidad del producto.
- Para evaluar la efectividad del proceso.
- Para mejorar la calidad del trabajo llevado a cabo al nivel del proyecto.

Las medidas y métricas para una clase individual, la jerarquía de clases y las colaboraciones de clases poseen un alto valor en la evaluación de la calidad del diseño. (27)

Lorenz y Kidd separan las métricas basadas en clases en cuatro amplias categorías: tamaño, herencia, valores internos y valores externos. (27).

Para la validación se utilizará una de las métricas propuestas por Lorenz y Kidd, Tamaño de clase (TC). El tamaño general de una clase puede medirse determinando las siguientes medidas:

- El total de operaciones (métodos tanto heredados como privados de las clases), que se encapsulan dentro de la clase. (TOC)
- El número de atributos (atributos tanto heredados como privados de la clases), encapsulados por la clase. (RC)

Para evaluar la calidad del diseño propuesto se tendrá en cuenta la cantidad de procedimientos y cantidad de instancias que poseen las clases siguientes (Tabla 10):

Clases utilizadas	Cantidad de procedimientos	Cantidad de relaciones
ActualizarAcuerdoSimpleFormController	3	1
CargarDatosAcuerdoMultiActionController	8	1
ConsultarAcuerdoAbstractCommandController	1	1
RegistrarAcuerdoSimpleFormController	3	1
GestionarAcuerdoManagerImpl	17	5
AcuerdoDAOImpl	2	0
ContabilizarObjetivoMultiAction	4	1
CargarDatosObjetivoMultiActionController	5	1
GestionarObjetivoManagerImpl	8	8
CargarDatosRenegociarMultiActionController	9	1
RenegociarVencimientoMultiAction	20	1
RenegociarVencimientoMangerImpl	26	10

Tabla 10: Clases utilizadas para las métricas TC y RC

2.4.2 Tamaño Operacional de Clase (TOC)

Variables TOC

Total de clases	12
Promedio (PO)	106
2* Promedio (2*PO)	8,833333333

Tabla 11: Variables de comparación en las métricas de TOC

Atributos de calidad	Descripción	Aumento del TOC	Categoría	Criterio
Responsabilidad	Responsabilidad que posee una clase en un marco conceptual correspondiente al modelado de la solución propuesta.	Implica un aumento de la responsabilidad asignada a la clase.	Baja	< =PO
			Media	Entre PO y 2* PO
			Alta	> 2* PO
Complejidad de implementación	Grado de dificultad que tiene implementar un diseño de clases determinado.	Implica un aumento de la complejidad de implementación de la clase.	Baja	< =PO
			Media	Entre PO y 2* PO
			Alta	> 2* PO
Reutilización	Significa cuán reutilizada es una clase o estructura de clase dentro de un diseño de software.	Implica una disminución del grado de reutilización de la clase.	Baja	> 2* PO
			Media	Entre PO y 2* PO
			Alta	< =PO

Tabla 12: Descripción de atributos de calidad de la métrica de TOC

Resultados de la aplicación de métrica TOC

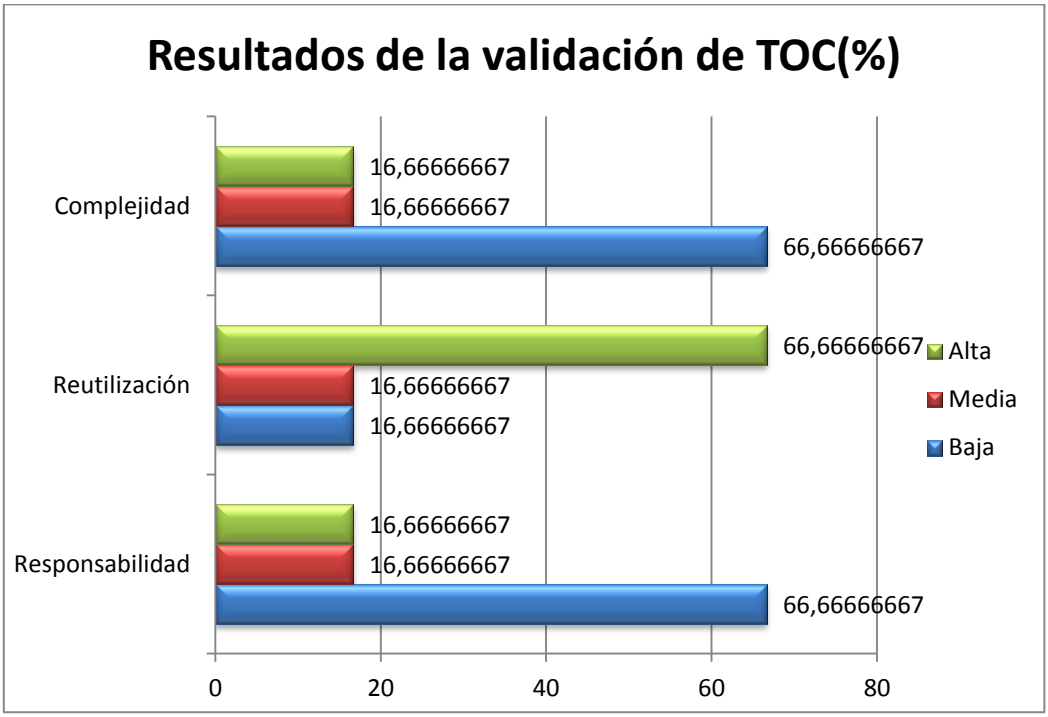


Figura 9: Resultados obtenidos al aplicar la métrica TOC

Después del análisis realizado basándose en los atributos de calidad definidos se evidencia una baja complejidad en la implementación (66,67%), una alta reutilización (66,67%) y una baja responsabilidad asignada a las clases (66,67%).

2.4.3 Relaciones entre Clases (RC)

Variables RC	
Total de clases	12
Promedio (PO)	31
2* Promedio (2*PO)	2,583333333

Tabla 13: Variables de comparación en las métricas de RC

Atributos de calidad	Descripción	Aumento del RC	Categoría	Criterio
Acoplamiento	Dependencia o interconexión de una clase o estructura de clase respecto a otra.	Implica un aumento del acoplamiento de la clase.	Ninguno	0
			Bajo	1
			Medio	2
Complejidad del mantenimiento	Nivel de esfuerzo necesario para	Implica un aumento de la complejidad de	Alta	>2

	sustentar, mejorar o corregir el diseño de software propuesto. Puede influir significativamente en los costes y la planificación del proyecto.	mantenimiento de la clase.	Baja	<= PO
			Media	Entre PO y 2*PO
Reutilización	Significa cuán reutilizada es una clase o estructura de clase dentro de un diseño de software.	Implica una disminución en el grado de reutilización de la clase.	Alta	> 2*PO
			Baja	>2* PO
			Media	Entre PO y 2*PO
Cantidad de pruebas	Dependencia o interconexión de una clase o estructura de clase respecto a otra.	Implica un aumento de la cantidad de pruebas de unidad necesarias para probar una clase.	Alta	<= PO
			Baja	<= PO
			Media	Entre PO y 2*PO

Tabla 14: Descripción de atributos de calidad de la métrica de RC

Resultados de la métrica RC

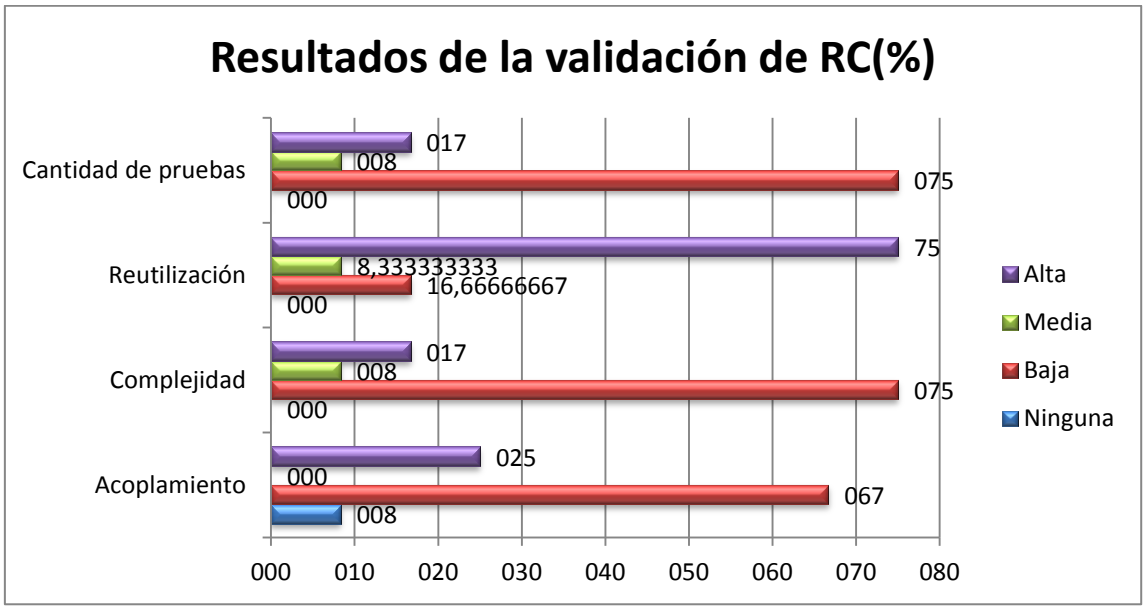


Figura 10: Resultados obtenidos al aplicar la métrica RC

Luego de aplicarse la métrica de diseño RC, los resultados obtenidos de la evaluación basándose en los atributos de calidad definidos son positivos, evidenciándose una baja cantidad de pruebas (75%), una alta reutilización (75%), una baja complejidad en el mantenimiento (75%) y un bajo acoplamiento de las clases (66,67%).

2.4.4 Matriz de inferencia de indicadores de calidad

Se representan los atributos de calidad de las métricas usadas para evaluar la calidad del diseño de los componentes de la solución propuesta. La misma asigna a la intersección de la casilla atributo/métrica el valor de 1 en caso de que los resultados fueran positivos para esa atributo en esa métrica, 0 en caso que fueran negativos y (-) en caso de que ese atributo de calidad no esté presente en la métrica analizada.

Atributos de calidad	TOC	RC	Promedio
Complejidad de implementación	1	(-)	1
Reutilización	1	1	1
Responsabilidad	1	(-)	1
Cantidad de pruebas	(-)	1	1
Acoplamiento	(-)	1	1
Complejidad de mantenimiento	(-)	1	1

Tabla 15: Matriz de inferencia de indicadores de calidad

Como reflejan los resultados recogidos en la matriz de inferencia de indicadores de calidad todos los resultados son positivos, por lo cual el equipo de desarrollo puede continuar con la próxima disciplina que propone el modelo de desarrollo.

Conclusiones parciales

- Como resultado de la disciplina Requisitos se obtuvo la descripción un total de 8 requisitos funcionales agrupados en tres módulo validados por el cliente.
- Los patrones de diseño utilizados, permiten ganar en reutilización y brindan robustez a la solución.
- La elaboración de los diagramas de la disciplina Análisis y Diseño facilitó el entendimiento previo de los requisitos para la implementación.
- La aplicación de las métricas TOC y RC permitió indicar que el diseño realizado cuenta con una buena calidad, mostrando valores entre el 66% y 75% en cada atributo de calidad.

CAPÍTULO 3. IMPLEMENTACIÓN Y PRUEBAS

Introducción

El presente capítulo se encarga de dar cumplimiento a las disciplinas: implementación y pruebas internas. Se exponen los estándares de codificación utilizados, así como los resultados de realizar pruebas de caja blanca con el framework JUnit y pruebas de caja negra utilizando la técnica Partición de Equivalencia. Por último se muestran los resultados de la validación de la propuesta de solución.

3.1 Implementación

La implementación es el principal flujo de trabajo en la fase de construcción. En él se describe cómo los elementos del modelo de diseño se implementan en función de componentes y por ende en piezas más manejables por el lenguaje del programador. Tiene como objetivo implementar de cada una de las clases significativas del diseño. Su importancia se debe a que se obtiene como consecuencia un sistema ejecutable, siendo uno de los principales objetivos en el desarrollo de software. (37)

3.1.1 Estándares de codificación

El propósito fundamental de los estándares de codificación es que el software desarrollado tenga una arquitectura y un estilo consistente, independiente del autor, permite que todos los participantes lo puedan entender en menos tiempo y que el código en consecuencia sea mantenible. (38)

Los estándares de codificación se definen por el equipo de desarrollo para lograr estandarización en la programación del software. Estos se basan en la estructura y apariencia física de un programa con el fin de facilitar la lectura, comprensión, mantenimiento del código, reutilización a lo largo del proceso de desarrollo de un software y no en la lógica del negocio. (38)

Los estándares de codificación que se utilizaron son los definidos por el proyecto Sistema de Gestión Bancaria Quarxo desde su primera fase para el desarrollo del producto Quarxo.

PascalCasing: Para nombrar las clases, definiendo que deben comenzar con letra mayúscula, y en caso de estar compuesta por más de una palabra, todas deben iniciar con mayúscula. (Ejemplo: RenegociarAcuerdoMangerImpl).

CamelCasing: Para nombrar los métodos y variables que se encuentran en las clases, definiendo que deben comenzar con minúscula y en caso de contener más de una palabra deben comenzar con letra mayúscula. (Ejemplo: listarObjetivosAcuerdos()).

3.2 Pruebas Internas

Luego de concluida la disciplina Implementación se realizan las actividades de la disciplina Pruebas Internas, desarrollando pruebas por el grupo de calidad interna de CEIGE y el equipo de desarrollo, verificando el resultado de la implementación.

Una vez generado el código fuente, el software debe ser probado para descubrir y corregir el máximo de errores posibles antes de su entrega al cliente. Es aquí donde se aplican las técnicas de pruebas del software, las cuales facilitan una guía sistemática para diseñar pruebas que comprueben la lógica interna de los componentes software y verifiquen los dominios de entrada y salida del programa para descubrir errores en la funcionalidad, el comportamiento y rendimiento. (27)

3.2.1 Pruebas de Caja blanca

La prueba de caja blanca del software se basa en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado. (27)

Aplicación de la prueba de Caja blanca:

Dentro de las pruebas de caja blanca se realizaron pruebas unitarias con el framework JUnit, que permite evaluar el funcionamiento de cada uno de los métodos de las clases, comprobando si se comporta de manera esperada, en caso que el valor de salida sea diferente al valor de entrada devolverá un error en el método correspondiente. Para lograr el control de la entrada y salida se utilizó las librerías EasyMock en complemento con JUnit. Estas pruebas se realizan creando un diseño o prototipo de un objeto fantasma, de forma que se compruebe que los resultados devueltos a ciertas entradas introducidas sean los correctos. (39)

Para la realización de esta prueba se realizaron los siguientes pasos:

Crear una clase JUnit Test Case, seleccionando los métodos setUp (donde se inicializan todos los atributos) y tearDown (encargado de las tareas a realizar en cada test), así como los métodos que desea probar.

- Se crea un objeto de la clase donde se encuentra el método a probar y tres mocks configurados para simular un objeto que reemplaza de una manera sencilla que posibilita la librería EasyMock, siendo en este caso: moHttpServletRequest, moHttpServletResponse, moHttpSession (Figura 11).
- Se crea un Test Suite seleccionando todos los Test Case que se desean probar y ejecutamos la prueba (Figura 12**Error! Reference source not found.**).

```
public class CargarDatosRenegociarMultiActionControllerTestJUnit extends TestCase{
    CargarDatosRenegociarMultiActionController cargarDatosRenegociarMultiActionController;
    private MockControl controlHttpServlet;
    private HttpServletRequest moHttpServletRequest;

    private MockControl controlHttpResponse;
    private HttpServletResponse moHttpServletResponse;

    private MockControl controlHttpSession;
    private HttpSession moHttpSession;
```

Figura 11: Atributos de la clase

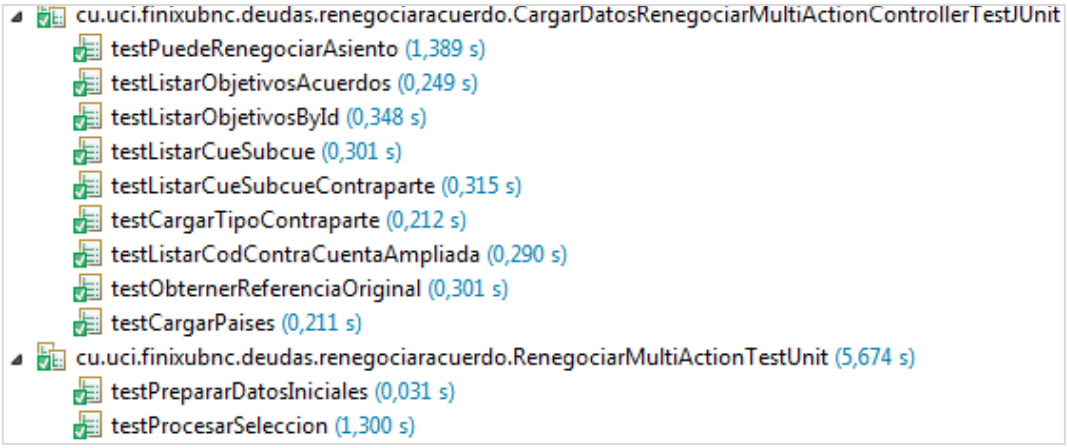


Figura 12: Resultados de la prueba con JUnit

Durante la realización de la prueba se comprobaron las funcionalidades implementadas de mayor importancia, detectándose durante la primera iteración de las pruebas 2 errores, logrando una correcta ejecución de las funcionalidades durante la segunda realización de la prueba.

3.2.2 Pruebas de Caja negra

Las pruebas de caja negra, también denominadas pruebas funcionales o de comportamiento, se centran en los requisitos funcionales del software. Permiten obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. Con este tipo de pruebas se pretende encontrar funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos o en accesos a las bases de datos externas, errores de rendimiento y errores de inicialización y terminación. (27)

Aplicación de la prueba de Caja negra:

Para la ejecución de las pruebas se utilizó la técnica Partición de Equivalencia divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. (27)

Para la ejecución de la técnica Partición de Equivalencia se realizaron los diseños de caso de prueba:

- CIG-QF2-N-DEU-i5101 Registrar Acuerdo
- CIG-QF2-N-DEU-i5102 Registrar Acuerdo utilizando Maqueta
- CIG-QF2-N-DEU-i5103 Actualizar Acuerdo
- CIG-QF2-N-DEU-i5104 Consultar Acuerdo
- CIG-QF2-N-DEU-i5105 Eliminar Acuerdo
- CIG-QF2-N-DEU-i5106 Registrar Vencimiento
- CIG-QF2-N-DEU-i5107 Enmendar Objetivo
- CIG-QF2-N-DEU-i5108 Renegociar Vencimiento
- CIG-QF2-N-DEU-i5109 Actualizar Renegociación
- CIG-QF2-N-DEU-i5110 Actualizar Objetivo
- CIG-QF2-N-DEU-i5111 Registrar Objetivo
- CIG-QF2-N-DEU-i5112 Eliminar Objetivo

En la primera iteración de las pruebas realizadas se detectaron 18 no conformidades detectadas en la documentación y 14 en la aplicación, en la segunda iteración

realizada se detectaron un total de 14 no conformidades de las cuales 11 fueron en la documentación y 3 en la aplicación. El equipo de desarrollo le dio solución a cada una de las no conformidades detectadas, hasta la iteración número 3 donde el subsistema quedó libre de errores (Figura 13), emitiéndose el Acta de Liberación, la cual puede ser consultada en el **Error! Reference source not found.**.

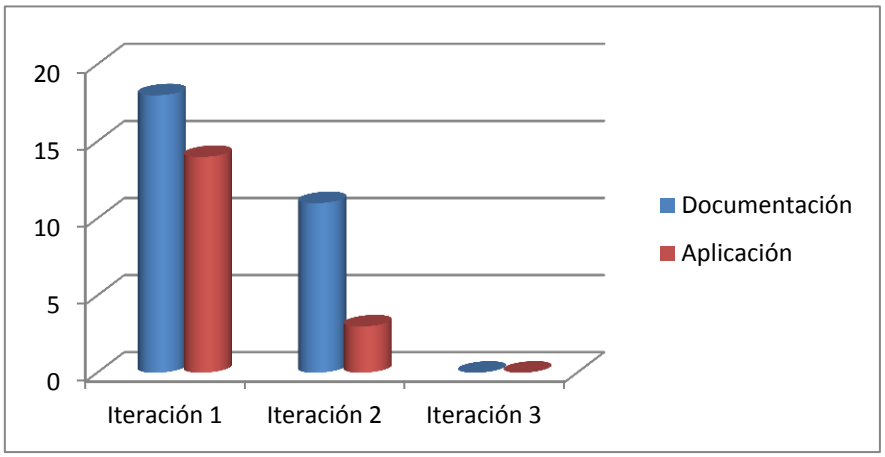


Figura 13: Iteraciones de la prueba realizada

3.3 Validación de los resultados

Uno de ellos es el Experimento, que cuenta con tres variantes (Cuasi-experimento, pre-experimento y experimento puro). Para validar la propuesta de solución planteada en este trabajo se utiliza un pre-experimento, en el que el grado de control es mínimo.

Con el objetivo de validar la investigación, se escogió un especialista del área de Deudas del BNC, a quien se le solicitó realizar el proceso de renegociación de dos formas posibles: antes y después de incorporar el subsistema Deudas al sistema Quarxo. El especialista fue ubicado en una estación de trabajo con buenas prestaciones, enfocado solamente en la ejecución de la prueba, sin la afectación de factores ajenos al proceso.

Para realizar una renegociación es necesario seguir los siguientes pasos:

- Registrar datos. Tiempo de ejecución = **Rdm**
- Asignar vencimientos. Tiempo de ejecución = **Avm**
- Renegociar vencimientos. Tiempo de ejecución = **Rm**

La tabla (Tabla 16) muestra las actividades que se realizan en cada paso, reflejando los dos momentos: antes y después de la incorporación del subsistema Deudas al sistema Quarxo. Se puede observar que el total de actividades a realizar para la

renegociación antes de aplicada la propuesta es de ocho, mientras que después de aplicada la cantidad total de actividades es de cuatro.

Actividades para la renegociación		
Pasos	**Antes (A)**	**Después (D)**
1. Registrar datos	1. El usuario selecciona el subsistema Contabilidad. 2. El usuario selecciona el subsistema Contabilidad. 3. Selecciona la opción actualizar que se encuentra en Gestionar nomencladores. 4. Selecciona la tabla Renegociación e inserta un nuevo registro con los datos asociados.	1. El usuario selecciona el subsistema Deudas. 2. Selecciona la opción Registrar Acuerdo que se encuentra en Gestionar Acuerdo.
2. Asignar vencimientos	5. El usuario selecciona el subsistema Vencimiento. 6. Selecciona la opción Registrar vencimiento e introduce todos los vencimientos de principal e intereses.	3. Selecciona la opción Buscar objetivo que se encuentra en Gestionar Objetivo y luego la operación Asignar vencimiento.
3. Renegociar vencimientos	7. Selecciona la opción Buscar vencimiento y realiza búsquedas de vencimientos y selecciona los vencimientos a renegociar. 8. Realiza la renegociación del vencimiento seleccionado.	4. Selecciona la opción Buscar Vencimiento que se encuentra en Renegociar acuerdo y luego la operación Renegociar.

Tabla 16: Actividades para la renegociación

Para validar la propuesta se realizó la ejecución de cada uno de los tres pasos antes descritos, cuyos resultados contribuyeron a obtener un promedio del tiempo necesario para realizar la renegociación. Se realizaron tres iteraciones teniendo en cuenta que por las características del negocio, para cada transacción o cada gestión de deudas, el juego de datos es diferente. La tabla (Tabla 17) muestra el tiempo de duración de estas tareas representado en minutos.

	Rdm			**Avm**			**Rm**			**Promedio Total**
Iteración	I	II	III	I	II	III	I	II	III	
Antes de incorporar el subsistema Deudas	1,03	0,59	1,01	2,05	2,13	2,10	2,01	2,30	2,16	5,26
Promedio	1.01			2.09			2.16			
Después de incorporar el subsistema Deudas	1	0,50	0,55	0,37	0,49	0,45	1,35	1,28	1,32	3,11
Promedio	0.55			0.44			1.32			

Tabla 17: Resultados de la validación

Antes de incorporar el subsistema Deudas se observa para la ejecución de los tres pasos un tiempo promedio total de 5,26 minutos. Mientras que después de incorporado el subsistema Deudas, se puede observar a un tiempo promedio total de 3,11 minutos.

En la figura (Figura 14) se muestran los tiempos promedios obtenidos en cada paso después de realizar las tres iteraciones:

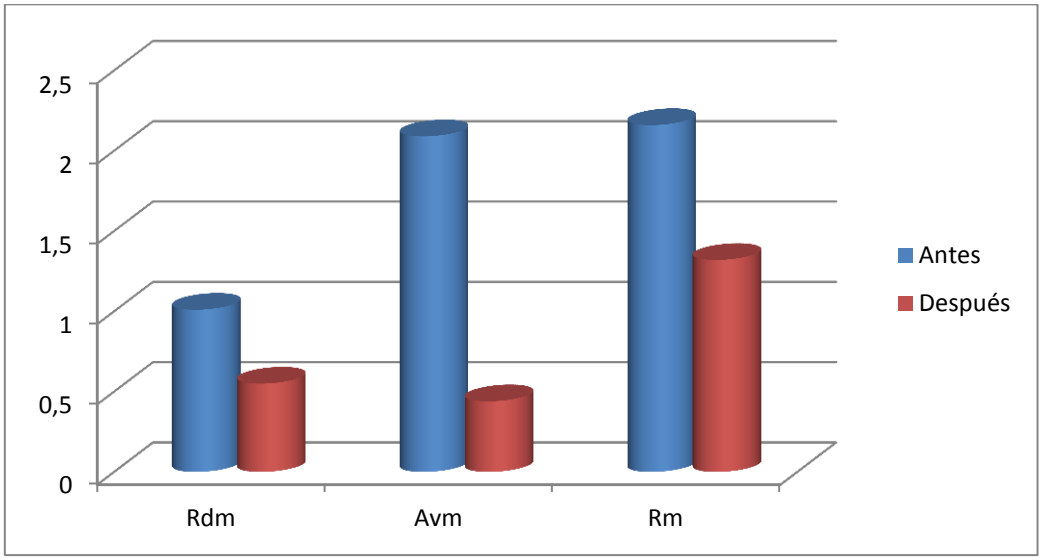


Figura 14: Iteraciones de la prueba realizada

La gráfica evidencia una reducción de 2,15 minutos después de incorporado el subsistema Deudas al sistema Quarxo. Además la aplicación de esta propuesta reduce en un 50% la cantidad de pasos a realizar para la renegociación de los vencimientos y centraliza todas las actividades relacionadas con este proceso en un solo subsistema, como evidencia la tabla (Tabla 16).

Con lo expresado anteriormente se valida el cumplimiento de lo planteado en la idea a defender: Si se desarrolla el subsistema Deudas se permitirá agilizar los procesos de gestión de deudas del Sistema de Gestión Bancaria Quarxo.

Conclusiones parciales

- Los estándares de código definidos permitieron obtener un código legible.
- Con la ejecución de las pruebas unitarias con JUnit se demostró la correcta implementación de las funcionalidades comprobadas.
- Con la ejecución de las técnica Partición de equivalencia se detectaron un total de 32 no conformidades que fueron resueltas, permitiendo que el departamento de calidad del CEIGE emitiera el acta de liberación del subsistema Deudas.
- A través de la realización del método para la validación de la investigación pre-experimento como método para validar la investigación permitió evidenciar una disminución aproximada de 2.15 minutos representando un 41% después de incorporado el subsistema Deudas al sistema Quarxo.

CONCLUSIONES GENERALES

- El estudio de sistemas SIGADE y Tallyman permitió estructurar y agrupar los procesos e informaciones concernientes a las deudas de forma diferente a la que se realizaba mientras que el estudio del sistema Quarxo permitió conocer problemas no identificados que afectan la gestión de Deudas en el BNC, así como algunas funcionalidades que pueden ser usadas durante la implementación del subsistema Deudas.
- Como resultado de la disciplina Requisitos se obtuvo la descripción un total de 8 requisitos funcionales agrupados en tres módulo validados por el cliente, además se elaboraron los diagramas de la disciplina Análisis y Diseño facilitando el entendimiento previo de los requisitos para la implementación
- La aplicación de las métricas TOC y RC permitió indicar que el diseño realizado cuenta con una buena calidad, mostrando valores entre el 66% y 75% en cada atributo de calidad.
- Con la ejecución de las pruebas unitarias con JUnit se demostró la correcta implementación de las funcionalidades comprobadas y con la ejecución de las técnica Partición de equivalencia se detectaron un total de 32 no conformidades que fueron resueltas, permitiendo que el departamento de calidad del CEIGE emitiera el acta de liberación del subsistema Deudas.
- A través de la realización del método para la validación de la investigación pre-experimento como método para validar la investigación permitió evidenciar una disminución aproximada de 2.15 minutos representando un 41% después de incorporado el subsistema Deudas al sistema Quarxo.

Referencia Bibliográfica

1. Banco Nacional de Cuba. [En línea] [Citado el: 12 de Enero de 2014.] http://www.ecured.cu/index.php/Banco_Nacional_de_Cuba.

2. **Rafael Bello Lara, Susana Gonzalez, Manuel Borroto.** Arquitectura del Sistema Quarxo, una solución para el Banco Nacional de Cuba. [En línea] [Citado el: 20 de Diciembre de 2013.] http://www.bc.gov.cu/Anteriores/RevistaBCC/2013/Rev%202%20WEB%202013/Arquitectura%20Quarzo.html.

3. Sitio del Banco Nacional de Cuba. [En línea] [Citado el: 10 de Diciembre de 2013.] http://interbancario.bnc.cu.

4. Cuba. Sitio del Gobierno de la República de Cuba. [En línea] [Citado el: 21 de Enero de 2014.] http://www.cubagob.cu/..

5. Diccionario digital. [En línea] [Citado el: 2 de Febrero de 2014.] http://es.thefreedictionary.com.

6. Economía condonación. *Significado de condonación.* [En línea] [Citado el: 2 de Febrero de 2014.] http://www.economia48.com/spa/d/condonacion/condonacion.htm.

7. Economía mora. *Significado de mora.* [En línea] [Citado el: 2 de Febrero de 2014.] http://es.mimi.hu/economia/mora.html.

8. Economía renegociación. *Significado de renegociación.* [En línea] [Citado el: 2 de Febrero de 2014.] http://es.mimi.hu/economia/renegociacion_de_la_deuda.html.

9. Econmía enmendar. *Significado de enmendar.* [En línea] [Citado el: 2 de Febrero de 2014.] http://es.mimi.hu/economia/enmendar.html.

10. Sitio de SIGADE. [En línea] [Citado el: 18 de Febrero de 2013.] www.unctad.org/dmfas.

11. Sitio de Tallyman. [En línea] [Citado el: 18 de Febrero de 2014.] http://Gestión de recobro _ Experian España.htm.

12. **Obregón, Ing. William González.** *CEIGE-Modelo de Desarrollo de Software v1.2.* 2013.

13. Bizagi . [En línea] [Citado el: 2 de Febrero de 2014.] www.bizagi.com/docs/BPMNbyExampleSPA.pdf.

14. **Schmuller, Joseph.** *APRENDIENDO UML EN 24 HORAS.* México : PEARSON EDUCACION, 2000.

15. Tecnología Java. [En línea] [Citado el: 24 de Febrero de 2014.] http://java.ciberaula.com/articulo/tecnologia_java/.

16. **Pérez, Javier Eguíluz.** *Introducción a JavaScript.* 2008.

17. *Extensión de Visual Paradigm for UML para el desarrollo dirigido por modelos de aplicaciones de gestión de información.* **Lianet Cabrera González, Enrique Roberto Pompa Torre.** 10, La Habana, Cuba : s.n., 2012, Vol. 5.

18. Eclipse. [En línea] [Citado el: 26 de Febrero de 2014.] http://www.eclipse.org/galileo/galileoinaction.php.

19. Microsoft. [En línea] [Citado el: 15 de Febrero de 2014.] http://www.microsoft.com/en-us/sqlserver/product-info.aspx.

20. Apache Tomcat. [En línea] [Citado el: 20 de Febrero de 2014.] http://tomcat.apache.org/.

21. Control de versiones con Subversion. [En línea] [Citado el: 23 de Febrero de 2014.] http://svnbook.red-bean.com/nightly/es/svn-ch-1-sect-1.html..

22. **Bernard, Emmanuel.** *Hibernate Annotations.* 2005.

23. **Rusell, M.A.** *Dojo, The Definitive Guide.* s.l. : Estados Unidos de América., 2008.

24. **Thomas Van de Velde, Bruce Snyder, Christian Dupuis, Sing Li, Anne Horton y Naveen Balani.** *Beginning Sping Framework 2.* Estados Unidos de América. : s.n., 2008.

25. **Craig Walls, R.B.** *Spring in Action Second Edition.* Estados Unidos : Manning Publications, 2007.

26. **Group, The Business Rules.** *Manifiesto de Reglas de Negocio.*

27. **Pressman, Roger S.** *Ingenería del Software. Un enfoque práctico.* España : s.n., 2001. Quinta Edición.

28. Técnica de captura de requisitos. [En línea] [Citado el: 29 de Marzo de 2014.] http://www.ecured.cu/index.php/T%C3%A9cnicas_de_recopilaci%C3%B3n_de_requisitos.

29. **Iglesias, Miguel Adolfo.** *Documento de Arquitectura de Software, Modernizaciondel Sistema Bancario Cubano.* 2009.

30. **Orea, Sergio Valero.** *Desarrollo de Software.* 2009.

31. **Ortiz, Antonio Moreno.** Base de Datos. [En línea] [Citado el: 3 de Abril de 2014.] http://elies.rediris.es/elies9/4-2.htm. ISSN: 1139-8736.

32. **Gutierrez, Demián.** UML Diagramas de Paquetes. [En línea] Septiembre de 2009. [Citado el: 14 de Abril de 2014.] www.codecompiling.net/files/slides/UML_clase_05_UML_paquetes.pdf.

33. **Otero, Mari Carmen.** Diagramas de Clases. [En línea] [Citado el: 24 de Abril de 2014.] www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r25387.PDF.

34. Diagrama de Secuencia. [En línea] [Citado el: 26 de Abril de 2014.] http://www.w3schools.com/html5/html5_reference.asp..

35. **Larman, Craig.** *UML y Patrones. S.l.: Prentice Hall.* 1999.

36. **Erich Gamma, Richard Helm, Ralph Johnson John Vlissides.** *Design Patterns.*

37. **Pressman, Roger S.** *Ingenería del Software. Un enfoque práctico.* Quinta Edición.

38. **Manzano, Mailin Dayana Romero.** *Modelo de desarrollo de software para el departamento Soluciones Financieras del CEIGE.* 2012.

39. **GAMMA, Erich and BECK, Kent.** *JUnit: A cook's tour.* 1999.

GLOSARIO DE TÉRMINOS

- **Campo:** Se refiere campo como al atributo que forma parte del valor de un elemento dentro de un mensaje XML, describe generalmente una propiedad o atributo de un objeto del dominio del negocio.
- **Código abierto (Open Source):** Término con el que se conoce al software distribuido y desarrollado libremente.
- **Command:** Es la representación específica de la información con la cual el sistema opera. Se limita a lo relativo de la vista y su controlador facilitando las presentaciones visuales complejas. Los command también pueden operar con más datos no relativos a la presentación, haciendo uso integrado de otras lógicas de negocio y de datos afines con el sistema modelado.
- **Framework**: Es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.
- **Módulo:** Cada módulo es una parte del sistema, que se relaciona con otros módulos con los que intercambia información.
- **Plugins:** Un plugins es un módulo de hardware o software que adiciona una característica o un servicio específico a un sistema existente.
- **Servlet**: Son objetos que están presentes dentro del contexto de un contenedor de servlets como el Tomcat, es comúnmente utilizado para generar páginas web de forma dinámica a partir de parámetros de una petición de un navegador web.
- **XML** (en inglés Extensible Markup Language)**:** Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium permitiendo definir la gramática de lenguajes específicos. XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

ANEXOS

Anexo 1: Reglas del negocio

No	Tipo	Nombre	Descripción
1	Reglas del Modelo de Datos	Número de contrato	El número de contrato no puede estar en blanco.
2	Reglas del Modelo de Datos	Sigla de moneda	La sigla de moneda debe estar entre su clasificador moneda.
3	Reglas del Modelo de Datos	Banco	El banco debe estar entre su clasificador banco.
4	Reglas del Modelo de Datos	Código de país	El código país debe estar entre su clasificador país.
5	Reglas del Modelo de Datos	Cliente	El cliente debe estar entre su clasificador cliente.
6	Reglas de Relación	Renegociación	Todas las renegociaciones deben de partir de un objetivo de un acuerdo.
7	Reglas Textuales	Acuerdo	Todo acuerdo debe definir al menos un objetivo.

Tabla 18: Reglas del negocio

Anexo 2: Diagrama de paquetes

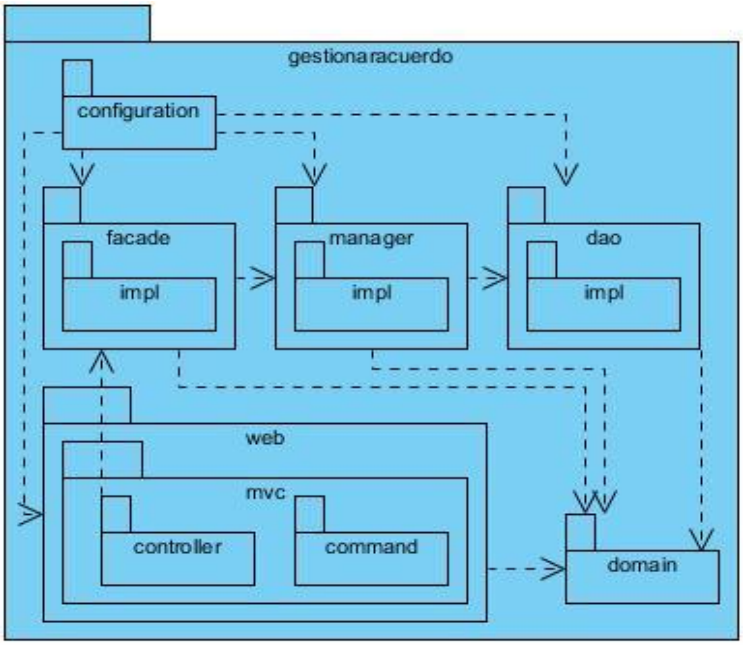


Figura 15: Diagrama de paquetes del módulo gestionar Acuerdo

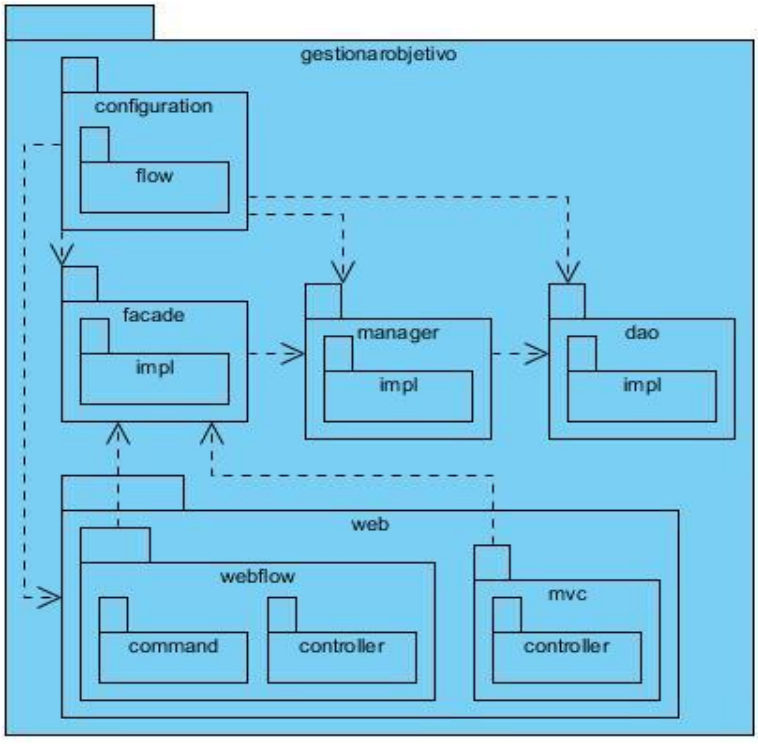


Figura 16: Diagrama de paquetes del módulo gestionar Objetivo

Anexo 3: Diagrama de clases del diseño

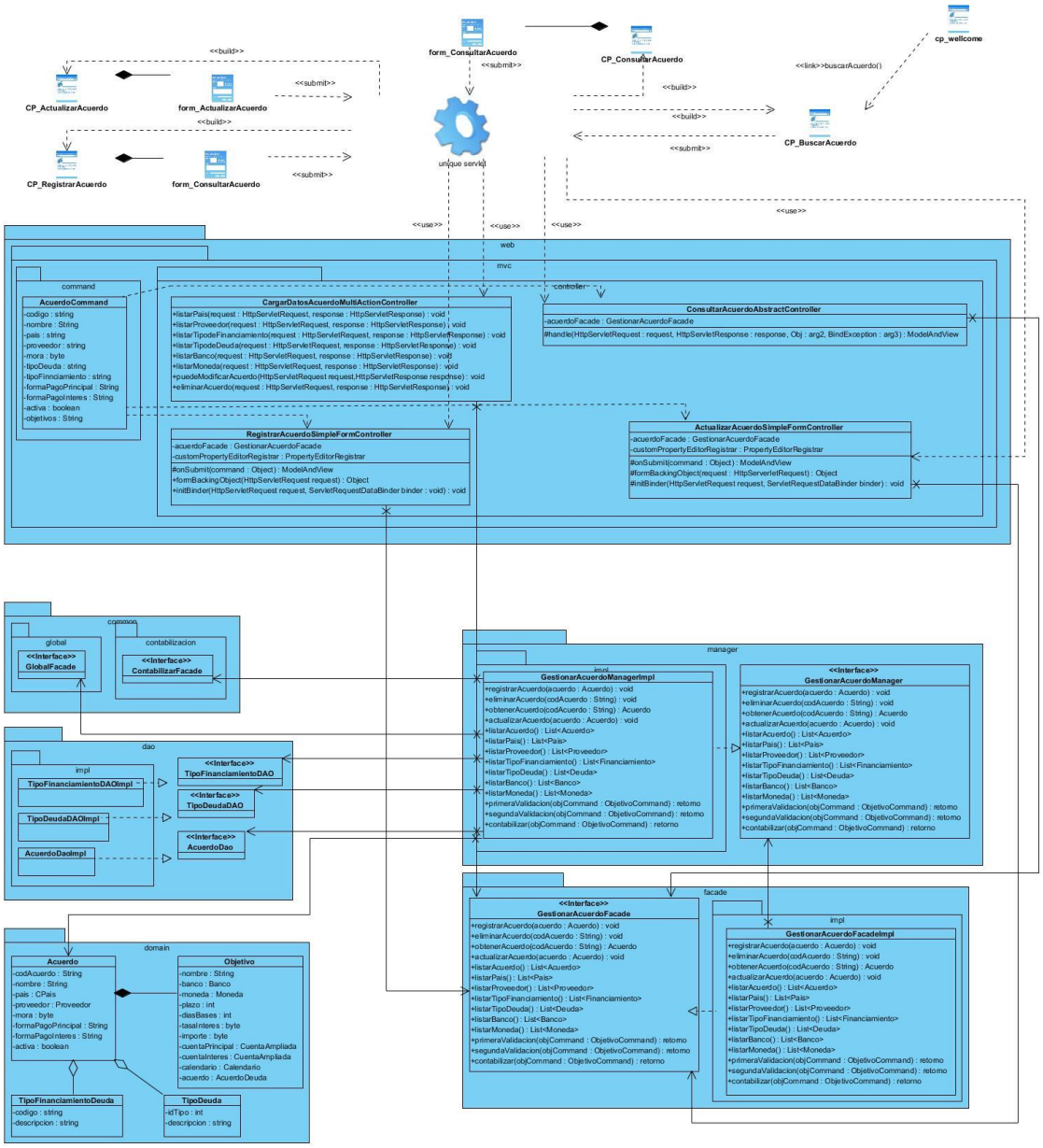


Figura 17: Diagrama de clases del diseño del módulo gestionar Acuerdo

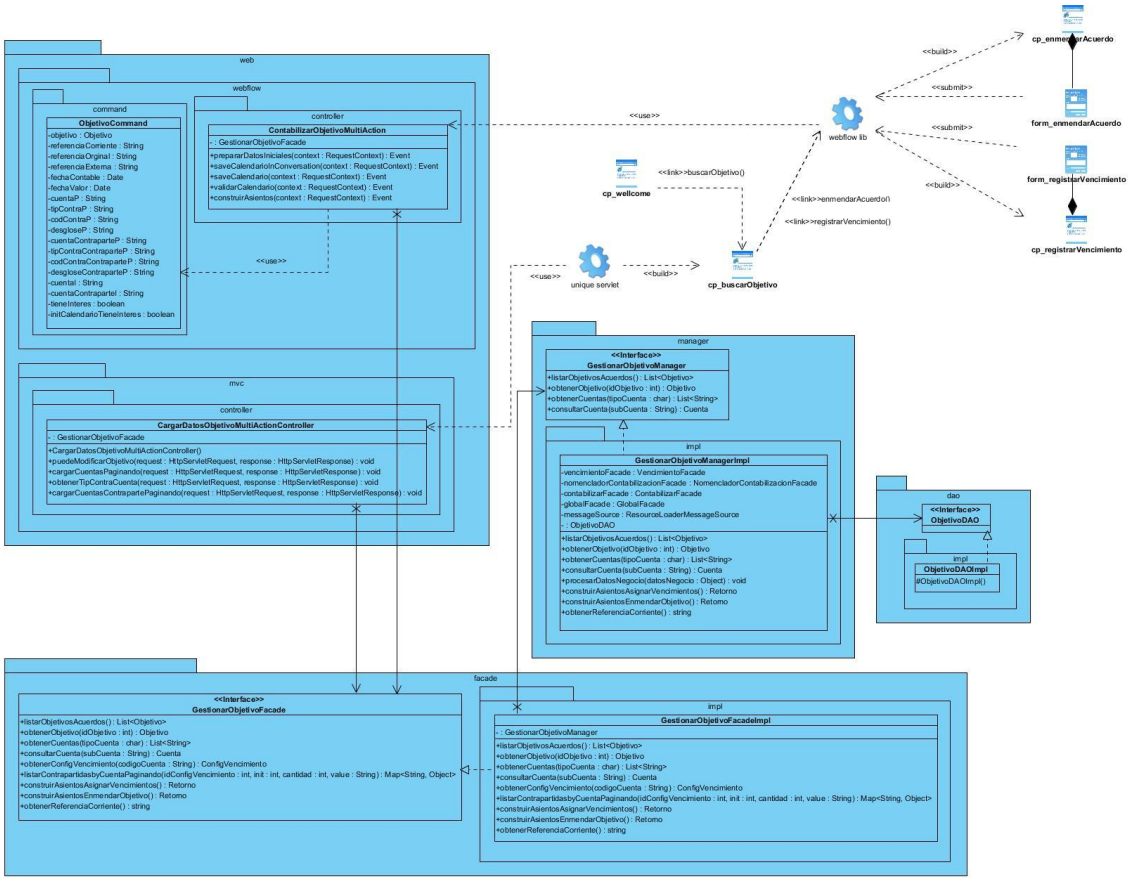


Figura 18: Diagrama de clases del diseño del módulo gestionar Objetivo

Anexo 4: Diagramas de Secuencia

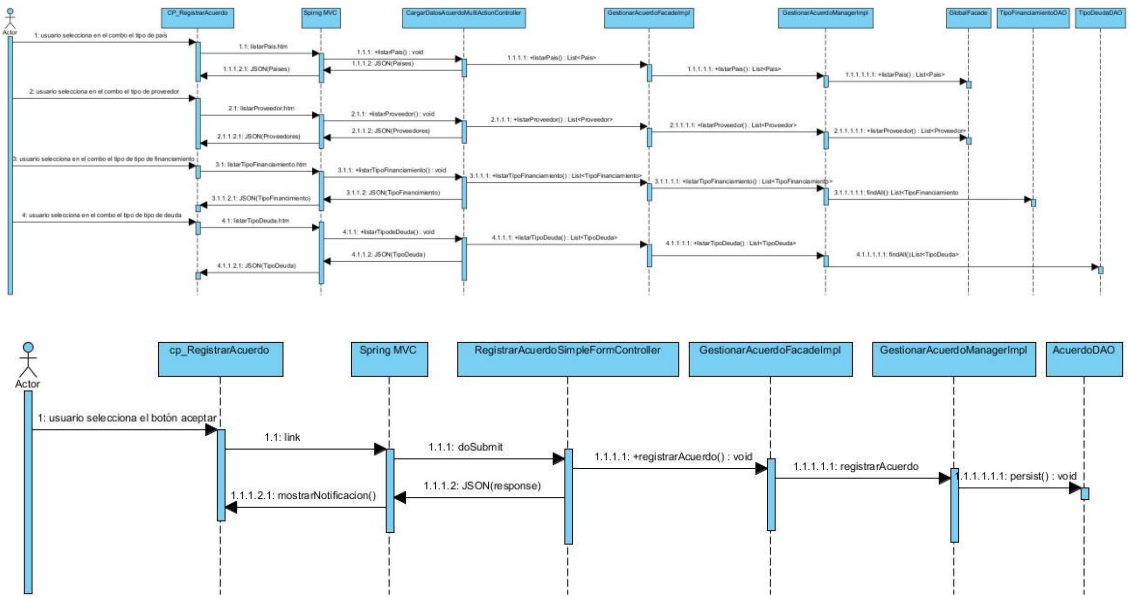


Figura 19: Diagrama de secuencia registrar Acuerdo

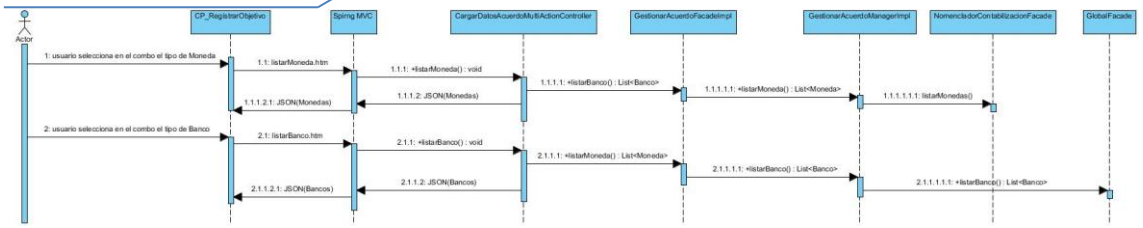


Figura 20: Diagrama de secuencia registrar Objetivo

Anexo 5: Prototipos de Interfaz de Usuario

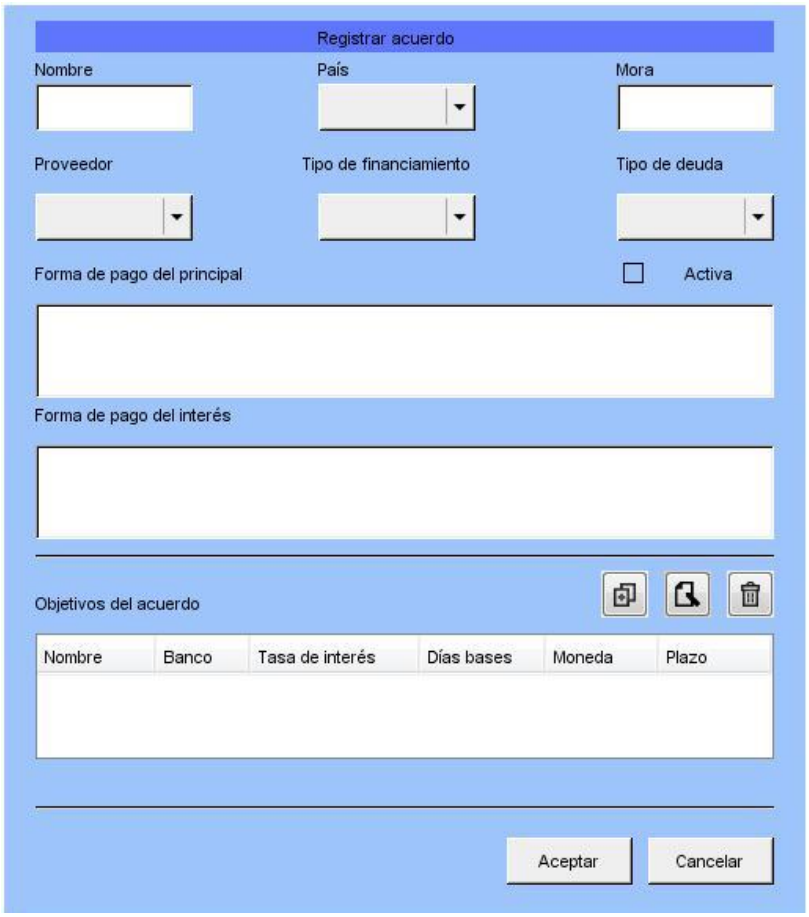


Figura 21: Registrar acuerdo

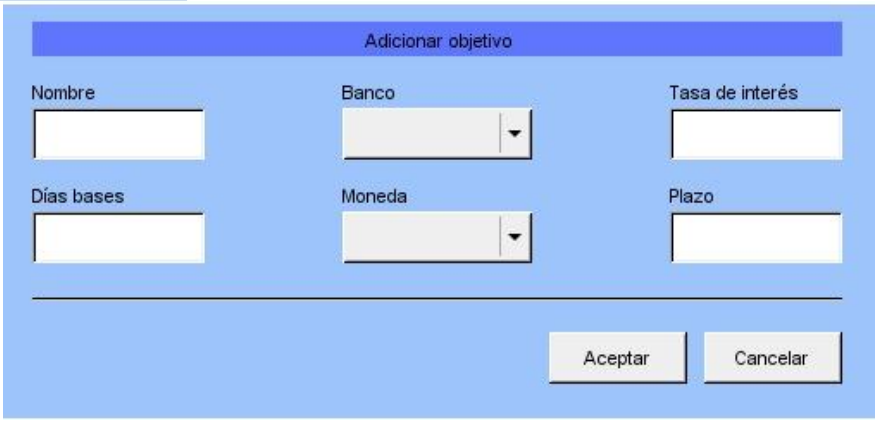


Figura 22: Registrar objetivo

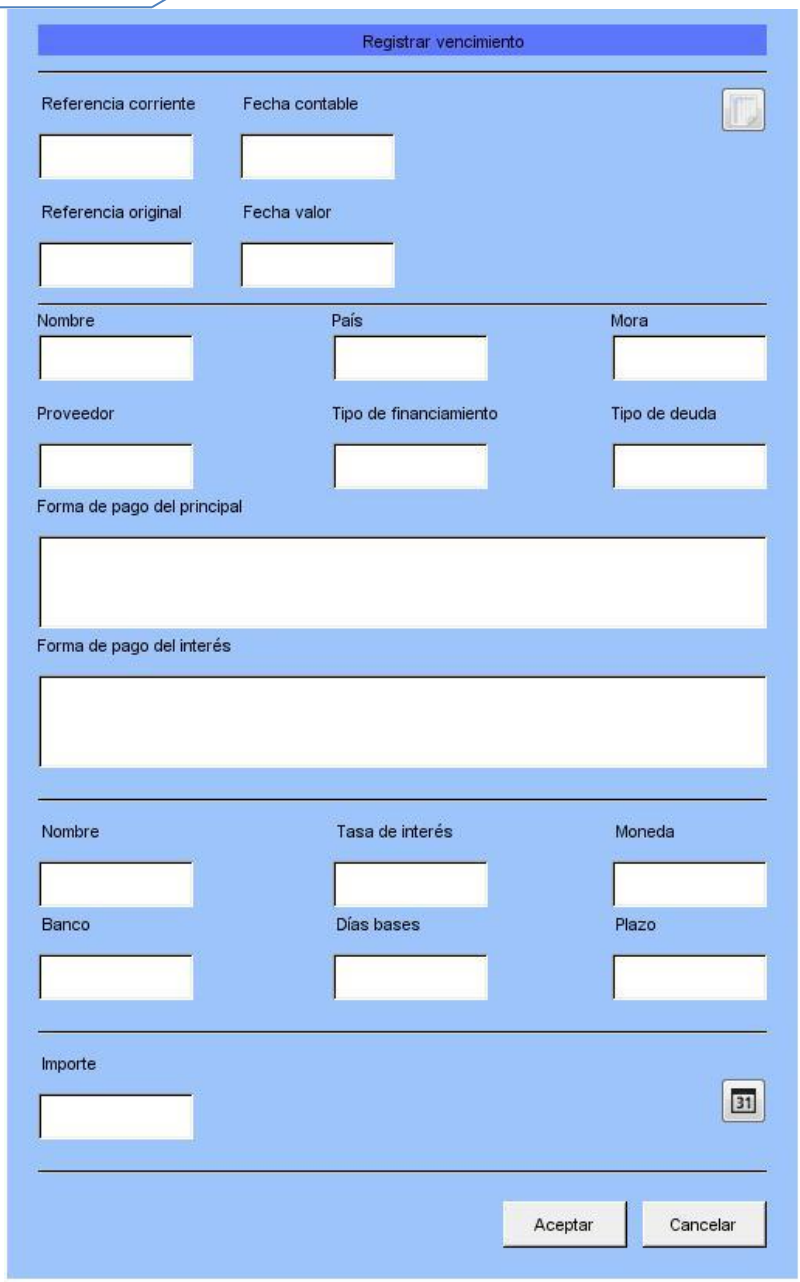


Figura 23: Registrar vencimientos

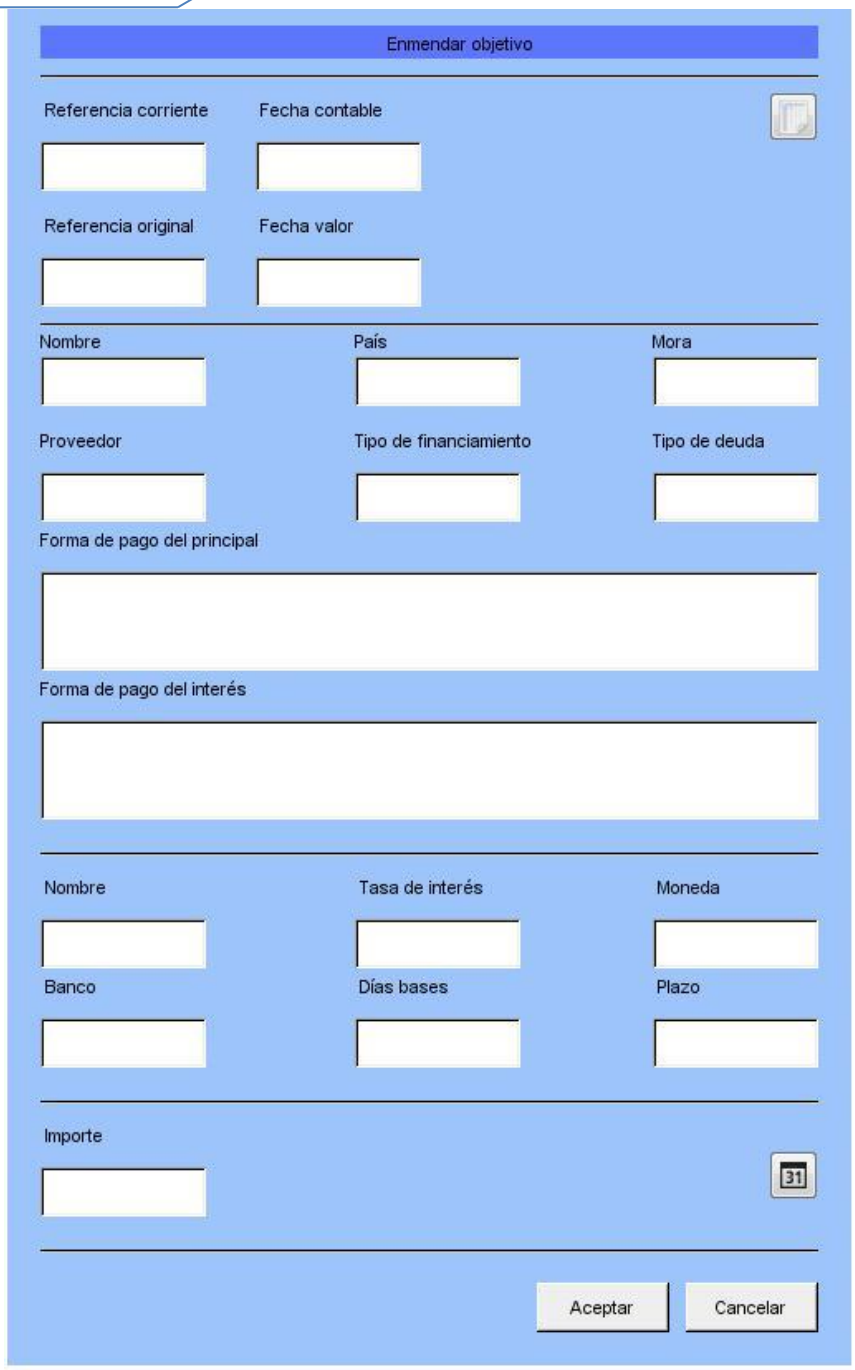


Figura 24: Enmendar objetivo

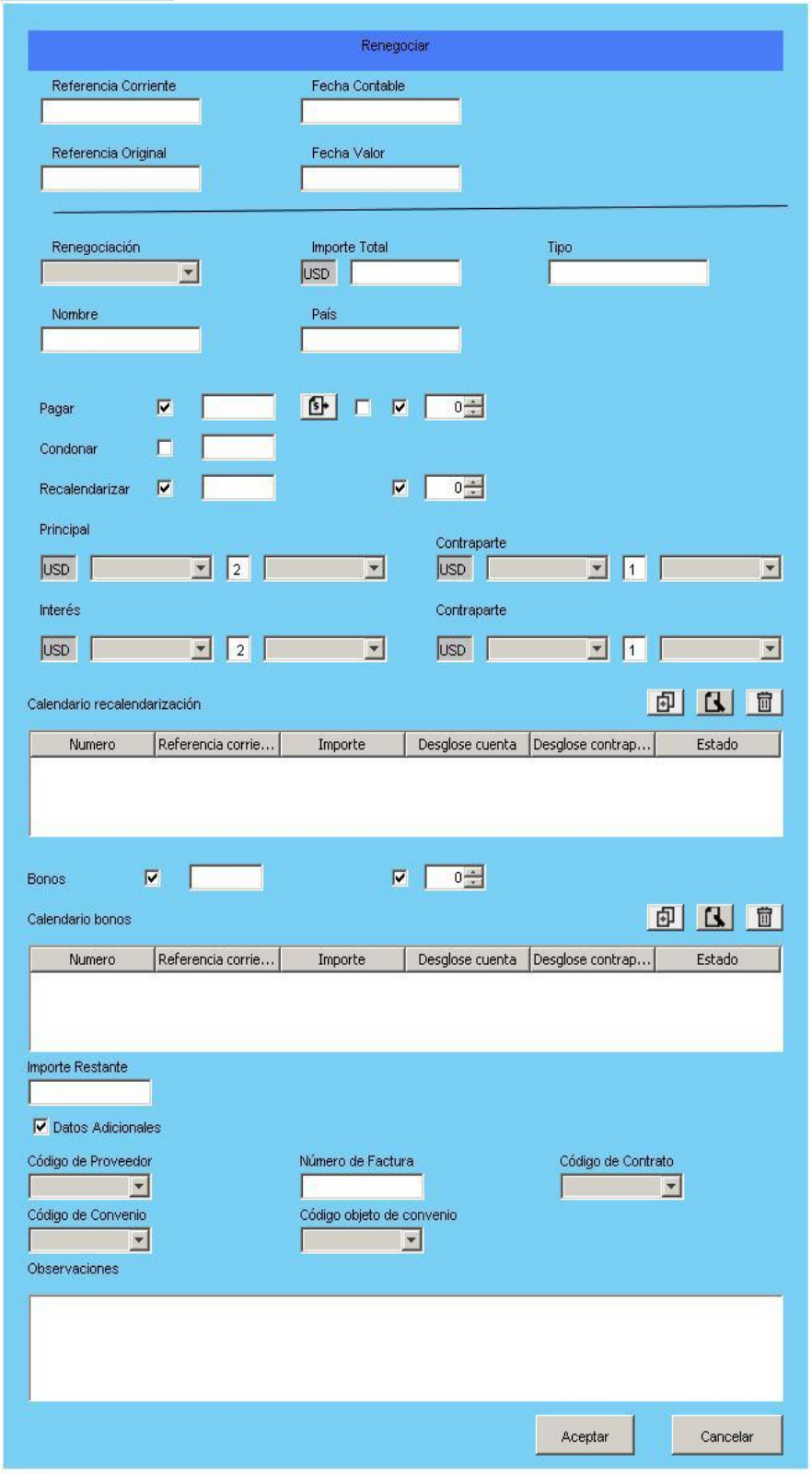


Figura 25: Renegociar vencimiento

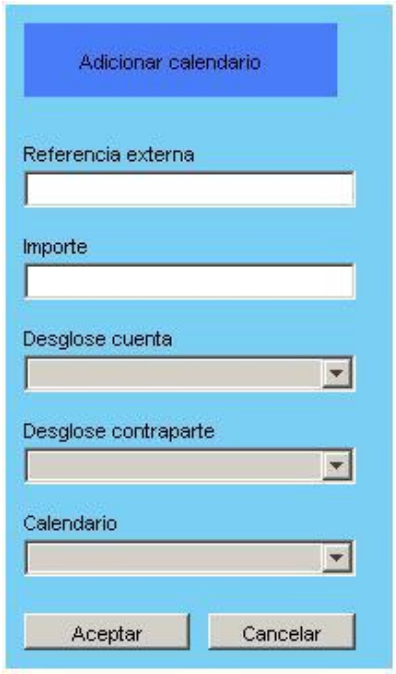


Figura 26: Adicionar calendario