Universidad de las Ciencias Informáticas
Facultad 3

Título: Diseño e implementación del subsistema Tesorería del sistema Quarxo

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

Autor: Danis Lazaro Ramos Rodríguez

Tutor: Ing. Yoan Antonio López Rodríguez

Ciudad de la Habana 2012

Declaración de Autoría

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

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

_____	_____
Danis Ramos Rodriguez	Ing. Yoan López Rodríguez
Autor	Tutor

Agradecimientos

Agradecer a mi familia por apoyarme estos 5 años y por darme todo lo que necesité.

A todos mis compañeros que de una forma u otra fueron mi familia aquí. A Wilfredo por ser un hermano para mí, a Yanelis por siempre preocuparse por mí y ayudarme en todo, a Noelio por ser amigo y guía, a Eduard que aunque lo conocí recientemente me ha brindado su apoyo y su amistad.

A todas las personas que de una forma u otra contribuyeron con la realización de este trabajo, en especial a Manuel, Emilio y Evelio que tantas veces los molesté.

A Susel por siempre estar ahí para mí en los buenos y malos momentos, por ser más que mi novia, mi amiga, por sus consejos, y por todo lo que hemos vivido y aprendido el uno del otro todo este tiempo.

A Paita que aunque ya no está entre nosotros fue y será una persona muy importante en mi vida. A mi abuela por ser lo que más quiero en el mundo, por hacerme la persona que soy, por todo su amor y dedicación. A mi mamá por ser tan comprensiva por quererme y siempre estar atenta de mis cosas. A mi tío por jugar el papel de padre en mi vida, a mi padrastro por quererme como un hijo, a mi hermanito y a mi hermana por su amor. A mi tía Eneida por ser una persona muy especial para mí, por su cariño y su preocupación.

A mi familia del Mónaco: Norge, Daniel, Orlenys, Beto, Minerva y Adriana, por acogerme con tanto cariño y brindarme su casa.

Dedicatoria

A toda mi familia por todo lo que me han dado, a ellos dedico este resultado y todo lo que pueda obtener en la vida, en especial a mamita.

Resumen

La informatización del sistema bancario cubano como parte del desarrollo tecnológico que precisa el país, demanda una elevada capacidad tecnológica y operativa, lo que trae consigo la utilización de herramientas computacionales para el procesamiento de la información.

La Universidad de las Ciencias Informáticas ha desempeñado un papel protagónico en ese sentido, de manera que está inmersa en el desarrollo del sistema Quarxo, que responde de forma rápida y certera a la actividad contable y financiera que se lleva a cabo en el Banco Nacional de Cuba (BNC), a partir de que en este se utiliza el Sistema Automatizado para la Banca Internacional de Comercio (SABIC), el cual está desarrollado sobre tecnología obsoleta, lo que hoy en día dificulta el manejo de los grandes volúmenes de datos, así como la gestión de tesorería proceso clave dentro de las operaciones bancarias. Con el propósito de llevar a cabo dichas funcionalidades en el BNC, el presente trabajo comprende el Diseño y la Implementación del subsistema Tesorería que será incorporado al sistema Quarxo. La solución abarca el estudio de tecnologías, herramientas y lenguajes a utilizar, así como la elaboración de artefactos propios de los flujos diseño e implementación como parte del proceso de desarrollo de software.

Palabras clave: Compraventa, Transferencia, Tesorería, Quarxo, Diseño, Implementación.

Contenido

Tabla de contenidos

Tabla de contenidos

Introducción

El desarrollo de las nuevas tecnologías es una de las tendencias que está alterando fundamentalmente el mundo financiero. La informatización de los procesos empresariales es uno de los mecanismos indispensables para el desarrollo económico de una empresa en busca del aumento en los niveles de eficiencia, organización de las actividades, control del flujo de información y optimización de los procesos. El entorno en que se desarrolla la actividad bancaria, exige un constante esfuerzo de mejora en muchos frentes tales como: el rediseño de procesos, la mejora de la productividad, la reducción de costos y el alcance de una buena calidad para mejorar la satisfacción de los clientes, dejando clara la necesidad de perfeccionamiento y evolución de las soluciones informáticas.

La tesorería es una de las funciones críticas dentro del área financiera, está vinculada a los procesos de liquidez de las empresas, además debe administrar el flujo de caja con una cierta tasa de rentabilidad de oportunidad. Ese flujo es muy propio de cada industria, de cada banco y de cada empresa en particular, con sus productos y servicios. Los bancos necesitan disponer de liquidez, pues ella representa el grado de disponibilidad con la que los diferentes activos pueden convertirse en dinero, el cual es el medio de pago más líquido de todos los existentes.

El Sistema Bancario Cubano y en específico el Banco Nacional de Cuba (BNC) en función de lograr el manejo adecuado de la información y la interconexión entre los diferentes bancos y sucursales ha estado empleando el Sistema Automatizado para la Banca Internacional de Comercio (SABIC). El SABIC tiene como principal desventaja el ser monotarea, lo que entorpece en gran medida el trabajo del operador. Este sistema no permite realizar los procesos de compraventa y transferencia pertenecientes al área de tesorería. Cuando la institución requiere realizar una compraventa o una transferencia los especialistas tienen que hacer uso de hojas de cálculo en los cuales se conforma el modelo correspondiente al proceso a ejecutar, luego este modelo es impreso y llevado al área de préstamos y depósitos para ser contabilizado.

Lo antes mencionado provoca que el tiempo de respuesta a estas solicitudes tarden 20 minutos aproximadamente, siendo este periodo inadecuado teniendo en cuenta la cantidad de operaciones de este tipo que se deben realizar diariamente en la entidad, además esta demora incurre en el atraso de la contabilización de estas operaciones, convirtiéndose así en un proceso lento, engorroso y poco confiable. Además el tener que imprimir en papel el modelo antes mencionado incrementa el consumo de recursos materiales de la institución.

Introducción

Como parte de la modernización del Sistema Bancario Cubano, la Universidad de las Ciencias Informáticas (UCI) ha desarrollado un sistema informático para el BNC capaz de gestionar la mayoría de los procesos.

La solución denominada Quarxo abarca un conjunto de subsistemas, entre ellos: Contabilidad, Cuentas de Clientes, Cartas de Créditos, existiendo una adecuada comunicación entre ellos.

Atendiendo a la situación problémica descrita con anterioridad, se define como **problema a resolver*:*** Las deficiencias en el proceso de gestión de Tesorería incrementan el consumo de recursos materiales y provocan aumento del tiempo de respuesta de las operaciones de transferencia y/o compraventa en el Banco Nacional de Cuba. A partir del problema planteado se define como **objeto de estudio** los procesos de gestión de Tesorería en las entidades bancarias y el **campo de acción** los procesos de gestión de Tesorería en el Banco Nacional de Cuba.

Como **idea a defender** se ha establecido que con el desarrollo del subsistema Tesorería para el sistema Quarxo se contribuirá a la gestión de los procesos de transferencia y compraventa en el Banco Nacional de Cuba, permitiendo disminuir los tiempos de respuesta y ahorrar los recursos materiales de la institución. El **objetivo general** definido para el siguiente trabajo de diploma es: Realizar el diseño e implementación el subsistema Tesorería del sistema Quarxo para contribuir a la gestión de los procesos de transferencia y compraventa en el Banco Nacional de Cuba, permitiendo disminuir los tiempos de respuesta y ahorrar los recursos materiales de la institución.

Objetivos específicos:

1. Elaborar el marco teórico de la investigación.
2. Diseñar el subsistema Tesorería del sistema Quarxo.
3. Validar el diseño propuesto.
4. Implementar el subsistema Tesorería del sistema Quarxo.
5. Validar el subsistema implementado.

Tareas para cumplir los objetivos:

1. Analizar las metodologías, lenguajes y herramientas definidos en el proyecto.
2. Analizar las tecnologías y patrones de diseño establecidos en el proyecto.

3. Realizar un estudio del estado del arte de los sistemas informáticos de gestión bancaria.
4. Analizar los procesos de Tesorería en el Banco Nacional de Cuba.
5. Analizar los sistemas informáticos existentes en el mundo para realizar los procesos de Tesorería y determinar si existe alguno que sirva de apoyo al sistema a desarrollar.
6. Obtener el modelo de diseño para el subsistema Tesorería.
7. Aplicar métricas de calidad para validar el diseño.
8. Realizar la implementación del subsistema Tesorería.
9. Realizar pruebas de unidad, integración y aceptación al subsistema Tesorería.

El **posible resultado** que se espera es: realizar la implementación del subsistema Tesorería del sistema Quarxo.

Estructura del Documento:

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

Capítulo 2: En este capítulo se elabora el diseño a partir de la especificación de requerimientos del subsistema Tesorería, como base para la implementación del mismo. Está enfocado a la construcción de diagramas de clases de diseño, diagramas de paquetes y diagramas de secuencias para conformar el modelo de diseño, según la Arquitectura Base y los requerimientos del sistema, así como la realización del modelo de datos.

Capítulo 3: Estará centrado principalmente en la implementación del subsistema Tesorería, mostrando y explicando en cada caso los estándares de codificación, la interacción entre componentes del sistema a través de diagramas, descripción de un subconjunto de clases y funcionalidades.

Por otra parte se realiza la validación a partir de la realización de las pruebas de unidad, brindando una explicación detallada de las pruebas de caja blanca que fueron realizadas al código del software y las pruebas de caja negra que se realizaron a la interfaz del mismo.

Capítulo 1: Fundamentación Teórica.

1.1 Introducción

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

1.2 Conceptos asociados al dominio del problema

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

1.2.1 Tesorería

- ✓ La tesorería es aquella área de la empresa o institución en la cual se gestionan y concretan todas las acciones relacionadas con operaciones de tipo monetario. (1)
- ✓ La tesorería es la disponibilidad de medios líquidos en la caja de una sociedad o entidad de crédito. (2)
- ✓ La tesorería es la actividad de dirección responsable de la custodia e inversión del dinero, garantía del crédito, cobro de cuentas, suministro de fondos y seguimiento del mercado de valores en una empresa. (3)

Al analizar dichos conceptos se concluye que la Tesorería ofrece información de hechos tanto económicos como financieros de una entidad, se encarga de la custodia de los fondos y valores, además de realizar los cobros y pagos de una entidad o empresa. De esta forma se considera una herramienta clave para un buen funcionamiento de la entidad.

1.2.2 Gestión de Tesorería

- ✓ Conjunto de técnicas y procedimientos destinados a gestionar óptimamente los fondos monetarios de la entidad. (4)

1.2.3 Transferencia bancaria

Operación mediante la cual el titular de una cuenta disponible a la vista en una entidad de crédito, ordena a esta que transfiera determinados fondos de la misma, a una cuenta también disponible a la vista a nombre de un tercero, abierta en la misma u otra entidad de depósito. La transferencia es iniciada por una orden del titular (ordenante) de una cuenta, que solicita en la misma un traspaso de fondos entre dos cuentas, donde la cuenta destino puede ser de él o de otro titular. El banco emisor lo puede hacer de forma directa o utilizando los servicios de otra entidad. (5)

1.2.4 Compraventa de divisas

La compraventa son cambios de una determinada moneda por otra debido a razones comerciales o de otro tipo, teniendo en cuenta el tipo de cambio correspondiente a cada moneda. El tipo de cambio es el precio de una divisa en función de otra.

1.3 Necesidad de informatizar el proceso de Tesorería

Al ser un tipo de operación basada en información, la tecnología debe acompañar a los procesos y reflejarse en la organización. Una operación de tesorería, es principalmente una operación sin papeles, la cual podría virtualizarse totalmente bajo un diseño agresivo e innovador con altos niveles de delegación y con fortaleza de control en los sistemas de información. Debe apoyarse en información actualizada y confiable. (6)

1.4 Sistema Financiero

Las instituciones financieras son uno de los mayores inversores en los sistemas de información. De acuerdo a una encuesta realizada por Ernst & Young y American Banker, los gastos de estas instituciones en la industria tecnológica alcanzaron niveles sin precedentes a finales de 1990. Se estima que sólo en 1994, los bancos invirtieron cerca de $ 20 mil millones en sistemas y tecnologías en un esfuerzo por mejorar la eficiencia de sus procesos y los servicios brindados al cliente. Muchas instituciones financieras tienen éxito demostrando que los sistemas de información y las tecnologías pueden ser una poderosa arma competitiva que puede utilizarse para capturar cuota de mercado, mejorar el servicio al cliente, reducir los costos operativos y crear nuevos productos y servicios. (7)

1.5 Sistemas Informáticos Bancarios

Hoy en día todos los sectores de la sociedad se han visto beneficiados de una forma u otra con el empleo de las tecnologías de la información. Los sistemas informáticos bancarios son un ejemplo claro de los beneficios que trae aparejada la informática. Los mismos son programas contables destinados a sistematizar y simplificar las tareas de contabilidad, tesorería, análisis financieros, entre otros. Sólo necesitan que sea ingresada la información requerida y posteriormente el programa se encarga de hacer las operaciones necesarias. A continuación serán descritos y caracterizados algunos sistemas de este tipo, tanto nacionales como internacionales, con el objetivo de buscar características que contribuyan a la solución del problema.

1.5.1 Sistema Bancario Financiero BYTE

Es un conjunto de módulos independientes e integrables, que permiten la automatización de todos los departamentos, según sean las necesidades. Este sistema contempla todas las operaciones que una institución financiera realiza, tanto administrativas como operativas y esto se logra a través de la interacción de los módulos que lo integran y que constituyen además su eje central:

Clientes: permite una visión global de las operaciones que un cliente ha realizado con la institución, además de poder agruparlos por grupos o sectores.

Captaciones: permite las operaciones de cuenta corriente, ahorros, depósitos a plazo, cuentas de traslado automático, cheques certificados, cuentas over night y administración de valores.

Colocaciones: realiza las operaciones de evaluación de sujetos de créditos, análisis financiero, control de solicitudes y presión de créditos, préstamos, documentos descontados, créditos en cuenta corriente, valuación de activos, control de recuperación y análisis de morosidad.

Se caracteriza por ser una aplicación completamente WEB que ha sido desarrollada siguiendo el modelo MVC, implementada completamente en el lenguaje Java, presenta soporte para bases de datos MS SQL Server, Oracle y DB2. (8)

1.5.2 Sage XRT Treasury 3.0

Sage XRT Treasury es una solución informática que te ayuda a gestionar la tesorería, controlar el presupuesto, conciliar los movimientos, gestionar los cobros y pagos, tratar los riesgos financieros y administrar las comunicaciones bancarias. Ofrece, sencillez y facilidad de despliegue sin perder en funcionalidad.

Sus equipos estarán operativos en cualquier momento, asegurándose así de que aprovecha su inversión, utilizando tecnología estándar y potente para implementar procesos lineales, para una mejor colaboración y mayor visibilidad de sus flujos de tesorería. (9)

Algunas de sus características son:

- ✓ SQL Server 2000/2005 y Oracle v10
- ✓ Único registro, soportes Active Directory
- ✓ LDAP/Active Directory authentication

1.5.3 BANTOTAL-Core Bancario

Es un software desarrollado por la compañía "De Larrobla & Asociados Internacional", el mismo comprende el procesamiento integral de todas las operaciones de una entidad financiera. Este sistema a su vez ofrece soluciones especializadas para el área de Tesorería (incluyendo operaciones relacionadas con los Mercados de Dinero, Divisas y Valores), incorpora como parte de sus funcionalidades la gestión de cuentas corrientes, cajas de ahorro, custodia y administración de valores entre otras.

Posee un módulo de tesorería que involucra cuatro aspectos fundamentales:

- ✓ **Sistema de Cambios / Dinero**

  Administra las funciones asignadas generalmente al sector de Cambios y Mesa de Dinero.

- ✓ **Sistema de Títulos**

  Es el sistema que permite administrar y procesar las operaciones sobre instrumentos públicos y privados de inversión, con destino a inversión propia del Banco o para reventa. Permite el ingreso y actualización de operaciones de compraventa, registrando los vencimientos pactados, las tasas, y calculando los intereses a cobrar o pagar y devengados.

- ✓ **Sistema de Títulos con Renta Variable**

  Tiene por finalidad registrar y administrar las inversiones en activos cuya rentabilidad no es conocida en forma anticipada.

- ✓ **Sistema de Títulos con Renta Fija**

  Es el sistema que permite administrar y procesar las operaciones sobre instrumentos públicos y privados de inversión con rentas conocidas en forma anticipadas.

El sistema BANTOTAL puede ejecutarse en arquitecturas Java o .NET además de ser una aplicación implementada sobre la WEB y emplea el patrón MVC. En cuanto al almacenamiento y procesamiento de los datos la misma fue desarrollada con la capacidad de soportar bases de datos IBM i-Series, MS SQLServer, DB2 y Oracle. (10)

1.5.4 Cash Flow Manager

Sofware Cash Flow Manager es un sistema de gestión de tesorería que automatiza todos los procesos necesarios para la obtención de información financiera, su clasificación, comprobación y optimización en la toma de decisiones, generando posteriormente todos los asientos contables derivados de las operaciones bancarias y financieras de la empresa. Complementado con sistemas de comunicaciones bancarias automáticas, e interfaces directos con sus sistemas contables o ERP (como por ejemplo, Microsoft Dynamics NAV o Microsoft Dynamics AX), Cash Flow Manager transforma la información para darle una visión financiera en vez de contable. Pertenece al grupo de software propietario. Está diseñado para poder definir los interfaces necesarios para capturar y transformar, de forma sencilla, toda la información contenida en sus sistemas o en el presupuesto económico de su empresa. (11)

1.5.5 ELK-Cash

ELK-Cash es un software creado por la Consultoría de ELKARGI, S.G.R. para la Gestión Integral de la Tesorería de la empresa. Va dirigido a empresas usuarias de cualquier programa de gestión (ERP y similares) que precisen disponer de una herramienta específica para la gestión de su tesorería. Posee los módulos de Conciliación y contabilización automática, Banca electrónica, Presupuestos de tesorería a distintos plazos y Previsiones de cobros y pagos. Además es multimoneda y de fácil manejo, al igual que Cash Flow Manager es un sistema propietario. (12)

1.5.6 Sistemas Integrados de Gestión

Los Sistemas Integrados de Gestión son la expresión más acabada de sistemas de información orientados al registro de las operaciones económicas en su sentido más amplio. Dentro de estos sistemas podemos encontrar los ERP (Enterprise Resourcing Planning) traducido al español Planificación de Recursos de la Empresa. Muchos ERP, indican en su publicidad que disponen de un módulo de gestión de tesorería.

Después, la realidad demuestra que requieren de enormes desarrollos para siquiera aproximarse a las mínimas prestaciones y herramientas de gestión de tesorería. El hecho es que los ERP no proporcionan una herramienta de calidad para la gestión de tesorería de las empresas.

1.5.7 Sistema Automatizado para la Banca Internacional de Comercio (SABIC)

El SABIC es un sistema diseñado y desarrollado por la Dirección de Sistemas Automatizados del Banco Central de Cuba para satisfacer las necesidades de procesamiento de datos de bancos e instituciones no bancarias, utilizando los medios técnicos de computación disponibles en el mercado. El sistema ha sido modificado en varias ocasiones en busca de satisfacer los requerimientos de las operaciones propias del Banco Nacional, logrando que el personal que opere mediante él pueda tramitar las operaciones sin necesidad de recurrir a archivos ni a la actividad manual, incrementando así la productividad, la eficiencia y seguridad en el trabajo.

Entre las características fundamentales que presenta el sistema se encuentran: la contabilización en tiempo real; que permite mantener actualizados los archivos contables en todo momento, la contabilización en varias monedas; para contabilizar los activos y pasivos en sus monedas de origen sin tener que realizar las conversiones correspondientes y la operación con transacciones; permitiendo realizar las operaciones usando transacciones que crean asientos automáticamente. Como otra de sus características está además su estructura modular que permite que le sean incorporados otros módulos de acuerdo a las particularidades de las instituciones.

El SABIC fue desarrollado empleando el lenguaje de programación FOXPRO, y su lógica de negocio se encuentra en su mayoría implementada en forma de procedimientos almacenados en la Base de Datos, realizada en SQL Server, además de ser ejecutado sobre MS DOS. Independientemente de los beneficios que ha representado el SABIC para el sistema bancario cubano, presenta incompatibilidad con el sistema de mensajería SISCOM que funciona sobre Windows y es el que emplea el Banco Nacional, presenta una pobre gestión de los reportes y no permite la realización de los procesos de tesorería.

Existe otra versión del SABIC, en el lenguaje de programación Visual FoxPro, que se ejecuta sobre Windows, con Base de Datos SQL Server realizada para el Banco Central de Cuba, la cual mejora la situación del mismo, pero sus características no se ajustan a las del Banco Nacional. (13)

1.6 Metodología definida para el desarrollo de la investigación

Una metodología es una colección de procedimientos, técnicas, herramientas y documentos auxiliares que ayudan a los desarrolladores de software en sus esfuerzos por implementar nuevos sistemas de información. Una metodología está formada por fases, cada una de las cuales se puede dividir en sub-fases, que guiarán a los desarrolladores de sistemas a elegir las técnicas más apropiadas en cada momento del proyecto y también a planificarlo, gestionarlo, controlarlo y evaluarlo. (14)

En la actualidad y a partir de la concepción de diferentes ideologías sobre cómo desarrollar un software han surgido un gran cúmulo de metodologías de desarrollo que atienden principalmente a aspectos importantes tales como, el tamaño del proyecto, la criticidad que presenta, la calificación del personal con el que se cuenta, así como la cultura que presenta el mismo. Comúnmente se dividen en dos grupos: las metodologías ágiles y las robustas. Las ágiles buscan minimizar los riesgos desarrollando software en cortos lapsos de tiempo y resaltan las comunicaciones cara a cara en vez de la documentación, además acentúan que la primera medida de progreso es el software funcional, entre ellas están: eXtreme Programming (XP), Scrum, Crystal, Open Unified Process (OpenUP), Agile Unified Proccess (AUP). Por su parte las metodologías robustas destacan por ser apropiadas para productos y proyectos grandes en las que existe un rigor de requisitos y diseño adecuado para los procesos de prueba, como ejemplo de estas se encuentran: RUP, Microsoft Solution Framework (MSF) e Iconix.

Para el desarrollo de esta investigación se ha seleccionado la metodología RUP (en inglés: Rational Unified Process) debido al alcance del proyecto y a que la misma aplica muchas de las mejores prácticas del desarrollo de software moderno, enfocadas a la producción de software con calidad.

1.6.1 Rational Unified Process

El Proceso Unificado es un proceso de desarrollo de software. Un proceso de desarrollo de software es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. El Proceso Unificado es más que un simple proceso, es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes niveles de aptitud y diferentes tamaños de proyecto.

El Proceso Unificado utiliza el UML (en inglés: Unified Modeling Language), para realizar todos los esquemas de un sistema de software, pero existen tres aspectos que realmente lo caracterizan y ellos son:

- ✓ **Dirigido por casos de uso:** Los casos de uso describen lo que los usuarios futuros necesitan y desean, lo que es captado en el modelado del negocio y representado a través de los requerimientos. A partir de este momento todos los artefactos que se obtienen en el desarrollo representan la realización de los casos de uso.
- ✓ **Centrado en la arquitectura:** La arquitectura provee al equipo de desarrollo y a los usuarios de una visión general del sistema en la que ambas partes deben de estar de acuerdo y describe los elementos del modelo que son más importantes para su construcción. La relación entre casos de usos y arquitectura es estrecha, es decir los casos de usos deben encajar en la arquitectura cuando se llevan a cabo mientras que la arquitectura debe permitir el desarrollo de todos los casos de usos requeridos.
- ✓ **Iterativo e incremental:** El desarrollo de un proyecto supone un gran esfuerzo que puede durar entre varios meses hasta posiblemente un año o más. Es práctico dividir el trabajo en partes más pequeñas o mini proyectos. Cada proyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en el flujo y los incrementos al crecimiento del producto. Para una efectividad máxima las iteraciones deben estar controladas.

1.6.2 Lenguaje de Modelado

El lenguaje propuesto por la metodología de desarrollo RUP para modelar elementos de software es UML (*Unified Modeling Language)*, UML es un lenguaje de modelado visual que permite visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software, soportando además el paradigma orientado a objetos. Se caracteriza por dividir a los sistemas en una estructura estática y un comportamiento dinámico**.** (15)

1.7 Patrones de Diseño

En el complejo mundo del desarrollo del software se ha hecho muy común el uso de los patrones, esto no se debe solamente a un gusto, sino que su empleo brinda a los equipos de desarrollo un arma que agiliza el diseño del sistema, debido a que establecen soluciones a problemas particulares del diseño, facilitan la reutilización del código y permiten una fácil comprensión debido a la documentación estándar que presentan. Una definición formal para el término patrón sería:

Un patrón es una pareja problema-solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas. Los patrones de diseño son la base para la

búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Según el libro “UML y Patrones” del autor Craig Larman publicado por la editorial Prentice Hall en el año 1999, se definen dos grupos de patrones:

Patrones GRASP

Los Patrones Generales para Asignar Responsabilidades: GRASP (en inglés: General Responsibility Asignment Software Patterns) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones, su nombre se debe a la importancia de captar estos principios, si se quiere diseñar eficazmente el software orientado a objetos. GRASP destaca 5 patrones principales: Experto, Creador, Bajo acoplamiento, Alta cohesión, Controlador.

Patrón Experto: este patrón consiste en asignar una responsabilidad al experto en información, es decir a la clase que cuenta con la información necesaria para cumplir con la responsabilidad. Se refuerza el encapsulamiento y se favorece el bajo acoplamiento.

Patrón Creador: guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. Su propósito principal es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Brinda soporte al bajo acoplamiento.

Patrón Bajo Acoplamiento: pretende asignar una responsabilidad para mantener el bajo acoplamiento, es decir el diseño de clases más independientes, que no se relacionen con muchas otras, que reducen el impacto de los cambios, que son más reutilizables y acrecienten la oportunidad de una mayor productividad.

Patrón Alta Cohesión: su objetivo es asignar una responsabilidad de modo que la cohesión siga siendo alta. Las clases con alta cohesión se caracterizan por tener responsabilidades estrechamente relacionadas y no realizar un trabajo enorme. Una clase con alta cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizarla. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño.

Patrón Controlador: consiste en asignar la responsabilidad a una clase de manejar los mensajes correspondientes a eventos de un sistema. El controlador es un intermediario entre la interfaz de usuario y el núcleo de las clases donde reside la lógica de la aplicación. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación para aumentar la reutilización de código y a la vez tener un mayor control. (15)

Patrones GoF

Por sus siglas en inglés Gang of Four. Llamados así debido a que fueron cuatro autores los que escribieron el libro “Design Patterns” que ilustra sus funciones. Los patrones de diseño GoF se clasifican en 3 grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

Patrón Fachada: patrón estructural que trata de simplificar la interfaz entre dos sistemas o componentes de software ocultando un sistema complejo detrás de una clase que hace las veces de pantalla o fachada. La idea principal es la de ocultar todo lo posible la complejidad de un sistema, el conjunto de clases o componentes que lo forman, de forma que solo se ofrezca un punto de entrada al sistema tapado por la fachada. Su uso aísla los posibles cambios que se puedan producir en alguna de las partes.

Patrón Mediador: patrón de comportamiento que coordina las relaciones entre sus asociados. Permite la interacción de varios objetos, sin generar acoples fuertes en esas relaciones.

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

Patrón Singleton: patrón creacional diseñado para restringir la creación de objetos pertenecientes a una clase. Su objetivo es garantizar que una determinada clase solo tenga una instancia y proporcionar un punto de acceso global a esta. Es un patrón muy sencillo de diseñar y a menudo es implementado por otros patrones. (16)

Patrón DAO

Por sus siglas en inglés Data Access Object. Este patrón maneja la conexión con la fuente de datos para obtener y almacenar datos. Su uso ofrece varios beneficios para la persistencia de datos como son: la separación el acceso a datos de la lógica de negocio, lo cual suele ser altamente recomendable en sistemas medianos o grandes, o que manejen lógica de negocio compleja. Permite el encapsulamiento de la fuente de datos, lo cual es especialmente beneficioso en sistemas con acceso a múltiples entradas; posibilita el ocultamiento de la API con la que se accede a los datos, lo que viabiliza que se pueda cambiar el negocio del manejo de los datos persistentes sin que afecte el resto, por ejemplo cambiar de framework de acceso a datos, centralizando todo el acceso a datos en una capa independiente. (17)

1.8 Ambiente de Desarrollo

1.8.1 Lenguajes de programación

Lenguaje de programación en el lado del servidor

Hoy en día existe una fuerte tendencia al desarrollo de aplicaciones Web favorecido por el desarrollo de Internet y las facilidades que este brinda como se explicaba al principio de este capítulo. Un aspecto importante de este tipo de aplicaciones lo constituye el lenguaje que se emplee en el servidor, que es el encargado de ejecutarse en el mismo y del cual los usuarios obtienen el beneficio del procesamiento de la información.

En la actualidad coexisten gran número de lenguajes por los que un equipo de desarrollo puede optar para realizar esta función, pero el lenguaje Java en particular marca la delantera en este sentido, atendiendo a la popularidad que presenta dentro de la comunidad de desarrollo de software, el cúmulo de sistemas que han sido desarrollados sobre el mismo, además de los resultados que se han obtenido mediante la explotación de aplicaciones basadas en el, lo antes expuesto evidencia su robustez y dominio. (18)

Java

Java es un lenguaje de programación orientado a objetos, que permite a los programadores realizar aplicaciones de múltiples tipos, ya sean de escritorio o web. Se caracteriza por ser un lenguaje simple, robusto y poderoso que se torna fácil de aprender, debido a que elimina sentencias de bajo nivel además del Garbage Collector (en español: Recolector de Basura) haciendo transparente para los programadores el manejo de la memoria. (19)

Se destaca por ser un lenguaje de código abierto, multiplataforma por lo cual ha logrado una gran expansión por todo el mundo. En la actualidad incluye un gran número de librerías para múltiples trabajos como: el trabajo con la red, tratamiento de excepciones, hilos para el procesamiento concurrente entre otras.

Plataforma JEE

JEE (en inglés: Java Enterprise Edition) es una plataforma de programación, que define estándares para desarrollar y ejecutar aplicaciones en el lenguaje de programación Java, empleando arquitecturas que

definen un modelo multicapa y que se apoyan en componentes de software modulares. JEE incluye tecnologías, tales como Servlets, JSP (en inglés: Java Server Pages) y varias tecnologías de servicios web. Las aplicaciones desarrolladas en esta plataforma tienden a ser portables, escalables, robustas y seguras a la vez que son integrables con tecnologías anteriores. (20)

Lenguaje de programación en el lado del cliente

En el caso de los lenguajes que sustentan la aplicación en el cliente, se definieron por parte del proyecto el uso de:

XHTML

XHTML (en inglés: Extensible Hypertext Markup Languaje) es el lenguaje de marcado pensado para sustituir a HTML (en inglés: Hypertext Markup Languaje), es una reformulación del mismo que es compatible con XML (en inglés: Extensible Markup Languaje) Se utiliza para generar documentos y contenidos de hipertexto generalmente publicados en la WEB.

El documento se escribe en forma de etiquetas, que hasta cierto punto pueden establecer la apariencia del documento, aunque en la actualidad se suele mezclar con lenguajes script como JavaScript, para lograr mayor interacción con los usuarios. (21)

JavaScript

Es un lenguaje interpretado, es decir que no requiere compilación y se utiliza comúnmente para la construcción de páginas web en combinación con el XHTML. JavaScript no es orientado a objetos debido a que no soporta la herencia de objetos, sino que está basado en objetos que incorpora para su funcionalidad, aunque permite la creación de objetos propios.

Se caracteriza por ser un lenguaje manejado por eventos por el hecho de responder a eventos generados ya sea por el usuario o por el navegador, es independiente de la plataforma debido a que solo se necesita un navegador para ejecutar el código, permite un desarrollo rápido y es relativamente fácil de aprender. Es soportado por la mayoría de los navegadores como Internet Explorer, Netscape y Mozilla Firefox.

1.8.2 Frameworks

Un framework es una estructura conceptual y tecnológica de soporte definido, normalmente, con artefactos o módulos de software concretos, en base a la cual otro proyecto de software puede ser organizado y desarrollado. 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. (22)

En el contexto actual de la industria de software se cuenta con grupo considerable de frameworks que facilitan y agilizan el desarrollo de sistemas. Específicamente para los lenguajes seleccionados con anterioridad existen un conjunto de ellos que se emplean con mucha frecuencia. Para lo referente al trabajo de acceso a datos se pueden mencionar a algunos frameworks de elevado prestigio como Athena Framework, Ebean ORM e Hibernate Framework.

Por otro lado están un conjunto de frameworks que hacen posible la aplicación del patrón MVC; propuesta de la arquitectura base, que poseen una elevada aceptación a nivel mundial, entre ellos: Struts, JBoss Seam y Spring Framework, vale destacar que los dos últimos presentan un marcado uso en la universidad.

De igual manera para la lógica de presentación se utilizan frameworks, debido a la importancia que esta posee en los sistemas informáticos, teniendo en cuenta su interacción con los usuarios. Diversas son las librerías desarrolladas para JavaScript, que enriquecen las interfaz de usuario y brindan un ambiente de trabajo más limpio en el uso de Ajax (en inglés: Asynchronous JavaScript and XML), entre ellas destacan Qooxdoo, Ext JS y Dojo Toolkit. Ateniendo a las características del proyecto y las bondades que brindan cada uno de estos frameworks el equipo de arquitectura decidió usar los que se caracterizan a continuación:

Spring Framework

Es un framework bajo licencia de código abierto concebido para el desarrollo de aplicaciones basadas en la plataforma Java/JEE. Spring ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas que permiten desarrollar un software seguro y robusto haciendo uso de las prácticas comunes en la industria de software. Su funcionamiento se basa en inversión de control e inyección de

dependencia, apoyándose en el API de Java Reflection. El framework para su correcto funcionamiento está dividido en los siguientes módulos:

Spring Core: Este representa el núcleo de Spring, es donde se encuentran funcionalidades fundamentales que provee el framework.

Spring AOP: Este módulo ofrece un extenso soporte para Programación Orientada a Aspectos, permitiendo definir entre otras cosas la política transaccional y de seguridad de una aplicación.

Spring ORM: En este módulo se brinda el soporte necesario para la integración con los frameworks Hibernate e iBates.

Spring DAO: Provee un trabajo con JDBC en aras de hacer el código de acceso a datos más limpio y entendible, ofrece además una capa de excepciones para el manejo de los errores emitidos por los servidores de base de datos.

Spring Web: Es el encargado de crear el contexto para aplicaciones web, incluye soporte para una variedad de tareas como la subida de archivos, la vinculación de parámetros de las peticiones a objetos del negocio, etc.

Spring Web MVC: Ofrece gran soporte para el desarrollo de aplicaciones web utilizando la filosofía Modelo-Vista-Controlador, emplea la inversión de control para brindar una separación entre la lógica de los controladores y los objetos del negocio.

Spring Context: Este módulo es el que consagra a Spring como un framework, ofrece soporte para la internacionalización, la aplicación de eventos de ciclo de vida. Brinda soporte a servicios como correo electrónico, acceso JNDI, integración con EJB, etc.

Se hace uso de Spring Framework en su versión 2.5.

JUnit

Framework desarrollado para realizar pruebas automatizadas a los sistemas informáticos durante la etapa de construcción. Su objetivo principal es evaluar el funcionamiento de cada uno de los métodos dentro de las clases que conforman el software. Facilita la aplicación de la práctica Test Driven Development (TDD). Incluye formas de ver los resultados que pueden ser en modo texto o gráfico. Es uno de los más utilizados de su tipo en el lenguaje Java, cuenta con una amplia comunidad de desarrollo y es bastante maduro. (23)

Para llevar a cabo las pruebas unitarias se hace uso de la versión 4.10.

Hibernate Framework

Hibernate es una solución ORM (en inglés: Object Relacional Mapping) para el lenguaje de programación Java, concebido bajo la filosofía del código abierto. Busca solucionar el problema de la diferencia entre el modelo de objetos y el modelo relacional, realizando un mapeo entre tablas de la base de datos y los objetos del negocio a través de archivos declarativos, en este caso archivos XML. Soporta la conexión a una gran variedad de servidores de base de datos, como PostgreSQL, Oracle, SQLServer y otros, permite además adaptarse a una base de datos ya existente, así como generar la base de datos a partir de un modelo objetual.

Entre sus funciones se encuentran: permitir transacciones, asociaciones, polimorfismo, herencia, carga mediante referencia, persistencia transitiva. Ofrece también un lenguaje de consulta denominado HQL (en inglés: Hibernate Query Language), siendo este una poderosa vía de comunicación entre el programador y la base de datos, debido a que permite realizar consultas basándose en los objetos del negocio y no en las tablas de la base de datos. Permite la ejecución de consultas SQL (en inglés: Standrad Query Language). (24)

Se hace uso de Hibernate Framework en su versión 3.5.

Dojo Toolkit

Dojo Toolkit es una colección de scripts estáticos que permiten el desarrollo de aplicaciones web enriquecidas en el cliente, incorpora soporte para el trabajo con la tecnología AJAX. Se destaca por permitir desarrollar aplicaciones que pueden funcionar con independencia del navegador web donde se ejecuten, hace transparente el desarrollo para diferentes implementaciones del DOM, ofrece soporte para la internacionalización e incluye un gran cúmulo de componentes visuales basados en XHTML, CSS y JavaScript para enriquecer la interfaz de usuario. Presenta una arquitectura modular donde destacan los módulos: Dojo, Dijit y Dojox. (25)

Se hace uso de Dojo Toolkit en su versión 1.3.

1.8.3 Herramientas de desarrollo

Herramienta CASE

Una herramienta CASE (en inglés: Computer Aided Software Engineering) consiste en diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas. Este tipo de herramientas ayuda en todos los aspectos del ciclo de vida del desarrollo del software como la realización de su diseño, generación de código a partir de un diseño dado, documentación, entre otros.

En sentido general estas herramientas intentan dar ayuda automatizada al proceso de desarrollo de software, sirven de apoyo a la metodología de desarrollo empleada.

Visual Paradigm

Herramienta multiplataforma para el modelado UML que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue. Ayuda a una más rápida construcción de aplicaciones de calidad. Permite modelar diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Otra gran utilidad que presenta es la capacidad de integrarse con IDE (en inglés: Integrated Development Environment) como el Eclipse y el Netbeans. Su principal dificultad radica en que posee una licencia muy restringida. (26)

Se estará haciendo uso de Visual Paradigm en su versión 6.4.

Control de versiones

Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas. Un sistema de control de versiones debe proporcionar un mecanismo de almacenaje de los elementos que deba gestionar y un registro histórico de las acciones realizadas con cada elemento o conjunto de elementos entre otros aspectos.

Subversion

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. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. 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 Subeclipse para integrarlo con Eclipse. (27)

Se hace uso de Subversion en su versión 1.6.6.

Entorno integrado de desarrollo. Eclipse IDE

Eclipse es un entorno de desarrollo integrado de código abierto y multiplataforma desarrollado por IBM. Emplea plug-ins para proporcionar toda su funcionalidad, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. La arquitectura de plug-ins permite integrar diversos lenguajes, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo, tales como: herramientas UML, editores visuales de interfaces, ayuda en línea para librerías, entre otros. (28)

Se estará haciendo uso de Eclipse IDE en su versión 3.4.

Servidor de aplicaciones web. Tomcat

Tomcat es un contenedor de servlets bajo la filosofía del código abierto licenciado con Apache Sofware License que presenta la ventaja de ser multiplataforma. Implementa las especificaciones de Servlets 2.5 y JSP (en inglés: Java Server Pages) 2.1. Con frecuencia se presenta en combinación con el servidor web Apache aunque puede realizar esta función por sí mismo. En la actualidad es utilizado como un servidor web autónomo en entornos donde existe un alto nivel de tráfico y alta disponibilidad. (29)

Se hace uso de Tomcat en su versión 6.

Servidor de Base de Datos. Microsoft SQL Server 2005

SQL Server es un servidor de base de datos basado en el modelo relacional. 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. Sus lenguajes de consulta son el SQL y el T-SQL (en inglés: Transact-SQL), siendo este último el principal medio de programación y administración del servidor. Requiere para su funcionamiento un sistema operativo Microsoft Windows, representando esto un inconveniente para su utilización. (30)

1.9 Valoración del estado del arte

En sentido general los sistemas antes expuestos constituyen buenas soluciones informáticas para el mundo financiero, debido a la gama de funcionalidades que brindan, posibilitando la gestión de las operaciones bancarias. Los sistemas extranjeros antes mencionados están sustentados por tecnologías que hoy en día marcan un elevado nivel en la industria de software, atendiendo al número de aplicaciones desarrolladas sobre las mismas que han alcanzado gran prestigio internacionalmente. En estas soluciones informáticas se destaca como otra de sus características la multimoneda, elemento clave para la actividad económica y financiera a nivel mundial, que es a su vez de vital importancia para el Banco Nacional de Cuba debido al papel que este desempeña en el país como banco comercial, que interactúa directamente con el exterior.

Un aspecto a tener en cuenta en el sistema bancario cubano es la dualidad monetaria; característica de la economía del país resultante de las diversas transformaciones que se han venido desarrollando y que en el ámbito internacional es un término poco difundido, para lo cual no fueron concebidos estos sistemas.

Por otro lado las licencias de dichos productos constituyen una limitante; teniendo en cuenta que son privativas y que poseen altos precios, lo que trae consigo que se dificulte tanto la adquisición como el uso y mantenimiento de los mismos, a partir de que Cuba no está en condiciones de invertir en la compra de estas licencias, independientemente de esto pueden servir de base para la definición de tecnologías, herramientas y plataformas para el desarrollo de soluciones informáticas relacionadas con esta área de conocimientos.

1.10 Conclusiones del capítulo

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

Las tecnologías seleccionadas para el desarrollo de la solución están acordes a los requerimientos del cliente, a las políticas del centro y a los estándares internacionales: RUP como metodología de desarrollo, UML como lenguaje de modelado, Visual Paradigm como herramienta CASE, Java (en su especificación JEE) como lenguaje de programación, Spring como framework núcleo de la aplicación apoyado por los frameworks Hibernate y Dojo Toolkit, Eclipse como herramienta de desarrollo, SQL Server 2005 como gestor de base de datos, y como contenedor web el Apache Tomcat.

Capítulo 2: Arquitectura y Diseño de la solución.

2.1 Introducción

El presente capítulo enmarca su contenido en explicar la arquitectura del sistema Quarxo y el diseño de los módulos Compraventa y Transferencia del subsistema Tesoreria. Se presentan el modelo de clases del diseño, el modelo de datos, así como los diagramas de interacción de las funcionalidades más significativas.

2.2 Descripción de la arquitectura

"La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución". (31)

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

2.2.1 Estructura del sistema

El sistema está estructurado a través de subsistemas, módulos y componentes. Los subsistemas, módulos y componentes son definidos según las funcionalidades identificadas en la captura de requisitos. Los subsistemas agrupan un conjunto de módulos relacionados con los procesos que ejecutan. Los módulos agrupan un conjunto de casos de uso que representan uno o más procesos bancarios estrechamente relacionados y por último los componentes son un conjunto de funcionalidades comunes que son reutilizados por otros módulos del sistema. En la figura 1 se muestra una representación de las capas lógicas. Para una mejor comprensión de las capas lógicas por las que está compuesto el sistema de acuerdo a la figura 1, se realiza una breve descripción de cada una de ellas.

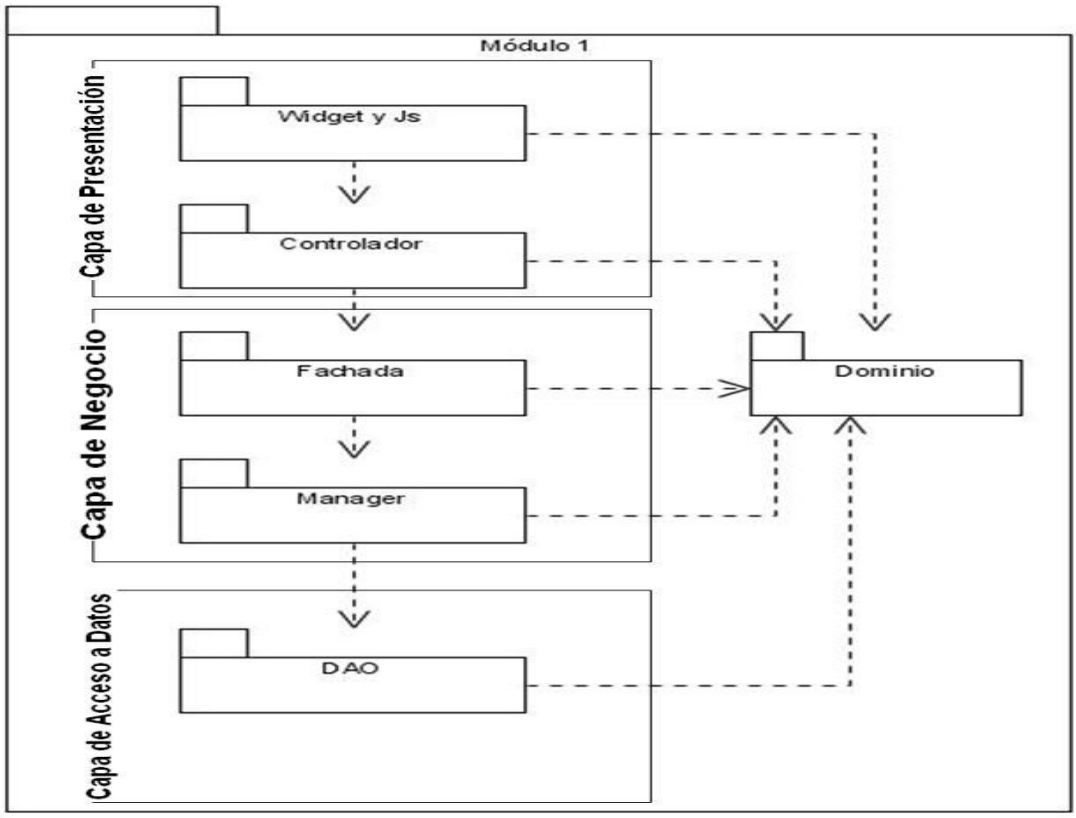


Figura 1. Arquitectura de las capas lógicas de Quarxo. (32)

Capa de presentación

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

Capa de negocio

Esta capa está divida en dos subcapas:

- La subcapa Fachada agrupará las funcionalidades según su naturaleza para que pueda ser invocada desde la capa de presentación, sin realizar la lógica de negocio, delegando de esta forma

a la subcapa Manager la realización de la lógica del negocio. Esta será además, el punto de intercambio entre la capa de presentación y la capa de negocio.

- La subcapa Manager es donde se realizará la lógica del negocio conteniendo la jerarquía de clases indicadas para su implementación. Además esta subcapa utilizará la capa de acceso a datos para el trabajo con la persistencia de los datos y la capa de dominio para la generación de los objetos del dominio.

Capa de acceso a datos

En esta capa se realizarán todas las operaciones relacionadas con el gestor de base de datos, permitiendo así la persistencia y el acceso a la información de la aplicación. La interacción con la capa de negocio se realizará a través de interfaces.

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

2.3 Elementos de entrada al diseño

Durante el desarrollo de un software el diseño se realiza con el fin de traducir los requisitos a una estructura que describe cómo implementar el sistema, siendo lo suficientemente específica para que no existan ambigüedades y ofreciendo la posibilidad de descomponer los trabajos de implementación en componentes más manejables, visualizándolos mediante una notación común. Para lograr una mejor comprensión del diseño e implementación del sistema a desarrollar se muestran a continuación los principales artefactos obtenidos en el análisis por parte del equipo de analistas del proyecto SAGEB.

2.3.1 Especificación de los requisitos funcionales

Los requisitos funcionales son los que definen el comportamiento interno del sistema. Abarcan las funcionalidades específicas que el software debe ser capaz de realizar. Condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente. (33)

Requisitos Funcionales		Prioridad
RF1:Registrar compraventa	El sistema muestra el formulario para registrar una compraventa.	Alta
RF2:Actualizar compraventa	El sistema muestra el formulario con los datos de la compraventa a actualizar.	Alta
RF3:Consultar compraventa	El sistema muestra el formulario con los datos de la compraventa a consultar.	Alta
RF4:Buscar compraventa	El sistema muestra el formulario para buscar la compraventa.	Alta
RF5:Adicionar cuenta	El sistema muestra el formulario para adicionar una cuenta.	Alta
RF6:Eliminar cuenta	El sistema muestra el formulario para eliminar una cuenta.	Alta
RF7: Registrar transferencia	El sistema muestra el formulario para registrar una transferencia.	Alta
RF8: Actualizar transferencia	El sistema muestra el formulario con los datos de la transferencia a actualizar.	Alta
RF9: Consultar transferencia	El sistema muestra el formulario con los datos de la transferencia a consultar.	Alta
RF10: Buscar transferencia	El sistema muestra el formulario para buscar la transferencia.	Alta
RF11: Emitir transferencia	El sistema muestra el formulario con los datos de la compraventa a emitir.	Alta
RF12: Emitir compraventa	El sistema muestra el formulario con los datos de la transferencia a emitir.	Alta

Tabla 1. Requisitos Funcionales

2.3.2 Especificación de los requisitos no funcionales

Especifican criterios que pueden usarse para calificar las funcionalidades de un sistema en lugar de sus comportamientos específicos. Definen propiedades y restricciones del sistema. (33)

De acuerdo con las características sobre las cuales se desarrolla este subsistema se toman en cuenta los siguientes requisitos no funcionales (RNF).

Funcionalidad

RNF 01 El sistema mostrará los errores en forma de mensajes.

- ✓ Todos los mensajes de error del sistema deberán incluir una descripción textual del error.

Usabilidad

RNF 02 Los formularios serán estandarizados, por tanto:

- ✓ Los campos de texto tendrán un tamaño estándar de acuerdo con el espacio que se tenga en el área de la página y en la medida que se llene esa área primaria agregar la barra de desplazamiento vertical.
- ✓ No se utilizarán textos extensos para las etiquetas de la interfaz de usuario.

RNF 03 El menú de navegación estará disponible en todas las páginas.

RNF 04 En caso de que los resultados de las consultas tengan más de 5 coincidencias, estos se mostrarán de forma paginada en una tabla.

- ✓ Se mostrará en la parte inferior de la tabla el total de elementos encontrados, enlaces de navegación: ir hacia delante, hacia atrás o ir al inicio de los resultados mostrados.

Fiabilidad

RNF 05 El sistema estará disponible durante toda la jornada laboral del BNC.

Seguridad

RNF 06 El sistema permitirá la visualización de la información según el usuario autenticado.

RNF 07 El sistema implementará el uso de campos obligatorios y validaciones para garantizar la integridad de la información que se introduce por el usuario.

Restricciones de diseño

RNF 08 El sistema se implementará usando la plataforma JEE.

RNF 09 El sistema estará basado en un estilo arquitectónico en capas.

Interfaz de Usuario

RNF 10 Todos los textos y mensajes en pantalla aparecerán en idioma español. Los errores serán visibles al usuario y en lo posible incluirán sugerencias de las posibles soluciones.

RNF 11 El sistema presentará los términos capitalizados, es decir, tendrán su primera letra en mayúsculas.

Uno de los principales artefactos del flujo de trabajo de requerimientos es el diagrama de casos de uso del sistema, que representa gráficamente a los procesos y su interacción con los actores del sistema. Este diagrama es uno de los artefactos fundamentales que dan paso al diseño. Un caso de uso es una unidad coherente de funcionalidad, expresada como transacción entre los actores y el sistema. El designio de esta vista de casos de uso es enumerar a los actores y los casos de uso y señalar qué actores participan en cada caso de uso.

A continuación se ilustra el diagrama de casos de uso del sistema.

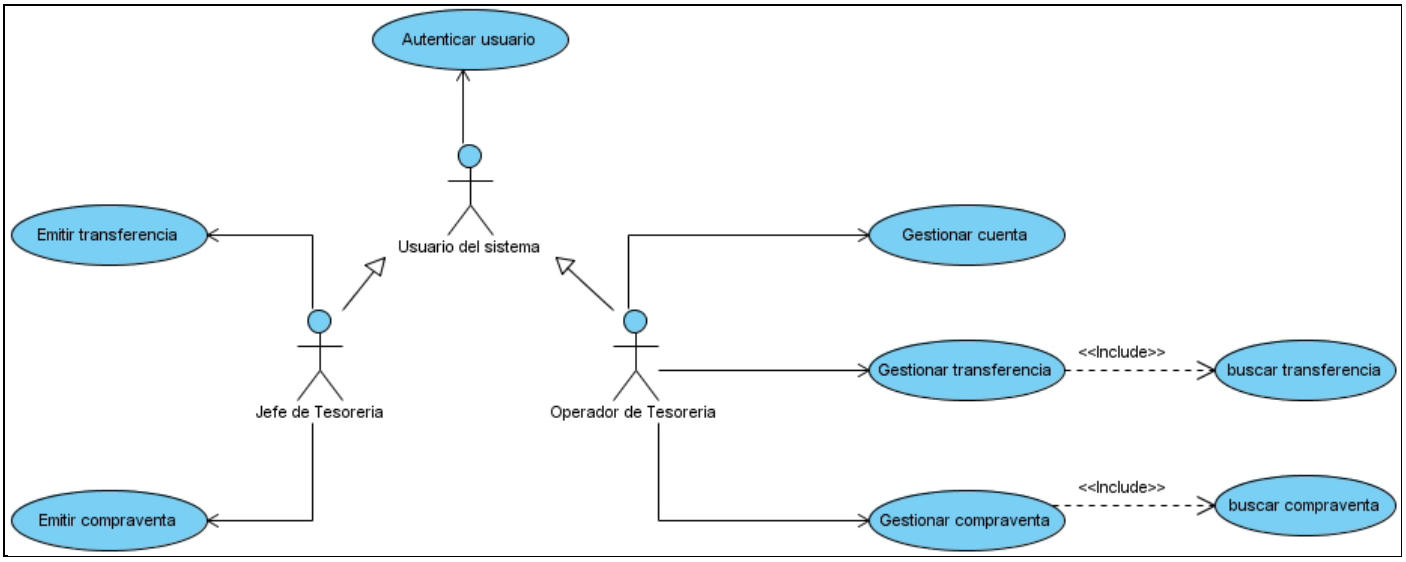


Figura 2. Diagrama de casos de uso del sistema.

2.4 Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales junto a otro grupo de restricciones relacionadas con el

entorno de implementación tienen impacto en el sistema, está compuesto por otros artefactos como los diagramas de paquetes y diagramas de clases, además sirve como abstracción a la implementación y se empleará como entrada fundamental de las actividades de esta etapa. (34)

Diagrama de paquetes

Durante el desarrollo de software resulta muy conveniente agrupar clases y ficheros por diferentes criterios que ayudarán a una fácil comprensión de la aplicación, resultando conveniente el desarrollo de los diagramas de paquetes, los cuales muestran como está dividido el sistema en agrupaciones lógicas mostrando las dependencias entre estas. En el caso del subsistema Tesorería la agrupación se realizó en dependencia de las funcionalidades que desempeñan estos ficheros en el sistema.

A continuación se muestra la estructura y dependencia de paquetes del módulo compraventa y una explicación de la composición de cada uno. El diagrama de paquetes que comprende al módulo de transferencia se puede consultar en el Anexo 6.

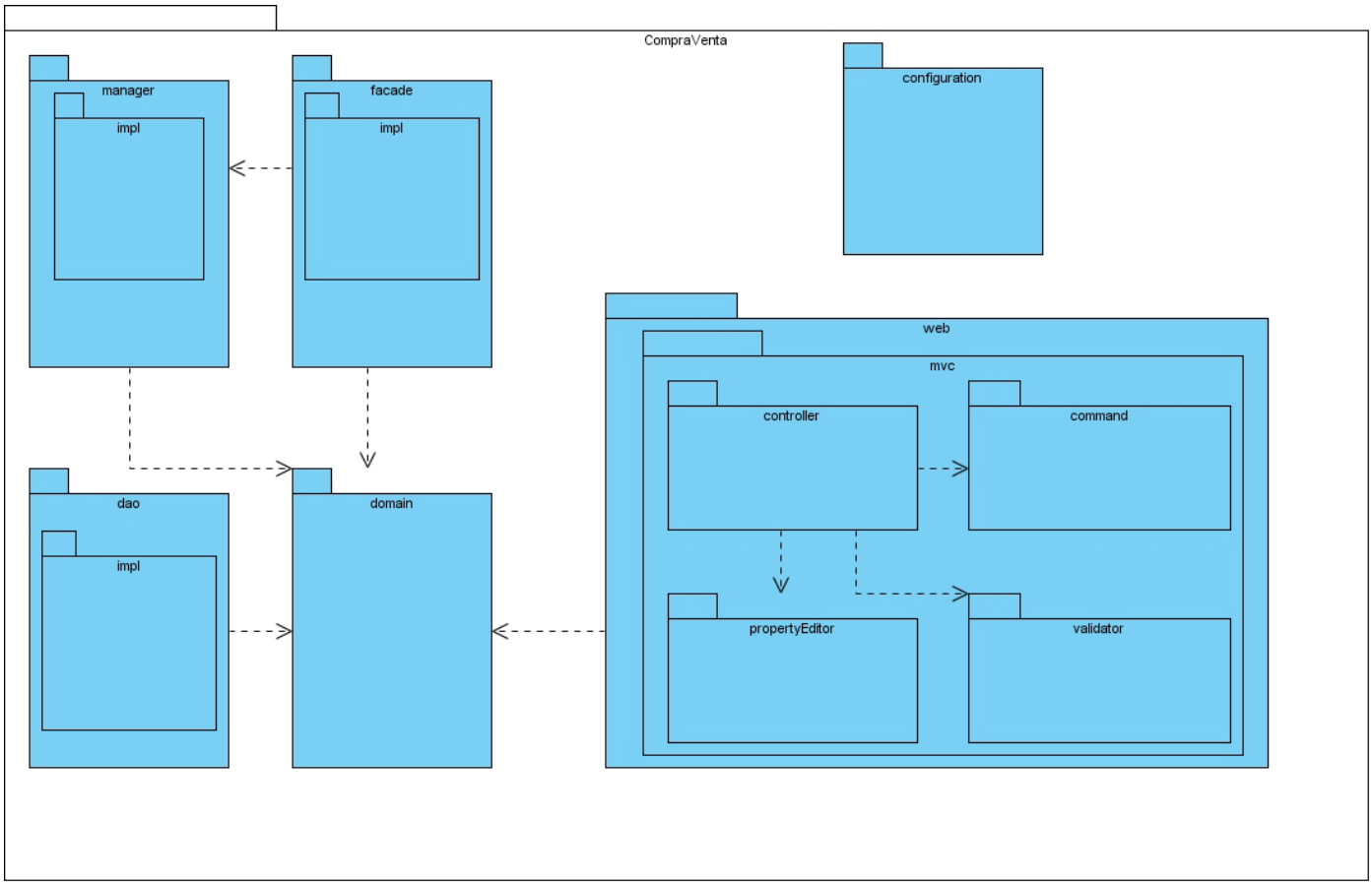


Figura 2. Diagrama de paquetes del módulo Compraventa

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

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

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

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

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

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

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

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

Paquete controller: En este paquete se hallan las clases que extienden de los controladores propuestos por Spring MVC, encargadas de responder a las peticiones realizadas por el cliente.

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

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

Paquete validator: Cuenta con las clases encargadas de realizar la validación de datos en el lado del servidor. (32)

Diagrama de clases del diseño

Una realización de caso de uso del diseño es una colaboración del modelo de diseño que describe como se realiza un caso de uso específico y como se ejecuta en término de clases de diseño y sus objetos. Una realización contiene diagramas de clases que muestran sus clases de diseño participantes y diagramas de interacción que muestran la realización de un flujo o escenario en concreto, en términos de interacción entre objetos. (34)

Específicamente los diagramas de clases de diseño son muy útiles porque muestran a través de atributos y métodos la estructura de las clases que después serán escritas en algún lenguaje de programación. A continuación se muestra el diagrama de clases del diseño del módulo Compraventa, el mismo está dividido por capas para una mejor comprensión. El diagrama de clases del diseño del módulo Transferencia se puede consultar en el Anexo 1.

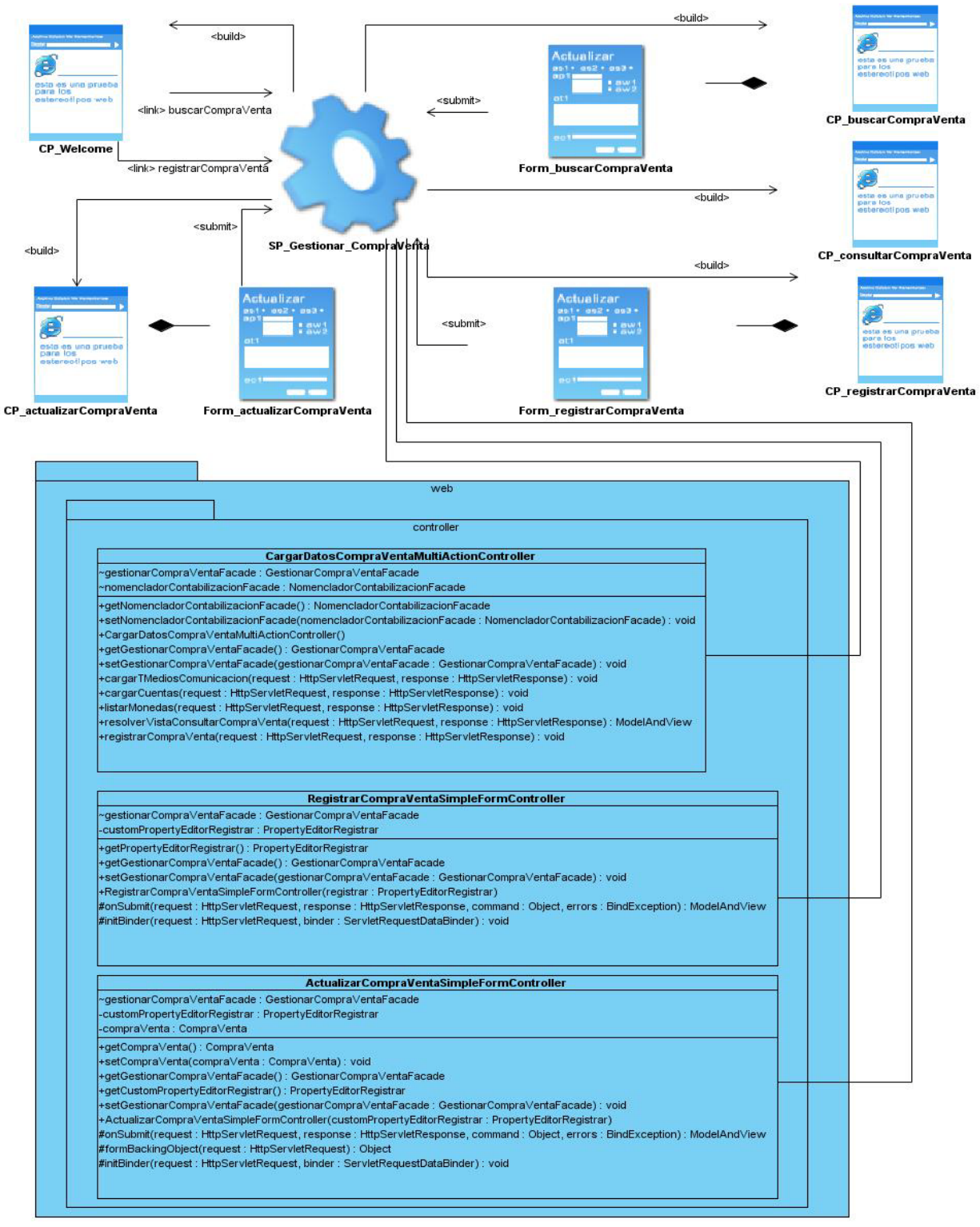


Figura 1. Capa de presentación del diagrama de clases del diseño del módulo Compraventa.

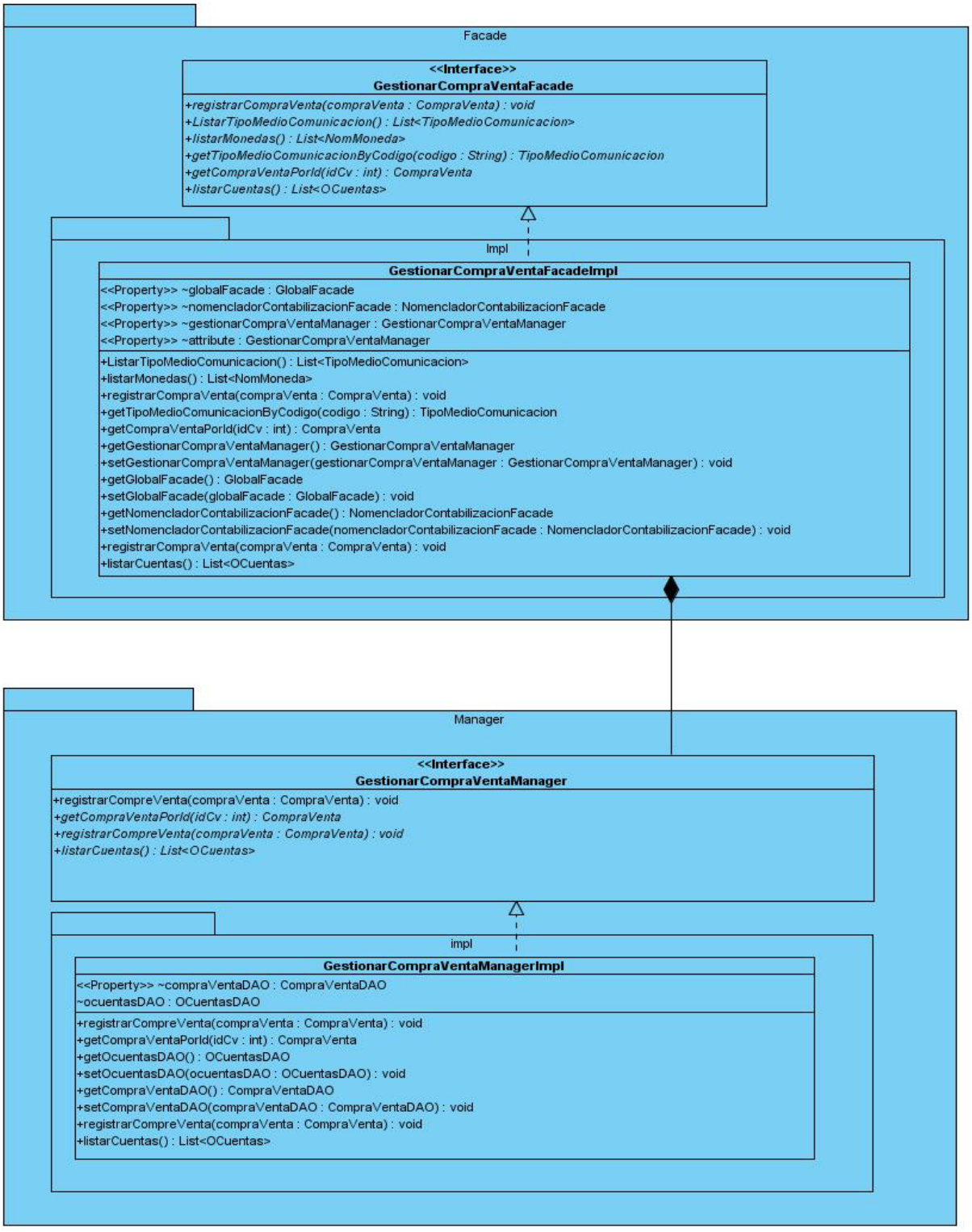


Figura 4. Capa de negocio del diagrama de clases del diseño del módulo Compraventa.

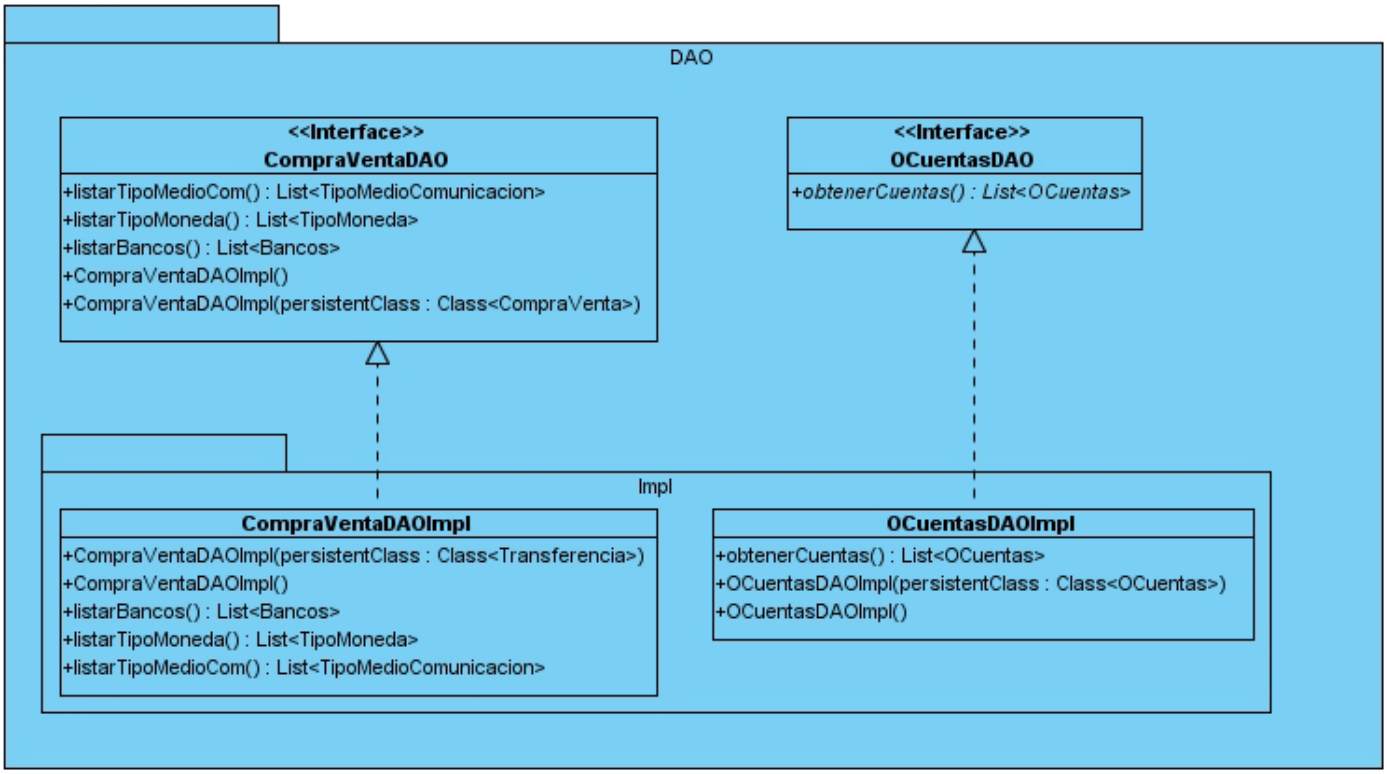


Figura 5. Capa de acceso a datos del diagrama de clases del diseño del módulo Compraventa.

Debido a la utilización de frameworks y los estándares definidos por el grupo arquitectura del proyecto, en la realización de los diagramas mostrados con anterioridad, fue necesario describir determinadas clases que pudieran crear confusión al tratar de comprender dichos diagramas, a continuación se brinda una breve descripción de estas para una mejor comprensión de los mismos:

Clases ClientPage (CP): páginas web encargadas de mostrar los formularios e información al usuario.

CargarDatosCompraVentaMultiActionController: clase que gestiona las acciones asociadas a registrar, buscar y consultar una compraventa.

GestionarCompraVentaFacade: interfaz encargada de brindar las funcionalidades del módulo a la capa de presentación y a otros módulos que la requieran.

GestionarCompraVentaFacadeImpl: clase encargada de implementar la lógica de programación de la Interfaz **GestionarCompraVentaFacade**.

GestionarCompraVentaManager: clase que contiene las funcionalidades que se deben implementar en la clase GestionarCompraVentaManagerImpl para dar respuesta a las acciones que se solicitan desde **GestionarCompraVentaFacadeImpl**.

CompraVentaDAO: interfaz de comunicación que se encarga de brindar las funcionalidades de la capa de acceso a datos.

CompraVentaDAOImpl: clase que implementa las funcionalidades de la capa de acceso a datos.

Diagramas de interacción

Un diagrama de interacción consiste en un conjunto de objetos y sus relaciones, incluyendo los mensajes que se pueden enviar entre ellos. Se les atribuye una gran importancia debido a la comprensión del sistema a la que puede llegarse a través de ellos. (35)

Se realizaron diagramas de secuencia, que son un ejemplo de diagramas de interacción que destacan principalmente el orden temporal de los mensajes para varios escenarios del subsistema en cuestión, a continuación se presenta el diagrama de secuencia asociado al escenario **Registrar Compraventa** perteneciente al **CU Gestionar Compraventa,** los restantes diagramas pueden ser consultados en el anexo 5 y en el Anexo 2 se pueden consultar los diagramas de secuencia correspondientes al **CU Gestionar Transferencia**.

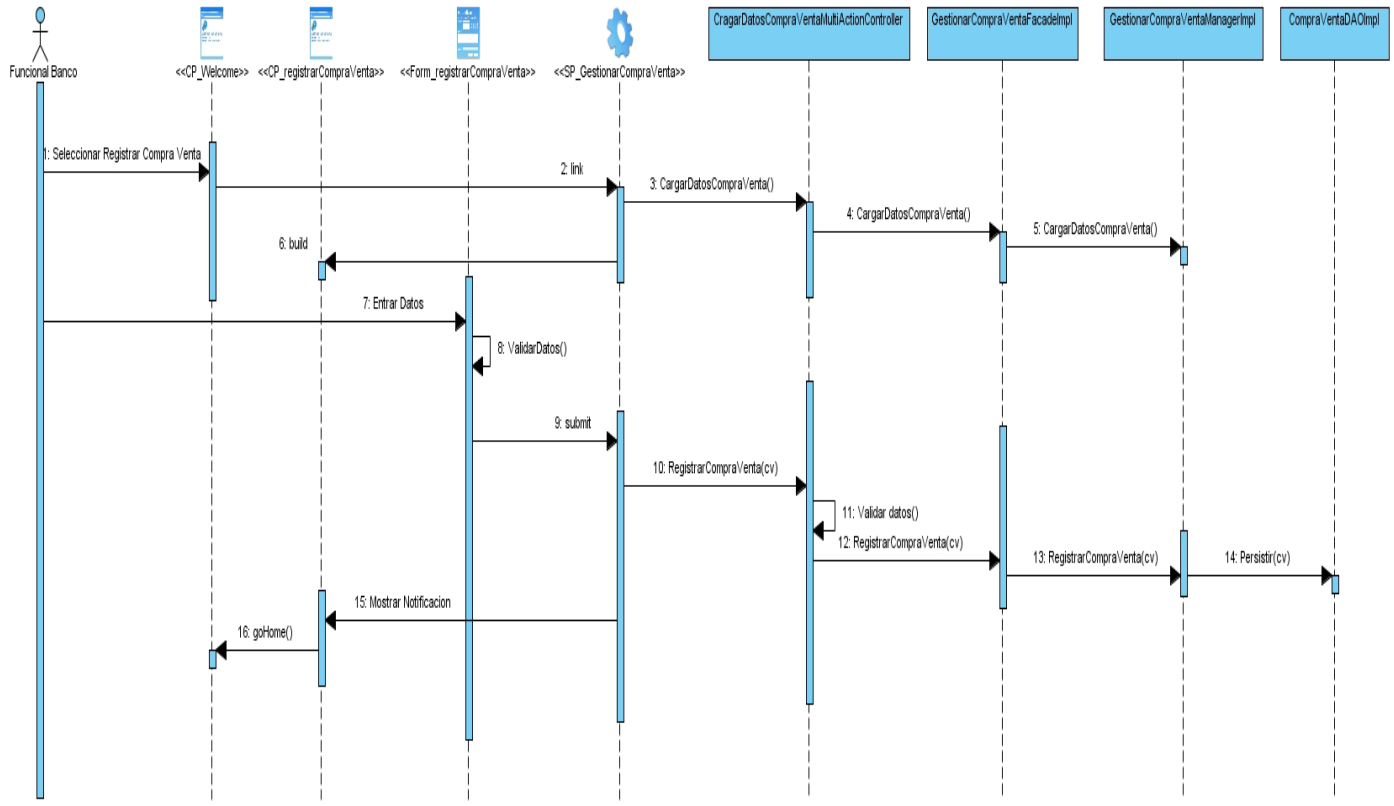


Figura 6. Diagrama de interacción del escenario Registrar Compraventa del CU Gestionar Compraventa.

2.5 Modelo de datos

El Modelo de datos se utiliza para la descripción de una base de datos. Este, por su parte, permite describir las estructuras de datos de la base (el tipo de datos que incluye la base y la forma en que se relacionan), las relaciones de integridad (las condiciones que los datos deben cumplir para reflejar correctamente la realidad deseada) y las operaciones de manipulación de los datos (agregado, borrado, modificación y recuperación de los datos de la base). Para la construcción del modelo de datos propuesto en la solución, se tuvo en cuenta la reducción a la mínima expresión de los campos nulos.

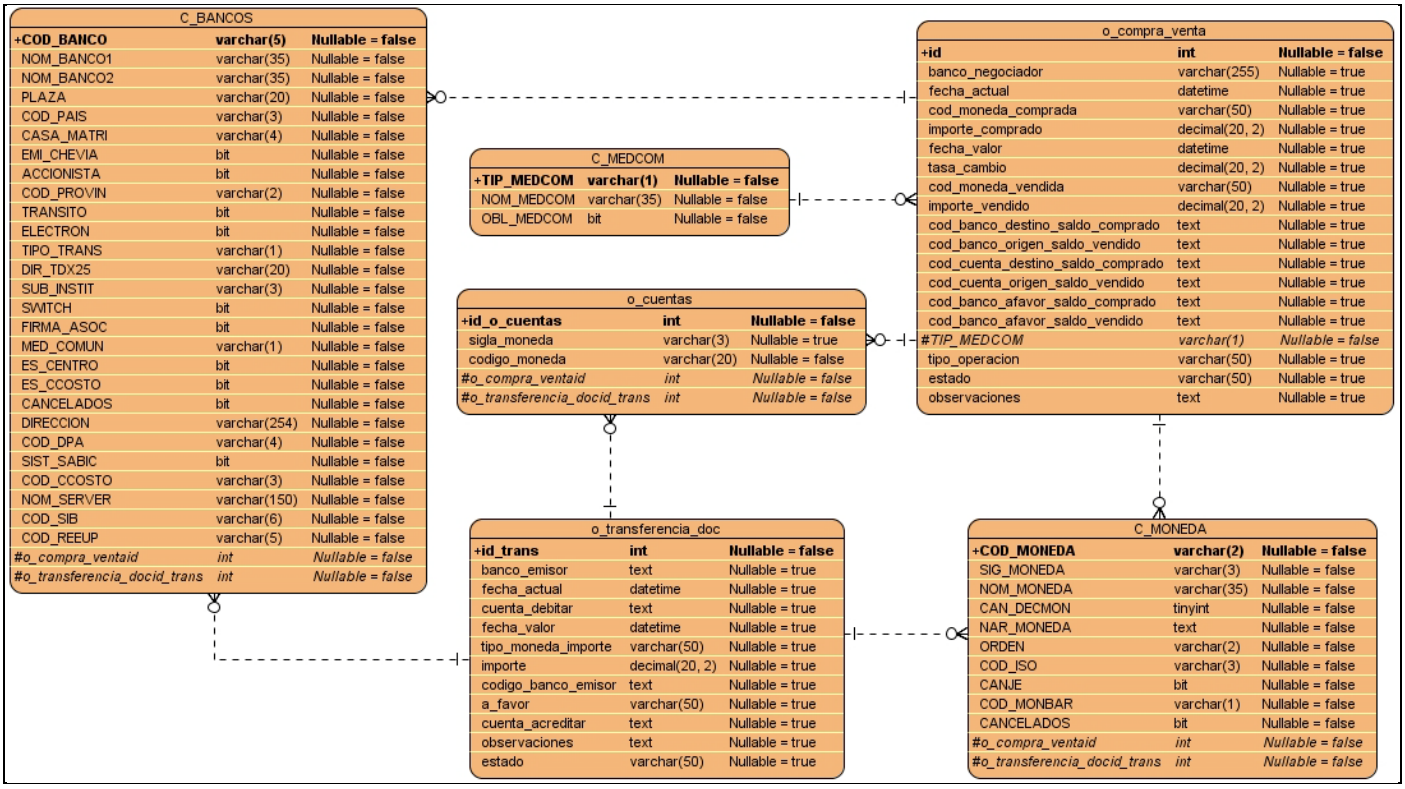


Figura 5. Modelo de datos del subsistema Tesorería.

2.6 Patrones de diseño empleados

El diseño fue elaborado siguiendo patrones basados en la experiencia, que de manera general constituyen soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Dentro de los patrones utilizados se encuentran la familia de los GRASP, ellos describen los principios fundamentales de la asignación de responsabilidades a objetos. Los patrones GRASP utilizados fueron los siguientes:

✓ **Controlador:** la clase controladora **CargarDatosCompraVentaMultiActionController**, constituye un ejemplo de la aplicación de este patrón, la misma tendrá en cuenta la responsabilidad de manejar los eventos que consisten en cargar los datos que serán mostrados al usuario.

✓ **Experto:** dicho patrón es evidenciado en la definición de las clases de acuerdo a las funcionalidades que deben realizar a partir de la información que manejan. Específicamente: la clase

CompraVentaDAO, será la responsable de efectuar las operaciones que conciernen a las funciones: insertar, eliminar, actualizar y consultar compraventa. Sobre este mismo principio se realiza el diseño de las restantes funcionalidades.

✓ **Bajo Acoplamiento:** este patrón se evidencia con la definición de interfaces e implementaciones, como puede ser la interfaz **GestionarCompraVentaFacade** y su implementación, que permiten que **CargarDatosCompraVentaMultiActionController** y **ActualizarCompraVentaSimpleFormController**, clases de la presentación se relacionen únicamente con ellas para realizar sus operaciones, reduciendo el impacto de cambios posteriores en el negocio del sistema.

✓ **Alta cohesión:** el sistema fue diseñado en módulos definidos a partir de que en los mismos se manejarán la menor cantidad de entidades posibles, de manera tal que a las clases pertenecientes a las diferentes capas se les asignarán las responsabilidades necesarias y bien delimitadas.

Durante el diseño se emplearon además dos patrones de la familia de los GoF, específicamente:

✓ **Fachada:** la utilización de este patrón se evidencia en la definición de la interfaz **GestionarCompraVentaFacade** y su implementación, responsables de la comunicación entre la presentación y el grupo de clases e interfaces más complejas que se encargan de la lógica de negocio y el acceso a datos.

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

✓ **Singleton:** la utilización de este patrón se evidencia en los beans declarados en los archivos de configuración de spring, lo cual garantiza la existencia de solo una instancia de una clase.

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

2.7 Descripción de clases y funcionalidades

A continuación se describen las clases más significativas desde el punto de vista funcional para el módulo compraventa del subsistema Tesorería.

NOMBRE	CargarDatosCompraVentaMultiActionController	
TIPO DE CLASE	Controladora	
ATRIBUTOS	**TIPO DE DATO**	
gestionarCompraVentaFacade	GestionarCompraVentaFacade	
nomencladorContabilizacionFacade	NomencladorContabilizacionFacade	
PARA CADA RESPONSABILIDAD		
NOMBRE	**DESCRIPCIÓN**	
cargarTMediosComunicacion (HttpServletRequest request, HttpServletResponse response)	Permite cargar los medios de comunicación correspondientes a una compraventa.	
resolverVistaConsultarCompraVenta (HttpServletRequest request, HttpServletResponse response)	Permite cargar la vista en la cual se consultan las compraventas.	
cargarCuentas (HttpServletRequest request, HttpServletResponse response)	Permite obtener un listado con todas las cuentas asociadas a una compraventa.	
registrarCompraVenta (HttpServletRequest request, HttpServletResponse response)	Recoge el identificador y los parámetros de la compraventa y registra esos datos.	
listarMonedas (HttpServletRequest request, HttpServletResponse response)	Permite obtener un listado con todas las monedas.	
cargarBancos (HttpServletRequest request, HttpServletResponse response)	Permite cargar los bancos implicados en una compraventa.	

Tabla 5. Descripción de la clase CargarDatosCompraVentaMultiActionController.

NOMBRE	ActualizarCompraVentaSimpleFormController	
TIPO DE CLASE	Controladora	
ATRIBUTOS		TIPO DE DATO
gestionarCompraVentaFacade		GestionarCompraVentaFacade
customPropertyEditorRegistrar		PropertyEditorRegistrar
compraVenta		CompraVenta
PARA CADA RESPONSABILIDAD		
NOMBRE		DESCRIPCIÓN
onSubmit(HttpServletRequest request, HttpServletResponse response, Object command, BindException errors)		Se ejecuta cuando se envía el formulario, permitiendo recoger los datos del objeto command.
formBackingObject(HttpServletRequest request)		Carga los datos de los componentes del formulario antes de que la página sea mostrada al usuario.
initBinder(HttpServletRequest request, ServletRequestDataBinder binder)		Permite definir conversores de datos específicos para cada formulario.

Tabla 6 Descripción de la clase ActualizarCompraVentaSimpleFormController.

2.8 Validación del diseño

La medición es fundamental para cualquier disciplina de ingeniería, y la ingeniería del software no es una excepción. La medición permite tener una visión más profunda, proporcionando un mecanismo para la evaluación objetiva. (36)

En el presente epígrafe se aplican un conjunto de métricas de diseño orientado a objeto y se analizan los resultados para determinar la calidad del diseño. Las métricas para diseño proporcionan al diseñador una mejor visión interna, ayudan a que el diseño evolucione a un nivel superior de calidad.

2.8.1 Métricas orientadas a objetos

Las métricas orientadas a objetos (OO) se han introducido para ayudar a los ingenieros del software a usar el análisis cuantitativo, para evaluar la calidad en el diseño antes de que un sistema se construya. El enfoque de las métricas orientadas a objetos está en la clase, piedra fundamental en la arquitectura orientada a objetos. (36)

Los objetivos de las métricas OO están centrados principalmente en tres puntos:

- ✓ Entender mejor la calidad del producto.
- ✓ Evaluar la efectividad del proceso.
- ✓ Mejorar la calidad del trabajo llevado a cabo al nivel del proyecto. (36)

2.8.2 Métricas orientadas a clases

Una clase es la unidad principal de todo sistema OO. Por lo que las medidas y métricas para una clase individual, la jerarquía de clases, y las colaboraciones de clases resultan sumamente valiosas para un ingeniero de software que necesite estimar la calidad de un diseño.

2.8.3 Métricas propuestas por Lorenz y Kidd

Lorenz y Kidd en su libro dividen las métricas basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas a tamaños para una clase se centran en cálculos de atributos y de operaciones para una clase individual, y promedian los valores para el sistema en su totalidad. Las métricas basadas en herencia se centran en la forma en que se reutilizan las operaciones a lo largo y ancho de la jerarquía de clases. Las métricas para valores internos de clase examinan la cohesión y asuntos relacionados con el código, y las métricas orientadas a valores externos examinan el acoplamiento y la reutilización.

De las métricas propuestas por Lorenz y Kidd se aplicarán dos al diseño propuesto.

Tamaño de clase (TC)

Lorenz y Kidd proponen que para medir el tamaño de clases se deben tener los siguientes aspectos en cuenta:

- ✓ El total de operaciones (operaciones tanto heredadas como privadas de la instancia), que se encapsulan dentro de la clase.
- ✓ El número de atributos (atributos tanto heredados como privados de la instancia), encapsulados por la clase.

Valores grandes de TC representa gran responsabilidad de la clase. Esto implica la reducción de la reutilización de la clase y complica la implementación y las pruebas. De forma general, operaciones y atributos deben ser ponderados al determinar el tamaño de la clase. Valores pequeños de TC para una clase indican mayor posibilidad de que la clase pueda ser reutilizada. (36)

Parámetros de calidad	Valores Grandes de TC
Reutilización	Reduce la reutilización de la clase
Implementación	Complica la implementación
Complejidad de las pruebas	Hace complejas las pruebas del sistema
Responsabilidad	La clase debe tener bastante responsabilidad

Tabla 2. Parámetros de calidad para valores grandes de TC. (37)

Para evaluar las métricas son necesarios los valores de los umbrales. Las medidas o umbrales han sido una polémica a nivel mundial en el diseño de sistemas. Los umbrales aplicados fueron los propuestos por Lorenz & Kidd.

Números de operaciones y/o atributos	
TC	**Umbrales**
Pequeño	**<=20**
Medio	**>20 y <=30**
Grande	**>30**

Tabla 3. Umbrales para TC. (37)

Estas métricas se aplicaron a un total de 34 clases, las cuales contaban en su totalidad con 260 atributos y 73 operaciones. Los resultados obtenidos para la métrica TC fueron los siguientes: promedio de 7.6 atributos y 2.14 operaciones por clases.

De acuerdo con los umbrales propuestos en la tabla 3 se obtuvo que 30 clases son de tamaño pequeño y 4 clases son de tamaño grande, como se muestra en la tabla 4, obteniendo que del total de clases analizadas existe un 88% de clases pequeñas y el 12% de clases grandes.

Cantidad de clases	Umbral	

		Tamaño
30	<=20	Pequeño
4	>30	Grande

Tabla 4. Cantidad de clases por tamaño.

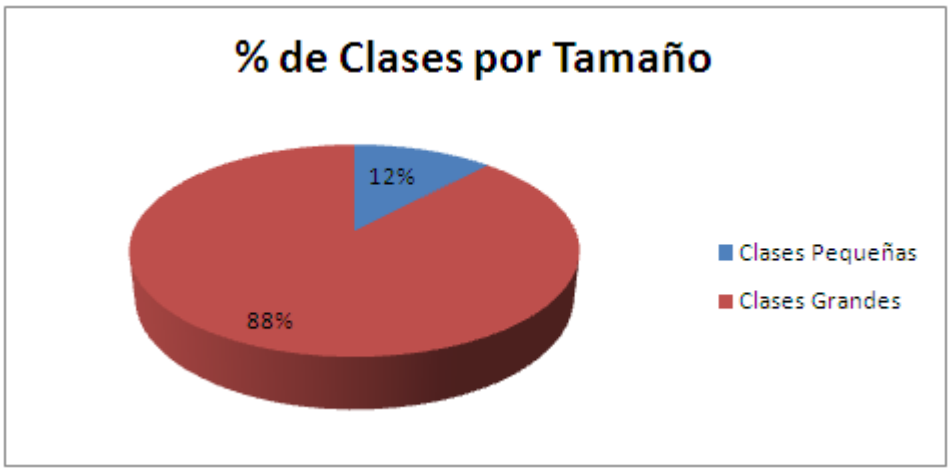


Figura 6. Representación de las clases según TC

Estos valores demuestran un resultado positivo según los parámetros de calidad (ver tabla 2) propuestos para esta métrica.

Número de operaciones redefinidas por una subclase (NOR)

NOR es la medida del número de operaciones que una subclase redefine de su superclase o clase padre, un valor grande para el NOR, generalmente indica un problema en el diseño, o sea si el NOR es grande el diseñador ha violado la abstracción representada por la superclase. Esto provoca una débil jerarquía de clases y un software orientado a objetos, que puede ser difícil de probar y modificar. (36)

Como resultado de la métrica aplicada se expone que el sistema que se desarrolló cuenta con 34 clases y de ellas solo 8 subclases realizan la redefinición de operaciones heredadas, estadísticas que a continuación se muestran en la tabla siguiente.

Cantidad de clases del sistema	Total de clases que redefinen operaciones
34	8

Tabla 4. Clases que redefinen operaciones.

Otra vista o representación en la que se pueden mostrar los datos obtenidos de esta métrica, es una vista del comportamiento del porciento de las subclases que redefinen funcionalidades de clases padres, contra las clases que no redefinen funcionalidades. En el siguiente esquema se representa que las subclases que redefinen funcionalidades representan un 24 % y las subclases que no redefinen funcionalidades representan un 76 % del total.

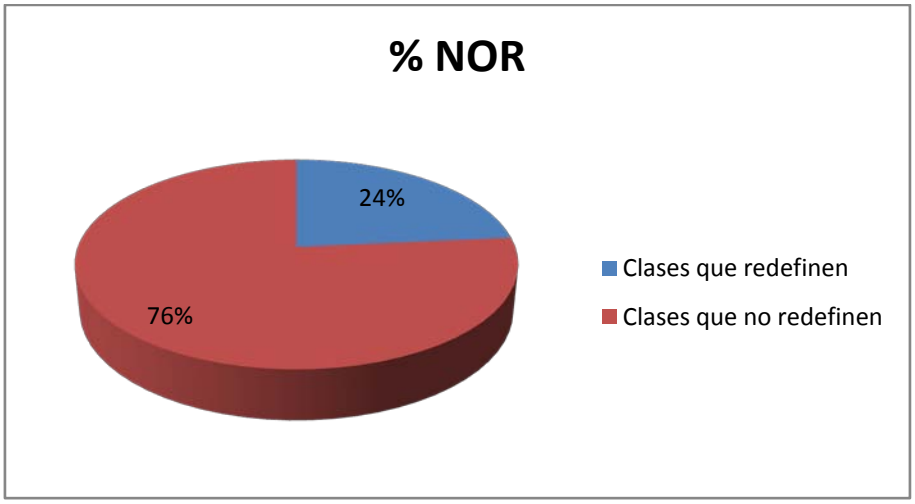


Figura 7. Total de clases que redefinen operaciones.

Se puede observar que los valores obtenidos para la aplicación de la norma NOR al diseño propuesto es aceptable, ya que el porciento de las clases que redefinen operaciones no superan el 25 %, medida que proponen autores como Basili, Harrison y Kidd. Este resultado demuestra que no se afecta la calidad del diseño y puede ser llevado a pruebas o modificado sin dificultad alguna.

2.9 Conclusiones del capítulo

La arquitectura base definida y los artefactos generados como resultado de la especificación de los requisitos correspondientes a los procesos de compraventa y transferencia, fueron la base para la realización del presente capítulo, además se realizó un correcto uso de patrones, con el fin de responder a las funcionalidades claves en la gestión de las operaciones asociadas a dichos procesos y proporcionar una entrada apropiada como punto de partida a las actividades de implementación.

Se lograron aplicar las métricas propuestas para la evaluación del diseño presentado. Con la aplicación de las métricas se determinó que el diseño propuesto no presenta una alta complejidad estructural de datos, posibilitando que las pruebas no sean complejas. Como el número de subclases que redefinen métodos no es alto, da la posibilidad de que el sistema pueda ser probado fácilmente y se pueda modificar en corto tiempo.

Capítulo 3: Implementación y prueba de la solución.

3.1 Introducción

En el presente capítulo se aborda lo referente a la implementación y prueba del sistema. Se describe el modelo de implementación utilizado, los diagramas de componente y de despliegue, así como las pruebas efectuadas para verificar el buen funcionamiento del sistema.

3.2 Modelo de implementación

Según la metodología RUP el modelo de implementación toma el resultado del modelo de diseño para generar el código final del sistema. Describe además, cómo los elementos de diseño se implementan en componentes (ficheros de código fuente, de código binario, ejecutable, scripts), como se organizan dichos componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados, y como dependen de los componentes unos de otros. (34)

Diagrama de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Los componentes representan todos los tipos de elementos del software que entran en la fabricación de aplicaciones informáticas. (38)

El diagrama de componentes que se presenta a continuación se ha elaborado de forma tal que muestra las relaciones existentes entre el subsistema Tesorería y los restantes componentes del sistema.

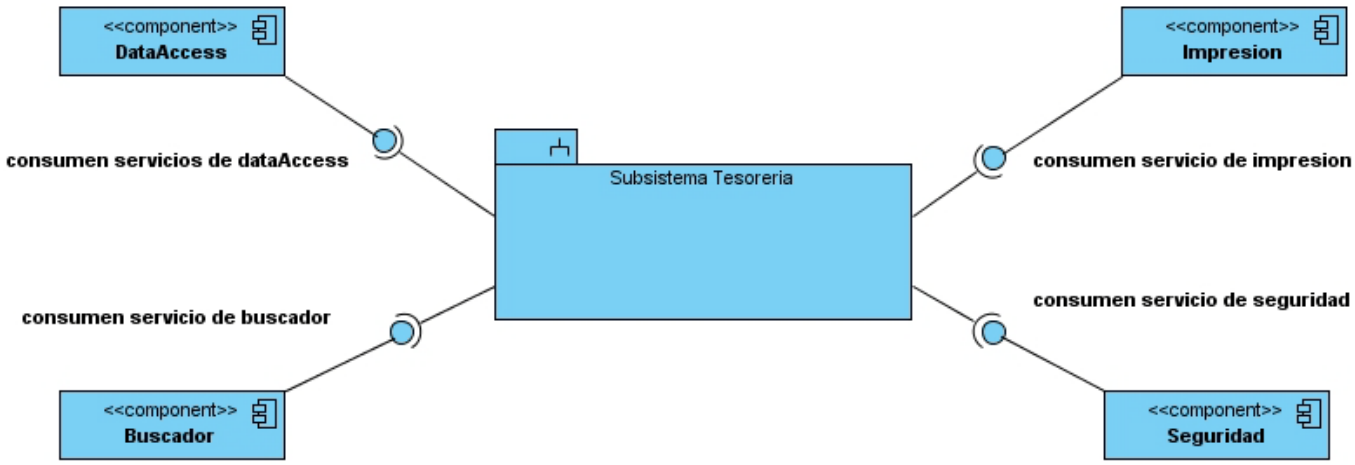


Figura 8. Diagrama de componentes asociado al subsistema Tesorería.

A continuación se explican de manera general cada uno de los componentes.

El componente **Buscador** ofrece un conjunto de clases que constituyen el motor de búsqueda, presentando una interfaz gráfica mediante la cual se solicitan los diferentes conceptos en dependencia del módulo donde se emplee. Para la correcta gestión de la información en el subsistema Tesorería se hace necesaria la búsqueda tanto de una compraventa como de una transferencia.

El componente **Seguridad** como su nombre lo indica es el encargado, entre otros aspectos, de mantener la seguridad en todo el sistema a través de la comprobación de las peticiones y sus permisos en dependencia del usuario que la realice, de aquí que sea necesaria la comunicación con el mismo para garantizar la seguridad del subsistema Tesorería.

El componente **Impresión** consiste en un componente que tiene como función brindar las clases que contienen las funcionalidades mediante las cuales es posible imprimir determinada información.

Por otro lado el **DataAccess** tiene la función de brindar todos los elementos necesarios para llevar a cabo la conexión a la base de datos y por ende todos los subsistemas dependen de él para poder realizar las funcionalidades que engloban.

Diagrama de despliegue

Es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. Se utiliza como entrada fundamental en las actividades de diseño e implementación debido a que la distribución del sistema tiene una influencia principal en su diseño. (34)

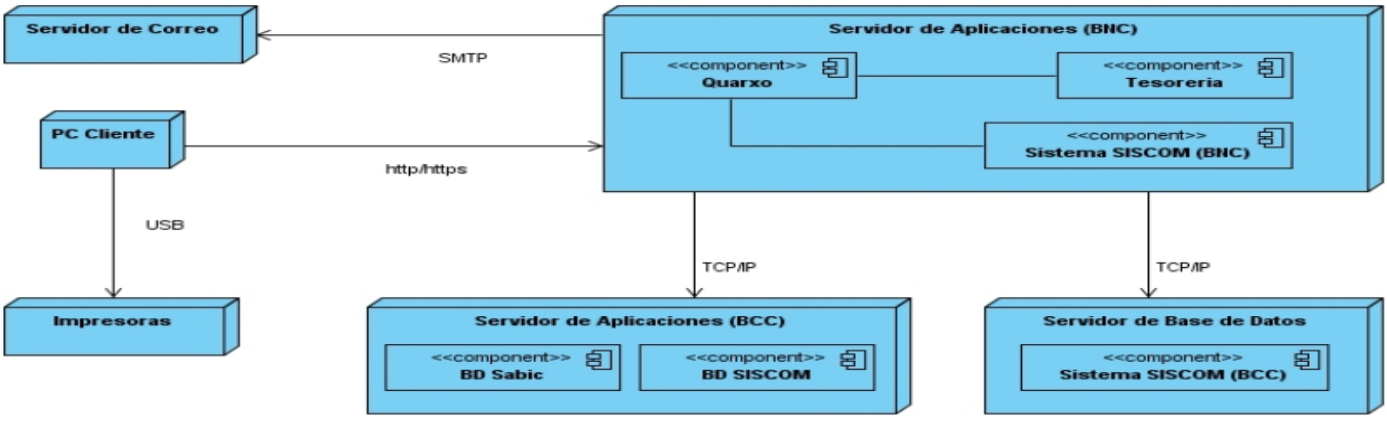


Figura 9. Diagrama de despliegue para el Banco Nacional de Cuba.

3.3 Estándares de codificación

Un estándar de codificación comprende todos los aspectos relacionados con la generación de código, de tal manera que sea prudente, práctico y entendible para todos los programadores. Un código fuente completo debe reflejar un estilo armonioso y uniforme, como si un único programador hubiera escrito todo el código. El desarrollo de una aplicación informática exige que se adopten estándares de codificación y estilo. El uso de estándares asegura la legibilidad del código entre distintos desarrolladores, permite la guía para el mantenimiento o actualización del sistema, además de que facilita la portabilidad entre plataformas y aplicaciones.

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

3.3.1 Convención de código general

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

Ejemplo: CompraVentaManager. En este caso el nombre de clase está compuesto por 3 palabras iniciadas cada una con letra mayúscula.

También se tomó en cuenta para el nombrado de las clases el tipo que esta posee, entiéndase como tipo el rol que ellas desempeñan en el sistema. A continuación se presentan las nomenclaturas organizadas por los paquetes a los que pertenecen las clases.

Controller: Las clases incluidas en este paquete, después del nombre se le incorporan el nombre del controlador de Spring del cual hereda.

Ejemplo: CargarDatosCompraVentaMultiActionController.

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

Ejemplo: CompraVentaCommand.

PropertyEditor: Las clases que se encuentran dentro del paquete propertyEditor después del nombre se le agrega la palabra PropertyEditor.

Ejemplo: TipoMedioComunicacionPropertyEditor.

Facade: A cada clase incluida en este paquete, después del nombre se le agrega la palabra facade y en el caso de subpaquete impl tendrán el mismo nombre de la interfaz que implemente más la palabra Impl.

Ejemplo: CompraVentaFacade, CompraVentaFacadeImpl.

Manager: A cada clase que se encuentra dentro de manager después del nombre se le pone la palabra manager y en el caso de subpaquete impl tendrán el mismo nombre de la interfaz que implemente más la palabra Impl.

Ejemplo: CompraVentaManager y CompraVentaManagerImpl.

DAO: A cada clase incluida en el paquete dao, después del nombre se le incorpora la abreviatura dao y en el caso de subpaquete impl tendrán el mismo nombre de la interfaz que implemente más la palabra Impl.

Ejemplo: CompraVentaDAO, CompraVentaDAOImpl.

De manera general el nombre de los métodos y los atributos de las clases se escriben con la inicial del identificador en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCasing, que es muy similar a Pascal Casing con la excepción de que la letra inicial comienza con minúsculas.

Lo mismo se aplica a los nombres de ficheros de código JavaScript y sus funciones y variables internas. Los comentarios deben ser lo más claros y precisos posibles de forma tal que se entienda el propósito de lo que se está desarrollando. En caso de ser una función complicada se debe comentar para lograr una mejor comprensión del código.

3.4 Pruebas

El principal objetivo de esta disciplina es evaluar la calidad del producto que se desarrolló a través de las diferentes fases por las cuales transita mediante la aplicación de pruebas concretas para validar que las suposiciones hechas en el diseño y que los requerimientos se estén cumpliendo satisfactoriamente, esto

quiere decir que se verifica que el producto funcione como se diseñó y que los requerimientos son satisfechos cabalmente.

3.4.1 Pruebas unitarias

El objetivo de las pruebas unitarias es el aislamiento de partes del código y la demostración de que estas partes no contienen errores. Las pruebas unitarias son una forma de probar el correcto funcionamiento de un módulo de código, esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. (39)

Las pruebas unitarias realizadas a los servicios del sistema sirven para validar que las salidas son correctas y aseguran al desarrollador que la solución no presenta errores en la lógica de programación y que las respuestas son las correctas ante una entrada de datos determinada.

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

Prueba de Caja Blanca

Objetivo

El objetivo de realizar este tipo de prueba al sistema es que se garantice que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo o método, todos los bucles en sus límites operacionales así como las estructuras internas de datos para asegurar su validez. (36)

Alcance

El proceso de pruebas de caja blanca se va a concentrar principalmente en validar a través del framework de software libre JUnit, que cada uno de los módulos o segmentos de códigos funcionen apropiadamente.

Descripción

La prueba de caja blanca es considerada como uno de los tipos de pruebas más importantes que se le aplican a los software, logrando como resultado que disminuya en un gran por ciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad. (36)

Para el desarrollo de estas pruebas unitarias se utilizó framework JUnit, ya que ofrece las funcionalidades necesarias para implementar pruebas en un proyecto desarrollado en Java. Además cuenta con una

interface simple que informa si cada una de las pruebas realizadas o conjunto de pruebas falló, pasó o fue ignorada.

Inicialmente se definieron los casos de prueba, uno por cada clase implementada. Luego se definieron e implementaron los ficheros de configuración necesarios para establecer la comunicación entre las diferentes capas de la aplicación. Se identificaron los métodos a probar dentro de cada caso de prueba, así como el resultado esperado en cada uno. Y por último se procedió a realizar los casos de prueba diseñados.

Con el objetivo de mostrar la realización de los casos de prueba, a continuación se muestran un grupo de imágenes con sus descripciones.

```
public class TesoreriaTest extends TestCase {

private CargarDatosCompraVentaMultiActionController
cargarDatosCompraVentaMultiActionController;

private MockControl controlHttpServletRequest;
private HttpServletRequest mockHttpServletRequest;

private MockControl controlHttpServletResponse;
private HttpServletResponse mockHttpServletResponse;

private SessionFactory sf;
```

Figura 10. Declaración de los atributos en la clase de prueba

Se crea una clase en la que se van a definir los casos de prueba implementados para probar las funcionalidades que se encuentran en la clase *CargarDatosCompraVentaMultiActionController*. Primero se declara un atributo de dicha clase, luego dos *mocks*[1] para simular los objetos *HttpServletRequest* y *HttpServletResponse* que se le deben pasar a los métodos a probar, estos objetos son controlados por

[1] **Mocks:** *son objetos simulados que imitan el comportamiento de los objetos reales en forma controlada. Normalmente se crea un objeto mock para probar el comportamiento de algún otro objeto.*

objetos de tipo *MockControl*, por último uno de tipo *SessionFactory* necesario para que Hibernate se comunique con la base de datos.

```
@Before
protected void setUp() throws Exception {

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

cargarDatosCompraVentaMultiActionController =
(CargarDatosCompraVentaMultiActionController) context
.getBean("cargarDatosCompraVenta");

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

controlHttpServletRequest = MockControl
.createControl(HttpServletRequest.class);
mockHttpServletRequest = (HttpServletRequest) controlHttpServletRequest
.getMock();

controlHttpServletResponse = MockControl
.createControl(HttpServletResponse.class);
mockHttpServletResponse = (HttpServletResponse)
controlHttpServletResponse
.getMock();

super.setUp();
}
```

Figura 11. Implementación del método setUp() en la clase de prueba.

El método *setUp()* es donde se inicializan todos los atributos declarados. Para hacer más entendible el caso de prueba, se decidió poner en el fichero *tesoreria-context.xml* todo el proceso de inicializar el objeto *cargarDatosCompraVentaMultiActionController* debido a su complejidad. Los objetos *mockHttpServletRequest, controlHttpServletRequest, mockHttpServletResponse, y controlHttpServletResponse* son creados usando la librería *EasyMock*.

```
protected void tearDown()
{
controlHttpServletRequest.verify();
}
```

Figura 12. Verificación de cada prueba.

El método *tearDown* es el encargado de las tareas a realizar en cada test. En este caso solo va a verificar los objetos que son pasados por el request.

```
@Test
public void testRegistrarCompraventa() {
mockHttpServletRequest.getParameter("id_cv");
controlHttpServletRequest.setReturnValue("50");
mockHttpServletRequest.getParameter("bancoNegociador");
controlHttpServletRequest.setReturnValue("BCC");
mockHttpServletRequest.getParameter("fechaActual");
controlHttpServletRequest.setReturnValue("9/6/2012");
mockHttpServletRequest.getParameter("codMonedaComprada");
controlHttpServletRequest.setReturnValue("CUC");
mockHttpServletRequest.getParameter("importeComprado");
controlHttpServletRequest.setReturnValue("2300");
mockHttpServletRequest.getParameter("tasaCambio");
controlHttpServletRequest.setReturnValue("1.0");
mockHttpServletRequest.getParameter("tipoMedioComunicacion");
controlHttpServletRequest.setReturnValue("FAX");
mockHttpServletRequest.getParameter("codCuentaDestinoSaldoComprado");
controlHttpServletRequest.setReturnValue("32322000203");

controlHttpServletRequest.replay();
assertTrue(cargarDatosCompraVentaMultiActionController.registrarCompraVenta(
mockHttpServletRequest, mockHttpServletResponse));
}
```

Figura 13. Parámetros utilizados por el método a probar.

Durante la realización de la prueba al método, se preparan los parámetros de pruebas que le son necesarios en algún momento al método registrarCompraVenta. Dichos parámetros serán pasados por el mockHttpServletRequest. Posteriormente se realiza la condición de prueba a través del método assertTrue el cual devuelve verdadero si se registró la compraventa de manera satisfactoria.

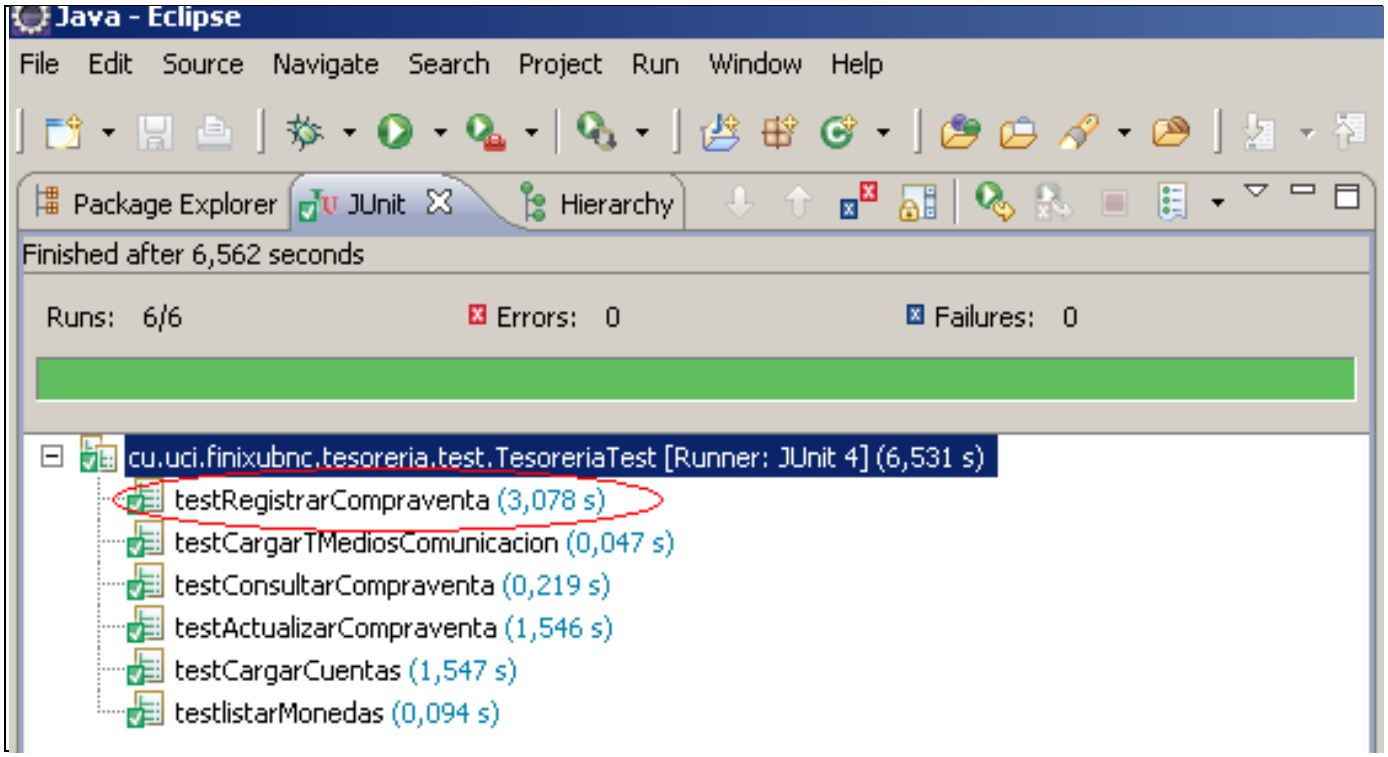


Figura 14. Resultado de la prueba realizada.

El resultado de la prueba realizada al método *registrarCompraVenta* fue satisfactorio al igual que los demás métodos de la clase *cargarDatosCompraVentaMultiActionController* los cuales se pueden observar en la figura 14.

Prueba de Caja Negra

Objetivo

El objetivo de realizar este tipo de prueba al sistema es para detectar el incorrecto o incompleto funcionamiento de este, así como los errores de interfaces, rendimiento y errores de inicialización y terminación.

Alcance

El proceso de pruebas de caja negra se va a centrar principalmente en los requisitos funcionales del software para verificar el comportamiento de la unidad observable externamente y la calidad funcional.

Descripción

Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales del programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

La prueba de caja negra no es una alternativa a las técnicas de prueba de la caja blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la caja blanca. (40)

Existen varias técnicas para desarrollar la prueba de caja negra, entre ellas están:

- Técnica de la Partición de Equivalencia: divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- Técnica del Análisis de Valores Límites: prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
- Técnica de Grafos de Causa-Efecto: permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones. (41)

Dentro de las técnicas de prueba de caja negra se utilizó la de Partición Equivalente. Esta técnica permite obtener un conjunto de pruebas que reducen el número de casos de prueba que se deben realizar para evaluar correctamente el software, esto se debe a que se centra en la evaluación de clases de equivalencia que representan un conjunto de estados válidos o no válidos para condiciones de entradas existentes en el software.

Para definir las clases de equivalencia se tuvieron en cuenta un conjunto de reglas: (40)

- Si una condición de entrada especifica un rango, entonces se confeccionan una clase de equivalencia válida y dos inválidas.
- Si una condición de entrada especifica la cantidad de valores, se identifica una clase de equivalencia válida y dos inválidas.

- Si una condición de entrada especifica un conjunto de valores de entrada y existen razones para creer que el programa trata en forma diferente a cada uno de ellos, se identifica una clase válida para cada uno de ellos y una clase inválida.
- Si una condición de entrada especifica una situación de tipo “debe ser”, se identifica una clase válida y una inválida.

Luego de tener las clases válidas e inválidas definidas, se diseñaron los casos de prueba para el escenario Registrar Transferencia del caso de uso Gestionar Transferencia, el mismo puede ser consultado en el Anexo 4. Cada uno de estos casos de prueba se definió teniendo en cuenta lo siguiente: (40)

- Escribir un nuevo caso de prueba que cubra tantas clases de equivalencia válidas no cubiertas como sea posible hasta que todas las clases de equivalencia hayan sido cubiertas por casos de prueba.
- Escribir un nuevo caso de prueba que cubra una y solo una clase de equivalencia inválida hasta que todas las clases de equivalencias inválidas hayan sido cubiertas por casos de prueba.

3.4.2 Resultados de las pruebas

Luego de la realización de los métodos de prueba, como parte de las pruebas internas realizadas a la propuesta de solución se obtuvieron resultados satisfactorios desde el punto de vista interno y funcional, atendiendo al correcto comportamiento de la misma ante diferentes situaciones. Se aplicaron los métodos de prueba de caja blanca y caja negra para validar tanto la interfaz como el correcto funcionamiento interno del software. Los resultados arrojados por ambas pruebas contribuyeron a validar la calidad de la solución implementada.

3.5 Conclusiones del capítulo

Una vez desarrolladas las tareas que dan por cumplido este capítulo, se concluye que la solución implementada y posteriormente validada, desde el punto de vista funcional, permite llevar a cabo los procesos relacionados con compraventa y transferencia en el BNC, de manera que ha sido incorporado el subsistema Tesorería al sistema Quarxo, desarrollado con tecnologías libres, y provisto de elementos de seguridad, estándares y un entorno amigable, gracias a la utilización del framework Spring y la librería

Dojo Toolkit. La solución exhibe valor técnico, además está integrada con componentes de carácter general dentro del sistema. El subsistema fue probado mediante pruebas de Caja Blanca y Caja Negra, y validado utilizando pruebas de aceptación.

Conclusiones generales

El proceso de desarrollo de software fue guiado por la metodología RUP, cumpliendo con los elementos que dicha metodología propone y logrando una efectiva ingeniería de software. Las tecnologías seleccionadas para el desarrollo de la solución están acordes a los requerimientos del cliente, a las políticas del centro y a los estándares internacionales. Se obtuvo el subsistema Tesorería del sistema Quarxo, desarrollado con tecnologías libres, y provisto de elementos de seguridad, estándares y un entorno amigable, gracias a la utilización del framework Spring y la librería Dojo Toolkit.

Se analizaron ventajas y deficiencias de sistemas informáticos tanto nacionales como internacionales vinculados a la gestión de tesorería, donde la dificultad principal para su utilización resultó ser el alto costo de sus licencias. Evidenciándose de esta manera la necesidad de desarrollar una solución informática capaz de ejecutar dichas funcionalidades, que cumpliera con los requisitos establecidos. La solución desarrollada contribuye a la gestión de tesorería en el BNC, permitiendo disminuir los tiempos de respuesta y ahorrar los recursos materiales de la institución.

Recomendaciones

A partir del estudio realizado en la presente investigación, teniendo en cuenta las experiencias obtenidas a lo largo de su desarrollo, se proponen las siguientes recomendaciones:

- ✓ Aprovechar los estudios realizados en esta investigación para mejorar la solución presentada en futuras versiones que se realicen del sistema.
- ✓ Incrementar el uso de las tecnologías y herramientas presentadas en la investigación en el desarrollo de otros sistemas de gestión bancaria.
- ✓ Dar continuidad a este trabajo con el objetivo de agregar nuevas funcionalidades a la solución.
- ✓ Extender el sistema hacia una nueva solución que sea genérica de manera tal que pueda ser utilizada por cualquier banco o entidad bancaria mundial.

Referencias bibliográficas

1. Definicion ABC. [En línea] http://www.definicionabc.com/economia/tesoreria.php.

2. Financiero.com. [En línea] http://www.financiero.com/diccionario_financiero/tesoreria.asp.

3. **Leandro, Gabriel.** AulaDeEconomía. [En línea] 2002. http://www.auladeeconomia.com/glosario-t.htm.

4. **Olsina, Xavier.** *GESTION DE TESORERIA.* s.l. : Profit Editorial , 2009.

5. **Martinez Muñoz, Javier.** *Definición De Los Requerimientos Funcionales Del Módulo Cuentas de Clientes y Órdenes de Pago Inmediato Del Proyecto Banco Nacional.* Cuidad Habana : s.n., 2008.

6. **Tripier, Benjamin.** Gerencia.com. [En línea] [Citado el: 20 de mayo de 2012.] http://www.degerencia.com/articulo/la_importancia_de_la_tesoreria.

7. Webs. [En línea] http://www.freewebs.com/nasrullah_djamil/AIS%20EFFICIENCY.pdf.

8. Sistema Bancario Financiero Byte. [En línea] [Citado el: 20 de noviembre de 2011.] http://www.bytesw.com/new/sistema_bancario_financierobyte.asp.

9. Sage XRT Teasurey. [En línea] [Citado el: 20 de enero de 2012.] http://www.sage.es/Productos/Tesoreria/Sage_XRT_Treasury.aspx.

10. Bantotal. [En línea] [Citado el: 25 de febrero de 2012.] http://www.bantotal.com/bantotal/productos_servicios.asp#Tesoreria.

11. Softwareseleccion. [En línea] [Citado el: 2 de marzo de 2012.] http://www.softwareseleccion.com/cash+flow+manager-p-2355.

12. Softwareseleccion. [En línea] [Citado el: 2 de marzo de 2012.] http://www.softwareseleccion.com/elk+cash-p-147.

13. **Mesa, Lissett Diaz.** *Proyecto Técnico para el Sistema Automatizado para la Gestión Bancaria(SAGEB).* 2009.

14. **Pérez, Pedro Piñero.** *Metodologías Ágiles y Robustas.* 2009.

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

Referencia bibliográfica

16. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.** *Design Patterns.* s.l. : Addison-Wesley Pub Co, 1995.

17. **ORACLE.** Core J2EE Patterns - Data Access Object. [En línea] http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html..

18. TIOBE Software. [En línea] [Citado el: 6 de junio de 2012.] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

19. desarrolloweb. [En línea] [Citado el: 20 de febrero de 2012.] http://www.desarrolloweb.com/articulos/497.php..

20. The Java EE 5 Tutorial. [En línea] [Citado el: 20 de febrero de 2012.] http://java.sun.com/javaee/5/docs/tutorial/doc/..

21. Manual xhtml. [En línea] [Citado el: 21 de febrero de 2012.] http://manual-xhtml.blogspot.com/2006/05/primer-documento-xhtml.html.

22. CodeBox. [En línea] [Citado el: 23 de febrero de 2012.] http://www.codebox.es/glosario.

23. **Milián, V.** Universidad de San Buenaventura seccional cali. [En línea] 2010. [Citado el: 16 de mayo de 2012.] http://usbvirtual.usbcali.edu.co/ijpm/images/stories/documentos/v1n2/003.pdf.

24. **Patrick Peak, Nick Heudecker.** *Hibernate Quickly.* s.l. : Manning Publications Co, 2006.

25. **A.Russell, Matthew.** *Dojo The Definitive Guide.* 2008. 978-0-596-51648-2..

26. Visual Paradigm. [En línea] [Citado el: 25 de febrero de 2012.] http://www.visual-paradigm.com.

27. Control de versiones con Subversion. [En línea] [Citado el: 26 de febrero de 2012.] http://svnbook.red-bean.com/nightly/en/index.html.

28. **Holzner, Steve.** *Eclipse: A Java Developer's Guide.* s.l. : O'Reilly & Associates, Inc. Sebastopol, 2004. ISBN:0596006411.

29. Apache Tomcat. [En línea] [Citado el: 20 de febrero de 2012.] http://tomcat.apache.org/..

30. Microsoft SQL Server . [En línea] [Citado el: 24 de febrero de 2012.] http://www.microsoft.com/sqlserver/en/us/product-info.aspx.

31. IEEE SA 1471-2000. [En línea] [Citado el: 12 de 4 de 2012.] http://standards.ieee.org/findstds/standard/1471-2000.html.

32. **Iglesias, Adolfo Miguel.** *Documento de Arquitectura Modernización Sistema del Banco Nacional de Cuba.* 2009.

33. **Group, Architecture Working.** *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.* s.l. : IEEE Standards Association, 2000. IEEE Standard IEEE Std 1471-2000.

34. **Ivar Jacobson, Grady Booch, James Rumbaugh.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Pearson Educación, S.A,, 2000. 84-7829-036-2..

35. **Guerrero, Luis A.** *Taller UML.* Universidad de Chile, Departamento de Ciencias de la Computación : s.n.

36. **Pressman, Roger S.** Biblioteca de la Universidad de las Ciencias Informaticás. [En línea] 2005. [Citado el: 5 de mayo de 2012.] http://bibliodoc.uci.cu/pdf/reg02689.pdf .

37. **Kidd, J. y Lorenz, M.** *Object-Oriented Software Metric.* 1994.

38. *Universidad de Castilla-La Mancha. Escuela Politécnica Superior de Albacete, Ingeniería de Software.* [En línea] [Citado el: 17 de abril de 2012.] http://www.dsi.uclm.es/asignaturas/42530/pdf/M2tema12.pdf.

39. **Kölling, D.J. Barnes and M.** *Programación orientada a objetos con Java. MANEJO EFICIENTE DEL TIEMPO.* 2007.

40. Universidad de Castilla-La Mancha. [En línea] Grupo Alarcos. http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema09.pdf.

41. **Myers, G., C. de Filecich, y A. Filevich.** *El arte de probar el software. .* s.l. : El Ateneo., 1983.

Bibliografía consultada

- Galíndez, Rodrigo. Control de Versiones Usando Subversion. [En línea] http://www.rodrigogalindez.com/files/14.pdf.
- Scribd. Conceptos de RUP. [En línea] http://es.scribd.com/doc/7844685/CONCEPTOS-DE-RUP.
- Desarrollo Web. Arquitectura cliente-servidor. [En línea] [Citado el: 10 de febrero de 2012.] http://www.desarrolloweb.com/articulos/arquitectura-cliente-servidor.html.
- Debug Mode On. El patrón MVC. [En línea] http://es.debugmodeon.com/articulo/el-patron-mvc.
- Pérez, Javier Eguíluz. Introducción a JavaScript. 2008.
- Apache. Documentación del Servidor HTTP Apache 2.0. [En línea] http://httpd.apache.org/docs/2.0/es/.
- Pressman, Roger. Ingeniería de Software. Un Enfoque Práctico. 1998.
- Rawld Gill, Craig Riecke, Alex Russell. Mastering Dojo. s.l.: Pragmatic Bookshelf, 2008. -934356-11-5
- Craig Walls, Ryan Breindenbach. Spring In Action. s.l.:Manning Publications Co, 2005. 1-932394-35-4
- Craig Walls, Ryan Breidenbach. Spring in Action Second Edition. s.l.: Manning Publications Co, 2008. 1-933988-13-4.
- Christian Bauer, Gavin King. Hibernate in Action. s.l.: Manning Publications Co, 2005. 1932394-15-X.
- Jason Hunter, Willian Crawford. Java Servlet Programing. s.l.: O'Reilly & Associates, Inc, 1998. 1-56592-391-X.
- Pruebas de Software. [En línea] [Citado el: 8 de abril de 2012.] http://lsi.ugr.es/~ig1/docis/pruso.pdf.
- Definicion ABC. [En línea] http://www.definicionabc.com/economia/tesoreria.php.
- Financiero.com. [En línea] http://www.financiero.com/diccionario_financiero/tesoreria.asp.
- Leandro, Gabriel. AulaDeEconomía. [En línea] 2002. http://www.auladeeconomia.com/glosario-t.htm.
- Olsina, Xavier. GESTION DE TESORERIA. s.l. : Profit Editorial , 2009.
- Martinez Muñoz, Javier. Definición De Los Requerimientos Funcionales Del Módulo Cuentas de Clientes y Órdenes de Pago Inmediato Del Proyecto Banco Nacional. Cuidad Habana : s.n., 2008.

Bibliografía Consultada

- Tripier, Benjamin. Gerencia.com. [En línea] [Citado el: 20 de mayo de 2012.] http://www.degerencia.com/articulo/la_importancia_de_la_tesoreria.
- Webs. [En línea] http://www.freewebs.com/nasrullah_djamil/AIS%20EFFICIENCY.pdf.
- Sistema Bancario Financiero Byte. [En línea] [Citado el: 20 de noviembre de 2011.] http://www.bytesw.com/new/sistema_bancario_financierobyte.asp.
- Sage XRT Teasurey. [En línea] [Citado el: 20 de enero de 2012.] http://www.sage.es/Productos/Tesoreria/Sage_XRT_Treasury.aspx.
- Bantotal. [En línea] [Citado el: 25 de febrero de 2012.] http://www.bantotal.com/bantotal/productos_servicios.asp#Tesoreria.
- Softwareseleccion. [En línea] [Citado el: 2 de marzo de 2012.] http://www.softwareseleccion.com/cash+flow+manager-p-2355.
- Softwareseleccion. [En línea] [Citado el: 2 de marzo de 2012.] http://www.softwareseleccion.com/elk+cash-p-147.
- Mesa, Lissett Diaz. Proyecto Técnico para el Sistema Automatizado para la Gestión Bancaria(SAGEB). 2009.
- Pérez, Pedro Piñero. Metodologías Ágiles y Robustas. 2009.
- Larman, Craig. UML y Patrones. s.l. : Prentice Hall, 1999. 970-17-0261-1.
- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns. s.l. : Addison-Wesley Pub Co, 1995.
- http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html..
- TIOBE Software. [En línea] [Citado el: 6 de junio de 2012.] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.
- desarrolloweb. [En línea] [Citado el: 20 de febrero de 2012.] http://www.desarrolloweb.com/articulos/497.php..
- The Java EE 5 Tutorial. [En línea] [Citado el: 20 de febrero de 2012.] http://java. 17. ORACLE. Core J2EE Patterns - Data Access Object. [En línea] sun.com/javaee/5/docs/tutorial/doc/..
- Manual xhtml. [En línea] [Citado el: 21 de febrero de 2012.] http://manual-xhtml.blogspot.com/2006/05/primer-documento-xhtml.html.
- CodeBox. [En línea] [Citado el: 23 de febrero de 2012.] http://www.codebox.es/glosario.
- Milián, V. Universidad de San Buenaventura seccional cali. [En línea] 2010. [Citado el: 16 de mayo de 2012.] http://usbvirtual.usbcali.edu.co/ijpm/images/stories/documentos/v1n2/003.pdf.

Bibliografía Consultada

- Patrick Peak, Nick Heudecker. Hibernate Quickly. s.l. : Manning Publications Co, 2006.
- A.Russell, Matthew. Dojo The Definitive Guide. 2008. 978-0-596-51648-2..
- Visual Paradigm. [En línea] [Citado el: 25 de febrero de 2012.] http://www.visual-paradigm.com.
- Control de versiones con Subversion. [En línea] [Citado el: 26 de febrero de 2012.] http://svnbook.red-bean.com/nightly/en/index.html.
- Holzner, Steve. Eclipse: A Java Developer's Guide. s.l. : O'Reilly & Associates, Inc. Sebastopol, 2004. ISBN:0596006411.
- Apache Tomcat. [En línea] [Citado el: 20 de febrero de 2012.] http://tomcat.apache.org/..
- Microsoft SQL Server . [En línea] [Citado el: 24 de febrero de 2012.] http://www.microsoft.com/sqlserver/en/us/product-info.aspx.
- IEEE SA 1471-2000. [En línea] [Citado el: 12 de 4 de 2012.] http://standards.ieee.org/findstds/standard/1471-2000.html.
- Group, Architecture Working. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. s.l. : IEEE Standards Association, 2000. IEEE Standard IEEE Std 1471-2000
- Ivar Jacobson, Grady Booch, James Rumbaugh. El Proceso Unificado de Desarrollo de Software. s.l. : Pearson Educación, S.A,, 2000. 84-7829-036-2..
- Universidad de Castilla-La Mancha. Escuela Politécnica Superior de Albacete, Ingeniería de Software. [En línea] [Citado el: 17 de abril de 2012.] http://www.dsi.uclm.es/asignaturas/42530/pdf/M2tema12.pdf.
- Kölling, D.J. Barnes and M. Programación orientada a objetos con Java. MANEJO EFICIENTE DEL TIEMPO. 2007.
- Universidad de Castilla-La Mancha. [En línea] Grupo Alarcos. http://alarcos.inf-cr.uclm.es/doc/ISOFTWAREI/Tema09.pdf.

Glosario de términos

API: aplication Programming Interface, es un conjunto de funciones y procedimientos que ofrece determinada biblioteca para ser utilizada por otro software

Beans: son componentes hechos en software que se pueden reutilizar y que pueden ser manipulados visualmente por una herramienta de programación en lenguaje Java.

CASE: del inglés (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Computadora) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero.

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

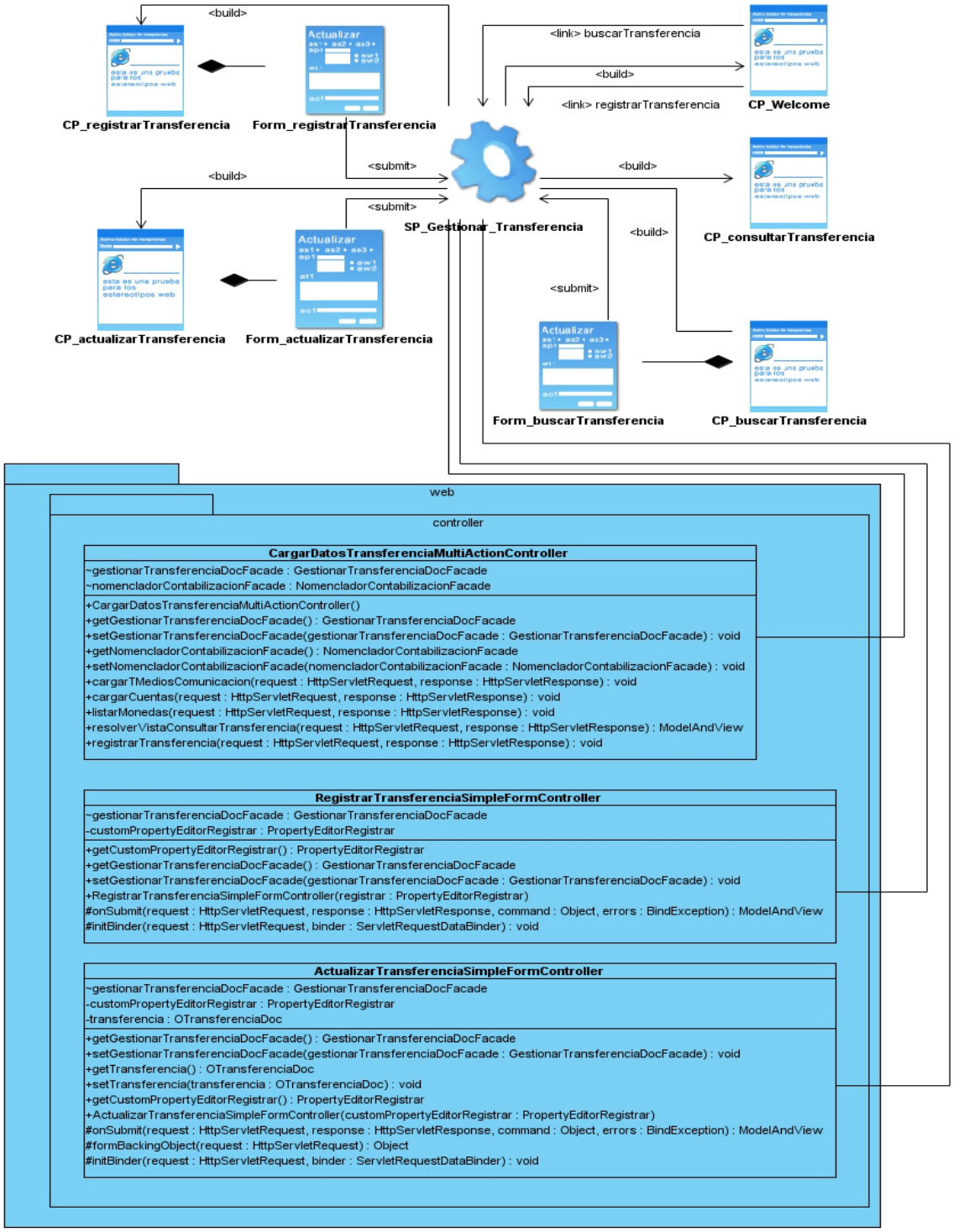
Framework: es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

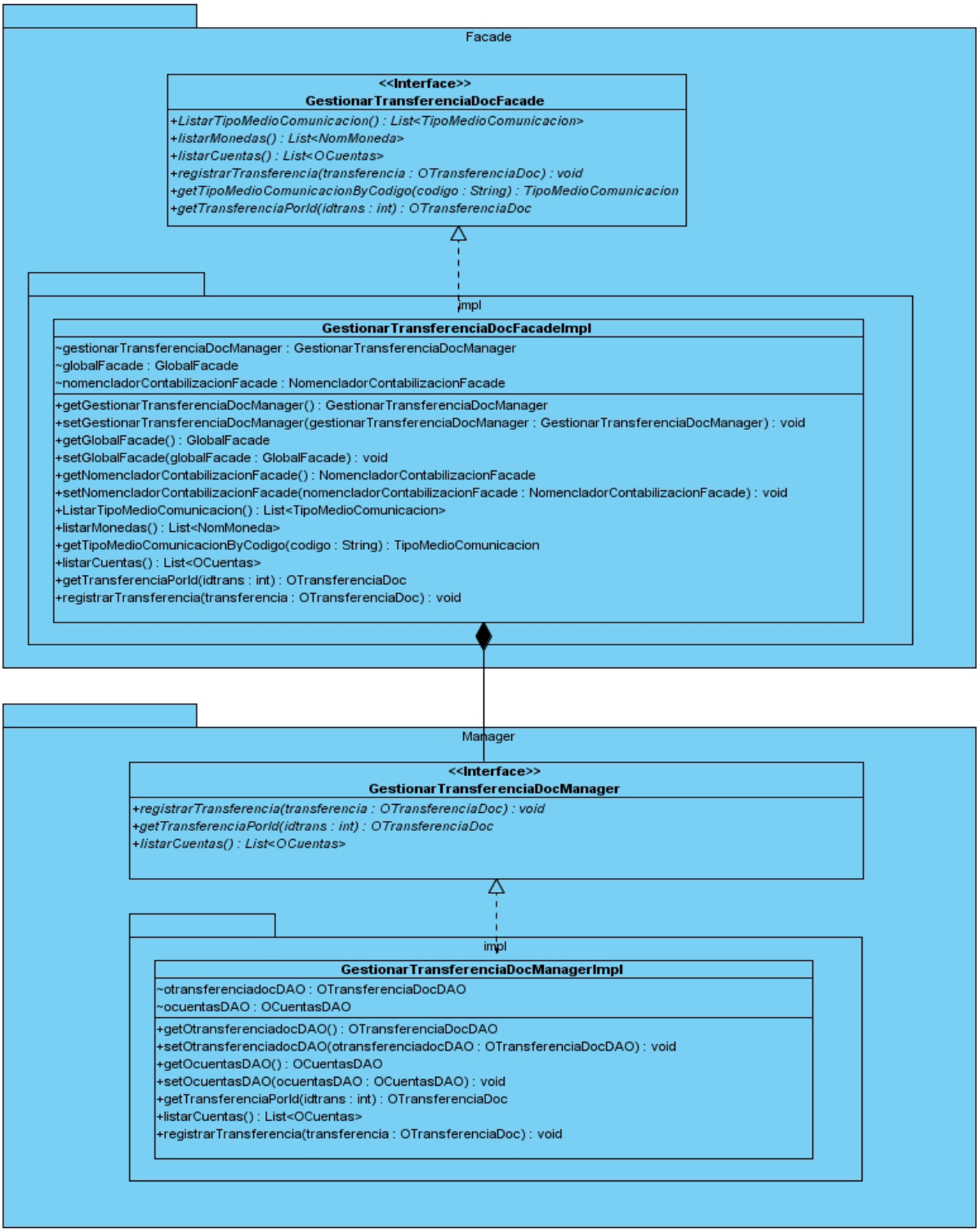
Garbage Collector: el recolector de basura es un mecanismo de gestión de memoria existente en algunos lenguajes de programación.

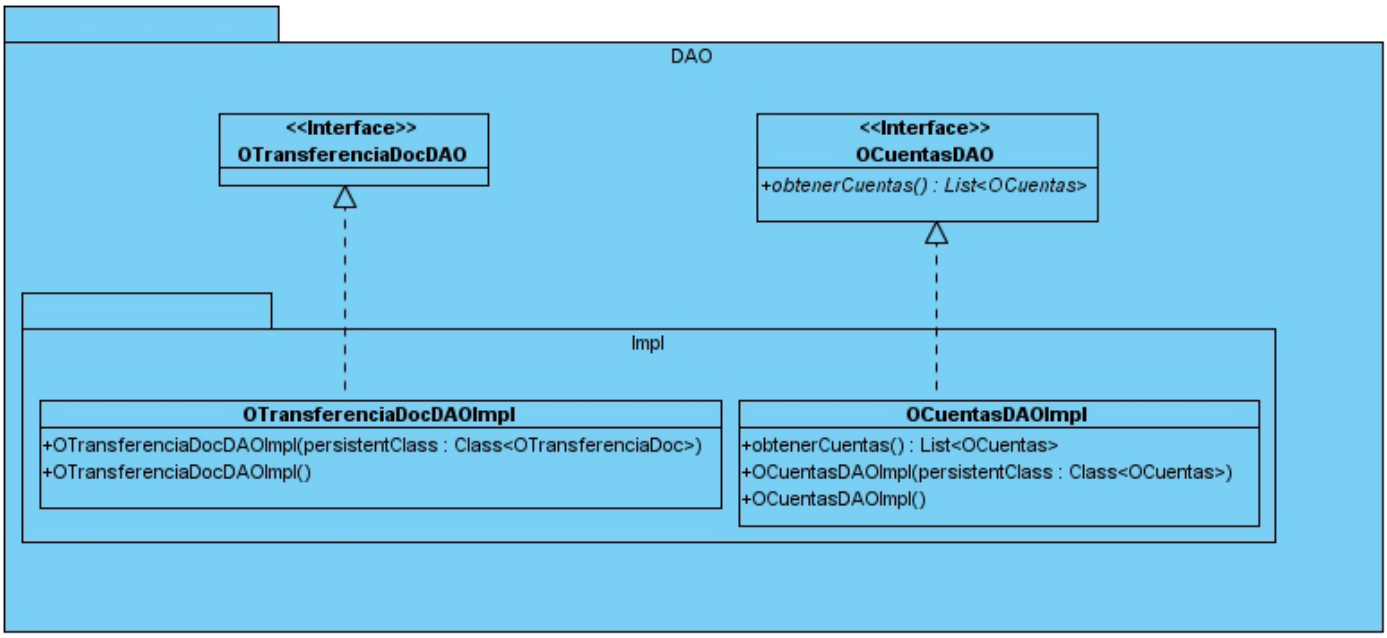
HTML: Lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

Anexos

Anexo 1: Diagrama de clases del diseño del módulo Transferencia.




Facade
<<Interface>>
GestionarTransferenciaDocFacade
+ListarTipoMedioComunicacion() : List<TipoMedioComunicacion>
+listarMonedas() : List<NomMoneda>
+listarCuentas() : List<OCuentas>
+registrarTransferencia(transferencia : OTransferenciaDoc) : void
+getTipoMedioComunicacionByCodigo(codigo : String) : TipoMedioComunicacion
+getTransferenciaPorId(idtrans : int) : OTransferenciaDoc
impl
GestionarTransferenciaDocFacadeImpl
~gestionarTransferenciaDocManager : GestionarTransferenciaDocManager
~globalFacade : GlobalFacade
~nomencladorContabilizacionFacade : NomencladorContabilizacionFacade
+getGestionarTransferenciaDocManager() : GestionarTransferenciaDocManager
+setGestionarTransferenciaDocManager(gestionarTransferenciaDocManager : GestionarTransferenciaDocManager) : void
+getGlobalFacade() : GlobalFacade
+setGlobalFacade(globalFacade : GlobalFacade) : void
+getNomencladorContabilizacionFacade() : NomencladorContabilizacionFacade
+setNomencladorContabilizacionFacade(nomencladorContabilizacionFacade : NomencladorContabilizacionFacade) : void
+ListarTipoMedioComunicacion() : List<TipoMedioComunicacion>
+listarMonedas() : List<NomMoneda>
+getTipoMedioComunicacionByCodigo(codigo : String) : TipoMedioComunicacion
+listarCuentas() : List<OCuentas>
+getTransferenciaPorId(idtrans : int) : OTransferenciaDoc
+registrarTransferencia(transferencia : OTransferenciaDoc) : void
Manager
<<Interface>>
GestionarTransferenciaDocManager
+registrarTransferencia(transferencia : OTransferenciaDoc) : void
+getTransferenciaPorId(idtrans : int) : OTransferenciaDoc
+listarCuentas() : List<OCuentas>
impl
GestionarTransferenciaDocManagerImpl
~otransferenciadocDAO : OTransferenciaDocDAO
~ocuentasDAO : OCuentasDAO
+getOtransferenciadocDAO() : OTransferenciaDocDAO
+setOtransferenciadocDAO(otransferenciadocDAO : OTransferenciaDocDAO) : void
+getOcuentasDAO() : OCuentasDAO
+setOcuentasDAO(ocuentasDAO : OCuentasDAO) : void
+getTransferenciaPorId(idtrans : int) : OTransferenciaDoc
+listarCuentas() : List<OCuentas>
+registrarTransferencia(transferencia : OTransferenciaDoc) : void


DAO
<<Interface>>
OTransferenciaDocDAO
<<Interface>>
OCuentasDAO
+obtenerCuentas() : List<OCuentas>
Impl
OTransferenciaDocDAOImpl
+OTransferenciaDocDAOImpl(persistentClass : Class<OTransferenciaDoc>)
+OTransferenciaDocDAOImpl()
OCuentasDAOImpl
+obtenerCuentas() : List<OCuentas>
+OCuentasDAOImpl(persistentClass : Class<OCuentas>)
+OCuentasDAOImpl()

Anexo 2: Diagramas de secuencia del caso de uso Gestionar Transferencia.

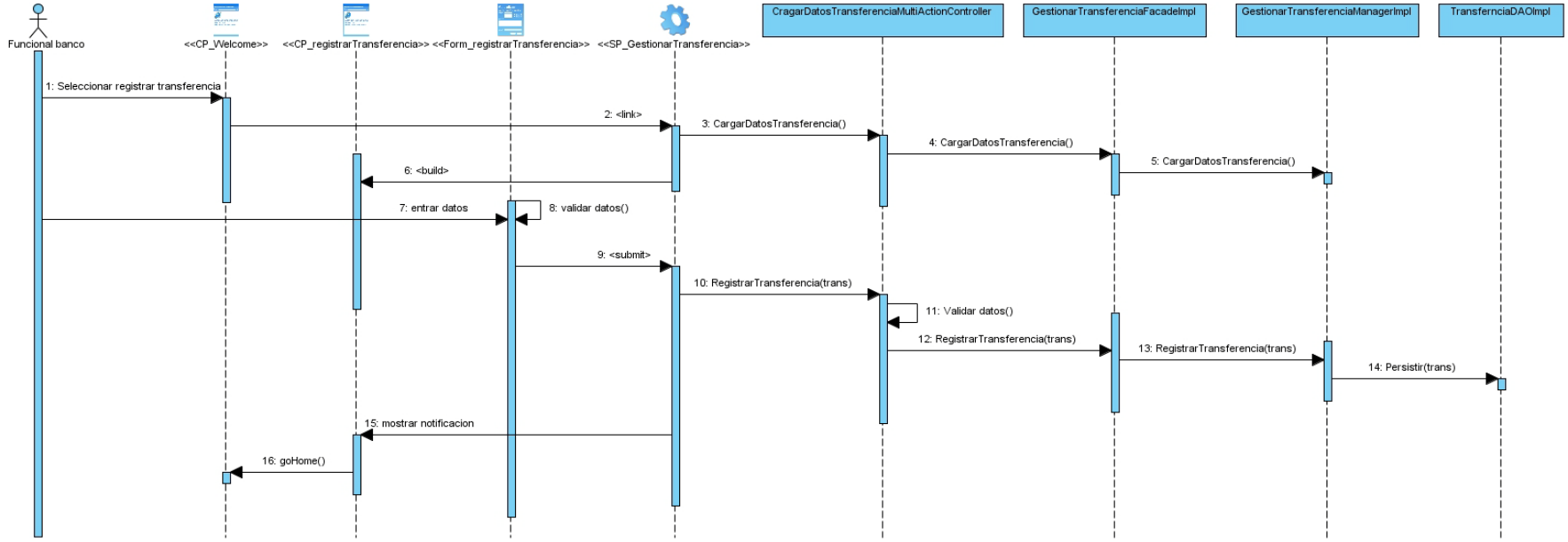


Diagrama de secuencia del escenario registrar transferencia del caso de uso Gestionar Transferencia.

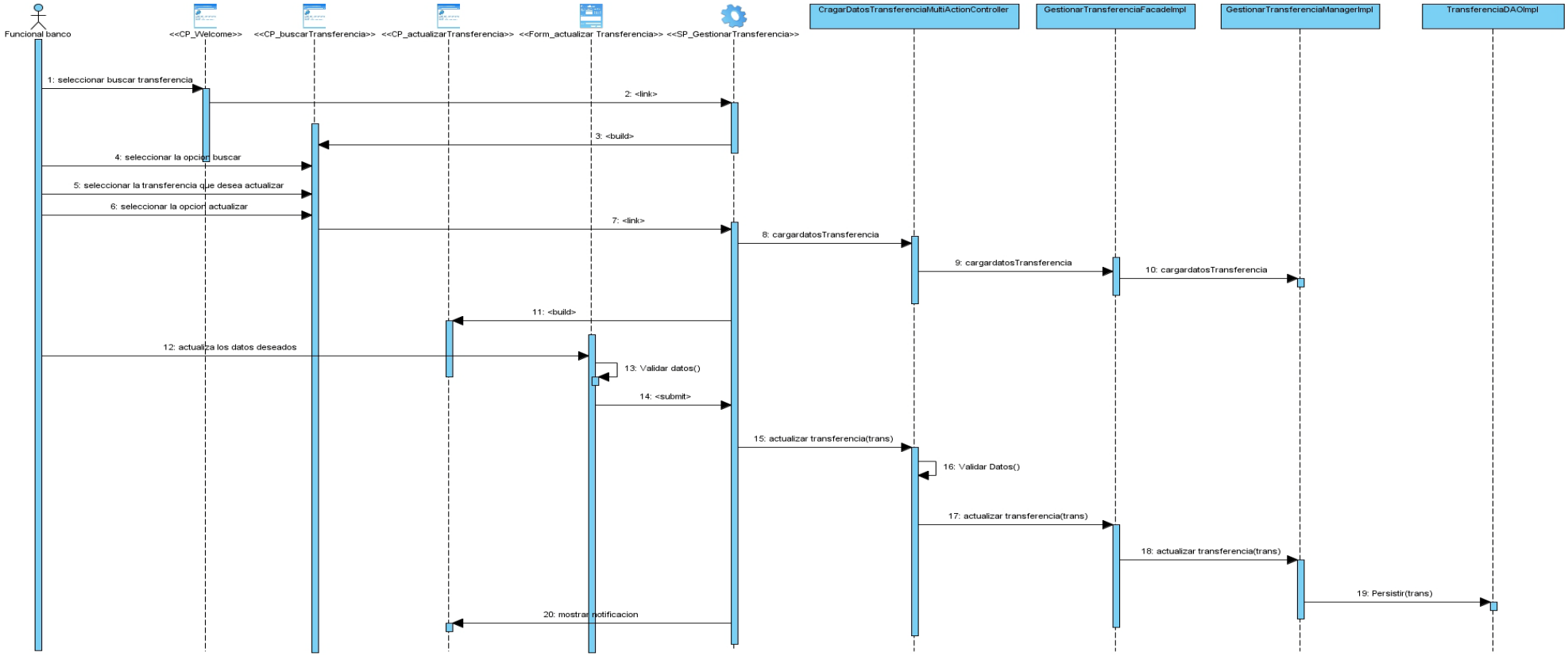


Diagrama de secuencia del escenario actualizar transferencia del caso de uso Gestionar Transferencia.

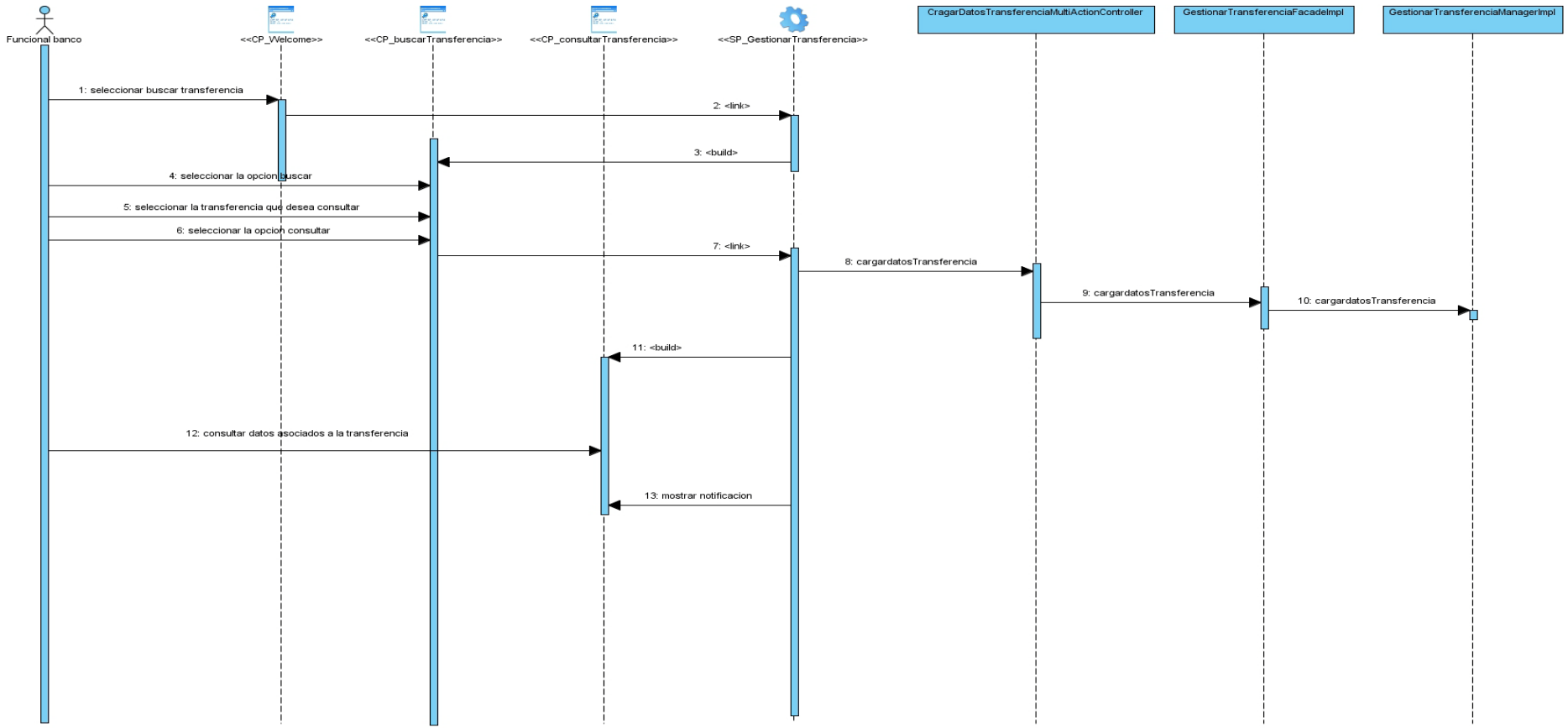
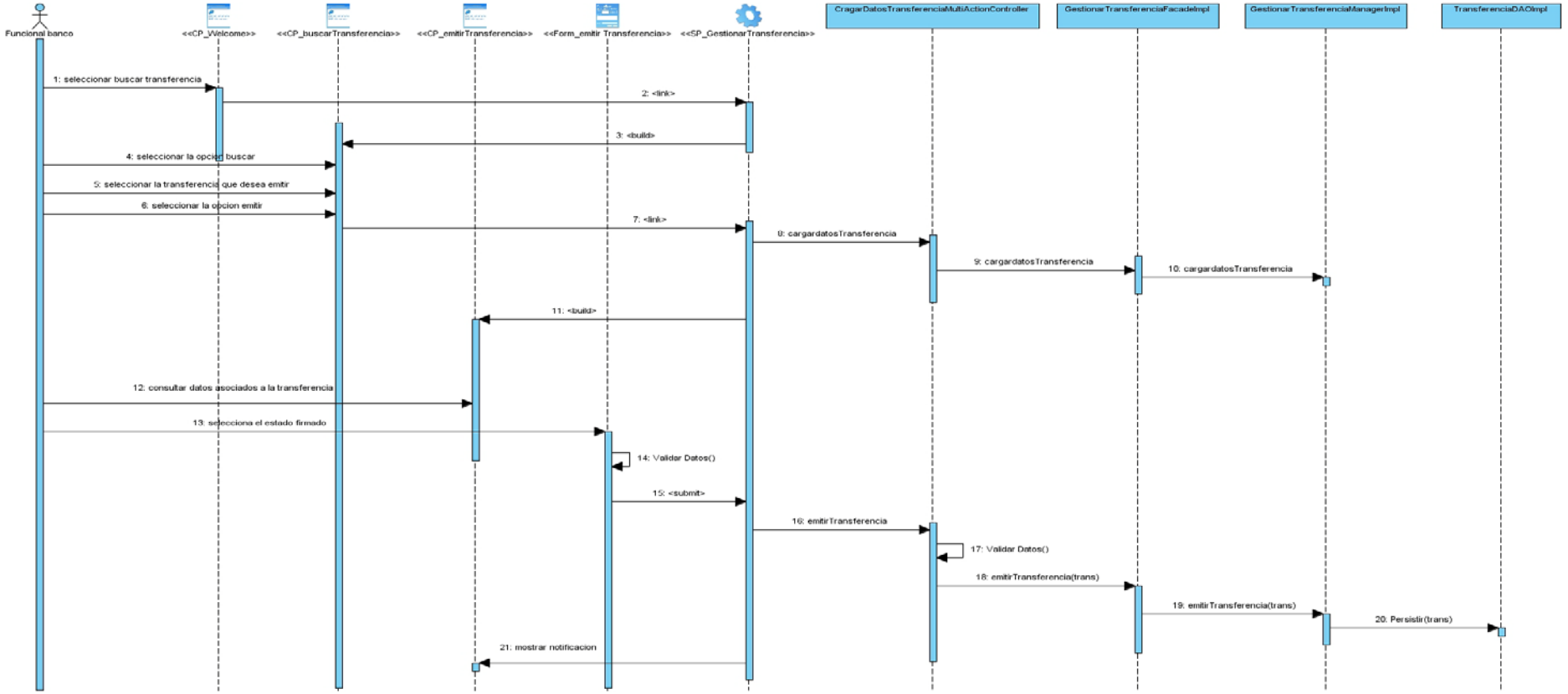


Diagrama de secuencia del escenario consultar transferencia del caso de uso Gestionar Transferencia.

Anexo 3: Diagramas de secuencia del caso de uso Emitir Transferencia.



Anexo 4: Caso de prueba del escenario Registrar Transferencia del caso de uso Gestionar Transferencia.

1. Descripción General.

- ✓ Se realiza el registro de una compraventa.

2. Condiciones de Ejecución.

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

3. Secciones a probar en el Caso de Uso.

Nombre de la sección	Escenario de la sección	Descripción de la funcionalidad
SC 1: Registrar compraventa	EC 1.1: Registrar compraventa	El usuario llena los campos necesarios y registra la compraventa.
	EC 1.2: Datos incorrectos	Se verifica que los datos entrados estén correctos. En caso contrario se notifica a través de un mensaje.
	EC 1.3: Cancelar operación	Se cancela la operación de registrar compraventa.

Tabla 7. Secciones a probar en el escenario registrar compraventa.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	bancoEmisor	Lista desplegable	No	Se carga automáticamente
2	fechaActual	Campo de Fecha	No	Se carga automáticamente
3	cuentaDebitar	Lista desplegable	No	Se selecciona
4	tipoMonedaImporte	Lista desplegable	No	Se selecciona

6	fechaValor	Campo de Fecha	No	Se selecciona
7	importe	Campo de texto	No	Es entrado por el usuario
8	codigoBancoEmisor	Lista desplegable	No	Se selecciona
9	aFavor	Lista desplegable	No	Se selecciona
10	cuentaAcreditar	Lista desplegable	No	Se selecciona
11	observaciones	área de texto	Sí	Es entrado por el usuario
12	tipoMedioComunicacion	Lista desplegable	No	Se selecciona

Tabla 8. Descripción de las variables de entrada.

Resp.	Flujo Central de Eventos
bancoEmisor	
fechaActual	
cuentaDebitar	
tipoMonedaImporte	
fechaValor	
Importe	
codigoBancoEmisor	
aFavor	
cuentaAcreditar	
Observaciones	
tipoMedioComunicacion	
Escenario	

EC 1.1: Registra Transferencia	1. El usuario introduce la URL en el navegador. 2. El sistema muestra una interfaz para que el usuario se autentique. El usuario selecciona la opción Aceptar. 3. El sistema muestra una interfaz con los subsistemas. El usuario selecciona el subsistema Tesorería. 4. El sistema muestra un menú con los módulos que corresponden al subsistema Tesorería. Selecciona el modulo transferencia. Y selecciona la opción Registrar transferencia. 5. El sistema muestra una interfaz para que el usuario realice el registro. 6. El usuario introduce los datos deseados y selecciona la opción Aceptar. 7. El sistema valida los datos, realiza el registro y muestra el mensaje : “Operación realizada satisfactoriamente”
FAX	
NA	
32303000201	
BNC	
BFI	
3000	
07/04/2012	
USD	
32302000203	
05/04/2012	
BFI	
Satisfactoria	

EC 1.2: Datos Incorrectos		Transferencia de BFI al BNC	3230300201	BNC	BFI	3000	07/04/2012	USD	3230200203	05/04/2012	BFI	Satisfactoria

1. El usuario introduce la URL en el navegador.
2. El sistema muestra una interfaz para que el usuario se autentique. El usuario selecciona la opción Aceptar.
3. El sistema muestra una interfaz con los subsistemas.
4. El usuario selecciona el subsistema Tesorería.
5. El sistema muestra un menú con los módulos que corresponden al subsistema Tesorería.
6. Selecciona el modulo transferencia.
7. Y selecciona la opción Registrar transferencia.
8. El sistema muestra una interfaz para que el usuario realice el registro.
9. El usuario introduce los datos deseados y selecciona la opción Aceptar.
10. El sistema valida los datos, y muestra el mensaje: “Debe seleccionar un medio de comunicación”

EC 1.3: Cancelar la operación	**1.** El usuario introduce la URL en el navegador. **2.** El sistema muestra una interfaz para que el usuario se autentique. El usuario selecciona la opción Aceptar. **3.** El sistema muestra una interfaz con los subsistemas. **4.** El usuario selecciona el subsistema Tesorería. **5.** El sistema muestra un menú con los módulos que corresponden al subsistema Tesorería. **6.** Selecciona el modulo transferencia. **7.** Y selecciona la opción registrar transferencia. **8.** El sistema muestra una interfaz para que el usuario realice el registro. **9.** El usuario selecciona la opción Cancelar. **10.** El sistema redirecciona a la página del subsistema.
FAX	Satisfactoria
Transferencia de BFI al BNC	BFI
32303000201	05/04/2012
BNC	32302000203
BFI	USD
3000	07/04/2012

Anexo 5: Diagramas de secuencia del caso de uso Gestionar Compraventa.

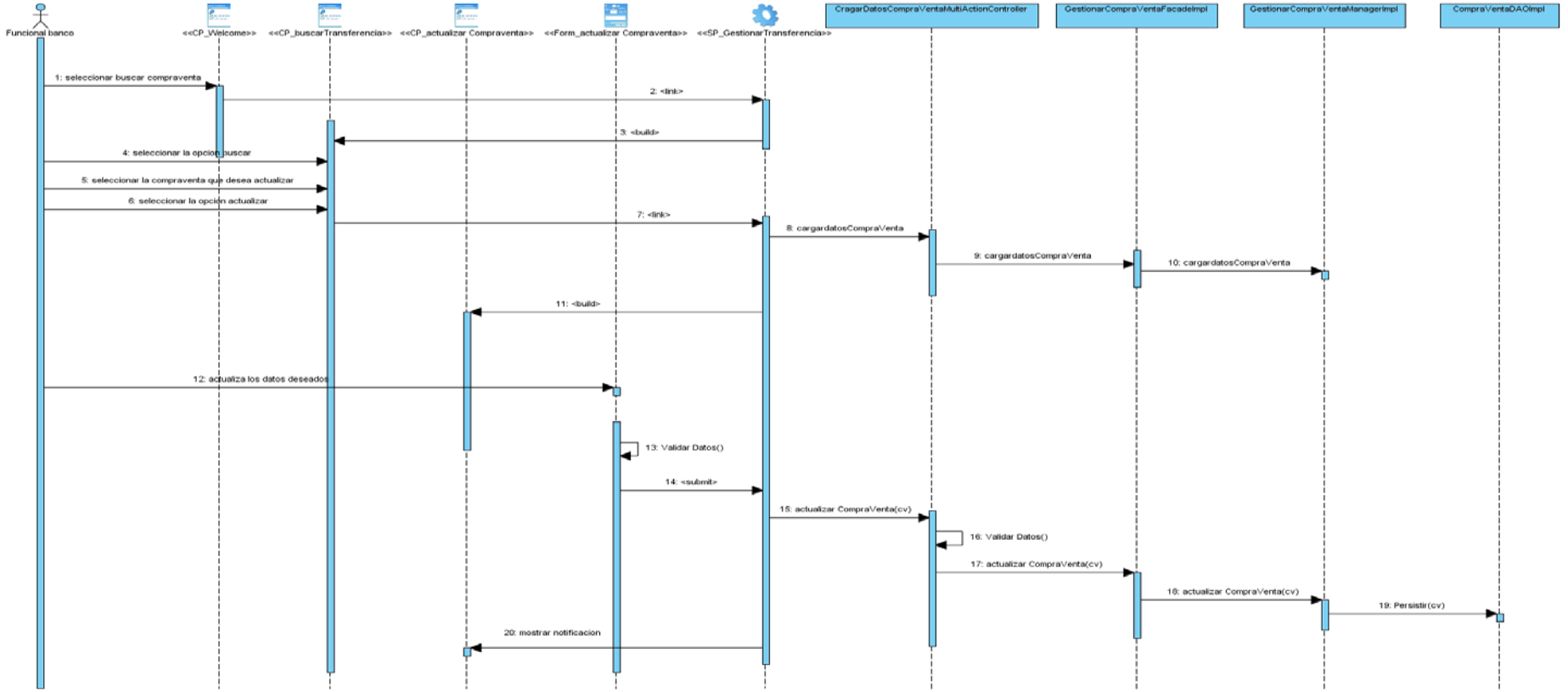


Diagrama de secuencia del escenario actualizar compraventa del caso de uso Gestionar Compraventa.

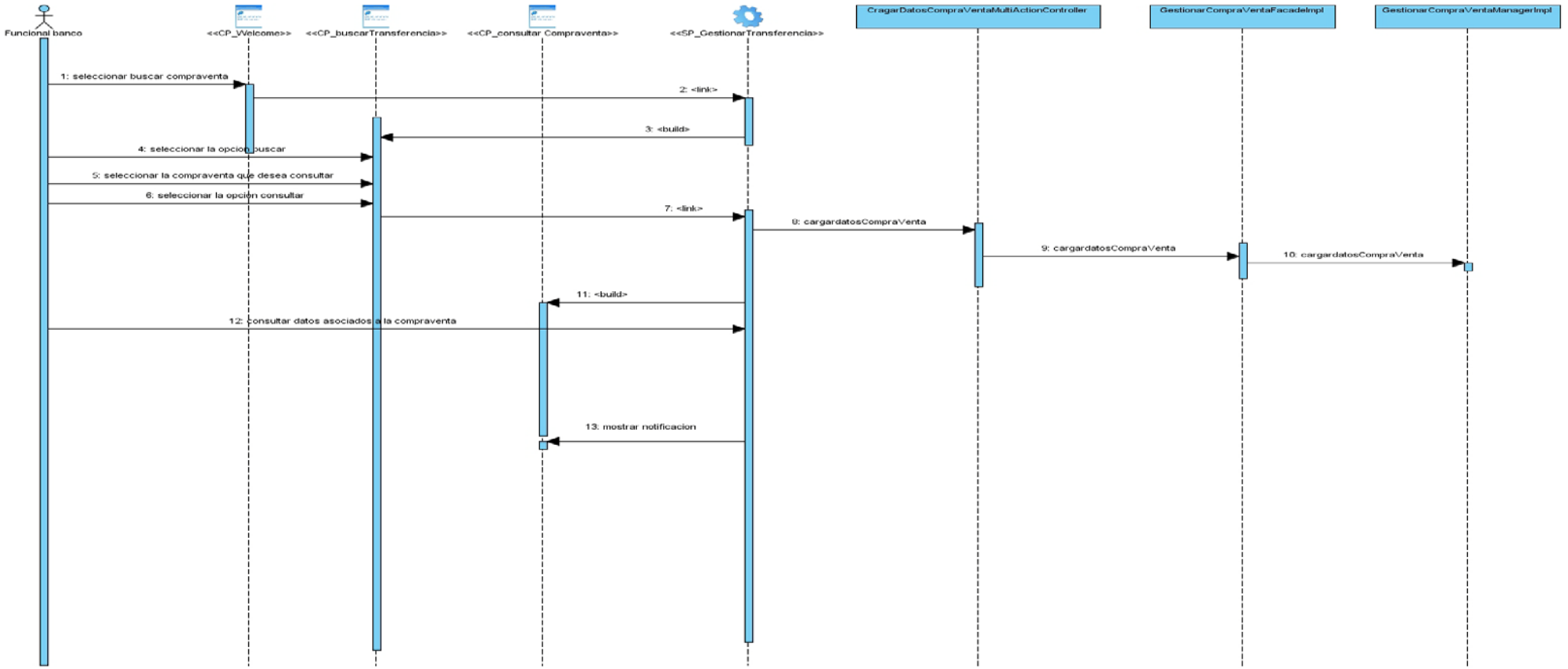


Diagrama de secuencia del escenario consultar compraventa del caso de uso Gestionar Compraventa.

Anexo 6: Diagrama de paquetes del módulo Transferencia.