Universidad de las Ciencias Informáticas

Facultad 15

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

Título: Diseño e implementación del módulo Pagaré del Proyecto Sistema Automatizado de Gestión Bancaria (SAGEB).

Autor: Yúrian Núñez Hernández

Tutor: Ing. Yesenia Perdomo Bello

Ciudad de la Habana, 29 de junio de 2010

DECLARACIÓN DE AUTORÍA

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

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

Yúrian Núñez Hernández	Yesenia Perdomo Bello
_____	_____
Firma del Autor	Firma del Tutor

FRASE

"Muchos me dirán aventurero, y lo soy, solo que de un tipo diferente y de los que ponen el pellejo para demostrar sus verdades."

ERNESTO CHE GUEVARA

DATOS DE CONTACTO

Tutor: Ing. Yesenia Perdomo Bello

Categoría Científica: Ingeniera

Categoría Docente:

Correo electrónico: yperdomo@uci.cu

Síntesis del Tutor:

Graduada en la Universidad de las Ciencias Informáticas. Analista del proyecto SAGEB. Adiestrada. Profesora de la asignatura GSW.

DEDICATORIA

A mi madre, por no dejarme caer nunca y por su apoyo durante toda mi carrera escolar. Te quiero mucho y eres lo más grande que tengo en la vida.

A mi padre, que aunque no lo demuestra se que se preocupa por mí y se siente orgulloso.

A mi hermanita, que aunque es menor la tengo como mi ídolo.

A mis abuelos, ellos saben.

A la crema original, saben quiénes son.

A la gente de la Villa Panamericana por sus preocupaciones.

A mi tutora por sus agallas.

Muchas gracias a todos.

AGRADECIMIENTOS

A mi familia en primer lugar por el apoyo brindado.

A mi tutora por no callarse en la predefensa.

A todos mis colegas: Osvaldo, Herson, Nordy, Damian, Raul, Rolando, Lazaron, Harold (apretador), Maurice, Mary, Irina, Peter, Yanet, Yary, Yisel, Raudel, Norlys, Daryx, Yarida, Walter, Chicoy, Yasmany, Angel Luis, en fin a todos.

A Lamothe por su gran ayuda en este trabajo. Gracias hermano, de veras muchas gracias.

A la gente del proyecto SAGEB que me ayudaron.

A Rosaura por su intenso trabajo.

A la que me dice “Chuchu”. Gracias por tu apoyo y tus noches conmigo.

A todos los que no creyeron que lo lograría. Gracias, eso me impulso bastante.

RESUMEN

En el presente trabajo de diploma se expone la implementación del módulo Gestión de Pagarés del proyecto Sistema Automatizado de Gestión Bancaria (SAGEB). Actualmente la situación general de los procesos de gestión de Pagarés en las entidades bancarias nacionales, se ha visto afectada por la utilización de sistemas informáticos que no cumplen con las expectativas de las nuevas estrategias de negocios de las entidades. En busca del perfeccionamiento de servicios y agilidad en los procesos que se realizan, la Universidad de las Ciencias Informáticas (UCI) es la encargada de desarrollar un sistema que automatice los procesos del Banco Nacional de Cuba (BNC), utilizando el núcleo del Sistema Automatizado para la Banca Internacional de Comercio (SABIC).

En la actualidad el proceso de gestión de Pagarés en el Banco Nacional de Cuba, toma más tiempo del que se necesita realmente, debido a que dichos procesos se encuentran aun sin automatizar, provocando esto, que las negociaciones se vuelvan lentas.

Con la utilización de herramientas de desarrollo se pretende automatizar los procesos vinculados a la gestión de los Pagarés en el Banco Nacional de Cuba, logrando una gestión de la información y la contabilización en tiempo real de estos medios de pago.

TABLA DE CONTENIDO

INTRODUCCIÓN

El desarrollo de sistemas automatizados de gestión empresarial ha tomado un gran auge a escala global, sobre todo a partir del avance de las nuevas tecnologías de la información y las comunicaciones. El mundo del mercado se encuentra sujeto a determinadas reglas y pautas propias del sector, que condicionan las operaciones comerciales entre los diferentes negociadores. Tales reglas se han ido conformando y universalizando en la medida que las relaciones comerciales han adquirido la madurez necesaria como para distinguir las conductas que reportan mayor ventaja al desarrollo de las negociaciones. Hoy, las reglas o normas que rigen la actividad mercantil, han adquirido un valor universal y se repiten por igual en todas partes del mundo donde se comercie. Básicamente la actividad de los mercados se manifiesta de una sola manera sin que sobre ellos influyan las condiciones políticas de los diferentes países. Las relaciones comerciales funcionan por igual en todas las latitudes mundiales, sometidas a las mismas disposiciones y dependiendo de análogos indicadores. Dentro de esta globalización de las conductas y regulaciones que rigen las negociaciones mundialmente, también se repite la utilización de similares herramientas financieras. Así sucede tanto para empresas estatales, como sectores privados o compañías de conformación mixta, todas emplean un mismo paquete de instrumentos financieros que se manifiestan con igual eficiencia en sus resultados y en sus controles de mercado.

En la última década la economía cubana ha experimentado una notable recuperación. En este impulso, ha jugado un papel determinante el sector empresarial al poner en práctica nuevas estrategias productivas y perfeccionar los métodos para la gestión de pagos. La necesidad de crear sistemas encaminados a garantizar el funcionamiento de la economía cubana en las nuevas circunstancias y en el marco de las transformaciones organizativas que se están desarrollando en la actualidad, constituye un reto. La informatización del sistema bancario como parte del avance tecnológico que precisa nuestro país, demanda una elevada capacidad tecnológica y operativa, lo que trae consigo, a gran escala, la utilización de modernos medios de procesamiento de la información. El propósito de la automatización de los procesos bancarios es tener sistemas que se adapten a la nueva estrategia de negocios de la entidad, apoyándose en el uso de herramientas computacionales, sin descuidar los principios contables.

Situación problemática

Como parte de la modernización del Sistema Bancario, el Banco Nacional de Cuba, le solicitó a la Universidad de las Ciencias Informáticas, el desarrollo de un software que fuera extensible a todos los procesos que se desarrollan dentro de las gerencias del propio banco, permitiendo la interacción de los mismos con otros externos desarrollados en los demás bancos y entidades financieras del país. En la actualidad, en su mayoría, las entidades financieras cubanas utilizan el SABIC para el manejo de sus funcionalidades contables y así integrarse al sistema bancario nacional en términos de transacciones bancarias y flujo monetario. El SABIC, partiendo de su diseño, ofrece fortaleza e integridad en la información que maneja y en su momento fue flexible a los requerimientos de cada entidad financiera. Su desarrollo sobre tecnología obsoleta y el creciente cambio de la actividad bancaria lo ha convertido en un sistema decadente, propiciando que no pueda cubrir en su totalidad las exigencias de los procesos que se ejecutan dentro de los bancos cubanos. Es un sistema puramente contable, con escasas funcionalidades para el control de los productos y servicios de cada entidad financiera bancaria.

Algunos de los problemas e insuficiencias que muestra el SABIC, enfocados a la gestión de los Pagarés se especifican a continuación:

- No ofrece una emisión digitalizada de los Pagarés, teniendo los trabajadores de la entidad que realizar el trabajo manualmente.
- No garantiza la gestión de los Pagarés, imposibilitando la obtención de reportes detallados y datos estadísticos para la toma de decisiones de cualquier entidad.
- Propicia una aglomeración de información, obstaculizando la realización de pagos y por tanto extiende las respuestas a peticiones del cliente.

Para el desarrollo del software “Sistema Automatizado de Gestión Bancaria”, se realizó un levantamiento de requisitos por módulos entre los que se encuentra el módulo de Pagaré. Además se definió una arquitectura para el sistema a desarrollar. Todo esto hace que sea necesario el diseño y la implementación de los componentes y artefactos necesarios para una solución que permita la gestión de los Pagarés en el Banco Nacional de Cuba.

Con la situación antes planteada se puede definir como **problema a resolver,** cómo garantizar una eficiente gestión del título valor Pagaré por parte de los funcionarios del Banco Nacional de Cuba. Para solucionar el problema planteado se define como **objetivo general**, diseñar e implementar el módulo

Pagaré del proyecto SAGEB. Como **objeto de estudio** se propone, modelos de diseño e implementación para el desarrollo de aplicaciones de gestión bancaria, enmarcando el **campo de acción** en los modelos de diseño e implementación para la gestión de los pagarés del Banco Nacional de Cuba.

Para dar cumplimiento al objetivo general planteado se definen las siguientes **Tareas a realizar:**

- Caracterización de las herramientas, lenguajes y notaciones utilizadas por el proyecto para facilitar la comprensión de la arquitectura definida por el mismo.
- Caracterización y comprensión de los procesos relacionados con la gestión de los Pagarés en el Banco Nacional de Cuba para la implementación de la aplicación.
- Realización del modelo de diseño del módulo Pagaré del proyecto SAGEB para el desarrollo de la solución.
- Realización del modelo de componentes del módulo Pagaré del proyecto SAGEB para el desarrollo de la solución.
- Implementación de los artefactos de todas las capas lógicas definidas por la arquitectura del proyecto SAGEB, con el objetivo de cumplir todos los requisitos identificados para la gestión de Pagarés.

Posibles resultados

Diseño e implementación de un sistema para la gestión de los Pagarés en el Banco Nacional de Cuba para su posterior despliegue.

Aportes prácticos

Con el desarrollo del módulo de Pagaré para el Proyecto Modernización del Sistema Bancario Cubano se pretende obtener una gestión eficiente de los procesos de pagarés en las entidades bancarias cubanas.

Estructura del trabajo

Con el propósito de organizar el trabajo de diploma y garantizar una mayor comprensión de su estructura, el presente documento se ha distribuido en tres capítulos que muestran la investigación realizada para realizar el diseño e implementación del módulo Pagaré para el Proyecto SAGEB.

CAPITULO 1. FUNDAMENTACIÓN TEÓRICA

En el primer capítulo se expone el estado del arte, se definen algunos conceptos necesarios para la comprensión del trabajo así como se valora la ventaja que tiene la utilización de un módulo como este. Se analiza el desarrollo de los sistemas bancarios enfocados a la gestión de los Pagarés. También se detalla la situación actual de los procesos y se fundamentan las técnicas, herramientas y lenguajes utilizados que apoyan el desarrollo del módulo.

CAPÍTULO 2. DISEÑO DE LA SOLUCIÓN

Se exponen, explican y caracterizan las tendencias y tecnologías utilizadas para el modelado de la aplicación y otras necesarias para su futura implementación. Se realiza la definición y descripción detallada de las funcionalidades del sistema, utilizando como herramienta de modelación Visual Paradigm. En el diseño, se explica la utilización de diferentes patrones de diseño y se presenta el diagrama de paquetes del módulo así como los diagramas de interacción.

CAPÍTULO 3. IMPLEMENTACIÓN DEL SISTEMA

En este capítulo se describe la implementación de la solución general de gestión de los Pagarés. Se analiza la organización del módulo mediante un diagrama de componentes y se muestran aspectos importantes de la implementación como la utilización del framework Spring Modelo Vista Controlador (MVC).

CAPÍTULO #1. FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

El éxito de una investigación encaminada a la solución de un problema, depende de un análisis bien detallado y conciso, orientada siempre a un punto específico, el cual no podemos obviar si deseamos un resultado que determine la solución de un problema. En este primer capítulo se definen los conceptos generales relacionados con los procesos de pagaré. Se realiza un análisis de los mismos así como una fundamentación de las herramientas, tecnologías y lenguajes utilizados.

1.2. Informatización de la sociedad en Cuba.

La informatización en la sociedad cubana, se basa en la utilización ordenada y masiva de las Tecnologías de la Información y las Comunicaciones(TIC) en todas sus esferas, en un esfuerzo por lograr cada vez más eficacia y eficiencia en todos los procesos y por consiguiente mayor generación de riqueza y aumento en la calidad de vida de los ciudadanos. La aplicación de las nuevas TIC, ha traído consigo el procesamiento de la información y ha permitido mejorar gradualmente el servicio en las propias sucursales, facilitando las operaciones de turistas y visitantes extranjeros en general, que pueden encontrar ahora un sistema bancario moderno, prácticamente a la altura de sus similares en el exterior. Cuba ha sabido utilizar con sabiduría y justeza los recursos informáticos y de comunicaciones a su alcance, y hoy, en su marcha hacia la informatización de la sociedad, cuenta, entre otros resultados, con más de 335 mil computadoras, numerosas redes sectoriales y 790 mil usuarios de correo electrónico y otra cifra importante de Internet.

Como aspectos fundamentales de la política, para el desarrollo de la informática en la sociedad cubana tenemos principalmente:

- Potenciar el uso masivo de la nuevas Tecnologías de la Información y las Comunicaciones a favor del desarrollo de la economía nacional, la sociedad y el servicio ciudadano.
- Potenciar el desarrollo de la Informática, Comunicaciones, Electrónica y Automática.
- Facilitar todo tipo de proyectos de los organismos, órganos de gobierno o instituciones, dirigidos a apoyar el proceso de informatización hacia el interior o exterior de los mismos.
- Apoyar el desarrollo e implementación de sistemas verticales de informatización, la informática aplicada a la gestión económica y de recursos humanos.[Bello,2009]

1.3. Sistema Bancario Nacional

Cuba exhibe hoy un dinamismo alentador en el sector de la informática y las telecomunicaciones, diez años después de trazarse como estrategia lo que entendidos en la materia califican de informatización de la sociedad. Nuestro país está consciente de que una sociedad para ser más eficaz, eficiente y competitiva debe aplicar la informatización en todas sus esferas y procesos y está convencida de que para los países subdesarrollados resulta imprescindible el logro de este propósito, ya que su principal objetivo es lograr la supervivencia de sus pueblos.

Actualmente la estructura del sistema bancario y financiero en Cuba está conformada por:

1 Banco Central; 8 Bancos Comerciales, 18 Instituciones Financieras no Bancarias; 13 Oficinas de Representación de Bancos Extranjeros y 4 Oficinas de Representación de Instituciones Financieras no Bancarias.

Banco Central de Cuba (BCC)

Proporciona financiamientos y refinanciamientos a los bancos, concede créditos a fondos de desarrollo o de inversión, realiza descuentos y redescuentos, concede anticipos a los bancos por motivos de iliquidez transitoria, y mantiene informadas a las instituciones bancarias y financieras nacionales (oficiales y sociedades anónimas) e internacionales de la evolución de la economía cubana. Sus objetivos fundamentales consisten en emitir la moneda nacional y velar por su estabilidad, proponer e implementar la política monetaria del país y actuar como órgano rector del sistema bancario y financiero. [**estudios-economicos-cubanos.org**]

Banco Nacional de Cuba (BNC)

Este banco fue creado el 23 de diciembre de 1948 como banco central del Estado y mediante la Ley No. 13, con autonomía orgánica, personalidad jurídica independiente y patrimonio propio. Entre sus principales funciones esta:

- Obtener y otorgar créditos en moneda libremente convertible.
- Librar, aceptar, descontar, avalar y negociar, en cualquier forma, letras de cambio, pagarés, cheques y otros documentos mercantiles negociables denominados en moneda nacional y divisas.

- Fijar las tasas de interés que deberán aplicar a las operaciones que efectúe el Banco, dentro de los límites que establezca el Banco Central de Cuba.[**BNC. 2007**]

1.4. Títulos Valores. Pagaré

Los Títulos Valores son documentos mercantiles en los que está incorporado un derecho privado patrimonial, por lo que el ejercicio del derecho está vinculado jurídicamente a la posesión del documento. En las compraventas que se realizan dentro del tráfico mercantil, está muy extendida la utilización de algunos Títulos Valores como medio de pago y, en ocasiones, como instrumento de crédito. Dentro de los títulos valores más usados están los cheques, letras de cambio y pagarés. [**J. GARRIGUES, 1994**]

Un pagaré es un documento que contiene una promesa incondicional de pago. La persona que lo emite, que se conoce como suscriptora o giradora, se compromete a pagar a una segunda persona, conocida como beneficiaria o tenedora, una cierta cantidad de dinero en un determinado plazo. [**Definición.de. 2008**]

Al constituir un documento formal de pago, un pagaré debe cumplir con ciertos requisitos de validez. Debe estar especificada, dentro del documento, la palabra que lo identifica como tal. Además debe detallar el importe que se pagará, ya sea en letras o en números. Algunos ejemplos de títulos valores se muestran en los anexos 1-5.

1.4.1. Requisitos que debe contener un Pagaré

1- Denominación de la entidad y oficina librada.

2- Datos de la cuenta librada, habitualmente código cuenta cliente e International Bank Account Number (IBAN).

3- Fecha de vencimiento del pagaré .

4- Importe del pagaré expresado en cifras.

5- Persona a la que se debe realizar el pago.

6- Importe del pagaré expresado en letras. En caso de discrepancia entre el importe expresado en letras y el importe expresado en cifras prevalecerá el primero.

7- Fecha y lugar de emisión.

8- Identificación del documento en sí mismo, serie y número del documento.

9- Firma del emisor del pagaré.

10- Identificación del pagaré codificada especialmente para una lectura automatizada del documento.

En el anexo 6 se muestra un pagaré con sus requisitos.

1.4.2. Personas que intervienen

- **El girador:** Persona que se compromete a pagar la suma de dinero. El girador coincide con el librador que es aquel que emite el pagaré.

- **El beneficiario o tenedor:** Persona a cuya orden debe hacerse el pago de la suma de dinero estipulada en el pagaré.

1.4.3. Vencimientos

- **Vencimientos a la vista:** Implica que una vez presentado el Pagaré al girado debe ser pagada.

- **Vencimientos a día cierto determinado:** Se fija en el mismo texto del título, el día, mes y año en que se va a realizar el pago.

- **Vencimientos a día cierto indeterminado:** Sólo es posible cuando ocurre un hecho futuro como la muerte de una persona.

- **Vencimientos ciertos sucesivos:** También llamado por cuotas, se utiliza para realizar pagos periódicos determinados.[**Bello, 2009**]

1.5. Análisis de software de gestión bancaria

La economía actualmente para los bancos constituye un sinónimo de cambio rápido. La constante exigencia de los usuarios, deviene la utilización de nuevas tecnologías y de sistemas informáticos capaces de automatizar los procesos bancarios lo más eficazmente posible.

1.5.1. Emisión de pagarés Banco Popular

Es una aplicación que permite rellenar el formulario del pagaré en formato carta (A4) de forma fácil. Estos formularios son los que se pueden solicitar en la sucursal del Banco. Es un programa esencial y

práctico para cualquier empresa que emita este tipo de documento a sus proveedores. [**abcdatos.com**]

Fecha: 15/7/2008.

Tamaño: 21,34 MB.

Autor: Mari Carmen Chorro.

1.5.2. Easy Pagarés TRV v4.25

Aplicación para la gestión de Pagarés y Talones. Easy Pagarés TRV v4.25 es una aplicación que pretende automatizar la gestión e impresión de pagarés. La facilidad de uso y la capacidad de personalizar el formato de impresión de los pagarés para los distintos bancos, son las características más destacables de este producto.

Easy Pagarés TRV v4.25 permite la emisión de cartas informativas a proveedores con las facturas saldadas tras la emisión del pagaré, y la gestión de vencimientos y fecha de cobro del pagaré por parte del beneficiario. [**boxsoftware.net**]

Tamaño: 8,43 MB.

Autor: Trevenque Sistemas de Información.

Idioma: Español.

1.5.3. Pagarés y Gestión de Cobros Pro 13.0

Pagarés y Gestión de Cobros permite realizar una completa gestión de todos los pagarés bancarios El programa consta de una interfaz muy intuitiva desde la que se puede llevar un registro de todas las actividades realizadas. Permite sellar los pagarés, firmarlos, cruzarlos, gestionar cobros, generar informes muy detallados y estar siempre al tanto de los avales bancarios correspondientes. Pagarés y Gestión de Cobros puede ser utilizado desde cualquier dispositivo extraíble USB o sobre cualquier sistema operativo Windows. [**pagares-y-gestion-de-cobros.softonic.com**]

Tamaño: 3.100 MB.

Valoración

En el mundo existen diversidades de sistemas que gestionan los Pagaré como documentos de pago en las negociaciones que se establecen entre bancos o entidades financieras. La característica fundamental que nos impide la utilización de estos sistemas es que son software propietarios, donde el usuario final tiene limitaciones para usarlo, modificarlo o redistribuirlo. En nuestro país, la economía que fluye es atípica, siendo el bloqueo, uno de los impedimentos para que no podamos comercializar directamente con otros países. Por estas razones los procesos asociados a los Pagaré que son los medios de pagos que se utilizan principalmente para el pago del combustible del país producto de las negociaciones que se establecen entre PDVSA y Cuba, son utilizados y aplicados de formas diferentes, esto propicia que casi ningún sistema a nivel mundial se ajuste a las características de negocio y funcionalidades que requiere la gestión de Pagaré en el Banco Nacional de Cuba.

1.6. Tecnologías

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

Las tecnologías y metodologías utilizadas en la elaboración del módulo en cuestión, fueron definidas por el grupo de arquitectura del proyecto SAGEB, siendo esta la razón por la cual en este fragmento del capítulo no se definen dichas herramientas, sino que se realiza una resumida caracterización de cada una de ellas y de los beneficios que aportan en el trabajo de diseño e implementación del módulo implicado.

1.6.1. Ingeniería del software

El término ingeniería de software abarca al grupo de métodos, técnicas y herramientas que se utilizan en la producción del software, más allá de la actividad principal de programación. La ingeniería de software constituye la disciplina o área de la informática que ofrece métodos y técnicas para desarrollar y mantener software de calidad.

Esta ingeniería trata con áreas muy diversas de la informática y de las ciencias de la computación, tales como construcción de compiladores, sistemas operativos, o desarrollos Intranet/Internet, abordando todas las fases del ciclo de vida del desarrollo de cualquier tipo de sistemas de información y aplicables a infinidad de áreas: negocios, investigación científica, medicina, producción, logística, **banca**, control de tráfico, meteorología, derecho, Internet, Intranet, etc. [**Bello, 2009**]

"Ingeniería de software es la aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar y mantenerlos. Se conoce también como Desarrollo de Software o Producción de Software" (Bohem, 1976).

1.6.2. Metodología de desarrollo RUP[1]

El proyecto decidió utilizar como metodología de desarrollo RUP. Esta incluye artefactos (productos tangibles del proceso) y roles (papel que desempeña una persona en un determinado momento).

RUP es una metodología que tiene como objetivo administrar el ciclo de vida de un proyecto minimizando así los riesgos. La gestión de los requerimientos de un proyecto es sustentada a través del análisis y diseño de Casos de Uso y el modelado es llevado a cabo usando UML. Según JACOBSON, 2004, RUP es un proceso que se caracteriza por ser:

- **Dirigido por casos de uso.** Los casos de uso describen los requisitos funcionales del sistema desde la perspectiva del usuario y se usan para determinar el alcance de cada iteración y el contenido de trabajo de cada persona del equipo de desarrollo.

- **Centrado en la arquitectura.** La arquitectura permite ganar control sobre el proyecto para manejar su complejidad y controlar su integridad. Hace posible la reutilización a gran escala y provee una base para la gestión del proyecto.

- **Iterativo e incremental.** Se divide en 4 fases: Inicio, Elaboración, Construcción y Transición, y cada una de ellas se divide en iteraciones. En cada iteración se trabaja en un número de disciplinas haciendo énfasis en algunas de ellas.

Además de estas características principales cabe destacar las siguientes:

- **Desarrollo basado en componentes:** La creación de sistemas intensivos en software requiere dividir el sistema en componentes con interfaces bien definidas, que posteriormente serán ensamblados para generar el sistema. Esta característica en un proceso de desarrollo permite que el sistema se vaya creando a medida que se obtienen o que se desarrollan y maduran sus componentes.

[1] ***R****ational* ***U****nified* ***P****rocess (Proceso Unificado de Rational).*

- **Utilización de un único lenguaje de modelado:** UML es adoptado como único lenguaje de modelado para el desarrollo de todos los modelos.

- **Proceso Integrado:** Se establece una estructura que abarque los ciclos, fases, flujos de trabajo, mitigación de riesgos, control de calidad, gestión del proyecto y control de configuración. El proceso unificado establece una estructura que integra todas estas facetas. Además esta estructura cubre a los vendedores y desarrolladores de herramientas para soportar la automatización del proceso, soportar flujos individuales de trabajo, para construir los diferentes modelos e integrar el trabajo a través del ciclo de vida y a través de todos los modelos.

1.6.3. Lenguaje de Modelado UML[2]

UML es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software, pero no especifica en sí mismo qué metodología o proceso usar.

La utilización de UML constituye una ventaja ya que es una notación resultante de la evolución de las notaciones previas en ingeniería de software. Toma los aspectos fuertes de tres metodologías anteriores: OMT, Booch y OOSE. La notación UML se fundamenta en principios de modelado, lo cual es importante para toda implementación de un sistema de información.

1.6.4. DynamicJasper

DynamicJasper es una biblioteca de código abierto libre que oculta la complejidad de Jasper Reports. Ayuda a los desarrolladores a ahorrar tiempo en el diseño de simple / mediana complejidad.

DynamicJasper permite al desarrollador crear rápidamente una gran variedad de reportes a través de una intuitiva API[3] escrita en Java. Esta permite definir programáticamente las columnas, grupos, totales, gráficos (charts), sub-reportes y el formato de salida (pdf, Excel, html, etc.) en tiempo de ejecución. La API maneja todo lo relacionado con la diagramación y posicionamiento de los elementos del reporte haciendo el proceso de diseño fácil y automático. [**DynamicJasper.com**]

[2] ***U****nified* ***M****odeling* ***L****anguage (Lenguaje Unificado de Modelado)*

[3] ***A****plication* ***P****rogramming* ***I****nterface*

1.7. Ambiente de desarrollo

Un ambiente de desarrollo o Entorno de Desarrollo Integrado (IDE por sus siglas en inglés) es un programa compuesto por un conjunto de herramientas para un programador. Pueden dedicarse en exclusivo a un solo lenguaje de programación o a varios. Un ambiente de desarrollo constituye un entorno de programación empaquetado como una aplicación, consistente en un editor de código, un compilador, un depurador y un constructor de interfaz. Los ambientes de desarrollo proveen un marco de desarrollo amigable para la mayor parte de los lenguajes de programación.

1.7.1. Eclipse

Eclipse es un IDE (Integrated Development Environment), en español Entorno de Desarrollo Integrado para Java muy potente. Fue creado por la IBM bajo la filosofía de software libre. Se está convirtiendo en el estándar de puntera de los entornos de desarrollo para Java. Es Eclipse un marco de trabajo que está compuesto por componentes que se pueden o no incluir en dependencia de las necesidades del desarrollador, a estos complementos se les llama plugins. [**Bello, 2009**]

1.7.2. Contenedor Web (Tomcat 6)

Tomcat es un contenedor de servlets que implementa las especificaciones de Java Server Page. Es desarrollado en un entorno abierto. Fue publicado bajo la licencia del software de Apache. Tomcat puede funcionar como servidor web por sí mismo. Es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. [**Bello, 2009**]

1.7.3. Control de Versiones. (SubVersion)

Subversión es un controlador de versiones empleado en la administración de archivos utilizados en el desarrollo de software o contenido. Concurrent Versions System (CVS), considerado su antecesor, es uno de los controladores de versiones más utilizados en proyectos de software libre, sin embargo, a pesar de su amplio uso, el mismo diseño de CVS resultó ineficiente para diversos grupos de usuarios, y ante estas inconformidades se dio inicio al proyecto que hoy es conocido como Subversión, el mismo que ha empezado a socavar el dominio de CVS.

Es una herramienta desarrollada bajo tecnologías open source. Uno de sus principales objetivos es mantener integro el control de versiones, se integra con el eclipse mediante el plugin SubEclipse.

1.7.4. Gestor de Base de Datos. (SQL Server)

SQL Server 2005 provee herramientas sólidas y conocidas a trabajadores de la información, reduciendo la complejidad de la creación, despliegue, administración y uso de aplicaciones analíticas y de datos empresariales en plataformas que van desde los dispositivos móviles hasta los sistemas de datos empresariales. A través de un conjunto global de características, la interoperabilidad con sistemas existentes y la automatización de tareas rutinarias; ofrece una solución completa de datos para empresas de todos los tamaños. [**IGP SQLServer, 2007**]

1.7.5. Herramienta CASE[4]. Visual Paradigm for UML 6.1

La principal ventaja de la utilización de una herramienta CASE, es la mejora de la calidad de los desarrollos realizados y, en segundo término, el aumento de la productividad. Para conseguir estos dos objetivos es conveniente contar con una organización y una metodología de trabajo, además de la propia herramienta.

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor costo. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Visual Paradigm constituye una herramienta software libre de probada utilidad para el analista. Dentro de sus características se aprecia que soporta notación BPMN[5] y UML versión 2.1. [**Fuentes, 2009**]

1.7.6. Plataforma J2EE[6]

J2EE ha sido diseñada para aplicaciones distribuidas que son construidas con base en componentes, los cuales interaccionan entre sí para formar parte de una aplicación J2EE. Un componente de esta plataforma debe formar parte de una aplicación y ser desplegado en un contenedor, o sea en la parte del servidor J2EE que le ofrece al componente ciertos servicios de bajo nivel y de sistema (tales como seguridad, manejo de concurrencia, persistencia y transacciones).

---

[4] ***C**omputer **A**ided **S**oftware **E**ngineering (Ingeniería de Software Asistida por Ordenador)*

[5] ***B**usiness **P**rocess **M**anagement **N**otation: Notación para el Modelado de Procesos de Negocio, notación gráfica estandarizada que permite el modelado de procesos de negocio.*

[6] ***J**ava**2** **E**nterprise **E**dition*

La plataforma J2EE es un conjunto de herramientas que crean un escenario ideal para el desarrollo y despliegue de aplicaciones escalables en la Web, y que cuentan con las siguientes características:

- **Escalable:** Si tu empresa ve incrementado su número de clientes, nada más sencillo que añadir nuevos componentes J2EE a una aplicación Web para soportar al aumento de clientes, sin tener que reescribir todo el código de nuevo.

- **Altamente Soportada:** Prácticamente cualquier gran empresa de software tiene un contenedor de componentes o servidor de aplicaciones web compatibles con J2EE, entre ellas IBM (Websphere), BEA (WebLogic), Apache (Tomcat), la propia Sun con su nuevo servidor de aplicaciones iPlanet, Macromedia con JRun, etc.

- **Segura:** Mientras que otros modelos de aplicaciones empresariales requieren medidas de seguridad específicas en cada aplicación, el entorno de seguridad de la plataforma J2EE permite que se definan unas restricciones de seguridad en el momento de despliegue de la aplicación, aislando así las aplicaciones de la complejidad de las implementaciones de seguridad, la plataforma J2EE hace portables una gran complejidad de implementaciones de seguridad.

La plataforma J2EE resulta una propuesta atractiva, interesante y de vanguardia que responde, de manera natural, a la demanda actual para el desarrollo de software, bajo el concepto de arquitectura en capas.

1.7.7. Lenguaje de programación JAVA

Java permite la modularidad, por lo que se pueden hacer rutinas individuales que sean usadas por más de una aplicación, por ejemplo tenemos una rutina de impresión que puede servir para el procesador de palabras, como para la hoja de cálculo.

La programación en Java, permite tanto el desarrollo de aplicaciones bajo el esquema de Cliente Servidor, como de aplicaciones distribuidas, lo que lo hace capaz de conectar dos o más computadoras u ordenadores, ejecutando tareas simultáneamente, y de esta forma logra distribuir el trabajo a realizar.

1.8. Frameworks

Un framework es una estructura de software compuesta por componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede

considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta. Puede incluir soporte de programas, bibliotecas y un lenguaje de scripting entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Los frameworks son diseñados con el intento de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional.

Un framework permite separar en capas la aplicación:

- La lógica de presentación que administra las interacciones entre el usuario y el software.
- La lógica de dominio o de negocio, que manipula los modelos de datos de acuerdo a los comandos recibidos desde la presentación.
- La lógica de datos que permite el acceso a un agente de almacenamiento persistente u otros.

Spring Framework

Spring es un framework bajo la concepción de código abierto diseñado para el desarrollo de aplicaciones de la plataforma J2EE. Ofrece mucha libertad a los desarrolladores en Java, brindando soluciones bien documentadas, seguras y robustas gracias a sus 11 grandes módulos como se muestra en el anexo 7.

1.8.1. Spring MVC

Se utilizará Spring MVC para atender las peticiones web simples con navegación lineal. Spring MVC brinda una jerarquía de clases “Controller” para recibir dichas peticiones.

Los diferentes tipos de “*Controller*” que brinda Spring MVC responden a diferentes momentos en la recepción de peticiones.

- **AbstractComandController:** Se debe utilizar cuando se necesita trabajar con diferentes parámetros que se reciben desde una petición y estos parámetros se pueden mapear a un objeto específico. Este controlador está preparado también para insertarle diferentes validaciones al objeto mapeado sin que el desarrollador tenga que preocuparse por ello en la recepción del objeto.

- **SimpleFormController:** Cuando se envían datos desde el cliente a través de un formulario se debe utilizar esta clase que brinda Spring MVC, ya tiene el comportamiento del AsbtractComand e incluye, la carga inicial del formulario a procesar y la generación de la página posterior al procesamiento de los datos.

- **WizardController:** Cuando se procesa gran cantidad de información de manera secuencial a través de varias páginas (en forma de un asistente) se complejiza el desarrollo con SimpleFormController. Para lograr un correcto procesamiento de los datos, Spring propone la utilización del WizardController. Esta clase incorpora varios elementos que facilitan el procesamiento secuencial de cada página, validaciones de los datos introducidos por páginas y mantenimiento en memoria de todos éstos.

- **MultiActionController:** Esta clase se debe utilizar para atender varias peticiones que estén relacionadas con el negocio.

1.8.2. Spring WebFlow

Spring WebFlow es un framework que permite manejar la navegación de la aplicación web. Surge como una respuesta a la funcionalidad limitada del flujo de página ofrecida por los frameworks MVC clásicos. Los flujos Web en este framework son diseñados para ser auto-controlados, dando la posibilidad de definir reglas (múltiples y complejas) de navegación.

En Spring WebFlow un flujo maneja la conversación (ámbito nuevo que define el vacío entre Sesión y Petición) completa, desde que inicia hasta que culmina y en este punto limpia la memoria automáticamente siempre y cuando se termine el flujo, lo cual es una mejoría ya que el usuario no tiene que preocuparse por borrar los recursos en memoria que no estén siendo utilizados. Spring WebFlow es un complemento a Spring MVC.

1.8.3. Data Access Object (DAO) y Data Access Object Genérico

El framework Spring propone la utilización del patrón DAO para interactuar con la base de datos. Este soporte lo brinda a través de "*Spring DAO*". El patrón DAO tiene como objetivo separar la manera en que la capa de acceso a datos resuelve el manejo de los datos persistentes con la lógica de la aplicación.

Por otra parte, existe un componente nombrado *¨DAO genérico¨* que tiene programadas las funciones básicas para interactuar con la base de datos (Inserción, Actualización, Eliminación y Consultas por criterios).

1.8.4. Hibernate

Spring Object Relation Mapping (ORM), es el soporte a la integración de Spring con varios frameworks que interactúan con la base de datos. Dentro de esta integración está Hibernate. El framework Hibernate es utilizado para el trabajo con la base de datos. Este framework es considerado un ORM. Una de las características principales es que abstrae elegantemente el trabajo con la base de datos, soportando la manipulación de los datos persistentes objetualmente, a través de ficheros que mapean clases Java contra tablas. Hibernate es recomendable utilizarlo fundamentalmente en la interacción directa con las tablas de la base de datos.

1.8.5. Dojo Toolkit

Dojo Toolkit es un framework Javascript que permite el desarrollo de aplicaciones web enriquecidas en el cliente. Es popular porque está integrada en numerosos IDEs y otros frameworks para desarrollo de webs. Dojo proporciona variadas opciones en una sola biblioteca haciendo un mejor trabajo que sustenta los nuevos y viejos Browsers, resolviendo los problemas de compatibilidad entre los navegadores. Tiene múltiples puntos de entrada, es independiente del intérprete y unifica estándares de codificación.

1.8.6. Spring JDBC

Spring JDBC es un módulo perteneciente al framework Spring. El mismo posee algunas librerías de clases para trabajar con base de datos a través del API de Java JDBC. Unas de las características principales de este módulo es el soporte que brinda para invocar procedimientos y funciones almacenadas.

Por otra parte en el proyecto se desarrolló un componente para facilitar la interacción con los procedimientos y funciones almacenadas utilizando las ventajas que brinda el módulo Spring JDBC. El componente se nombra “StoreProcedureDAOSupport”.

1.8.7. Spring Transaction

Una transacción es la manera de agrupar una o varias tareas en una única y consistente unidad de trabajo, de forma tal que no puede verse el funcionamiento de ellas por separado; sino como un todo armónico. En el caso de que falle algunas de estas tareas se deshacen las tareas ejecutadas inicialmente en esa transacción y se vuelve al estado inicial. En la aplicación podrían ejecutarse múltiples transacciones concurrentemente y a menudo estas trabajan con el mismo dato para realizar su trabajo.

Spring Transaction brinda soporte para asegurar transaccionalmente las ejecuciones de las funcionalidades de la aplicación. Además de soportar el manejo de transacciones para varios gestores de base de datos y recursos compartidos a través del API JTA, también se pueden definir las transacciones en el código de la aplicación y declarativamente en los ficheros de declaración.

1.9. Conclusiones del capítulo

En el presente capítulo se ha tratado de forma general y resumida los diferentes conceptos y procesos que abarcan la gestión de pagarés, determinando las ventajas, desventajas y factibilidad de su utilización en las entidades cubanas. Además se realizó un estudio de las herramientas, tecnologías, lenguajes y la metodología propuesta por la dirección de arquitectura del proyecto SAGEB, permitiendo sentar las bases para el desarrollo de la aplicación.

CAPÍTULO #2: DISEÑO DE LA SOLUCIÓN

2.1. Introducción

En este capítulo se da una panorámica de la arquitectura del sistema SAGEB, definiendo el diseño, la cual responderá a las necesidades planteadas en el proyecto de modernización del sistema bancario nacional. El sistema se desarrolla bajo la tecnología de Java Enterprise Edition, donde el framework que se utiliza como núcleo de la aplicación es Spring Framework. Para la persistencia de datos el framework Hibernate y el almacenamiento se realiza con SQLServer.

2.2. Definición de Requerimientos Funcionales

Producto de la elicitación de requisitos, se obtuvieron los requerimientos funcionales para la gestión de los Pagarés en Entidades Financieras Bancarias. Los requerimientos han sido agrupados con el propósito de facilitar la comprensión de la estructura definida.

Gestionar Pagaré.

- Registrar Pagaré.
- Actualizar Pagaré.
- Consultar Pagaré.
- Buscar Pagaré.
- Cobrar Comisión
- Mostrar Asientos contables
- Crear Asientos contables
- Extraer asientos contables
- Confirmar pago de Pagaré

2.2.1. Validación de los requerimientos funcionales

La validación de requisitos es considerada la actividad donde clientes y usuarios, con ayuda de los desarrolladores, revisan los requerimientos definidos para confirmar que realmente reflejan sus necesidades y que definen el producto deseado. El objetivo de la validación de requisitos es descubrir problemas en los requisitos identificados, antes de comprometer recursos a su implementación. Como consecuencia de esta actividad se suele producir una nueva iteración de adquisición de requisitos, debido a que conforme se perfila el sistema, suelen ir apareciendo nuevas necesidades hasta entonces ocultas, sobre todo cuando se utilizan prototipos. En el desarrollo de la ingeniería de requerimientos aplicada al módulo Pagaré, se utilizó la técnica de validación de prototipos orientados a clientes y usuarios. Esta técnica hace posible que el usuario tenga una idea clara del producto que va a recibir. Se tiene una mayor fluidez en la comunicación desarrollador cliente. El desarrollador porque es quién está desarrollando los prototipos y los usuarios porque de su aceptación o no, depende el desarrollo del producto que en un momento van a utilizar. En los anexos 8-11 se muestran las interfaces correspondientes a este epígrafe.

2.3. Fundamentación de la arquitectura

2.3.1. Capas lógicas definidas

- Capa de Presentación.
- Capa de Negocios.
- Capa de Acceso a Datos.
- Capa de Dominio.

Para ganar en organización en el desarrollo y en el despliegue del sistema, se agruparon los Módulos y Componentes por Subsistemas. Cada Subsistema tendrá uno o más Módulos y/o Componentes estrechamente relacionados con las funcionalidades que ejecutan. Los Módulos y/o Componentes estarán separados por diferentes capas lógicas según la naturaleza de los mismos.

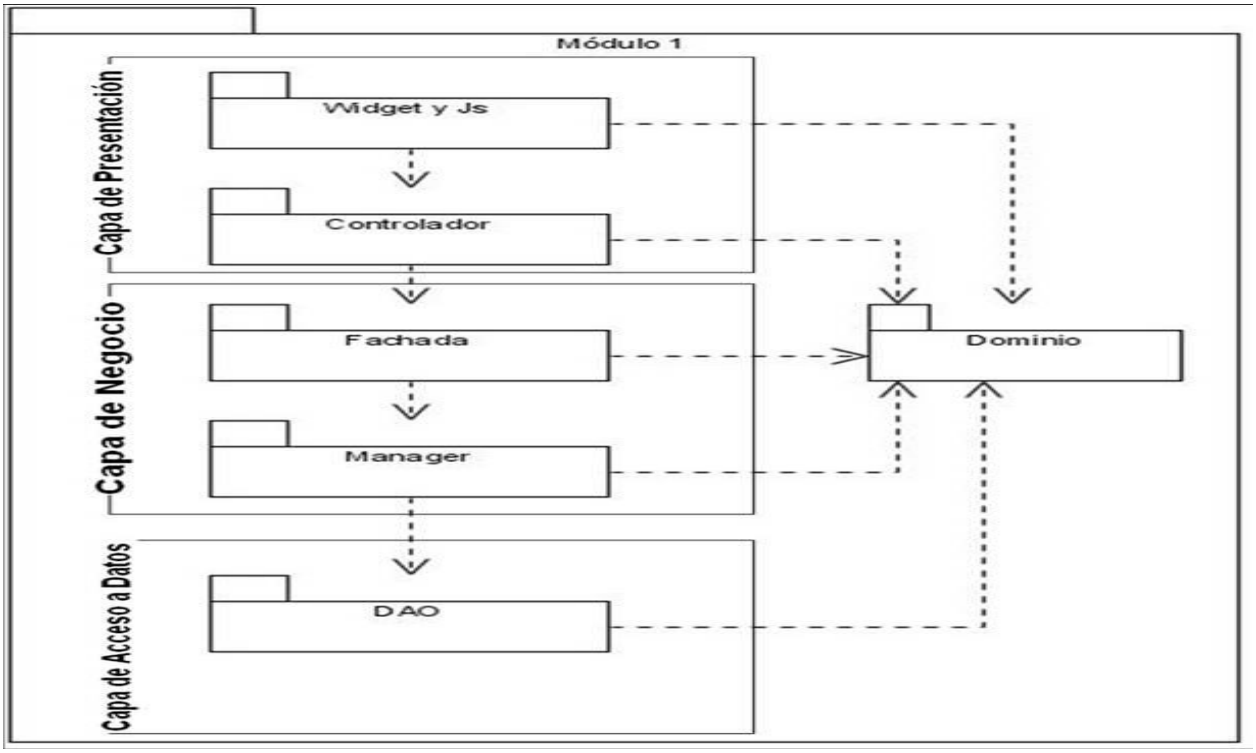


Ilustración 1: Diseño de las capas lógicas.

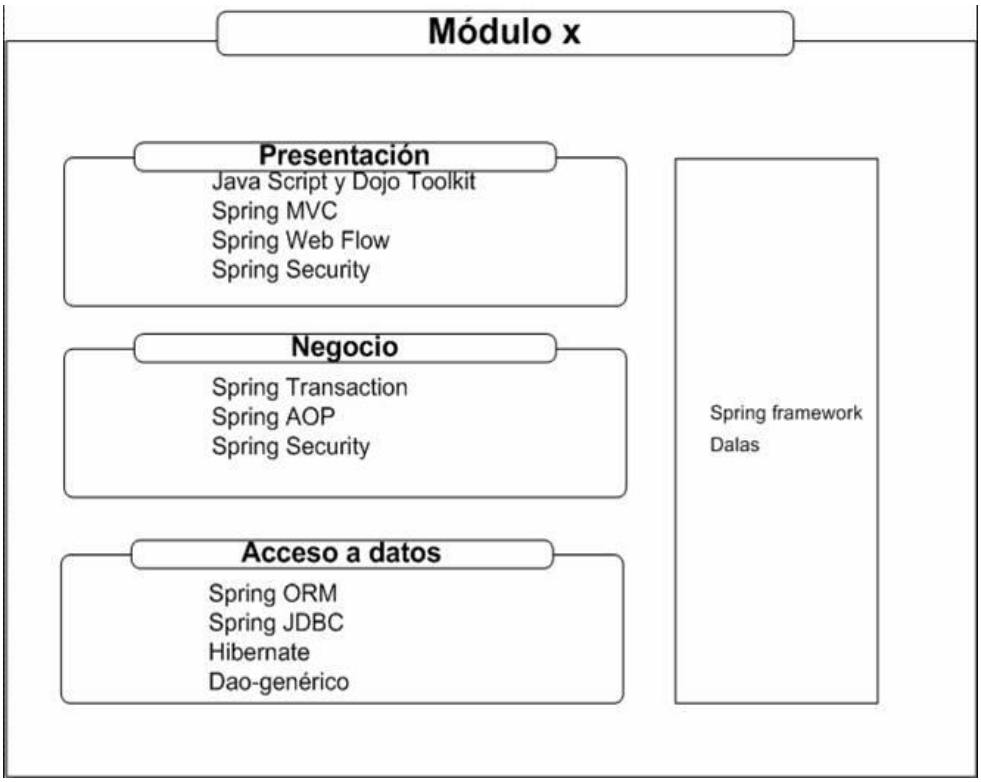


Ilustración 2: Relación entre las capas lógicas y los frameworks.

2.3.2. Capa de Presentación

Esta capa estará dividida en dos partes. Una subcapa del lado del servidor, encargada de recibir todos los pedidos de la interfaz de usuario, controlar el flujo de presentación del sistema y enviar las respuestas correspondientes a la interfaz de usuario. La otra subcapa estará en el cliente, utilizándose los componentes visuales de Java Script para manejar los eventos y validaciones del lado del cliente. La subcapa colocada en el lado del servidor estará relacionada con la capa de Negocios y de Dominio. [**Chaviano, 2009**]

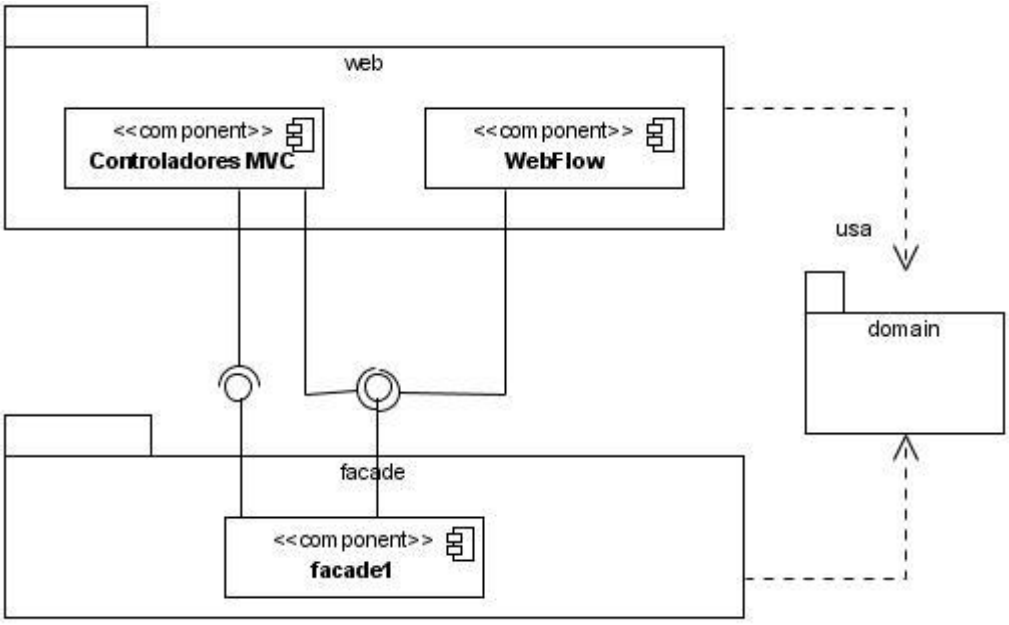


Ilustración 3: Estructura Capa de Presentación.

2.3.3. Capa de Negocio

La capa de negocio del SAGEB está compuesta por dos Capas. La subcapa Fachada que será el puente intermediario entre la capa de presentación y la del negocio, donde su función es agrupar las funcionalidades que serán invocadas desde la capa de presentación, sin realizar la lógica del negocio. Y la subcapa Manager donde se realizara la lógica del negocio conteniendo la jerarquía de clases indicadas para su implementación. Esta subcapa hace uso de la capa de acceso a datos para el trabajo con la persistencia de datos y de la capa de dominio para la generación de los objetos del dominio. La siguiente imagen muestra la estructura de la capa de negocio [**Chaviano, 2009**]. En la

figura 12 de los anexos se muestra la estructura de clases dentro del paquete manager y facade de la Capa de Negocio.

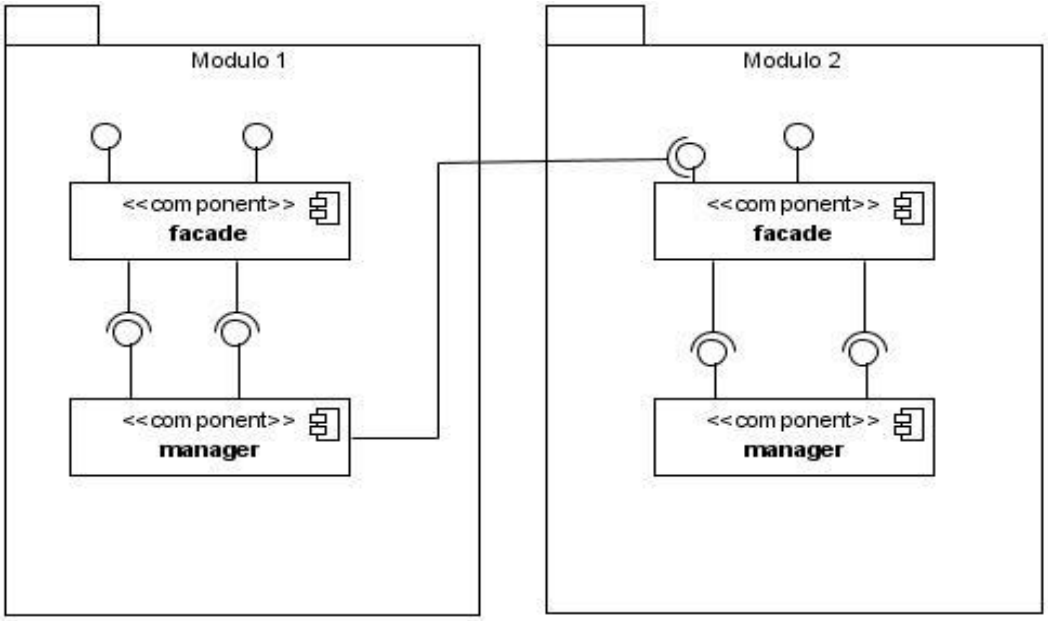


Ilustración 4: Estructura Capa de Negocio.

2.3.4. Capa de Acceso a Datos

En esta capa se implementarán los métodos encargados en interactuar con el gestor de Base de Datos. Esta capa tendrá solamente dependencia con la Capa de Dominio [**Chaviano, 2009**]. En la figura 13 de los anexos se muestra la estructura de los paquetes de clases en la Capa de Acceso a Datos.

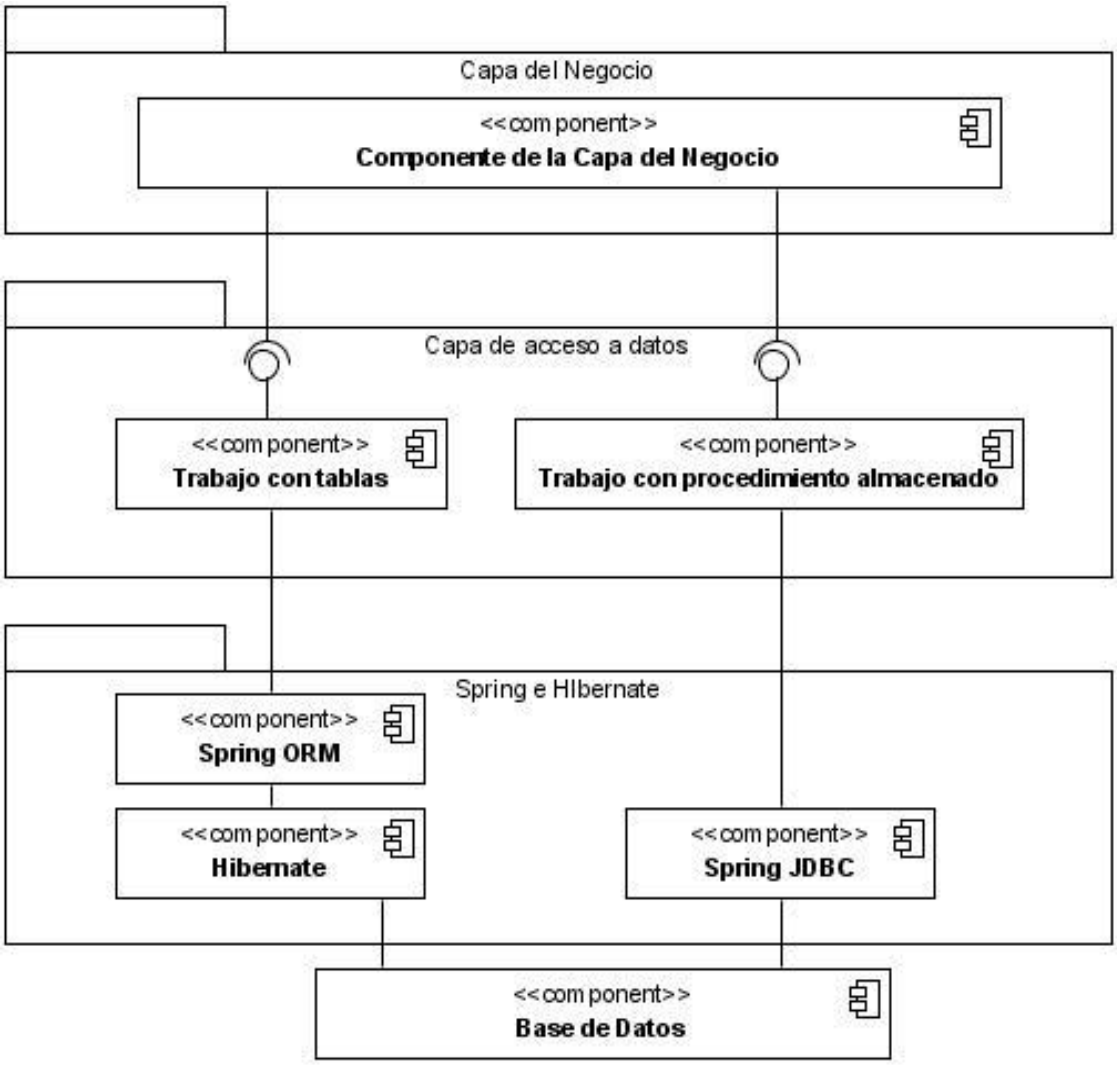


Ilustración 5: Estructura Capa de Acceso a datos.

2.3.5. Capa de Dominio

En esta capa se declararán todas las clases que representan entidades del negocio. Estas clases de dominio estarán presentes en todas las capas anteriormente descritas.[**Chaviano, 2009**]

2.4. Comunicación entre Módulos y Componentes

Las dependencias entre módulos y componentes están dadas por la necesidad de ejecutar en un módulo, funcionalidades que estén en otro módulo. El módulo que necesite una funcionalidad desarrollada en otro módulo, invocará dicha funcionalidad. A esta funcionalidades se les invocará a través de la capa de negocio del módulo. Por tanto, las dependencias entre módulos se realizarán a través de las capas del negocio.

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

Subsistema: Conjunto de módulos relacionados con los procesos que ejecutan.

Módulo: Conjunto de Casos de Uso relacionados con uno o más procesos bancarios estrechamente relacionados.

Componentes: Conjunto de funcionalidades comunes que serán reutilizados por el resto de los módulos del sistema. Estos componentes en algunas ocasiones se comportarán como módulos visuales en el sistema, y en otras ocasiones solamente recogerán funcionales del negocio.

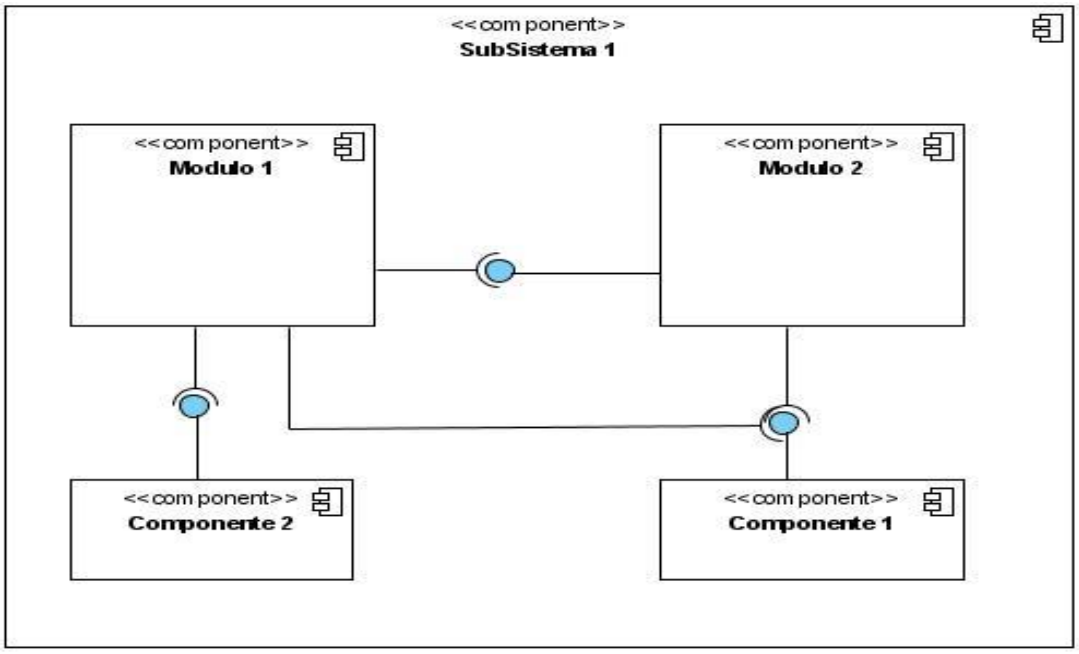


Ilustración 6: Estructura de un Subsistema.

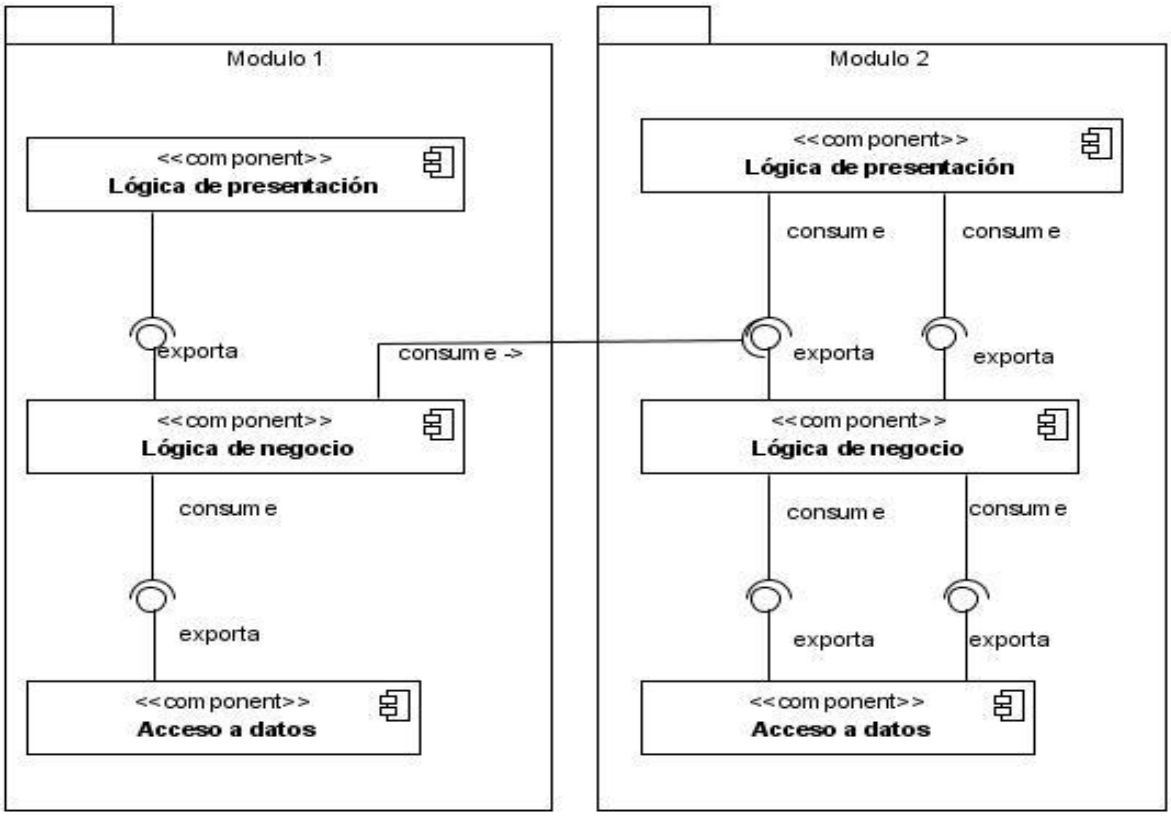


Ilustración 7: Comunicación entre módulos.

Nota: *Si el paquete facade del módulo 1 necesita funcionalidades que brinda el módulo 2, entonces la invocación de las funcionalidades del módulo 2 se hará mediante su paquete facade.*

2.5. Diseño. Aplicación de patrones de diseño

El proceso de diseño es un conjunto de pasos que permiten al diseñador describir todos los aspectos del sistema a construir. Constituye un enfoque de la ingeniería de software que modela un sistema como un grupo de objetos que interactúan entre sí. Representa un dominio en términos de conceptos clasificados de acuerdo a su dependencia funcional. [**Bello, 2009**]

Para mejorar la calidad del diseño fueron aplicados patrones de diseño durante la realización de los diagramas de interacción permitiendo asignar las responsabilidades a los objetos y diseñar la colaboración entre ellos. Los patrones de diseño no son más que la descripción de un problema y la solución del mismo, de forma que se puedan utilizar en diferentes contextos dando respuesta a interrogantes comunes. No es más que la solución efectiva que se le dio a un problema en un momento dado y puede ser reusable aplicándose en diferentes problemas de diseño en distintas circunstancias.

2.5.1. Patrones de asignación de Responsabilidades GRASP

En los patrones GRASP se codifican algunos de los principios, que se aplican al preparar los diagramas de interacción.

- Experto.

  La aplicación de este patrón permite a cada clase desarrollar las tareas que pueden realizar según la información que poseen.

- Creador.

  Permite crear instancias de otras clases en correspondencia con la responsabilidad dada. Con esto se logró conservar el encapsulamiento ya que los objetos logran valerse de su propia información para realizar lo que se les pide.

- Bajo acoplamiento.

  Este patrón soluciona el inconveniente de dar soporte a una dependencia escasa y a un aumento de la reutilización.

- Alta cohesión.

  Este patrón es utilizado para mantener la complejidad dentro de los límites manejables.

El diseño obtenido cumple con los patrones de Bajo acoplamiento y Alta cohesión permitiendo la colaboración entre los elementos del diseño, sin verse afectados la reutilización de los mismos y el entendimiento de estos cuando se encuentran aislados. La creación de clases controladores facilitó realizar las operaciones del sistema, debido a que estas operaciones reflejan los procesos de la empresa o dominio y no es factible manejarse en la capa de interfaz o presentación.

2.5.2. Patrones Estructurales

- Facade.

El propósito de utilizar este patrón es proveer de una interfaz unificada y sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas. Este patrón permite que una biblioteca de software sea más fácil de usar y entender. Esto es posible porque el facade implementa métodos convenientes para tareas comunes, puede reducir la dependencia de código externo en los trabajos internos, permitiendo así más flexibilidad en el desarrollo de sistemas.

2.5.3. Patrones de Comportamiento

- Command.

El objetivo de utilizar este patrón es tener parametrizados los objetos por las acciones que realizan. Este patrón permite especificar, administrar y ejecutar solicitudes en tiempos distintos.

El objeto Command puede guardar un estado que permita deshacer la ejecución del comando. Soporta la capacidad de generar bitácoras que permitan la recuperación del estado en caso de que el sistema falle. Facilita la estructuración un sistema en torno a operaciones de alto nivel construidas con base en operaciones primitivas o de bajo nivel. Un comando nos desliga el objeto invocador del objeto receptor, en otras palabras, independiza la parte de la aplicación que invoca la acción de la implementación de las mismas. Permite que las acciones sean objetos de primera clase, y se puedan agrupar comandos de uso frecuente en comandos compuestos.

2.5.4. Patrón de acceso a datos

- DAO

La utilización de este patrón permitirá acceder a la fuente de datos y encapsular los objetos clientes, ocultando tanto la fuente como el modo de acceder a ella. Los DAOs deben implementar los métodos del interface (InterfaceDAO) que declaran. Pero además pueden implementar otros métodos que no están en el interfaz. DAO permite el acceso a reglas de validación, esto es posible porque tiene capacidad de especificar relaciones entre tablas.

2.5.5. Patrones de Presentación

- Composite View (Vistas Compuestas)

  Un objeto vista que está compuesto de otros objetos vista. Por ejemplo, una página JSP que incluye otras páginas JSP y HTML usando la directiva include.

- Modelo Vista Controlador

  Este patrón propone dividir la aplicación en tres capas el Modelo, la Vista y el Controlador.

El modelo es la representación del dominio o datos del sistema, la vista se encarga de presentar la interfaz al usuario, en sistemas web, esto es típicamente HTML, aunque pueden existir otro tipo de formatos. En la vista sólo se deben de hacer operaciones simples y el controlador es el encargado de escuchar los cambios en la vista y enviarlos al modelo, el cual le regresa los datos a la vista como un ciclo. Cuando se aplica este patrón los accesos a la base de datos se hacen en el modelo, la vista y el

controlador no deben de saber si se usa o no una base de datos. El controlador es el que decide que vista se debe de imprimir y que información es la que se envía.

2.6. Diagrama de paquetes

Organizar por criterios en paquetes las clases y ficheros de un módulo ayuda a la fácil comprensión de la aplicación, en el caso de Pagaré, las mismas se agruparon según la función que cumplen. En breve se mencionan las clases que componen cada paquete.

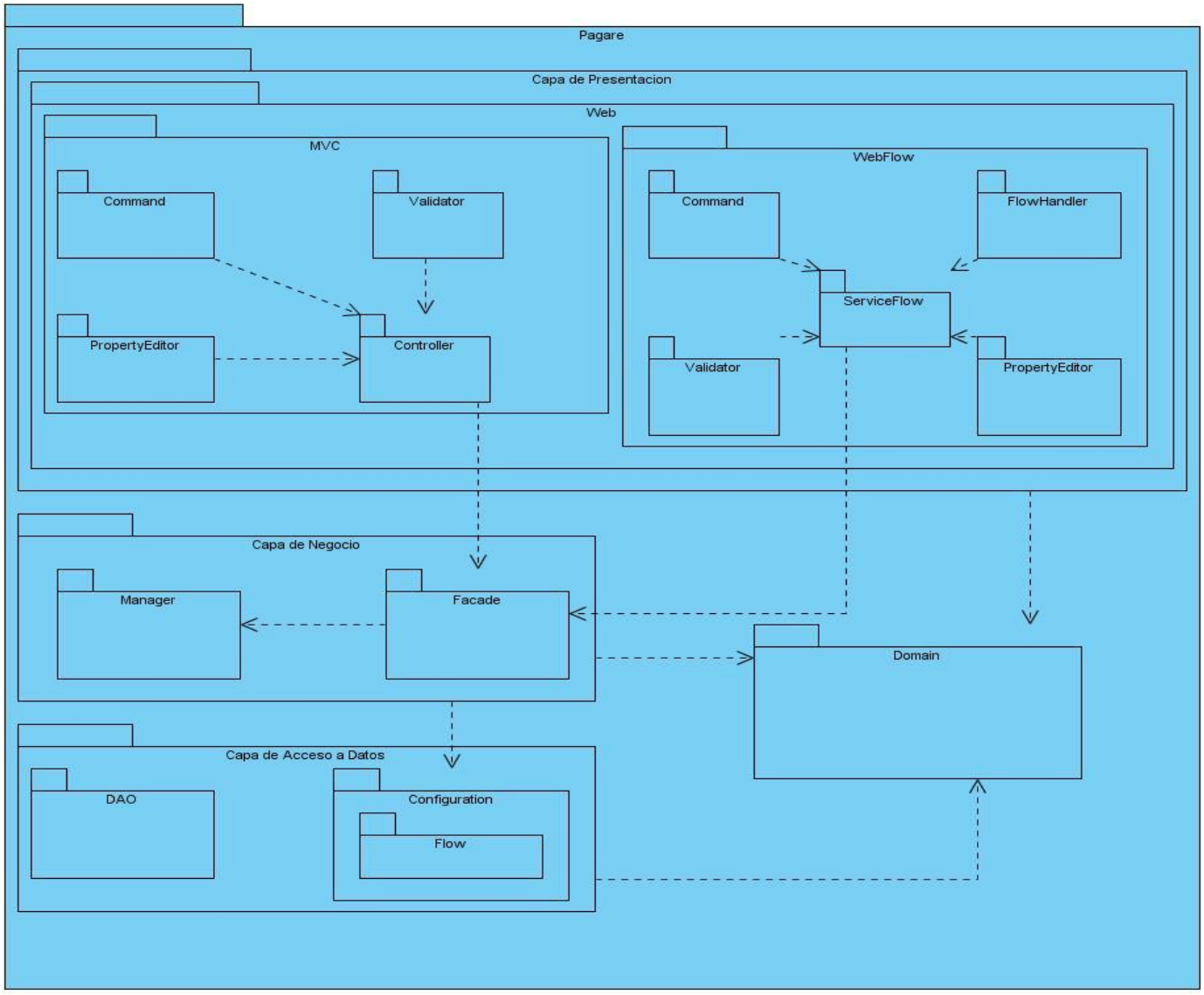


Ilustración 8: Diagrama de paquetes del módulo Pagaré.

Paquete Configuration: En este paquete estarán los ficheros XML de configuración de los diferentes contextos de Spring:

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

Paquete Flow: En este paquete estarán los flujos .xml del módulo correspondiente.

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

Paquete Manager: En este paquete se encontrarán las interfaces e implementaciones del negocio del módulo correspondiente.

Paquete Facade: En este paquete se encontrarán las interfaces y las implementaciones de las funcionalidades que se le brindaran a la presentación.

Paquete Domain: En este paquete estarán las clases del dominio del módulo.

Paquete Web: Este paquete será el contenedor de los paquetes mvc y webFlow.

Paquete MVC: En este estarán los paquetes que contendrán la lógica de presentación en el servidor para SpringMVC.

Paquete Commad: Contiene clases que representan objetos a manipular en los formularios.

Paquete Validator: Contiene las clases encargadas de validar los datos.

Paquete PropertyEditor: Clases para convertir objetos.

Paquete Controller: Contiene las diferentes clases que heredan de los controladores de Spring.

Paquete WebFlow: En este paquete estarán los paquetes que contendrán la lógica de presentación en el servidor para SpringWebFlow.

Paquete ServiceFlow: Contiene las diferentes clases que median entre el flujo y la fachada.

2.7. Diagramas de clases del 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 y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Además, el modelo de diseño sirve de abstracción de la implementación del sistema y es, de ese modo, utilizado como una entrada fundamental de las actividades de implementación. [**IVAR JACOBSON, 1999**]

A partir de los patrones de diseño explicados anteriormente se define el siguiente modelo de diseño para el módulo Pagaré.

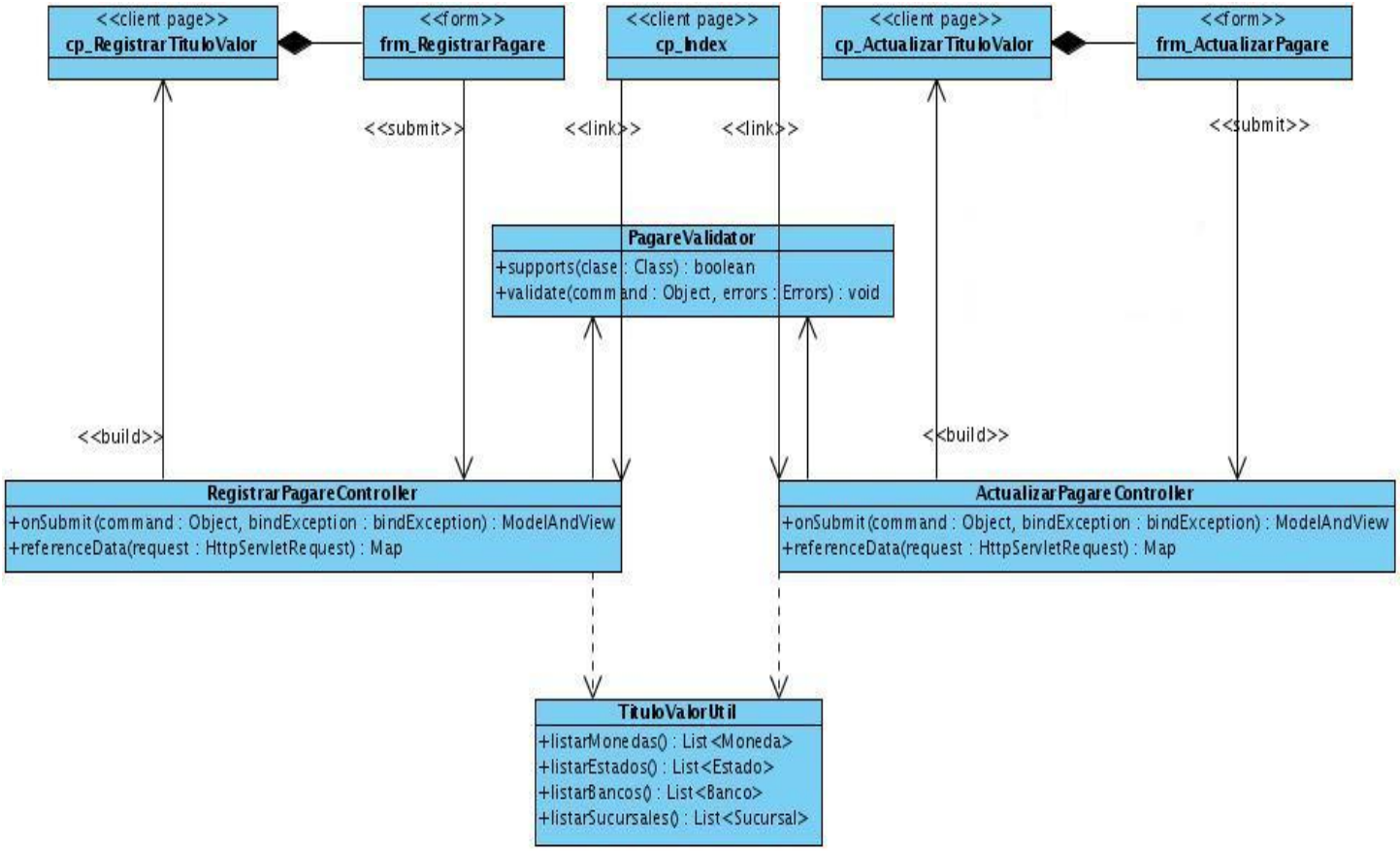


Ilustración 9: Diseño de la Capa de Presentación del módulo Pagaré.

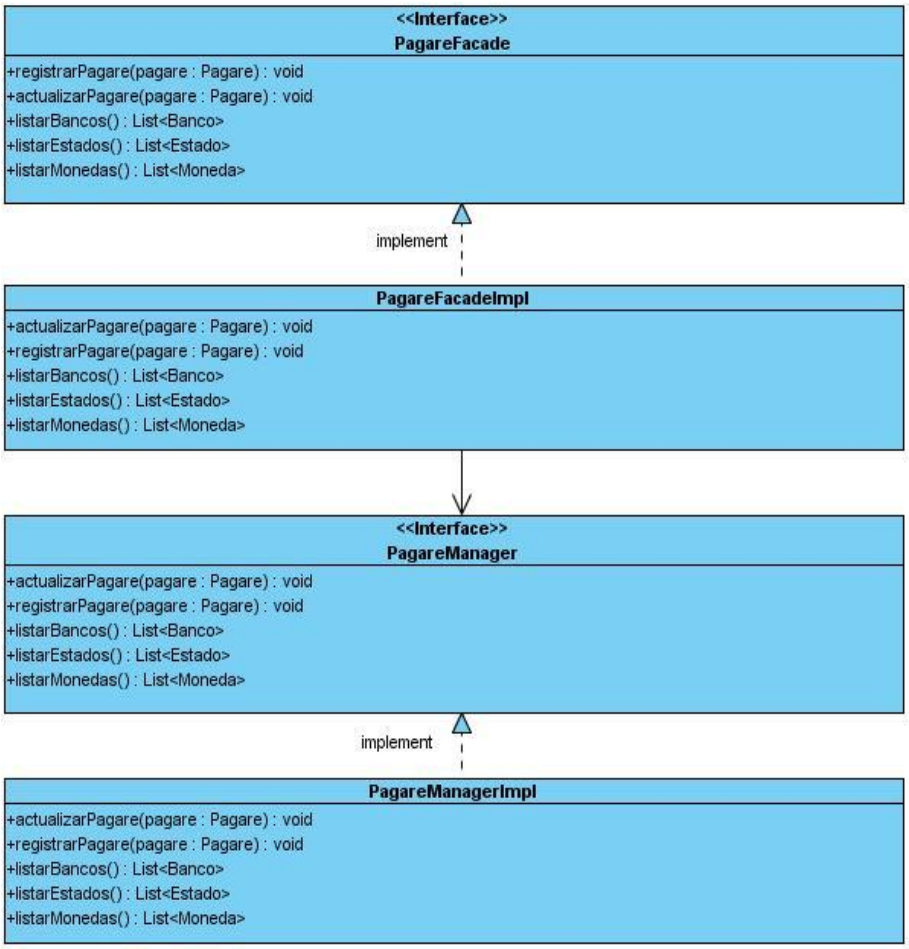


Ilustración 10: Diseño de la Capa de Negocio del módulo Pagaré.

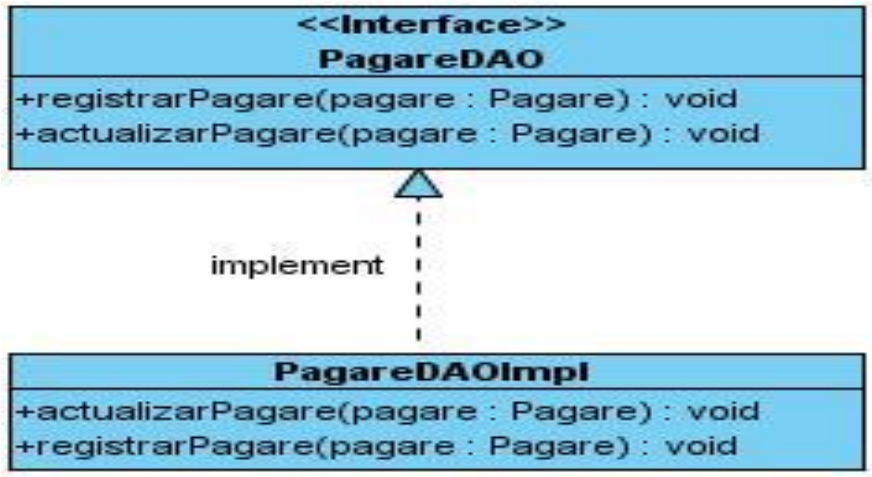


Ilustración 11: Diseño de la Capa de Acceso a Datos del módulo Pagaré.

2.8. Diagrama de interacción del diseño

En el flujo de diseño se utiliza fundamentalmente el diagrama de secuencia. En él se incluyen las interacciones entre las clases del diseño, a través de mensajes. Un mensaje significa una operación en la clase a la que va el mensaje.

A continuación los diagramas de interacción de las clases del diseño.

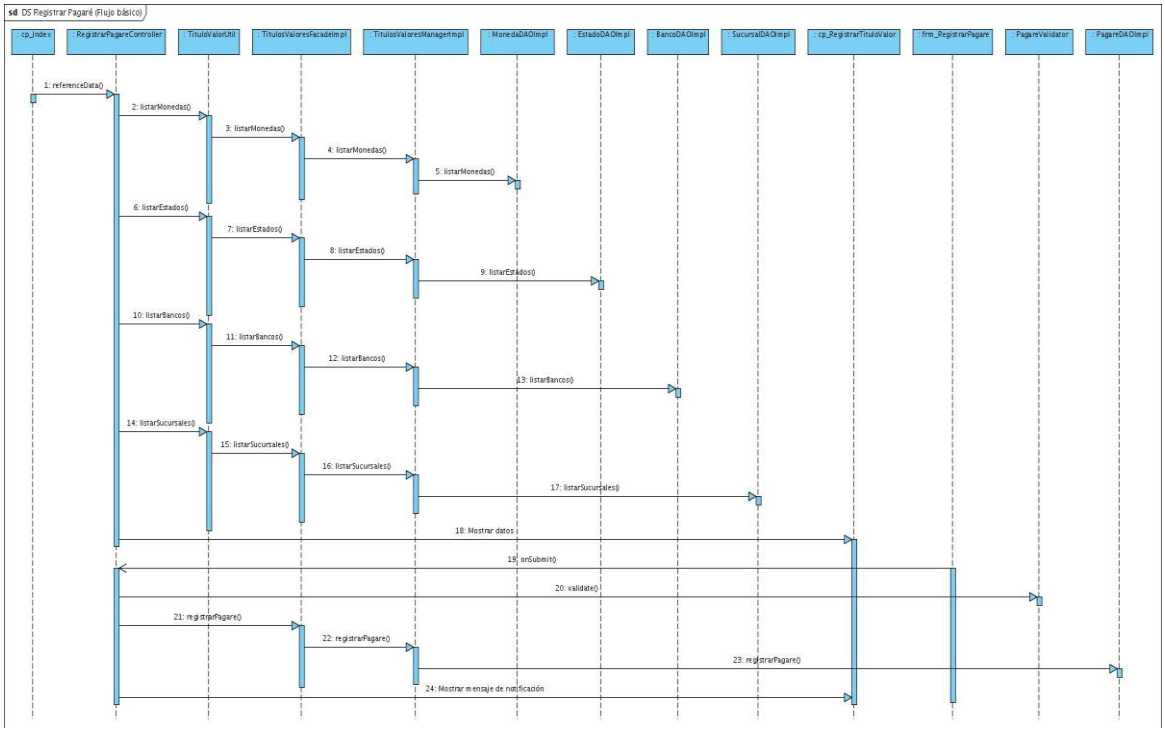


Ilustración 12: DS Registrar Pagaré (Flujo básico).

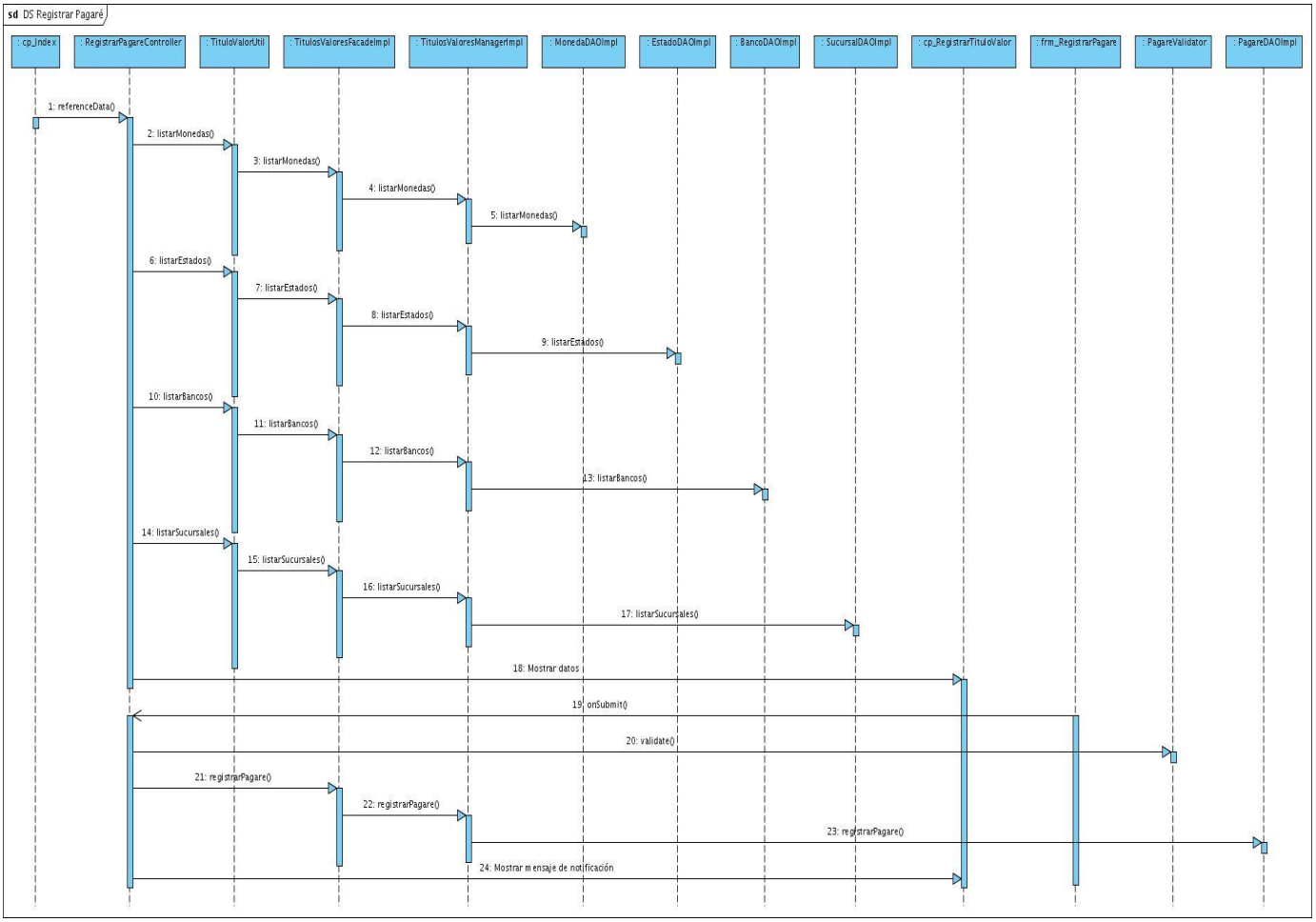


Ilustración 13: DS Registrar Pagaré (Flujo alterno)

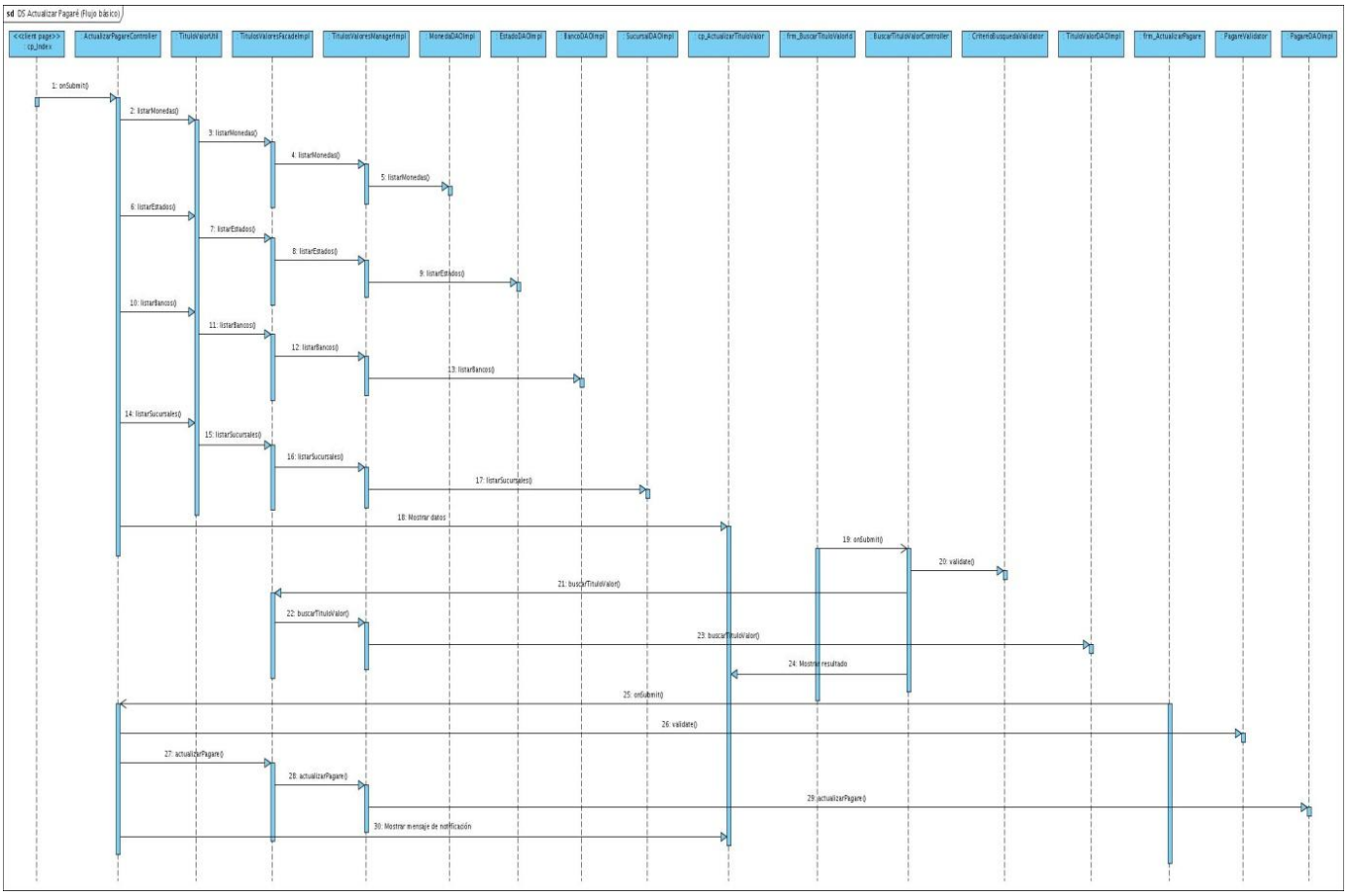


Ilustración 14: DS Actualizar Pagaré (Flujo básico)

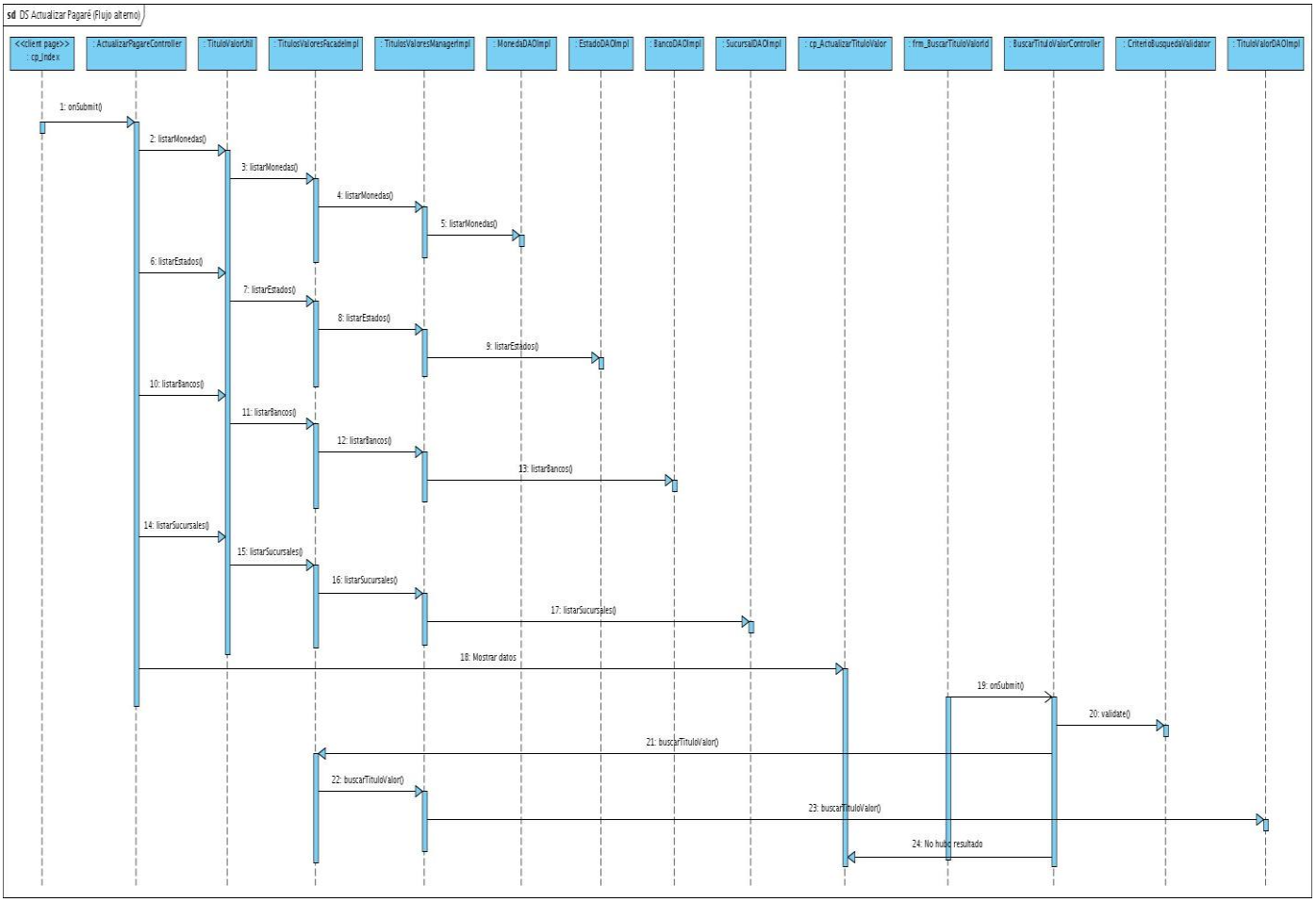


Ilustración 15: DS Actualizar Pagaré (Flujo alterno)

2.9. Conclusiones del capítulo

La panorámica de la arquitectura del proyecto SAGEB dada en este capítulo y el diseño del módulo Pagaré realizado en el mismo, facilitó:

- La comprensión de la estructura de paquetes existentes en el módulo.
- Un mejor entendimiento del diseño del módulo lo que implica una uniformidad y organización a la hora de implementar la solución propuesta.

CAPITULO # 3. IMPLEMENTACIÓN DEL SISTEMA

3.1. Introducción

En el último capítulo del presente trabajo de diploma se explica y se presenta la implementación del módulo Pagaré. Resulta de gran importancia ya que luego de esta fase se obtiene la aplicación funcional.

Se presenta el diagrama de componentes correspondiente al módulo. Además se muestran cuales son las filosofías de programación adoptadas para llegar a la solución, se exponen aspectos importantes de la implementación, así como los métodos más relevantes.

3.2. Diagrama de componentes del sistema

Este diagrama representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre los mismos. Los componentes físicos incluyen archivos, cabeceras, librerías compartidas, módulos, ejecutables, o paquetes.

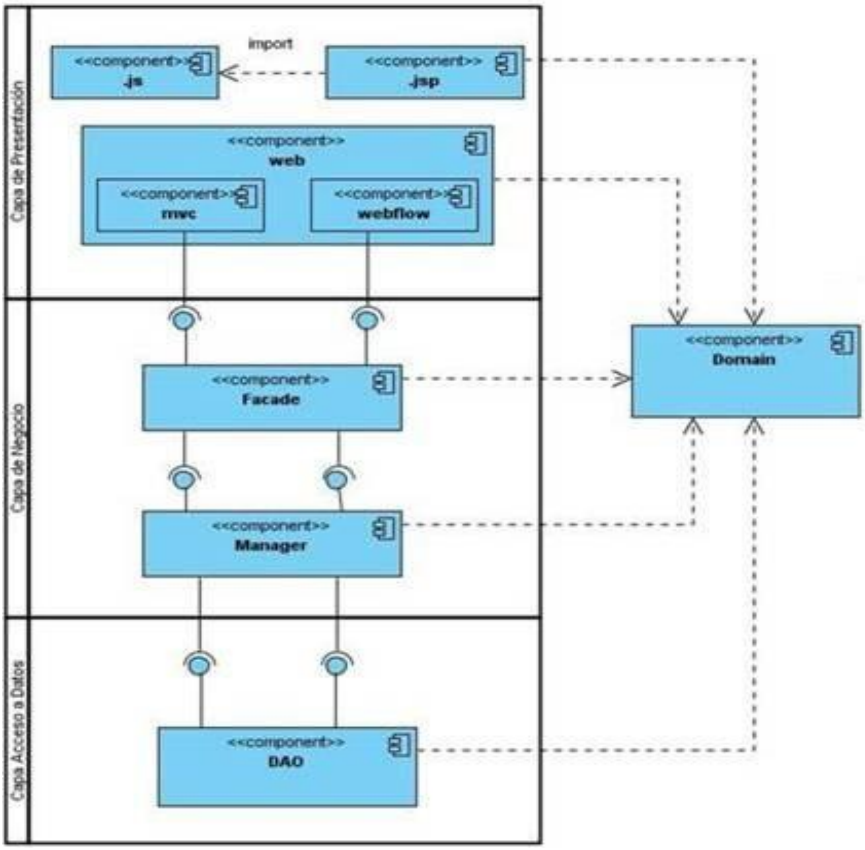


Ilustración 16: Diagrama de componentes del módulo Pagaré.

3.3. Diagrama de estado

Con la realización del diagrama de estado para los Pagarés, se muestran los diferentes estados por los que pasa el objeto durante toda su vida. Su principal función es mostrar los eventos que hacen que se pase de un estado a otro, así como las respuestas y acciones que genera.

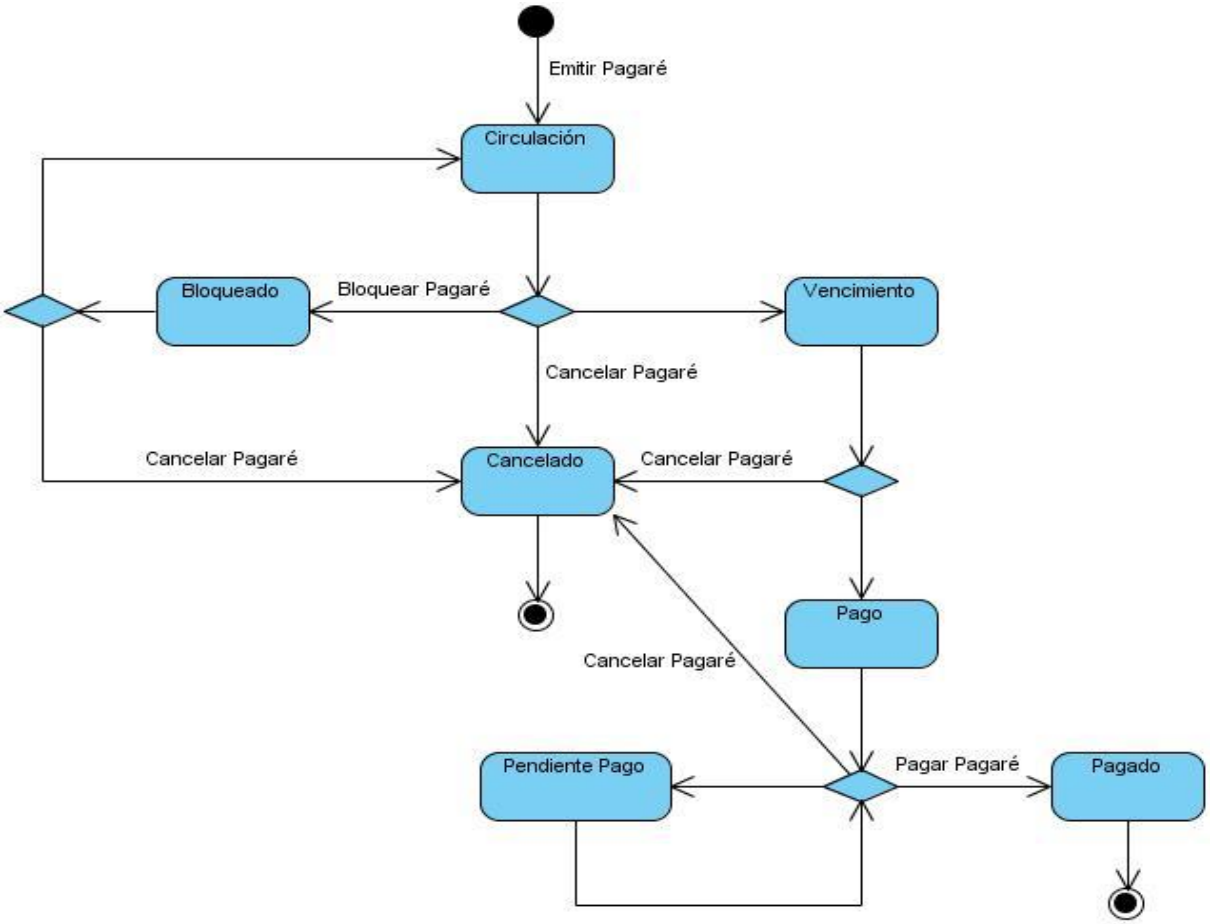


Ilustración 16: Diagrama de componentes del módulo Pagaré

3.4. Aspectos significativos de la Implementación. Utilización del framework Spring MVC

Con la utilización del framework Spring MVC, se resuelve el problema de atender múltiples peticiones. Usando la clase MultiActionController, pueden atenderse varias peticiones relacionadas con el negocio.

Primeramente se selecciona en el menudata, la opción “Buscar”, como se muestra a continuación:

```
id: "m53",
 label: "Buscar",
 labelNodeClass: "Level2",
_showBullet: true,
doAction: "true",
action: "finixubnc.core.menuRequest({url:'../pagare/buscar.htm?entidad='+ dojo.i18n.getLocalization('titulosv
```

La misma hace una referencia a la clase "`BuscarMultiActionController`", que a su vez ejecuta un método, el cual será el encargado de levantar la vista. Dicho método se muestra a continuación:

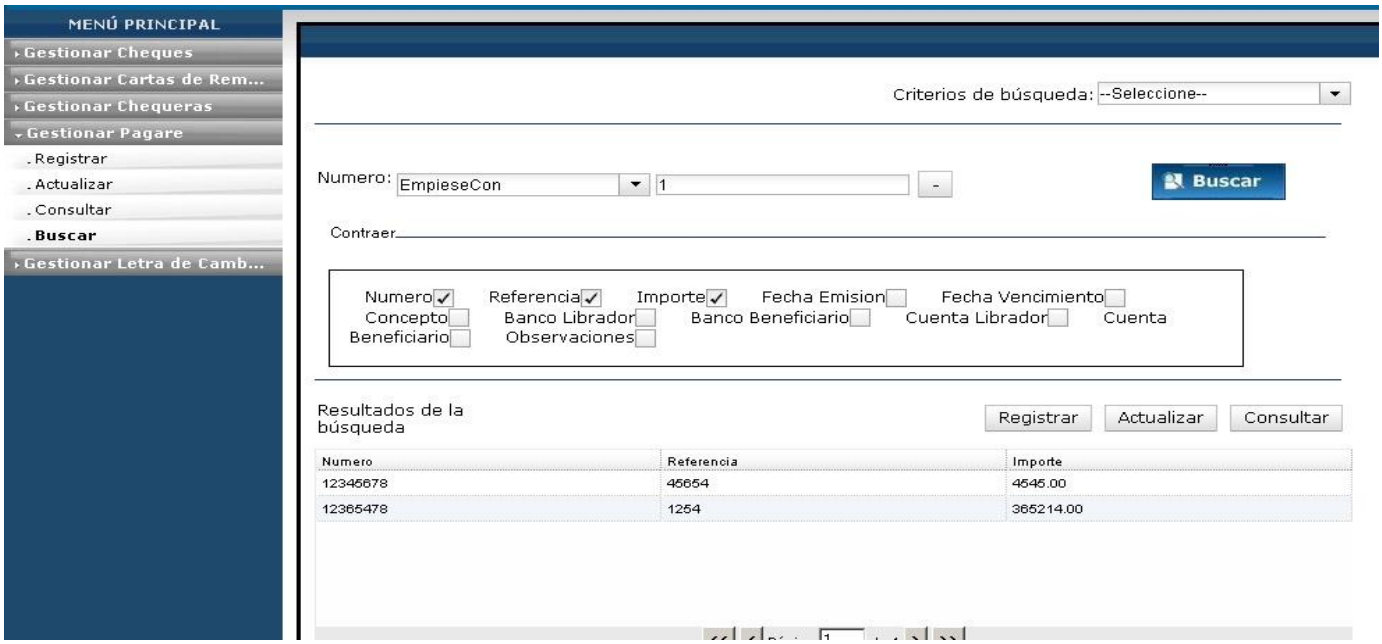
```
public ModelAndView resolverVista(HttpServletRequest request, HttpServletResponse response) {
    String modulo[] = request.getRequestURI().split("/");

    String entidad = (String) request.getParameter("entidad");
    String[] entiAux = entidad.replace('.', '~').split("~");
    String simpleName = entiAux[entiAux.length - 1];
    String view = "buscar" + simpleName;

    request.getSession().setAttribute("entidad", entidad);
    // JSONObject jsonCriterios = new JSONObject();
    Map<String, String> model = buscadorFacade.prepararDatos(modulo[modulo.length - 2], simpleName);
    return new ModelAndView(view, model);

}
```

Seguidamente al usuario se le muestra la vista "Buscador Dinámico", donde el mismo elegirá el criterio de búsqueda por el cual desea buscar el titulo valor



Al seleccionarse el criterio de búsqueda y ejecutar la misma, se hace una llamada al método mostrado a continuación.

```
public JSONObject crearJSonResultados(HttpServletRequest request, String urlentidad,
        List<FiltroValor> listCriterioFijo) {
    List<CriterioBusqueda> criterios =
        buscadorFacade.buscarCriterios(urlentidad.substring(urlentidad.lastIndexOf(".") + 1));
```

El mismo se encargara de recoger el criterio de búsqueda e insertarlo en un JSON, donde se almacenaran todos los datos, que finalmente le serán mostrados al usuario, de la manera que se muestra a continuación:

```
JSONObject elementos = new JSONObject();
JSONObject json = new JSONObject();
JSONArray array = new JSONArray();
```

3.5. Descripción de la clases y funcionalidades del sistema

Nombre: BuscarMultiActionController	
Tipo de clase: Controladora	
Atributo	**Tipo**
buscadorFacade	BuscadorFacade
Para cada responsabilidad:	
Nombre:	**Descripción:**
Model and View resolverVista(HTTPServletRequest request,HTTPServletResponse response)	Se encarga de levantar la vista que se le mostrara al usuario.
buscarDatos(HTTPServletRequest request,HTTPServletResponse response)	Obtiene los datos de búsqueda según el criterio seleccionado.
buscarDatosCriteriosEsta(HTTPServletRequest request,HTTPServletResponse response,Lista<FiltroValor>list)	
buscarDatosGenericos(HTTPServletRequest request,HTTPServletResponse response,String urlentidad, List<FiltroValor>list)	Obtiene los datos de búsqueda según el criterio seleccionado para Sprig WebFlow.

JSONObject crearJSONResultados(HTTPServletRequest request,HTTPServletResponse response, String urlentidad, List<FiltroValor>list CriterioFijo)	Este método crea el Json resultado que será enviado al cliente De ser usado tiene que llamarse luego a: ResponseUtil.escribirDatosEnElResponse(response, json) pasandole json retornado
CriterioBusqueda buscarCriterio(int id, List<CriterioBusqueda>buscar criterios)	Va comparando los criterios de búsqueda hasta encontrar el seleccionado

Nombre:Pagare	
Tipo de clase: Modelo	
Atributo	**Tipo**
numero	String
referencia	String
importe	Bigdecimal
tipoMoneda	nomMoneda
estado	*nomMonedaLetradecambioPagare*
fechaEmision	Date
fechaVencimiento	Date
concepto	String
bancoGirador	Banco
bancoBeneficiario	Banco
cuentaGirador	String
observaciones	String
cuentaBeneficiario	String

3.6. Flujos y beans establecidos para el trabajo con WebFlow

Para la utilización de Spring Web Flow en un módulo determinado se debe crear una carpeta flow y colocarla dentro de la carpeta Configuration. Dentro de la carpeta flow se colocarán todos los flujos (ficheros xml). Los bean o clases relacionados con Spring WebFlow se deben declarar en el fichero xml que cumplan con el formato siguiente según en el módulo: *finixubnc-subsistemaX-moduloX-webflow.xml.* Existen algunos beans o clases que se deben declarar siempre que se vaya a utilizar SpringWebFlow. Estos son:

```
<webflow:flow-builder-services id="flowBuilderServices"
            view-factory-creator="mvcViewFactoryCreator"
            conversion-service="conversionService" />

<bean id="mvcViewFactoryCreator"
      class="org.springframework.webflow.mvc.builder.MvcViewFactoryC
                                    reator">
      <property name="viewResolvers" ref="viewResolver" />
</bean>


<bean id="conversionService"

      class="cu.uci.finixubnc.common.global.web.webFlow.conversionServices.
FinixuConversionService">
            <property name="listaConverters">
                  <list>
                        <ref bean="" />
                  </list>
            </property>
            <property name="mapConverters">
                  <map>
                        <entry key="stringToHora" value-ref="stringToHora"
/>
                        <entry key="stringToDateJDeportivo"
                              value-ref="stringToDateJDeportivo" />
                  </map>
            </property>
</bean>
```

```
<webflow:flow-builder-services id="flowBuilderServices"
            view-factory-creator="mvcViewFactoryCreator"
                 conversion-service="conversionService" />

<bean id="mvcViewFactoryCreator"
            class="org.springframework.webflow.mvc.builder.MvcViewFactoryC
reator">
      <property name="viewResolvers" ref="viewResolver" />
</bean>


<bean id="conversionService"

      class="cu.uci.finixubnc.common.global.web.webFlow.conversionServices.
FinixuConversionService">
            <property name="listaConverters">
                  <list>
                        <ref bean="" />
                  </list>
            </property>
            <property name="mapConverters">
                  <map>
                        <entry key="stringToHora" value-ref="stringToHora"
/>
                        <entry key="stringToDateJDeportivo"
                              value-ref="stringToDateJDeportivo" />
                  </map>
            </property>
</bean>
```

3.6.1. Detalles de los bean o clases que se deben declarar

La dirección relativa de los flujos de Spring WebFlow es algo que debe indicarse. Debe ser consecuente con el módulo que corresponda. En este caso se instanciarán todos los flujos que estén en el subsistema common, en el módulo global, ubicado dentro de la carpeta configuration/flow.

```
<webflow:flow-location-pattern
value="classpath:cu/uci/finixubnc/common/global/configuration/flow/*-flow.xml" />
```

En el caso que se utilice algunas clases convertidoras de string a objeto y viceversa se deben declarar en el fichero xml correspondiente al trabajo con Spring Webflow y agregar los nombres de esos beans en la propiedad “listaConverters” del bean “conversionService”. En caso de personalizar algunos de los convertidores que posee Spring Web Flow se declaran estos en la propiedad “mapConverters” del “conversionService”.

```
<bean id="conversionService"
class="cu.uci.finixubnc.common.global.web.webFlow.conversionServices.FinixuConversionService">
```

3.7. Conclusiones del capítulo

En este capítulo se describieron las principales clases para facilitar la comprensión de la implementación realizada. Se explicaron los componentes que fueron reutilizados para el desarrollo de la aplicación además de dar una breve explicación de algunos aspectos importantes de la implementación.

CONCLUSIONES

Gracias al desarrollo del presente trabajo de diploma, donde está incluida una valoración de los sistemas de gestión de Pagarés a nivel nacional e internacional y la fundamentación de la metodología, herramientas, lenguajes y tecnologías de desarrollo de software propuestas por el proyecto SAGEB, puede concluirse que:

- Con la implementación realizada se logró un mayor entendimiento de los proceso de negocio de Pagarés en entidades financieras bancarias del Sistema Bancario Nacional y conocimientos de los procesos de gestión de los mismos a nivel internacional.

- Se implementaron los requerimientos funcionales definidos previamente, apoyándose en el uso de herramientas y tecnologías definidas por el proyecto SAGEB.

El objetivo general propuesto para este trabajo fue plenamente cumplido a través de las actividades realizadas. Durante estas actividades se generaron los artefactos relacionados con el proceso de desarrollo de software como por ejemplo el Modelo de diseño.

RECOMENDACIONES

Se recomienda la realización de nuevas versiones, basadas en los nuevos requerimientos que surjan por necesidades del cliente para ampliar las funcionalidades de la solución propuesta.

BIBLIOGRAFÍA CONSULTADA

R. URIA, 1998: Derecho Mercantil, R. URIA, Madrid, 1.998.

M. BROSETA PONT, 1994: Manual de Derecho Mercantil, M. BROSETA PONT, Madrid,1.994.

Affón: Affón Bravo Jimmy, Universidad privada Antenor Orrego – Facultad de Administración.

REFERENCIAS BIBLIOGRÁFICAS

J. GARRIGUES, 1994: Curso de Derecho Mercantil, J. GARRIGUES, Madrid, 1.994.

Fuentes, 2009: Fuentes, Alien Fernandez. *Implementación del Módulo Inventario Físico del Subsistema Inventario del Sistema de Gestión Integral Cedrux.*

Bello, 2009: Yesenia Perdomo. *Análisis y Diseño del Subsistema Títulos Valores del Proyecto Modernización del Sistema Bancario.*

Chaviano, 2009: Chaviano, Adolfo Miguel Iglesias. *Documento de arquitectura del proyecto SAGEB.*

DynamicJasper.com: [Online] http://dynamicjasper.sourceforge.net/index.html.

Definición.de. 2008: Definición de Pagaré y concepto » Definición ABC,. [Online] http://definicion.de/pagare/.

Javahispano.org. 2002-2007: Dynamic Jasper. [Online] http://www.javahispano.org/contenidos/es/agrega_dinamismo_a_los_reportes_en_jasperreports_con_dynamicjasper_11/.

IGP SQLServer, 2007: Información General del Producto SQLServer 2005 [En línea]

boxsoftware.net: Easy Pagarés TRV v4.25. [Online] http://www.boxsoftware.net/programas/easy_pagar%C3%A9s_trv_v4.25_29752/easy_pagar%C3%A9s_trv_v4.25_29752.asp.

abcdatos.com: Emisión de pagarés BancoPopular. [Online] http://www.abcdatos.com/programas/programa/z8954.html.

ciberconta.unizar.es: Los Títulos Valores: Letra de Cambio, Cheque y Pagaré. [Online] http://www.ciberconta.unizar.es/leccion/der021/100.HTM.

pagares-y-gestion-de-cobros.softonic.com: Pagarés y Gestión de Cobros Pro 13.0. [Online] http://pagares-y-gestion-de-cobros.softonic.com/.

estudios-economicos-cubanos.org: [Online] http://www.estudios-economicos-cubanos.org/content/download/277/1452/version/1/file/Estructura+y+Funciones+del+Sistema+Financiero+en+Cuba.pdf.

BNC. 2007: Manual de Instrucciones y procedimientos del Banco Nacional de Cuba. 2007.

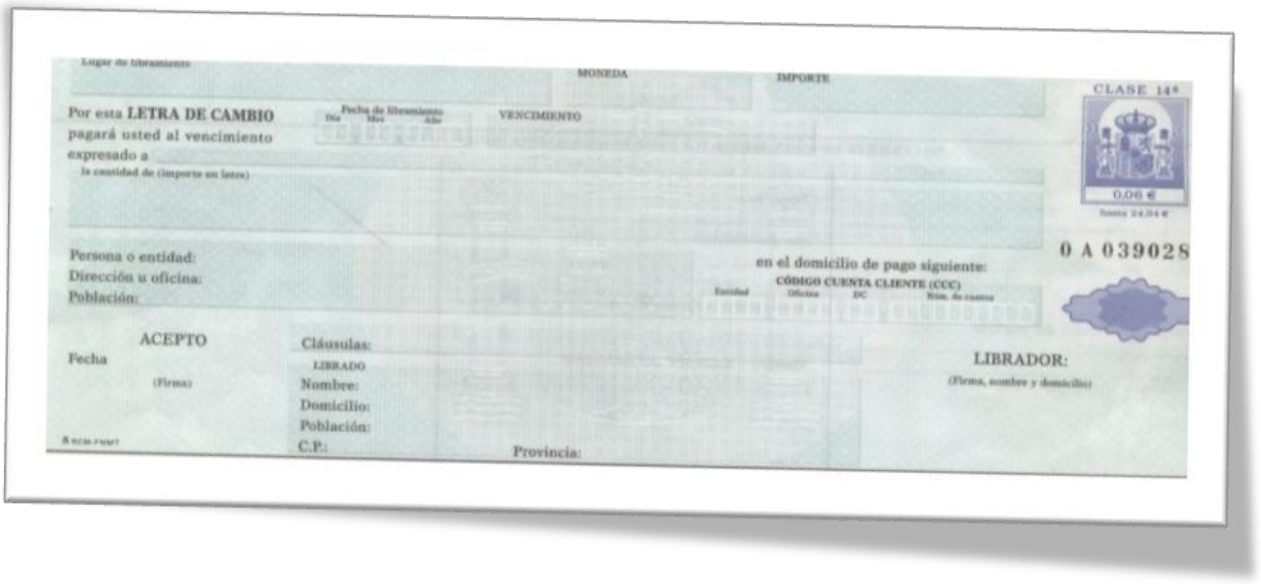
.

ANEXOS

Anexo 1: Pagaré a la orden

Anexo 2: Pagaré no a la orden

Anexo 3: Letra de Cambio

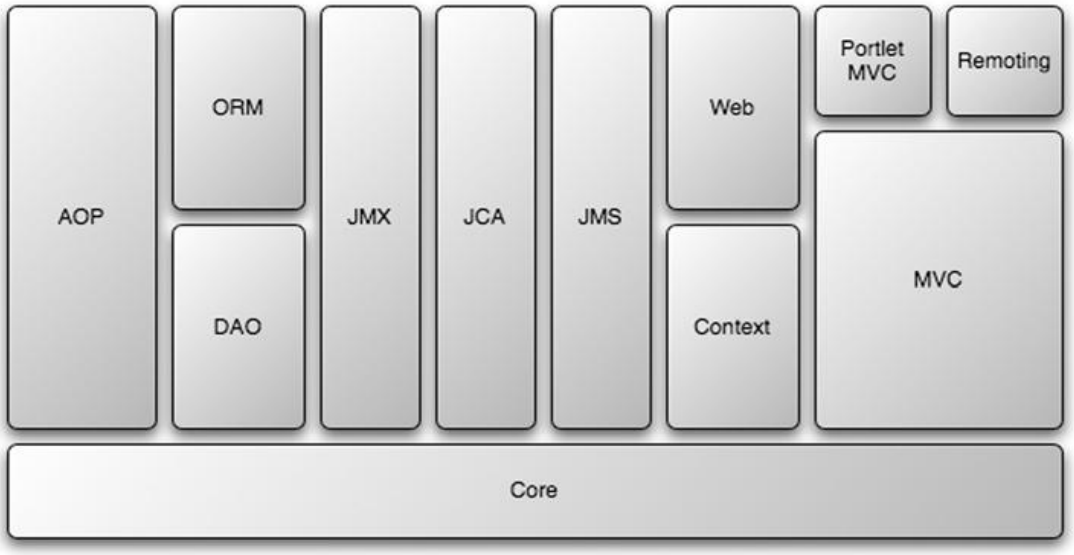


Anexo 4: Cheque

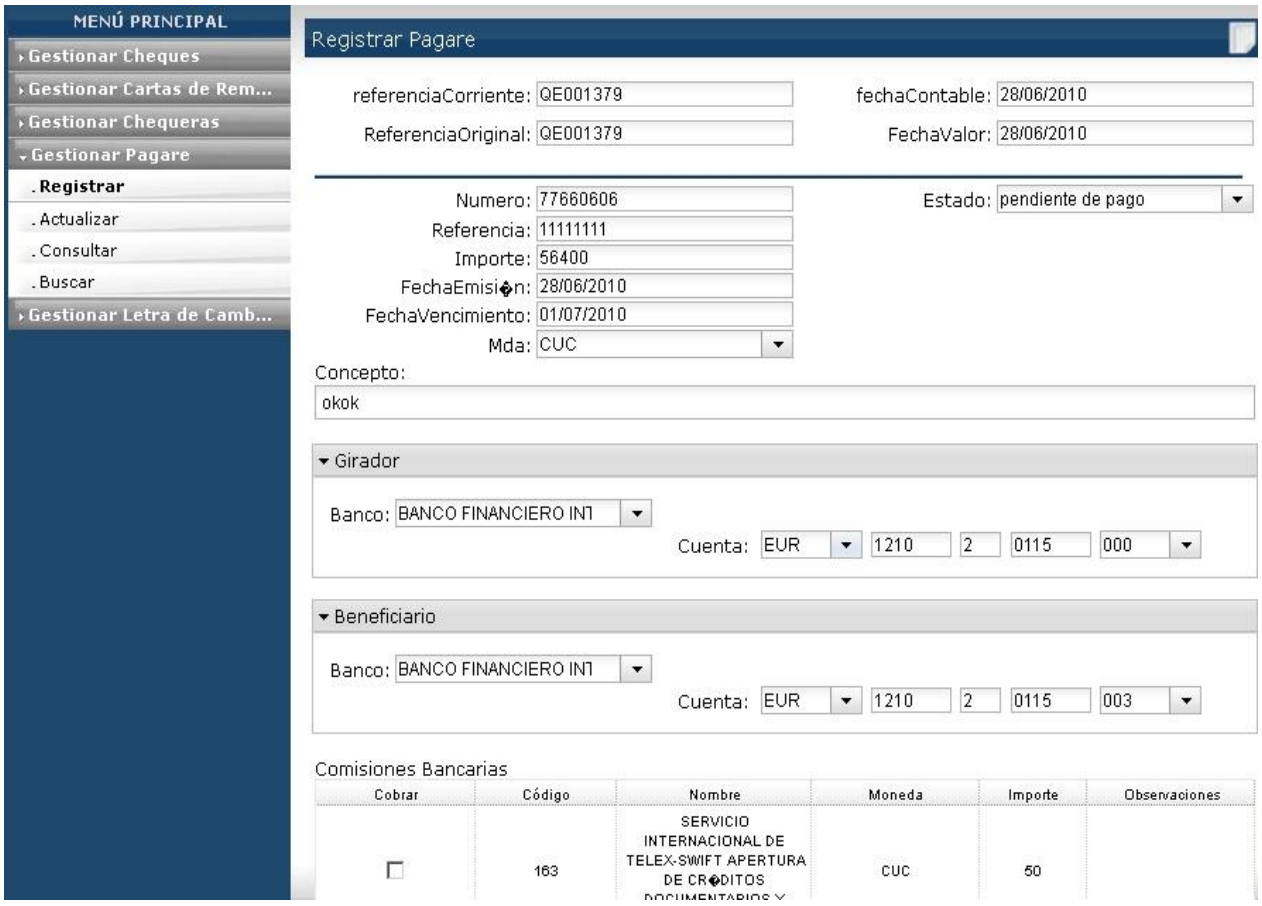
Anexo 5: Talonarios

Anexo 6: Pagaré con sus requisitos

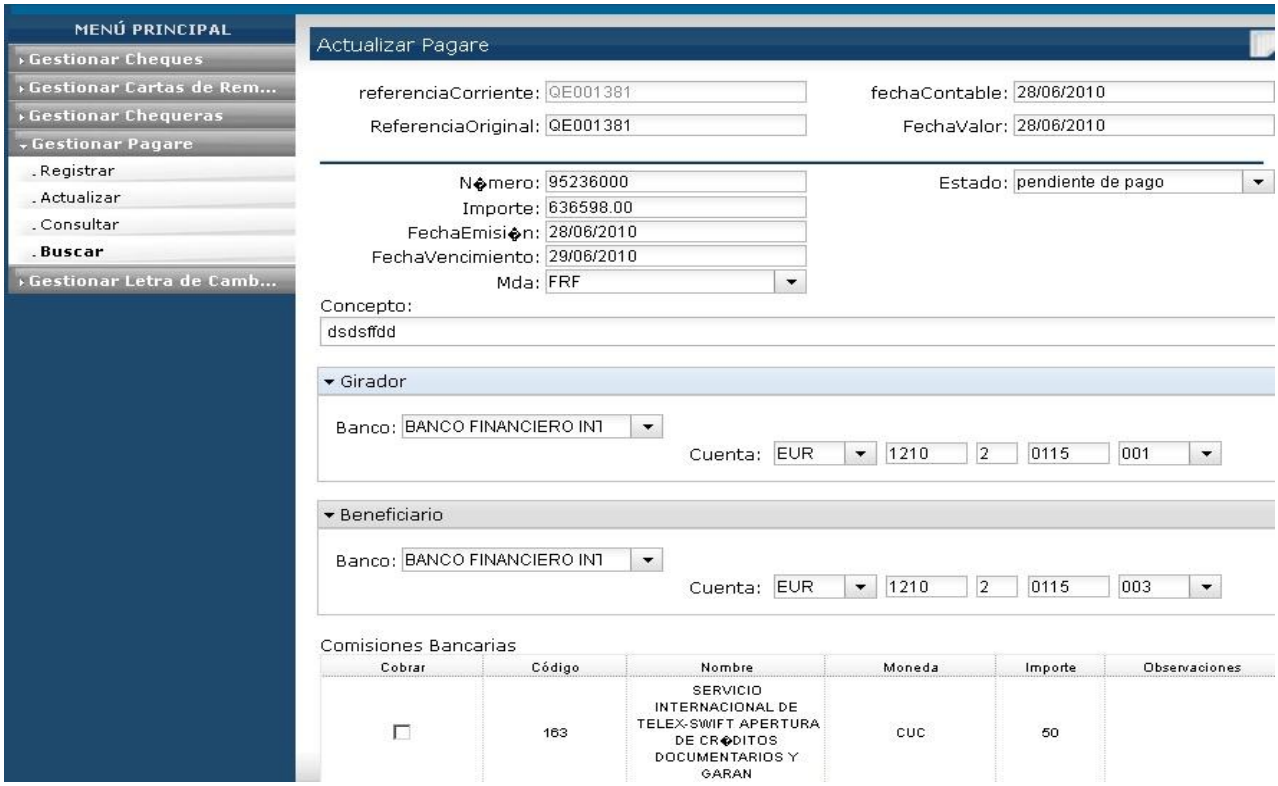
Anexo 7: Módulos de Spring framework



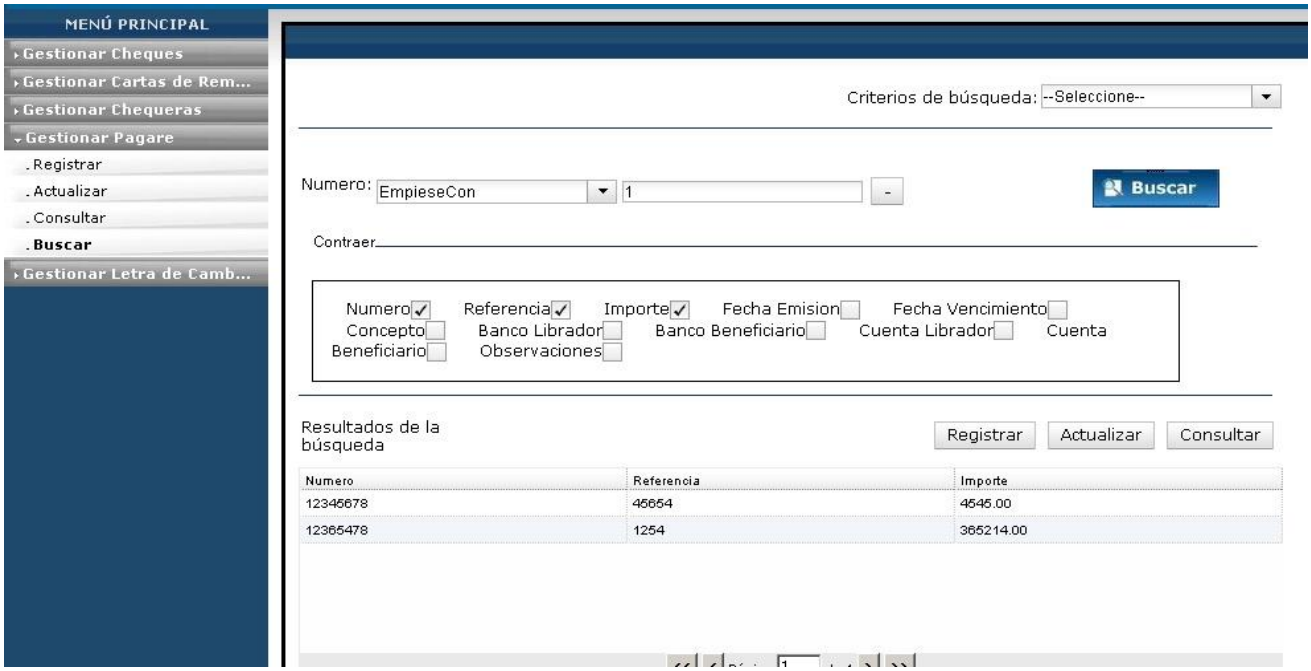
Anexo 8: Interfaz Registrar Pagaré



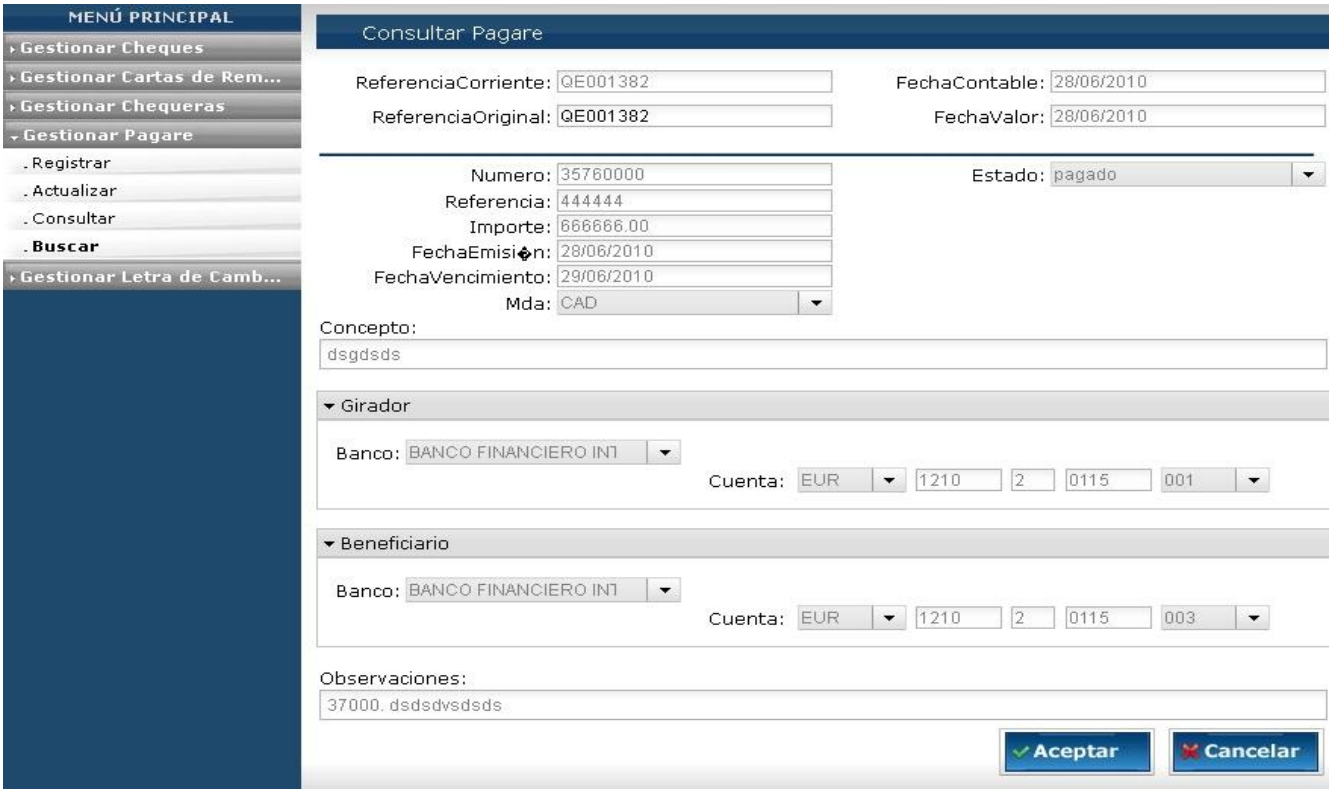
Anexo 9: Interfaz Actualizar Pagaré



Anexo 10: Interfaz Buscar Pagaré

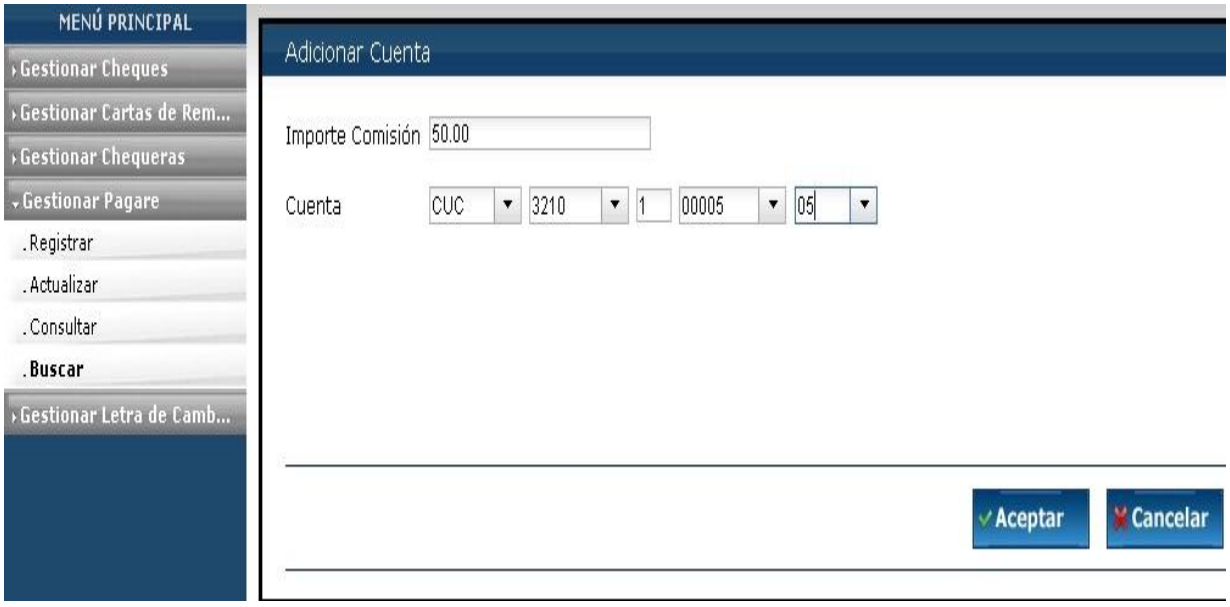


Anexo 11: Interfaz Consultar Pagaré

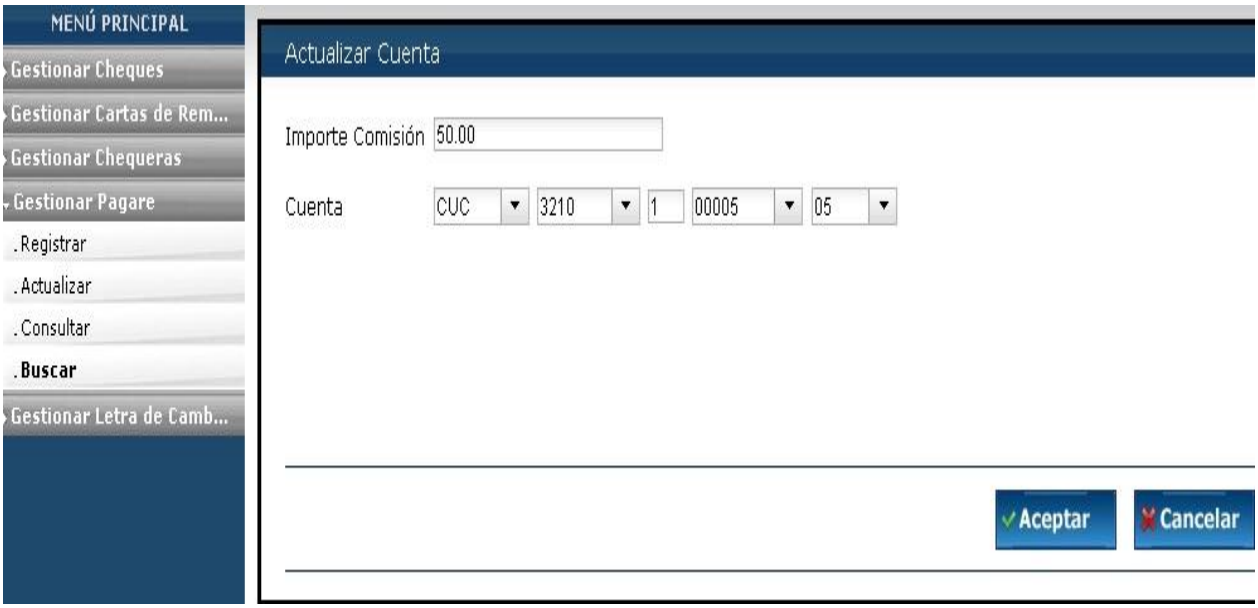


Anexo 12: Interfaz para el cobro de Comisiones Bancarias

Anexo 13: Interfaz para Adicionar cuentas para el cobro de Comisiones Bancarias



Anexo 14: Interfaz para Actualizar cuentas para el cobro de Comisiones Bancarias



Anexo 15: Estructura de clases dentro de los paquetes manager y facade de la Capa de Negocio

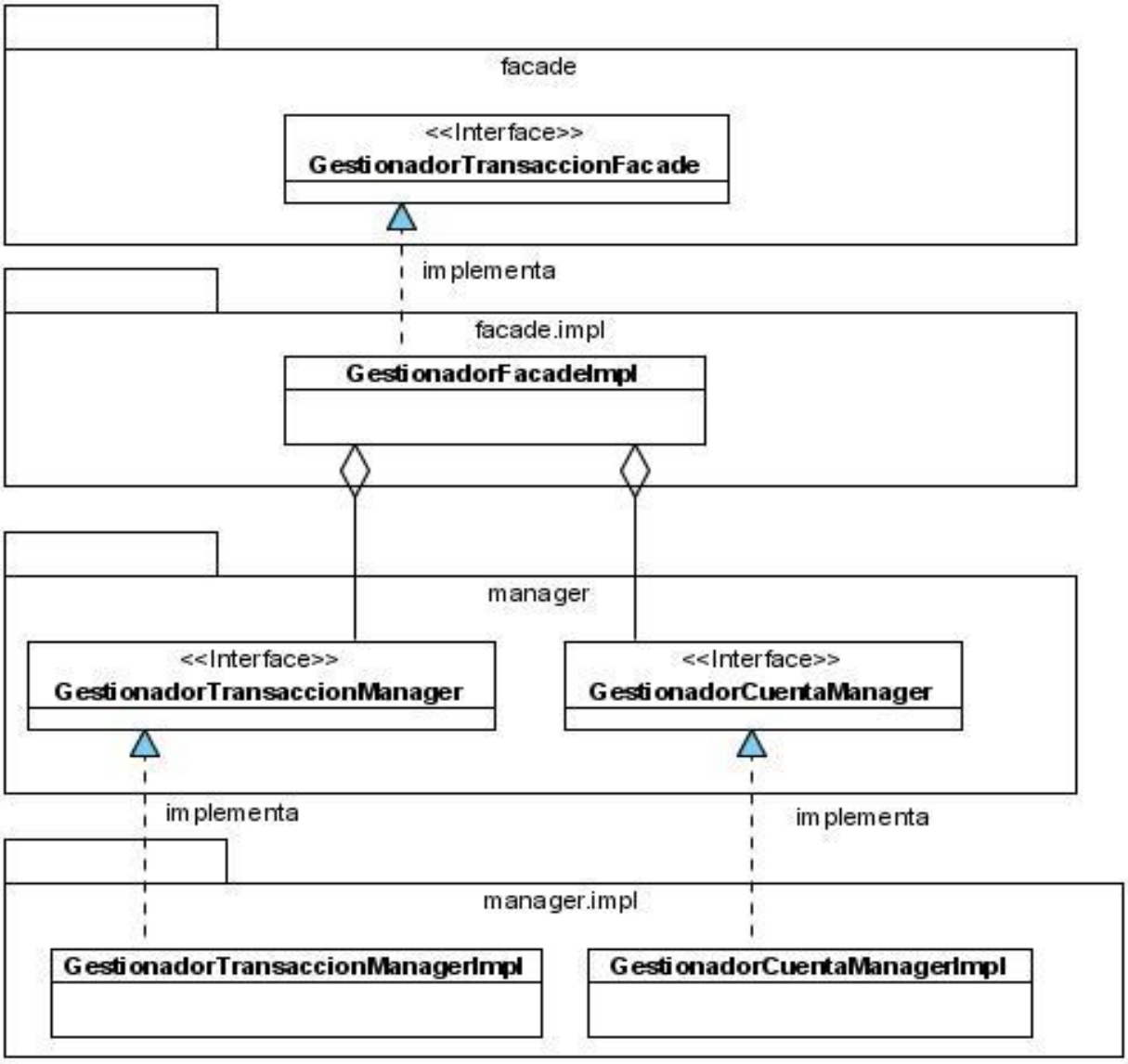
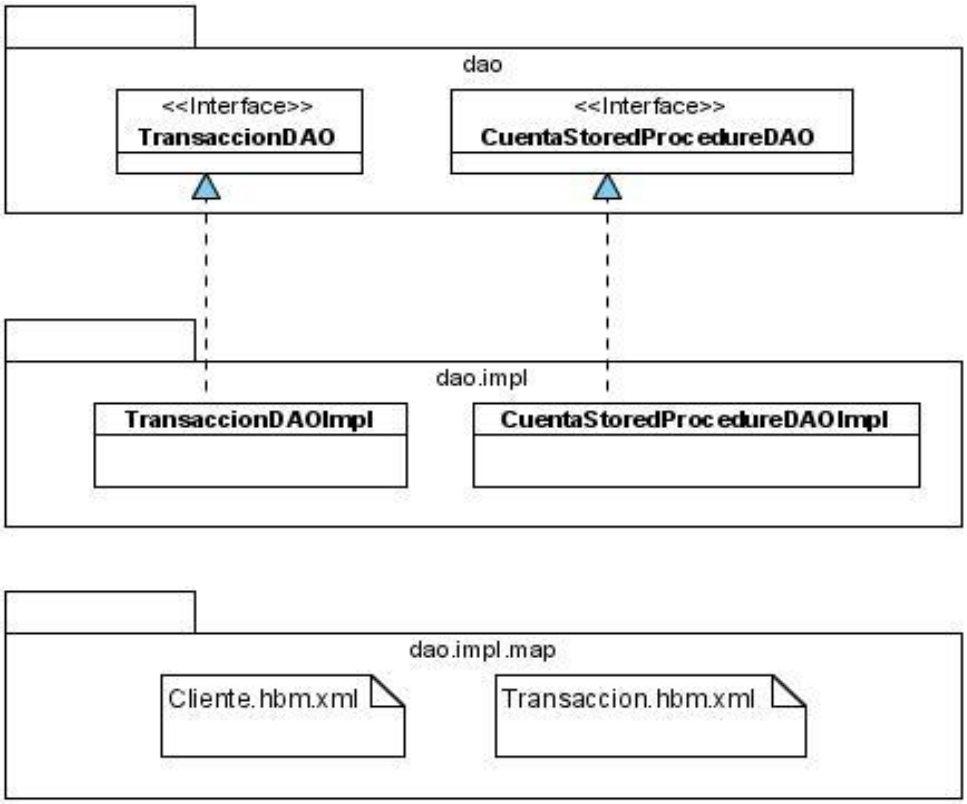


Figura 16: Estructura de los paquetes de clases en la Capa de Acceso a Datos.



GLOSARIO DE TÉRMINOS

Oficina Librada: Lugar de trabajo, generalmente de carácter administrativo o burocrático, tanto estatal como privado. (Bufete, despacho, agencia, estudio, secretaría, notaría, negociado, gabinete)

Cuenta librada: Cuenta del librador y es la cuenta del que paga el total del importe de la letra o del pagaré.

Bean: Es un componente software que tiene la particularidad de ser reutilizable y así evitar la tediosa tarea de programar los distintos componentes uno a uno.

Command: Objeto que representa los campos de una vista.

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

JavaServer Page (JSP): Tecnología orientada a crear páginas web con programación en Java.

API: Un API no es más que una serie de servicios o funciones que se le ofrece al programador.

Servlets: Programa que se ejecuta en el servidor.