UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
FACULTAD 4

ANÁLISIS Y DISEÑO DEL MÓDULO EMISIÓN DE CARTA DE CRÉDITO DEL SUBSISTEMA COMERCIO EXTERIOR DEL PROYECTO MODERNIZACIÓN DEL SISTEMA BANCARIO CUBANO

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

Autores
Alain Sánchez Ledón
Ismaury Pérez Figueroa

Tutora
Liliana María Hernández Toirac

Ciudad de La Habana
Junio, 2009

FRASE

"Las ideas nacen de los conocimientos y de los valores éticos. Una parte importante del problema estaría resuelta tecnológicamente, la otra hay que cultivarla sin descanso o de lo contrario se impondrán los instintos más primarios"

Fidel Castro Ruz

DECLARACIÓN DE AUDITORÍA

Declaramos que somos los únicos autores del trabajo titulado: Análisis y Diseño del módulo Emisión de Carta de Crédito del subsistema Comercio Exterior del proyecto Modernización del Sistema Bancario Cubano y autorizamos a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo.

Para que así conste firmamos la presente a los 15 días del mes de Junio del año 2009.

Ismaury Pérez Figueroa

_____

Firma del Autor

Alain Sánchez Ledón

_____

Firma del Autor

Liliana María Hernández Toirac

_____

Firma del Tutor

AGRADECIMIENTOS

Muchas han sido las personas involucradas en el logro de este sueño.
Quiero agradecer en primer lugar a mis padres por quererme con tanta intensidad y darme todo lo que tuvieron a su alcance, a quienes no sólo agradezco sus consejos y el apoyo constante, sino además, el privilegio de ser su hijo y en especial mi mamá que aunque no este presente se que esta muy orgullosa de mi.
A mis familiares.
A mis amigos por ayudarme en todo este tiempo y compartir conmigo alegrías y tristezas.
A todos los profesores que han formado en mí los valores necesarios para ser un buen profesional.
A Liliana por ser una tutora ejemplar, por ayudarme tanto y estar ahí cuando la necesité.
Por último quiero agradecer especialmente a la Revolución por darme la oportunidad de estudiar en la UCI que ha resultado tan positiva para mi formación personal y profesional.
A todos los que de una forma u otra han contribuido a la realización de este trabajo de diploma, de corazón, muchas gracias.

Ismaury

AGRADECIMIENTOS

De esta forma quiero agradecer a mi familia que siempre me ha dado aliento y apoyo en todo momento, agradecerles por la paciente espera de verme nacer como profesional este día.

A mi padre por siempre enseñarme lo que es correcto; por la honradez, la humildad y el amor con que lo ha hecho.

A mi madre por el amor y cariño que siempre me brindó, por los momentos de felicidad que me entregó y nunca olvidaré.

A mis compañeros por estar presentes en todo momento, gracias por todos por su ayuda.

A Liliana, tutora que siempre nos ayudó en todo momento.

A los profesores que me formaron como profesional para la vida laboral. Así como a la Revolución por permitirme estudiar en esta prestigiosa universidad.

Mi eterno agradecimiento a:

Lidiana Beltrán, Jennifer Céspedes, M. Teresa Herrera, Lázaro Ferrán, Antonia, Alina Brito, Migdalia Borges, Yaremis Morales, Rafaela Ruiz, Perla M Montero, María del Carmen Stocker, Yasiel Peña, Jorge Rodríguez, Yesenia, Bertha, Ariel Abreu, Eduardo Cruz, Juan C. Jiménez, Mario L. Rodríguez, Juan C. Matilla, Grabiel L. González.

Gracias a todos por su ayuda y apoyo incondicional.

Alain

DEDICATORIA

A mis padres que siempre confiaron en mí y me brindaron su apoyo para que viera realizados mis sueños, quiero regalarles este momento y honrarlos por tanto amor y dedicación. En Especial mi mamá que no me quita nunca los ojos de encima. Los quiero mucho.

Ismaury

A mi familia, que es lo más valioso que tengo en la vida.

A mi padre y a mi hermana por estar siempre conmigo, en las buenas y en las malas.

A mi madre que ya no la tengo pero que siempre esta presente

A todos los que como yo, nunca perdieron la esperanza.

Alain

RESUMEN

El presente trabajo aborda el análisis y diseño del subsistema Cartas de Créditos del proyecto Modernización del Sistema Bancario Cubano, el mismo se ajusta a los lineamientos arquitectónicos establecidos dentro del proyecto. Se describen las técnicas y herramientas utilizadas, así como la aplicación de patrones dentro del flujo de desarrollo de software siguiendo la metodología y procesos dictados por el Proceso Unificado de Desarrollo. Los artefactos se generaron usando como lenguaje de modelado el Lenguaje Unificado de Modelado y auxiliados por el Visual Paradigm como herramienta de Ingeniería de Software Asistida por Computadora. Mediante el flujo de trabajo de análisis y diseño se obtuvieron artefactos entendibles que viabilizan a los implementadores la automatización de una plataforma capaz de satisfacer las necesidades de los clientes. Para garantizar la calidad de los artefactos generados se aplicaron métricas, que arrojaron resultados positivos.

Palabras claves. Análisis, diseño, procesos, Cartas de Créditos.

ÍNDICE

Contenido

INTRODUCCIÓN

Hoy en día, debido al alto desarrollo alcanzado en la rama de la computación, la gran mayoría de las empresas se encuentran en algún grado informatizadas, convirtiéndose los sistemas computacionales en un recurso imprescindible para todas las ramas de la vida. Estos proporcionan la infraestructura suficiente y necesaria para la gestión de la información, elimina el trabajo engorroso de realizarlo a mano, evita que se comentan errores por el agotamiento del cerebro humano y permite obtener una información actualizada de todos los procesos que se encuentren digitalizados, obteniéndose un estricto control y seguimiento de los mismos. El Banco Nacional de Cuba (BNC) se encuentra enfrascado en la informatización de sus procesos financieros mediante el uso de las nuevas tecnologías de la información, lo que repercutirá en la elevación de la calidad de los servicios que se les ofrecen a los clientes.

Las Cartas de Créditos son uno de los principales instrumentos para asegurar la máxima satisfacción de los clientes. La exportación e importación de bienes extranjeros hoy en día es motivo de gran preocupación para la humanidad; de no ser por la existencia de las Cartas de Crédito, las cuales son utilizadas diariamente, estas operaciones representaran miles de millones de dólares a nivel mundial. Las Cartas de Crédito son un instrumento de pago, sujeto a regulaciones internacionales, mediante el cual un banco (Banco Emisor) obrando por solicitud y conformidad con las instrucciones de un cliente (ordenante) debe hacer un pago a un tercero (beneficiario) contra la entrega de los documentos exigidos, siempre y cuando se cumplan los términos y condiciones de crédito.

Actualmente el BNC utiliza una versión del Sistema Automatizado para la Banca Internacional de Comercio (SABIC) para aumentar la eficiencia en sus servicios. Este sistema fue desarrollado por la Dirección de Sistemas Automatizados del BCC para satisfacer las necesidades de procesamiento de datos de bancos e instituciones financieras utilizando microcomputadoras.

SABIC se centra principalmente en la contabilización, no se pueden generar muchos de los reportes estadísticos que el BNC necesita y no tiene la información centralizada, lo que implica que a los trabajadores del banco se les hace difícil acceder a la información. Usa MS-DOS como sistema de operativo. El sistema de mensajería interbancaria, utilizada por el BNC, es el SISCOM, el cual utiliza Windows y es una pérdida de tiempo y un maltrato a las computadoras cambiar varias veces de sistema operativo para realizar el proceso de emisión.

La máxima dirección del BNC al ver todas estas deficiencias le solicitó a la Universidad de las Ciencias Informáticas (UCI) el desarrollo de un software para perfeccionar las entidades financieras. Esta tarea se le asignó al proyecto Banco Nacional de la universidad, el cual decidió dividir el proyecto varios módulos.
Este trabajo se centrará en el módulo Emisión de Cartas de Crédito.

Problema a resolver

Los requerimientos para el nuevo software ya se encuentran capturados y documentados en el trabajo de diploma “Definición de los Requerimientos Funcionales del Módulo Emisión de Cartas de Crédito del Proyecto Banco Nacional”. Quedando expresado el problema a resolver de la siguiente forma:
¿Cómo realizar un análisis y diseño para la gestión de las Cartas de Crédito para el Proyecto Modernización del Sistema Bancario Cubano?

Objeto de estudio

Este trabajo tiene como objeto de estudio los procesos de Cartas de Crédito en entidades financieras bancarias.

Campo de acción

Los procesos de Emisión de Cartas de Crédito en entidades financieras bancarias de Cuba.

Objetivo general

Realizar el análisis y diseño del módulo Emisión de Cartas de Crédito para el Proyecto Modernización del Sistema Bancario Cubano.
Para darle cumplimiento a este objetivo y resolver el problema planteado, se realizaron las siguientes tareas:

- Análisis detallado de la tecnología y la arquitectura definidas en el proyecto.
- Análisis de los patrones de diseño para la búsqueda de soluciones a problemas comunes en el desarrollo.
- Análisis de los requisitos especificados para el desarrollo del subsistema propuesto. .
- Realización del diseño de la solución

El trabajo estará compuesto por tres capítulos:

1. Fundamentación Teórica: En este capitulo se argumentan las herramientas, metodologías y tecnologías a utilizar. Incluye además secciones como Estado del Arte, Cartas de Crédito.
2. Análisis y Diseño: Este capítulo se centra en el Análisis y Diseño, donde se explican los patrones a utilizar, la arquitectura propuesta por el proyecto Modernización del Sistema Bancario Cubano. Además de los artefactos que se generaran.
3. El 3er y último capítulo está destinado a la validación de la solución propuesta a través de métricas del diseño orientado a objetos y un análisis de los resultados.

1. CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En este capítulo se discuten conceptos fundamentales relacionados con el trabajo a desarrollar, además se argumentan las herramientas y metodología a utilizar en el desarrollo del proyecto. Los principios del análisis y el diseño están contenidos en este apartado.

Un estudio realizado sobre las Cartas de Crédito, tipos, funcionamiento, regulaciones y otras características, componen además esta sección, dando lugar a una disertación sobre sistemas contables nivel mundial y nacional que realizan operaciones sobre la gestión de Cartas de Crédito.

La Ingeniería de Software (IS) permite realizar un Análisis y Diseño con la calidad que requiere un software, proporcionando y aumentando la eficacia en el proceso de desarrollo de mismo, además posibilita de manera eficiente la traducción de requerimientos en diseño.

La IS es el establecimiento y uso de principios de ingeniería para obtener software que sea confiable y que funcione eficientemente. El proceso de desarrollo de software "es aquel en que las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos transformados en diseño y el diseño implementado en código, el código es probado, documentado y certificado para su uso operativo". Concretamente "define quién está haciendo qué, cuándo hacerlo y cómo alcanzar un cierto objetivo" (Jacobson, 1998).

Según la definición del IEEE[1], "software es la suma total de programas de computadora, procedimientos, reglas, la documentación asociada y los datos que pertenecen a un sistema de cómputo" (Lewis, 1994). Según el mismo autor, "un producto de software es un producto diseñado para un usuario". En este contexto, la Ingeniería de Software es un enfoque sistemático del desarrollo, operación, mantenimiento y retiro del software", que en palabras más llanas, se considera que "la Ingeniería de Software es la rama de la ingeniería que aplica los principios de la ciencia de la computación y las matemáticas para lograr soluciones costo-efectivas (eficaces en costo o económicas) a los problemas de desarrollo de software", es decir, "permite elaborar consistentemente productos correctos, utilizables y costo-efectivos" (Cota, 1994)

1. [1] IEEE: Instituto de Ingenieros Eléctricos y Electrónicos. Asociación técnico-profesional mundial dedicada a la estandarización.

1.2. Metodologías de desarrollo.

Se entiende por metodología de desarrollo una colección de documentación formal referente a los procesos, las políticas y los procedimientos que intervienen en el desarrollo del software.

1.2.1. Algunas metodologías.

XP o Programación Extrema.

XP define cuatro variables para proyectos de software: coste, tiempo, calidad y ámbito. Esta metodología trata de dar al cliente el software que él necesita y cuando lo necesita. XP es más una filosofía de trabajo que una metodología. Por otro lado ninguna de las prácticas defendidas por XP son invención de este método, XP lo que hace es ponerlas todas juntas.

SCRUM

Más que una metodología de desarrollo software, es una forma de auto-gestión de los equipos de programadores. Un grupo de programadores deciden cómo hacer sus tareas y cuánto van a tardar en ello. Scrum ayuda a que trabajen todos juntos, en la misma dirección, con un objetivo claro. Permite además seguir de forma clara el avance de las tareas a realizar, de forma que los "jefes" puedan ver día a día cómo progresa el trabajo.

MSF[2]

Metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso. Se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. (López Barrio, 2005)

CRYSTAL

Crystal da vital importancia a las personas que componen el equipo de un proyecto, y por tanto sus puntos de estudio son:

Aspecto humano del equipo

Tamaño de un equipo (número de componentes)

Comunicación entre los componentes

Distintas políticas a seguir

Espacio físico de trabajo

Crystal aconseja que el tamaño del equipo sea reducido.

FDD[3]

Es un proceso ágil para el desarrollo de sistemas.

No hace énfasis en la obtención de los requerimientos sino en como se realizan las fases de diseño y construcción.

Se preocupa por la calidad, por lo que incluye un monitoreo constante del proyecto.

Ayuda a contrarrestar situaciones como el exceso en el presupuesto, fallas en el programa o el hecho de entregar menos de lo deseado.

1.3. ¿Por qué RUP?

A pesar de ser definido por la dirección del proyecto, se realizó un estudio sobre algunas metodologías para analizar la conveniencia de utilizar RUP.

“Esta metodología proporciona un enfoque disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su meta es asegurar la producción del software de alta

2. [2] MSF: Microsoft Solution Framework.
3. [3] FDD: Desarrollo Manejado por Funcionalidades

calidad que resuelve las necesidades de los usuarios dentro de un presupuesto y tiempo establecidos."

Es Dirigido por Casos de Uso (CU), Iterativo e Incremental que tiene como ventaja que pequeños avances del proyectos son entregados al cliente, el cual puede probar mientras se esta desarrollando otra iteración del proyecto. Y además Centrado en la Arquitectura.

Esta disciplina define la arquitectura del sistema y tiene como objetivos trasladar requisitos en especificaciones de implementación, al decir análisis se refiere a transformar CU en clases, y al decir diseño se refiere a refinar el análisis para poder implementar los diagramas de clases de análisis de cada CU, los diagramas de colaboración de de cada CU, el de clases de diseño de cada CU, el de secuencia de diseño de CU, el de estados de las clases, el modelo de despliegue de la arquitectura. La realización de cada uno de estos diagramas depende de la complejidad y necesidades del desarrollo del software.

Existe una alta disponibilidad de la documentación sobre esta metodología así como personal calificado para su implementación. Es una inventiva probada y utilizada a nivel mundial con la cual se han obtenido resultados muy alentadores. Puede interpretarse según las necesidades y complejidad del software a desarrollar. A pesar de la variedad de metodologías existentes, donde todas tienen ventajas y desventajas, RUP es una de las más robustas y completas, ya que está destinada preferiblemente para proyectos de gran envergadura; pero esto no quiere decir que no sea aplicable a otros de menor complejidad. En cambio muchas metodologías no reúnen los requisitos necesarios para un buen desarrollo de software, aspecto que RUP garantiza en conjunto con la calidad de los resultados obtenidos, asegurando la complacencia del cliente con un software de gran calidad.

1.4. Framework utilizado.

Un framework es un término muy utilizado últimamente en el campo de la informática, se utiliza para referirse a un conjunto de bibliotecas, que se utilizan para implementar la estructura de un modelo para una aplicación. Esto se realiza con el objetivo de promover la reutilización de código, posibilitando que no sea necesario perder tiempo en reinventar la rueda. Existen diferentes tipos de framework, para diferentes propósitos, algunos orientados al desarrollo de aplicaciones web, o para un determinado sistema operativo o lenguaje. En un sentido muy amplio el framework que se utiliza determina la arquitectura del software.

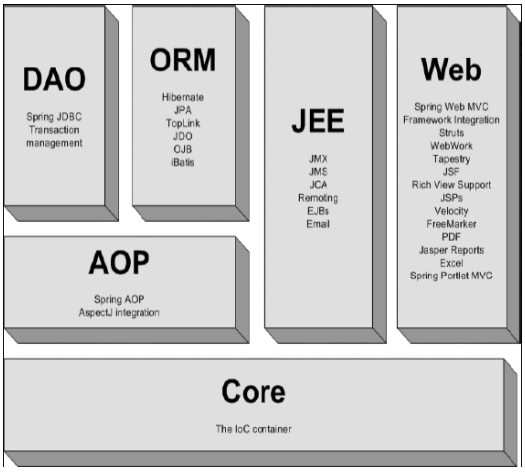
1.4.1. Spring

Los frameworks dan la posibilidad de definir la forma de realización de aplicaciones de software, dando soporte y simplificando parte de la complejidad. Spring; es un framework que tiene el objetivo de facilitar la construcción de aplicaciones Java, presenta un entorno diseñado para aumentar la productividad, liberando al desarrollador de tareas repetitivas, ayudándolo a hacer diseños más consistentes. Se puede utilizar en cualquier tipo de aplicación, y es ligero por el mínimo impacto que tiene en las aplicaciones. Spring se basa en la técnica Inversión de Control (IoC), técnica que promueve el bajo acoplamiento a partir de la inyección de dependencias (DI) entre los objetos y una implementación de desarrollo según el paradigma de Orientación a Aspectos (AOP) que presenta una estructura simplificada para el desarrollo y utilización de aspectos (módulos multiple object crosscutting).

El paradigma de Orientación a Aspectos (AOP) complementa la Programación Orientada a Objetos (POO), proponiendo otra manera de pensar sobre la estructura de un programa. Mientras que la POO descompone las aplicaciones en una jerarquía de objetos, la AOP descompone los programas en aspectos o preocupaciones. Dichas preocupaciones se convierten en servicios del sistema, separados de la lógica de negocio y que se ejecutan de manera trasversal a la funcionalidad base, lo que proporciona una definición de responsabilidades superior. Spring básicamente es un contenedor, que se encarga de gestionar, administrar el ciclo de vida y de la configuración de las clases de la aplicación. Es una aplicación open source, lo cual implica que no tiene ningún costo, ni se necesita licencia para utilizarlo, dando la libertad de incursionar en la

utilización de esta aplicación, además de que está disponible todo el código fuente de este framework en el paquete de la instalación. En sentido general Spring no intenta reinventar la rueda, sino que integra las diferentes tecnologías existentes en un solo framework posibilitando un desarrollo más sencillo y eficaz.

1Arquitectura del framework Spring



Core: Como su nombre indica, es el núcleo de Spring. Permite técnicas de Inversión del Control (IoC) como la inyección de dependencias.

AOP: Proporciona una implementación de programación orientada a aspectos, permitiendo definir puntos de corte e interceptores.

DAO: Proporciona el acceso a una capa de abstracción JDBC (Java Database Connectivity) con una forma de administrar transacciones desde el módulo AOP como es el caso de Hibernate y JDO.

ORM: Provee capas de integración para APIs de mapeo objeto-relacional.

Web: posibilita determinadas características apropiadas para el desarrollo de aplicaciones web e integración con otros frameworks (Struts, JSF, Tapestry, etc).

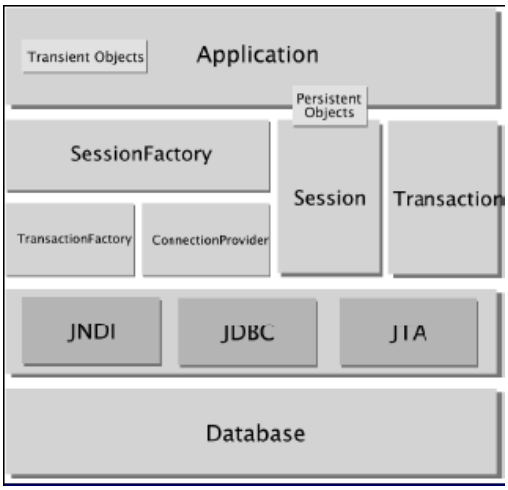
JEE: acceso e interacción con servicios enterprise.

1.4.2. Spring Web Flow.

Spring web flow forma parte del paquete de desarrollo de aplicaciones web de Spring. Se caracteriza por permitir la navegación múltiple y compleja permitiendo guiar al usuario a través de una serie de pasos para completar una transacción de aplicación y por ser una plataforma web de alto nivel, además incrementa la productividad, calidad y facilidad para realizar pruebas en el proceso de desarrollo.

1.4.3. Hibernate

Hibernate es una solución ORM (*Object-Relational Mapping*) para Java, es open source (código abierto) y la licencia está eximida de costo. *Hibernate* busca solucionar el problema de la diferencia entre el modelo orientado a objetos y el usado en las bases de datos modelo relacional mediante archivos declarativos. Está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de un modelo de objetos existente. Cabe mencionar que provee con Hibernate Query Language (HQL) una poderosa vía de comunicación entre el programador y la base de datos, al dotarlo de un lenguaje invariante con respecto a esta y además sintácticamente muy parecido al Structured Query Language (SQL).



2 Arquitectura de Hibernate

1.5. Lenguaje de programación y Entorno de Desarrollo Integrado (IDE).

Java es un lenguaje de desarrollo orientado a objetos, multiplataforma. Proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas. Es un lenguaje interpretado y compilado, esto quiere decir que su código fuente se transforma en algo similar al código máquina, los bytecodes y a la vez estos bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real (run-time). Además es considerado un lenguaje robusto, fiable y seguro, para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución.

Soporta la sincronización múltiple de hilos de ejecución (*multithreading*), lo cual da la posibilidad que un hilo puede ocuparse de interactuar con el cliente y otro de realizar una operación. Otra de las tantas características que lo hacen idóneo para su utilización, es que se utiliza para dos tipos de programas, aplicaciones independientes y applets.

1.6. Análisis y Diseño.

En la pasada década, se desarrollaron varios métodos de análisis y especificación del software. Los investigadores han identificado los problemas y sus causas y desarrollando reglas y procedimientos para resolverlos. Cada método de análisis tiene una única notación y punto de vista. Sin embargo, todos los métodos de análisis están relacionados por un conjunto de principios fundamentales.

El dominio de la información, así como el dominio funcional de un problema debe ser representado y comprendido. El problema debe subdividirse de forma que se descubran los detalles de una manera progresiva (o jerárquica) Deben desarrollarse las representaciones lógicas y físicas del sistema.

Aplicando estos principios, el analista enfoca el problema sistemáticamente. Se examina el dominio de la información de forma que pueda comprenderse su función más completamente. La partición se aplica para reducir la complejidad. La visión lógica y física del software, es necesaria para acomodar las ligaduras lógicas impuestas por los requerimientos de procesamiento, y las ligaduras físicas impuestas por otros elementos del sistema.

1.6.1. Principios operativos del análisis.

- Debe representarse y entenderse el dominio de información de un problema.
- Deben definirse las funciones que se realizarán con el software.
- Debe representarse el comportamiento del software (como consecuencia de acontecimientos externos).
- Deben dividirse los modelos que representan la información, función y comportamiento de tal manera que se descubran los detalles por capas o jerárquicamente.
- El proceso de análisis debe ir desde la información esencial hasta el detalle de implementación.

1.6.2. ¿Qué es el diseño?

Representación ingenieril de algo que se va a hacer. El “plano” del software.

Áreas que conforman el Diseño

- Datos, Arquitectura, Interfaces y Componentes.
- Resultados
- Varios modelos para las diferentes áreas.
- Proceso

Empieza con la descripción de los requerimientos y se construye poco a poco, siguiendo los principios de diseño, hasta obtener la especificación del diseño.

1.6.3. Principios del diseño.

- En el proceso deben tomarse enfoques alternativos.
- Deberá rastrearse hasta el análisis.
- Se debe reutilizar.
- Tratar de imitar el dominio del problema.
- Uniformidad e integración.
- Deberá estructurase para admitir cambios.
- Debe prever la adaptación a circunstancias inusuales.
- No codificar.
- Evaluarse en función de calidad mientras está creciendo.
- Minimizar errores conceptuales.

1.7. Patrones

Un patrón es una unidad de información nombrada, instructiva e intuitiva que captura la esencia de una familia exitosa de soluciones probadas a un problema recurrente dentro de un cierto contexto. El objetivo de los patrones es crear un lenguaje común a una comunidad de desarrolladores para comunicar experiencia sobre los problemas y sus soluciones. Pueden referirse a distintos niveles de abstracción, desde un proceso de desarrollo hasta la utilización eficiente de un lenguaje de programación.

Un buen patrón debe:

- Solucionar un problema
- Ser un concepto probado
- La solución no es obvia
- Describe participantes y relaciones entre ellos
- Tiene un alto componente humano: estética y utilidad

1.8. Lenguaje de modelado.

Un lenguaje de modelado es un conjunto estandarizado de símbolos y de modos de disponerlos para modelar (parte de) un diseño de software orientado a objetos.

1.8.1. Notación Unified Modeling Language (UML).

UML es un lenguaje estándar para los planos del software, que se utiliza para:

Visualizar: dispone de un conjunto de símbolos gráficos, cada uno de los cuales tiene una semántica bien definida.

Especificar: cubre la especificación de todas las decisiones de análisis, diseño e implementación que deben realizarse al desarrollar y desplegar un sistema con gran cantidad de software.

Construir: permitir la ejecución directa de modelos, la simulación de sistemas y la instrumentación de sistemas en ejecución.

Documentar: cubre la documentación de la arquitectura de un sistema y todos sus detalles; proporciona un lenguaje para expresar requisitos y pruebas así como un lenguaje para modelar las actividades de planificación de proyectos y gestión de versiones. Está pensado principalmente para sistemas con gran cantidad de software pero no está limitado al modelado de software.

1.9. Herramienta CASE.

Las Herramientas CASE[4] son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación y detección de errores.

Objetivos:

- Mejorar la productividad en el desarrollo y mantenimiento del software.
- Aumentar la calidad del software.
- Mejorar el tiempo y coste de desarrollo y mantenimiento de los sistemas informáticos.
- Mejorar la planificación de un proyecto.
- Aumentar la biblioteca de conocimiento informático de una empresa ayudando a la búsqueda de soluciones para los requisitos.
- Automatizar, desarrollo del software, documentación, generación de código, pruebas de errores y gestión del proyecto.
- Ayuda a la reutilización del software, portabilidad y estandarización de la documentación.
- Gestión global en todas las fases de desarrollo de software con una misma herramienta.
- Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

1.9.1. Rational Rose Enterprise Edition 2003.

Rational Rose Enterprise Edition es una herramienta CASE que soporta el modelado visual con UML, ofreciendo distintas perspectivas del sistema. Da soporte a RUP.

Algunas de sus características son:

Diseño dirigido por modelos que redundan en una mayor productividad de los desarrolladores.

Diseño centrado en casos de uso y enfocado al negocio, que generan un software de mayor calidad.

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

Cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases y entregables.
Sólo permite trabajar sobre el sistema operativo Windows.
Esta herramienta propone la utilización de cuatro tipos de modelo para realizar un diseño del sistema, utilizando una vista estática y otra dinámica de los modelos del sistema, uno lógico y otro físico. Permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software.

1.9.2. Visual Paradigm.

Visual Paradigm para UML es una herramienta CASE que soporta UML 2.1 como lenguaje de modelado y la BPMN16. Esta útil herramienta apoya el ciclo de vida completo de desarrollo del software, análisis, diseño, implementación y prueba. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.
Brinda la posibilidad de generar código a partir de los diagramas, para plataformas como .Net, Java y PHP, así como obtener diagramas a partir de código.
El análisis textual es una técnica útil y práctica para la captura de los requisitos del sistema y la identificación de las clases candidatas, Visual Paradigm es una de las pocas herramientas CASE que soporta el análisis textual.
Ofrece un diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad, además presenta disponibilidad de múltiples plataformas y en múltiples versiones. Esta característica es muy importante pues por ejemplo el Rational Rose, que es una herramienta muy recomendada y además profesional, tiene una desventaja en su contra pues obliga al usuario a desarrollar en máquinas con el sistema operativo Windows, mientras que el Visual Paradigm está disponible para varios sistemas operativos como Windows, Linux, Unix. Se decidió seleccionar esta herramienta CASE fundamentalmente por esta característica.

1.10. Sistema Bancario

1.10.1. ¿Qué es una Carta de Crédito?

Es el convenio en virtud del cual un Banco (Banco Emisor), obrando por cuenta propia o a petición de un cliente (el ordenante del crédito) y de conformidad con sus instrucciones, se obliga a efectuar un pago a un tercero (beneficiario) o autoriza a otro Banco a efectuar dicho pago, contra presentación de los documentos exigidos dentro del tiempo limite especificado, siempre y cuando se hayan cumplido los términos y condiciones del crédito.

1.10.1.1. Partes que intervienen.

Ordenante: Persona o entidad que solicita la apertura del crédito a su banco, comprometiéndose a efectuar el pago. Suele ser el importador o un agente.

Banco Emisor: Banco elegido por el comprador que confecciona y procede a la apertura del crédito. Efectúa el pago del crédito si se cumplen las condiciones exigidas en el mismo.

Banco Avisador (Notificador): Banco corresponsal del Banco Emisor en el país del exportador. Avisa al beneficiario de la apertura del crédito sin establecer ningún otro compromiso que el del propio aviso.

Banco Pagador: Banco, generalmente en el mismo país del exportador, que recibe el mandato del Banco Emisor para pagar o comprometerse al pago al vencimiento contra presentación de los documentos conformes con los términos y las condiciones del crédito. Para el beneficiario es como si de una delegación del propio Banco Emisor se tratara. Para el exportador es conveniente que exista un Banco Pagador en su propio país.

Banco Aceptador: Tiene un papel similar al del Banco Pagador, pero en este caso acepta un efecto al vencimiento en lugar de pagar o comprometerse al pago. El exportador dispone de un compromiso de pago documental (la letra de cambio) mientras que el Banco Pagador en un pago diferido se compromete al pago de forma no documental (no existe letra de cambio).

Banco Negociador: Tiene una función similar a la de los dos anteriores, sin embargo el Banco Negociador compra (descuenta) un efecto al exportador. Aunque el pago suele ser diferido, el exportador cobra a la vista, soportando o no los intereses del descuento de acuerdo con la condiciones del crédito. La compra es sin recurso contra el exportador si el Banco Negociador es también Banco Confirmador.

Beneficiario: Persona a cuyo favor se emite el crédito y que puede exigir el pago al Banco Emisor o al Banco Pagador, una vez haya cumplido con las condiciones estipuladas en el crédito.

Banco Confirmador: Banco que garantiza el pago por parte del Banco Emisor. Corrientemente en el lugar donde esta ubicado el beneficiario. Suele utilizarse cuando las garantías que ofrece el Banco Emisor no se consideran suficientes (países conflictivos, bancos de desconocida o dudosa solvencia) o cuando el beneficiario quiere asegurar pago o aceptación inmediatamente después de haber efectuado el despacho. En la mayoría de los casos es el propio Banco Avisador.

1.10.1.2. Funcionamiento y Circuitos que siguen las Cartas de Crédito.

- El exportador y el comprador cierran el contrato de compra-venta indicando en el mismo: "forma de pago: Carta de Crédito"
- El comprador solicita al Banco Emisor abrir el crédito a favor del exportador.
- El Banco Emisor solicita a un Banco Intermediario que avise y/o confirme su crédito.
- El Banco Avisador/Confirmador remite el crédito al beneficiario.
- El exportador envía la mercancía al comprador.
- El exportador presenta los documentos de embarque solicitados al Banco Avisador/Confirmador.
- El Banco Avisador/Confirmador revisa los documentos y paga, acepta o negocia (si el crédito lo designa como banco Pagador, Aceptador o Negociador) bajo los términos del crédito.
- El Banco Avisador/Confirmador remite los documentos al Banco Emisor.
- El Banco Emisor revisa los documentos y reembolsa al Banco Intermediario.
- El Banco Emisor adeuda al comprador según acuerdo.

- El Banco Emisor entrega los documentos al comprador.
- El comprador presenta el juego de documentos para retirar la mercancía.

1.10.1.3. Tipos de Cartas de Crédito.

- **De Importación.**

La Carta de Crédito de Importación se refiere a todo acuerdo, cualquiera que sea su denominación o descripción, por el que un banco (“Banco Emisor”), obrando a petición y de conformidad con las instrucciones de un cliente (“Ordenante”) o en su propio nombre: Se obliga a hacer un pago a un tercero (“Beneficiario”) por la compra de determinada mercancía (BNC, 2007).

- **De Exportación.**

Carta de Crédito de Exportación, es la que emite un banco cuando el comprador extranjero (Solicitante) solicita a su banco (Banco emisor) que le emita un crédito documentario a favor de un proveedor (Beneficiario) (BNC, 2007). La Garantía Bancaria constituye un compromiso de pago del banco emisor a favor de un beneficiario.

- **De Garantía.**

La Carta de Crédito de Garantía es el instrumento que acompaña contratos internacionales y suministro de bienes de toda clase, de prestación de servicios, de realización de mano de obras y otros contratos de préstamos. La emite el Banco garantizando el pago al Beneficiario en caso de que su cliente no efectúe el mismo (BNC, 2007).

1.10.2. Sistemas Contables.

Los sistemas contables son los programas de contabilidad o paquetes contables, destinados a sistematizar y simplificar las tareas de contabilidad. El software contable registra y procesa las transacciones históricas que se generan en una empresa o actividad productiva: las funciones de compra, ventas, cuentas por cobrar, cuentas por pagar, control de inventarios, balances, producción de artículos, nóminas. Para ello solo hay que ingresar la información requerida, como las pólizas contables, ingresos y egresos, y hacer que el programa realice los cálculos necesarios.

1.10.2.1. Sistemas Contables en el Mundo.

CARCRED es un sistema desarrollado por la compañía venezolana LA. Sistemas, con más de 20 años de trascendencia en el mercado financiero de ese país, el cual permite mantener el control y seguimiento de todos los procesos asociados a las Cartas de Créditos tales como Negociaciones y Pago de las mismas. Emite la Relación contable diaria en reporte y archivo de textos, genera las llamadas débitos/créditos de las operaciones efectuadas. Está dividido en 6 módulos. El módulo de Cartas de Crédito posibilita el registro de los instrumentos de pago utilizados en el comercio internacional.

Otro sistema contable utilizado es El Sistema Integral de Comercio Internacional(COBIS SCI), desarrollado con tecnología de punta y arquitectura cooperativa Cliente-Servidor, Permite informar al Cliente el estado de sus operaciones, el manejo eficiente de cobranzas, discrepancias, la generación de mensajes SWIFT en forma automática, las funciones de asignación, mantenimiento y control de las líneas de crédito de bancos corresponsales, con el propósito de canalizar las cartas de crédito en forma ágil y oportuna, puede realizar de forma automática la contabilización de las transacciones financieras generadas, maneja el proceso de negociación de una carta de crédito, con posibilidad de registrar discrepancias mediante la respectiva carta, indicándose también detalles de Embarques y formas de pago, liquidación, abono y cancelación. Se puede también indicar los costos a cobrar, que son adicionales a los que generalmente están asociadas las transacciones para Cartas de Crédito de Importación.

Estos sistemas son propietarios y muy costosos por lo que Cuba no puede adquirirlos. Constituyen además un riesgo en cuánto a seguridad se refiere, debido a que no está disponible una bibliografía suficiente para un estudio completo de cómo fueron concebidos.

1.10.2.2. Sistemas Contables en Cuba.

La Dirección de Sistemas Automatizados del Banco Central de Cuba para satisfacer las necesidades de procesamiento de datos de bancos e instituciones financieras desarrolló el Sistema Automatizado para la Banca Internacional de Comercio. Dentro de las principales características funcionales del SABIC se encuentran:
La Contabilización en tiempo real: permite que la extracción de dinero de una cuenta o el sobregiro de cualquier otra cuenta se realice de manera segura, controlando la existencia de los fondos requeridos para realizar la operación y haciendo posible que la institución mantenga sus ficheros contables actualizados permitiéndole saber en cualquier momento su situación financiera.

***La Contabilidad Multimoneda*:** permite poder registrar los activos y pasivos sin tener que hacer conversiones de moneda lo cual garantiza una mayor exactitud de la información sobre la situación financiera de la institución, al no tener que depender de las variaciones de los tipos de cambios.

La característica Multisucursal: se debe a que con su ayuda y utilizando una red de transmisión de datos X-25 o similares, se pueden enlazar entre sí todas las oficinas de un banco o institución financiera y realizar, también en tiempo real.

La característica Transaccional del Sistema: se basa en la contabilización de operaciones mediante transacciones las cuales son el conjunto de asientos requeridos para registrar una operación. Al ser un sistema modular facilita la adaptabilidad, flexibilidad y evolución del sistema sin tener que efectuar cambios en sus programas generales.

Debido a la evolución de las actividades que se llevan a cabo dentro de las instituciones financieras bancarias cubanas se hizo necesario la integración de otras funcionalidades al sistema por lo que existen varias versiones del SABIC: La primera versión del SABIC utiliza el MS-DOS como sistema de explotación lo cual constituye una limitación al ser este último monotarea, es decir, el microprocesador solo puede atender un único proceso. Esta primera versión fue realizada en FoxPro y utiliza un servidor de ficheros lo que conlleva a un tráfico excesivo dentro de la red.

Por esta razón se decidió realizar una nueva versión en un ambiente cliente-servidor. La segunda versión del SABIC se realizó bajo la filosofía de tener en un corto tiempo un sistema que utilizara las ventajas de la técnica cliente-servidor; pero sin realizar un nuevo diseño del mismo, por lo que siguieron persistiendo problemas de no adecuación con todos los procesos que se llevan a cabo dentro de un banco. Se escogió como lenguaje para programar al cliente a Visual FoxPro y para el servidor SQL Server. Con respecto al cliente la selección se basó en que el sistema anterior está programado en FoxPro y utilizar Visual FoxPro era más productivo ya que se podía aprovechar parte del código escrito para la versión anterior. Posteriormente a estas versiones al sistema se le han incluido algunas funcionalidades que dan soporte a la mayoría de los procesos que van surgiendo en la actividad financiera nacional o a las variaciones que han sufrido los mismos.

1.11. Conclusiones Parciales.

Uno de los principales objetivos del desarrollo de software a nivel mundial es la calidad de desarrollo y mantenimiento del software, es por ello que este capitulo se ha dedicado a realizar un estudio sobre metodologías de desarrollo de software, así como de herramientas Case y lenguajes de modelado para guiar el proceso de desarrollo. De este estudio resultó la selección de la metodología RUP, el Lenguaje Unificado de Modelado y la herramienta CASE Visual Paradigm.

2. CAPÍTULO 2: ANÁLISIS Y DISEÑO

2.1. Introducción

En este capítulo se explican los principales conceptos a tener en cuenta en el análisis del sistema, así como las principales funcionalidades con que debe contar el mismo.
Se aborda sobre los patrones de diseño que se emplearon para la realización del diseño en cuestión.

2.2. Requisitos del software.

El levantamiento de requisitos no es más que especificar y validar los servicios que debe proporcionar el sistema así como las restricciones sobre las que se deberá operar. Consiste en un proceso iterativo del análisis del problema a resolver, documentando los resultados en una variedad de formatos y probando la exactitud del conocimiento adquirido. Para modelar el sistema que se pretende construir se identifican sus requisitos, tanto funcionales como no funcionales, y se modelan los funcionales en términos de casos de uso del sistema.

2.2.1. Requerimientos Funcionales.

Los requerimientos funcionales son aquellos que representan capacidades que el sistema debe cumplir para satisfacer sus necesidades del cliente y que contribuyan a encontrar funcionalidades del sistema ha implementar.
Para cumplir los objetivos de este sistema el mismo debe ser capaz de:

1. **RF 1. Gestionar Carta de Crédito**

1.1. Registrar Carta de Crédito
1.2. Buscar Carta de Crédito
1.3. Modificar Carta de Crédito
1.4. Eliminar Carta de Crédito.

2. RF 2. Gestionar Enmienda

2.1. Registrar Enmienda

2.2. Buscar Enmienda

2.3. Modificar Enmienda

2.4. Eliminar Enmienda

3. RF 3. Contabilizar Apertura de Carta de Crédito.

4. RF 4. Contabilizar Enmienda.

4.1. Contabilizar Enmienda por Variación de Importe.

4.2. Contabilizar Enmienda por Cancelación

4.3. Contabilizar Enmienda por Variación de la Fecha de Vencimiento

4.4. Contabilizar Enmienda por Variación de Interés

5. RF 5. Contabilizar Cancelación

6. RF 6. Listar Cartas de Crédito por Vencer

6.1. Mostrar Cartas de Crédito por vencer.

6.2. Generar carta a la empresa.

7. RF 7. Gestionar Tipo de Financiamiento

7.1. Registrar Tipo de Financiamiento

7.2. Modificar Tipo de Financiamiento

7.3. Eliminar Tipo de Financiamiento

8. RF 8. Gestionar Forma de Pago

8.1. Registrar Forma de Pago

8.2. Modificar Forma de Pago

8.3. Eliminar Forma de Pago

9. RF 9. Gestionar Forma de Pago de Comisiones.

9.1. Registrar Forma de Pago de Comisiones.

9.2. Modificar Forma de Pago de Comisiones.

9.3. Eliminar Forma de Pago de Comisiones

10. RF 10. Contabilizar Pago.

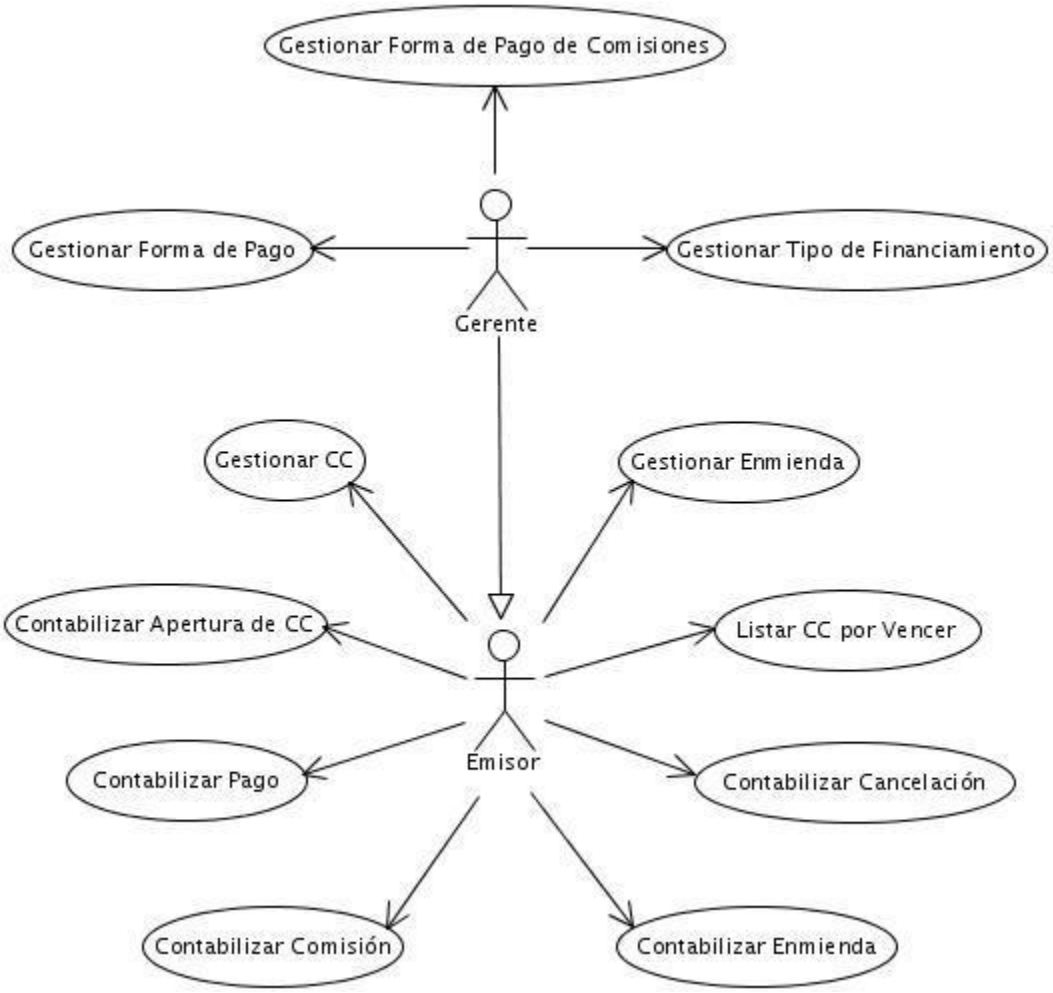
11. RF 11. Contabilizar Cobro de Comisiones.

12. RF 12. Imprimir.

A partir de estos requisitos se obtuvieron los siguientes casos de uso:

- CU Gestionar Forma de Pago
- CU Gestionar Forma de Pago de Comisiones
- CU Gestionar Tipo de Financiamiento
- CU GestionarCC
- CU Gestionar Enmienda
- CU Contabilizar Apertura de CC
- CU Listar CC por vencer
- CU Contabilizar Pago
- CU Contabilizar Cancelación
- CU Contabilizar Cancelación
- CU Contabilizar Enmienda

- **1 Diagrama de Casos de Uso**



Se trabajaron con artefactos de entrada como:

- **Diagrama de CU del sistema.**

Este diagrama muestra los actores y los CU, con la relación que existe entre ellos en de dependencia de que actor realice una determinada tarea función.

- **Descripción de CU.**

En la descripción de los CU se describe por escenarios los pasos que realizan los actores involucrados y el sistema para cumplimentar una determinada operación.

2.3. Patrones del diseño.

Un patrón es una unidad de información nombrada, instructiva e intuitiva que captura la esencia de una familia exitosa de soluciones probadas a un problema recurrente dentro de un cierto contexto.
El objetivo de los patrones es crear un lenguaje común a una comunidad de desarrolladores para comunicar experiencia sobre los problemas y sus soluciones.
Un buen patrón debe:

- Solucionar un problema
- Ser un concepto probado
- La solución no es obvia
- Describe participantes y relaciones entre ellos
- Tiene un componente humano alto: estética y utilidad

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.
Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

2.3.1. Patrones a utilizar.

2.3.1.1. Patrones de asignación de Responsabilidades GRASP.

En los patrones GRASP[5] 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.

2.3.1.2. 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

[5] General Responsibility Assignment Software Patterns, son patrones generales de software para asignación de responsabilidades, aunque se considera que más que patrones, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software.

código externo en los trabajos internos, permitiendo así más flexibilidad en el desarrollo de sistemas.

2.3.1.3. Patrones 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 los 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.

2.3.1.4. 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.

2.4. Análisis.

El análisis entra en el proceso de desarrollo de software como etapa decisiva, y uno de sus propósitos primarios es definir las clases y demás artefactos que intervienen en esta fase del RUP. Las primeras pautas para la implementación del sistema se definen en esta fase, de esta forma se definen las clases del análisis.

En el análisis se razona más sobre los aspectos internos del sistema. En el análisis se estructuran los requisitos de manera que nos facilite su comprensión, su preparación, su modificación, y en general su mantenimiento (Rumbaugh, et al., 2004)

En este flujo se generan artefactos como:

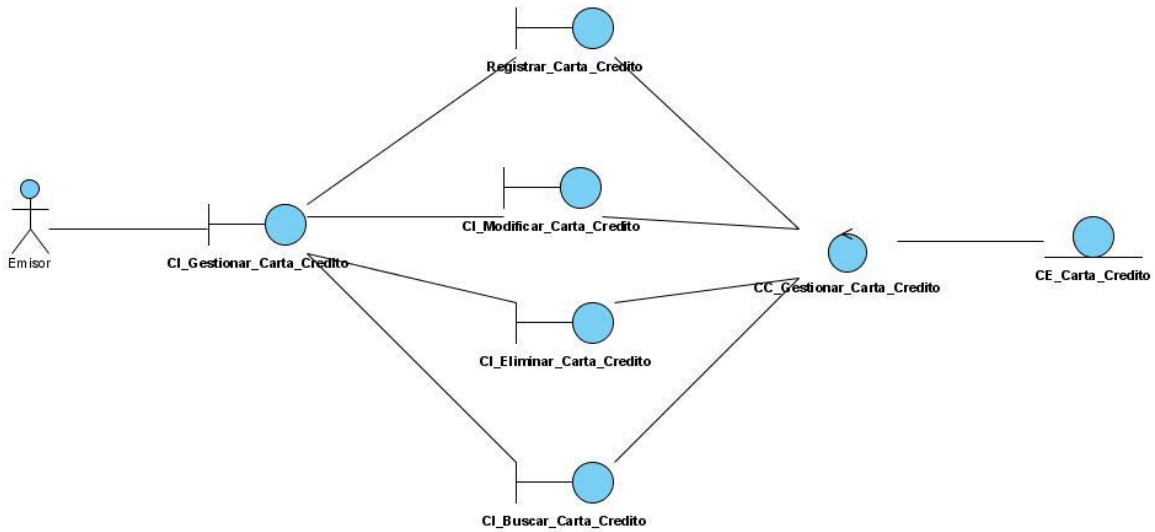
- *Diagrama de clases del análisis.*
- *Diagrama de colaboración.* **(Ver Anexos)**.

2.4.1. Modelo de análisis.

En el modelo del análisis se estructuran los requisitos de manera que nos facilite su comprensión, preparación, su modificación y su mantenimiento en general. Aquí se refinan los requisitos profundizando en el dominio de aplicación y se le asignan funcionalidades a un grupo de objetos.

2.4.1.1. Diagrama de clases del análisis

3 Gestionar Carta de Crédito



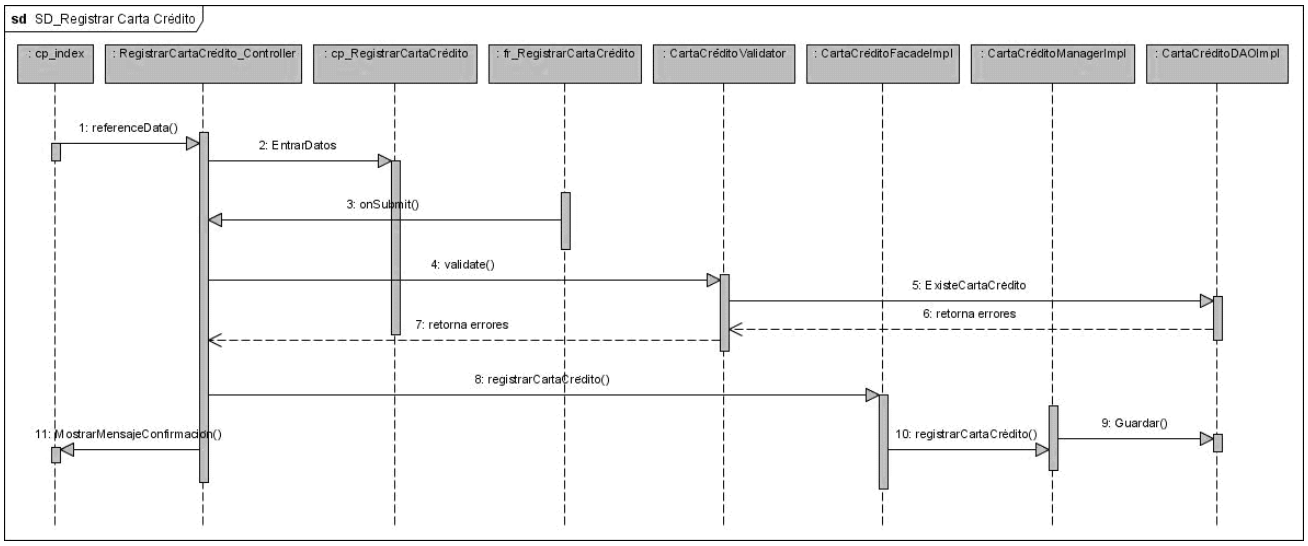
2.4.2. Diseño

El Modelo de Diseño es un modelo de objetos que adquiere una comprensión de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia y tecnología de interfaz de usuario. Además ayuda a descomponer los trabajos de implementación de partes mas manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo.

En esta etapa de diseño se obtienen: **(Ver Anexos)**.

- *Diagramas de secuencia*
- *Diagrama de clases del diseño*
- *Diagrama de clases de acceso a datos*.

2 Registrar Carta de Crédito.

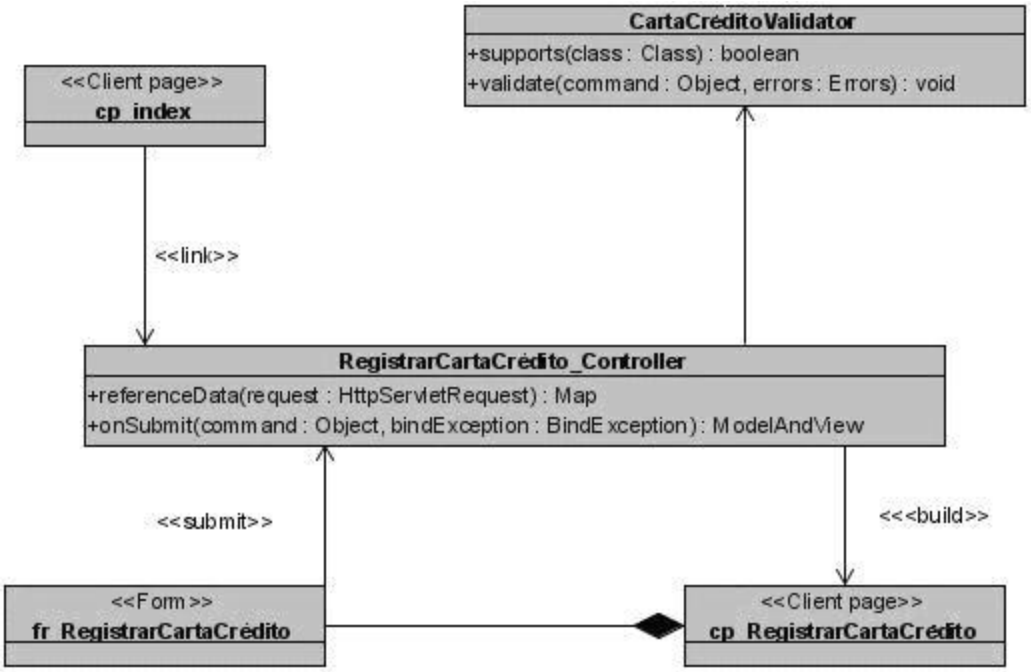


2.4.2.1. Modelo de Diseño

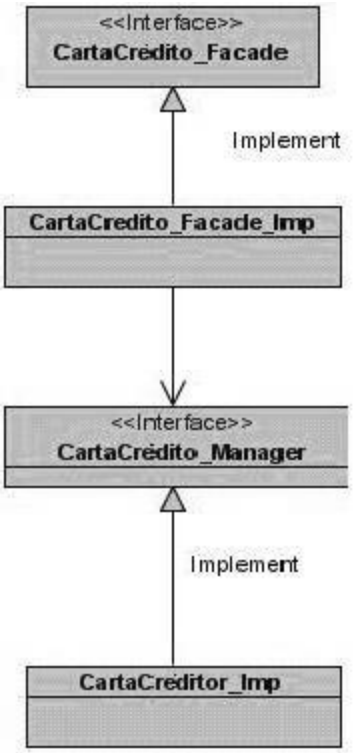
El Modelo de Diseño es un modelo de objetos que describe la realización de los casos de uso y al mismo tiempo constituye una abstracción del modelo de implementación y del código fuente.

2.4.2.2. Diagramas de clases del diseño.

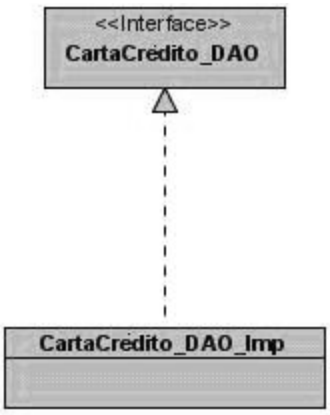
4 Registrar Carta de Crédito



5 Capa de Negocio Gestionar Carta de Crédito



6 Capa de Acceso a Datos Gestionar Carta de Crédito



2.5. Conclusiones parciales

En el capítulo se mostraron los artefactos generados en el flujo de trabajo de análisis y diseño, además se incluye una breve descripción de los mismos, con el objetivo de que se comprenda perfectamente los requisitos del software. Posibilitando la correcta transformación de los mismos a un diseño que indique como debe ser implementado el software. También se encuentra información referente a los patrones utilizados y a la arquitectura.

3. CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA.

3.1. Introducción

La medición es fundamental para cualquier disciplina de ingeniería, y la ingeniería del software no es una excepción. La medición nos permite tener una visión más profunda, proporcionando un mecanismo para la evaluación objetiva. (12) En el presente capítulo se aplican un conjunto de métricas de diseño orientado a objeto y se analizan los resultados para determinar la calidad del diseño.

3.2. Métricas para el Modelo de Diseño

Las métricas para diseño proporcionan al diseñador una mejor visión interna, ayudan a que el diseño evolucione a un nivel superior de calidad. Para la aplicación de estas métricas Berard [Laranjeira ´90] define cinco características fundamentales. 1. Localización. 2. Encapsulamiento. 3. Ocultamiento de la información. 4. Herencia. 5. Técnicas de abstracción de objetos.

3.2.1. 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.

3.2.1.1. 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 aplicaran dos al diseño propuesto.

3.2.1.1.1. Tamaño de Clase(TC)

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

- Total de operaciones, ya sean las de la clase o las heredadas de las clases padres o interfaces que implementen.
- Cantidad de atributos, al igual que el anterior, tanto los de ella, como los de los padres.
- Promedio de operaciones y atributos para el sistema completo.

7 Umbrales para TC

Nro. de operaciones y/o atributos	
TC	Umbral
Pequeño	<= 20
Medio	>20 y <=30
Grande	>30

Para evaluar las métricas son necesarios los valores de los umbrales. Las medidas o umbrales para los parámetros de calidad han sido una polémica a nivel mundial en el diseño de sistemas. Algunos especialistas plantean umbrales para estas métricas según se muestra en la tabla 1, estos fueron los aplicados en el diseño de este sistema.

8 Total de clases del diseño, cantidad de operaciones y atributos promedio

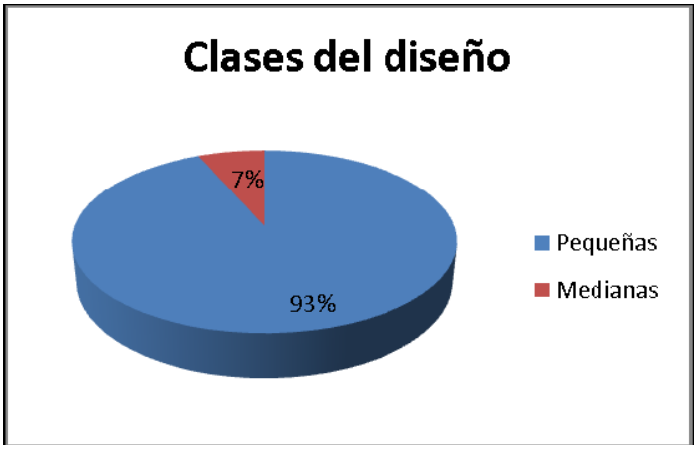
Total de Clases	**Operaciones**	**Atributos**
61	7.6	2.7

De acuerdo con los umbrales propuestos en la tabla 2 se obtuvo que para un total de 61 clases que tiene el diseño, 57 son de tamaño pequeño y 4 son de tamaño medio, como se muestra en la tabla 4, lo que representa el 7 % de clases medianas y el 93 % de clases pequeñas.

9 Tamaño de las clases de acuerdo al Umbral

Cantidad de clases	Umbral	Tamaño
57	<=20	Pequeño
4	>20<=30	Medio

10 Clases del diseño según TC



Estos valores demuestran que los indicadores de calidad, reutilización, implementación y responsabilidad no se ven afectados.

3.2.1.1.2. Número de Operaciones redefinidas por una subclase (NOR)

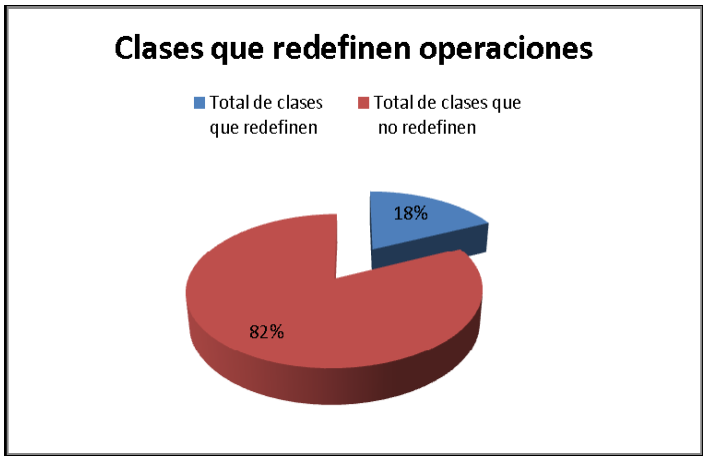
Para aplicar esta métrica es necesario conocer el número de clases que redefinen operaciones heredadas de otras clases.

11 Total de clases que redefinen operaciones

Total de clases	Total de clases que redefinen operaciones
61	11

Si se calcula el por ciento que representa el total de clases que redefinen funciones, nos da como resultado que el 18 % de las clases existentes en el diseño redefinen operaciones.

12 Clases que redefinen operaciones



3.2.1.2. Familia de métricas propuestas por Chidamber & Kemerer

Los conjuntos de métricas propuestos por Chidamber y Kemerer han sido uno de los más referenciados, las que son conocidas normalmente como la serie de métricas CK. Chidamber y Kemerer proponen seis métricas basadas en clases para sistemas orientados a objetos:

- Métodos ponderados por clase
- Profundidad árbol de herencia
- Número de descendientes
- Acoplamiento entre clases
- Respuesta para una clase
- Carencia de cohesión en los métodos

Estas métricas permiten conocer hasta qué punto están bien definidas las clases y el sistema, lo cual tiene un impacto directo en la mantenibilidad del mismo, tanto por la comprensión de lo desarrollado como por la dificultad de modificarlo con éxito. De estas seis se aplicaran dos al diseño propuesto.

3.2.1.2.1. Árbol de profundidad de herencia (APH).

Esta métrica está definida como la longitud máxima desde el nodo hasta la raíz del árbol. A medida que crece el APH, es más probable que las clases de niveles inferiores hereden muchos métodos. Esto da lugar a posibles dificultades cuando se intenta predecir el comportamiento de una clase. Una jerarquía de clases profunda (con un valor grande de APH) lleva también a una mayor complejidad de diseño. Por el lado positivo, los valores grandes de APH implican que se pueden reutilizar muchos métodos, lo que debe ser considerado como un elemento a favor de la mantenibilidad. En la tabla 6 se muestran los umbrales o medidas recomendados por algunos especialistas, estos fueron los aplicados en el diseño de este sistema.

13 Umbral para APH

Nivel de Jerarquía	
APH	**Umbral**
Sencilla	**<=5**
Compleja	**>5**

Al aplicar esta métrica al diseño propuesto, según los umbrales definidos, se obtiene como resultado que el APH presenta valor 4 como se muestra en la tabla 7, pues solo hay presencia de herencia entre las clases y las interfaces de las cuales heredan sus funciones y las redefinen, lo cual está dentro del umbral definido para determinar que el diseño es sencillo, por lo que evita que exista algún problema si se desea predecir el comportamiento de una clase y no es difícil de dar mantenimiento.

14 Resultados aplicando APH

Total de clases	**APH**
61	4

3.2.1.2.2. Número de descendientes (NDD)

Las subclases que son inmediatamente subordinadas a una clase son denominadas descendientes. A medida que crece el número de descendientes, se incrementa la reutilización, pero también es cierto, que a medida que crece NDD, la abstracción representada por la clase predecesora puede verse diluida y existe la posibilidad de que algunos de los descendientes no sean realmente miembros propios de la clase predecesora.

15 Cantidad de descendientes por clases

Número de clases	Número de descendientes
21	1
1	2

Como se puede observar en la tabla 15, existen 21 clases con NDD: 1 y una con 2 descendientes de un total de 61 clases. Los valores de NDD alcanzados para el diseño propuesto son pequeños lo que permite que cada descendiente sea realmente miembro de la clase predecesora y también se reduce la cantidad de pruebas necesarias para preparar cada uno de los miembros de la jerarquía.

3.3. Conclusiones parciales

En este capítulo se aplicaron métricas para la evaluación del diseño obtenido, dándole solución al objetivo planteado. El diseño propuesto no presenta una alta complejidad permitiendo que las pruebas no sean complejas y que el resto de los módulos puedan ser integrados sin muchas dificultades, además presenta un bajo acoplamiento pues la profundidad de los niveles de herencia están acorde con los umbrales. Se puede concluir que el diseño obtenido posee una calidad aceptable, facilitando la continuación eficiente del desarrollo en etapas posteriores.

4. CONCLUSIONES GENERALES

Una vez concluido el estudio de los procesos de Cartas de Créditos, se realizo el análisis y diseño de dicho subsistema, lo cual permitirá su posterior implementación. Como resultados del análisis se obtuvieron los diagramas de colaboración y los de las clases del análisis. Para la realización del diseño se tuvieron en cuenta algunos patrones para modelar el sistema, lo que permitió generar artefactos empleando buenas prácticas, necesarias para desarrollar un software con mayor robustez. Entre los artefactos generados en el diseño se pueden mencionar: diagramas de secuencia, diagramas de clases del diseño y el modelo de datos, imprescindibles para el buen desarrollo de una solución informática.

De forma general los resultados obtenidos a partir del análisis y diseño del sistema son positivos, tomando como referencia la aplicación de métodos y métricas para validar la solución del mismo.

5. RECOMENDACIONES

- Profundizar en el estudio del proceso de Cartas de Créditos en otras entidades bancarias para una futura versión ampliada del sistema.

- Realizar los restantes flujos de trabajo que propone la metodología utilizada, RUP, llegando a implementar las funcionalidades que hemos propuesto para el subsistema de Cartas de Créditos del Proyecto Modernización del Sistema Bancario Cubano, lo cual tributará a un incremento de la eficiencia y calidad en el Banco Nacional de Cuba.

6. BIBLIOGRAFÍA

Agüero, M. (Octubre de 2007). Introducción a Spring Framework. Buenos Aires, Argentina.

Cota, A. (1994). *Ingenieria de Software. Soluciones Avanzadas.*

González, H. S. (21 de 3 de 2003). Manual Hibernate.

Hernández Toirac, L. M., & Stalyn Perez, A. (junio de 2008). DEFINICIÓN DE LOS REQUERIMIENTOS FUNCIONALES DEL MÓDULO EMISIÓN DE CARTAS DE CRÉDITO DEL PROYECTO BANCO NACIONAL. La Habana, Cuba.

Jacobson, I. (1998). *Appying UML in The Unified Process.*

Lewis, G. (1994). *What is Software Engineering?*

Rueda Chacon, J. C. (2006). APLICACIÓN DE LA METODOLOGÍA RUP PARA EL DESARROLLORÁPIDO DE APLICACIONES BASADO EN EL ESTÁNDAR J2EE. Guatemala.

S. Pressman, R. (2002). *Ingeniería de Software. Un enfoque practico.* (5ta ed.).

Solís., M. C. Una explicación de la programación extrema (XP). *V Encuentro usuarios xBase 2003 MADRID.*

UCI. (2008). *EVA.* Recuperado el 25 de 3 de 2009, de Entorno Virtual de Aprendizaje: http://eva.uci.cu/file.php/259/CURSO_2008-2009/Materiales_Complementarios/Semana_1/CP/Diagramas_de_Interaccion.ppt

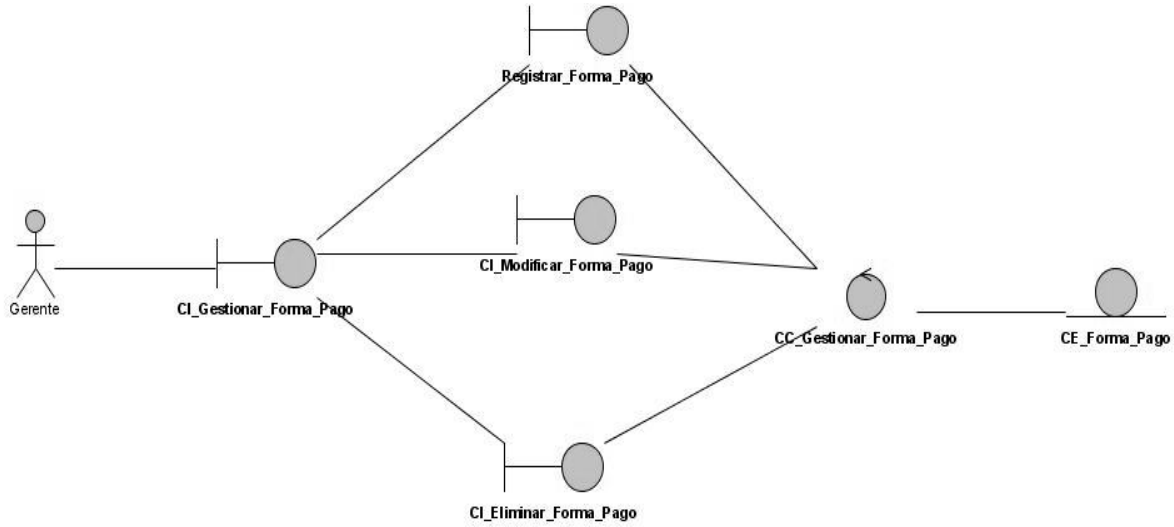
UCI. (2008). *EVA.* Recuperado el 25 de 3 de 2009, de Entorno Virtual de Aprendizaje: http://eva.uci.cu/file.php/259/CURSO_2008-2009/Materiales_Complementarios/Semana_1/CP/DIAGRAMA_DE_SECUENCIA.pdf

UCI. (2008). *EVA.* Recuperado el 12 de 2 de 2009, de Entorno Virtual de Aprendizaje: http://eva.uci.cu/file.php/102/Curso_2008-2009/Materiales_Basicos/Materiales_Basicos_Conf_7/Conf_7_FT_Analisis_y_Diseno.doc

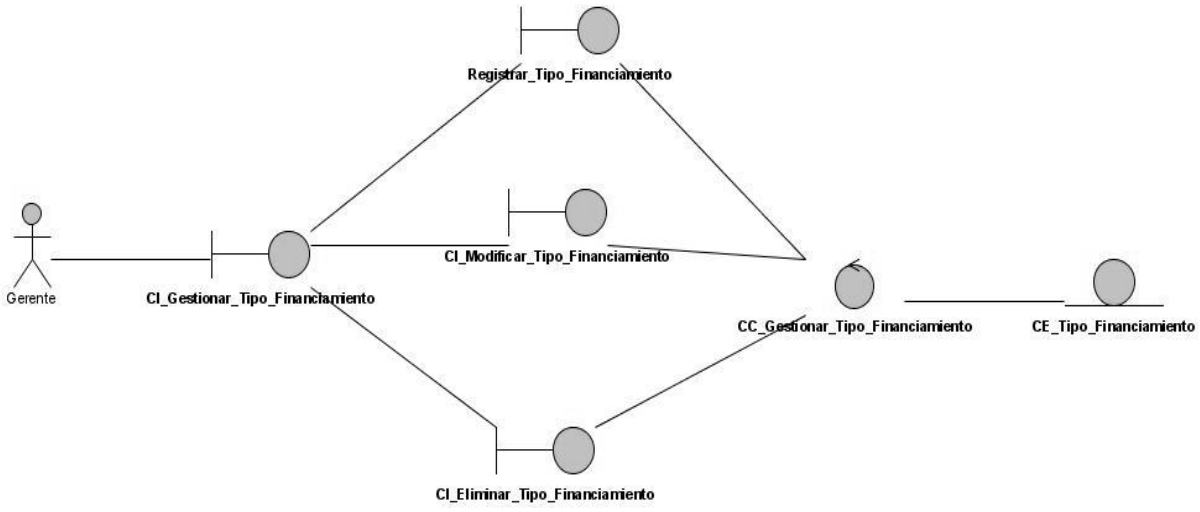
7. ANEXOS

7.1. Diagramas de clases del análisis

16 Gestionar Forma de Pago

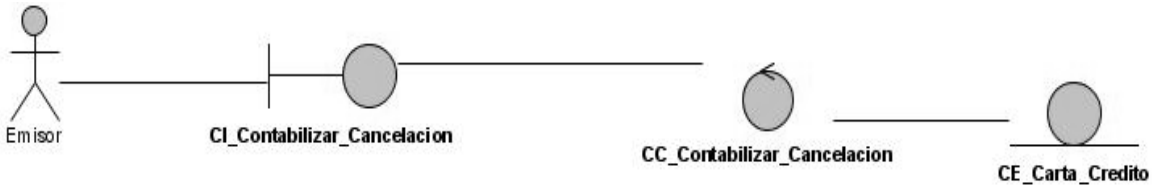


17 Gestionar Tipo de Financiamiento

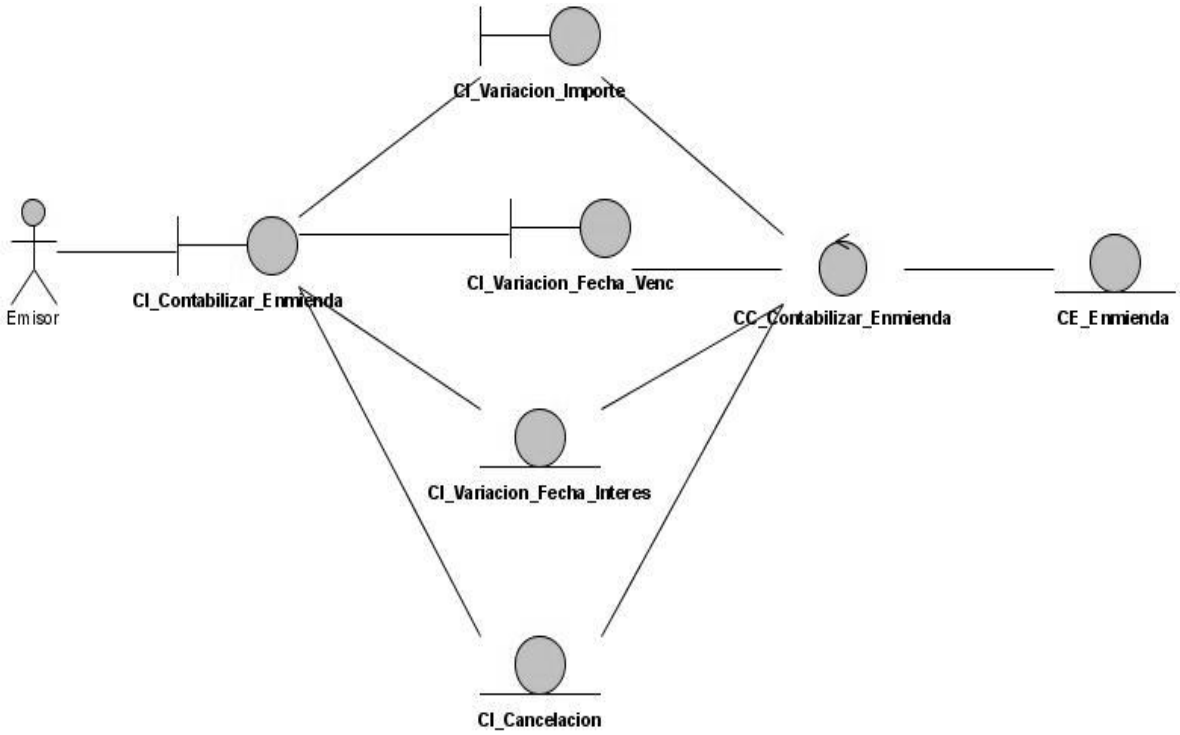


18 Contabilizar Apertura de carta de Crédito

19 Contabilizar Cancelación

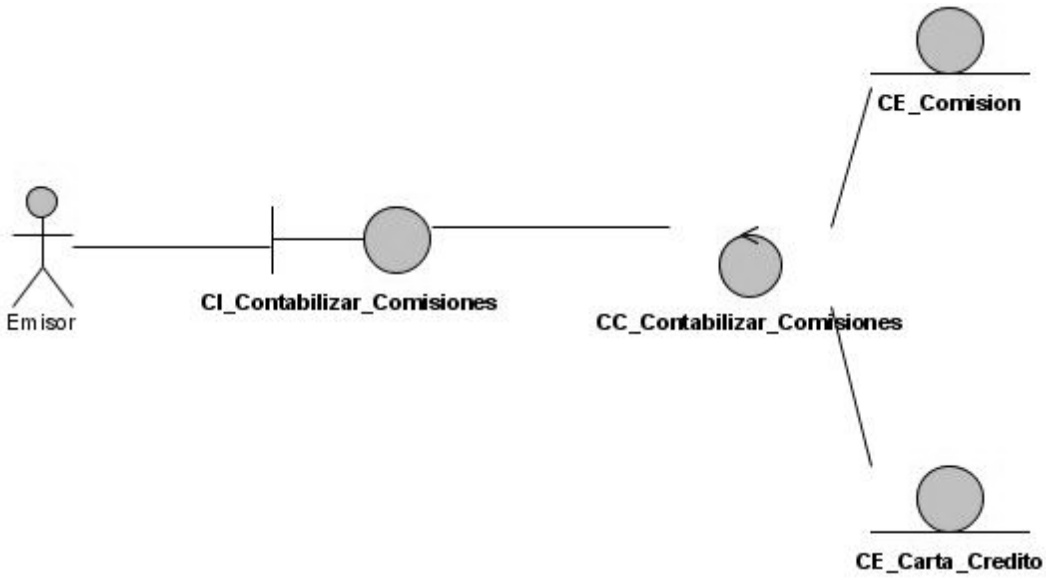


20 Contabilizar Enmienda

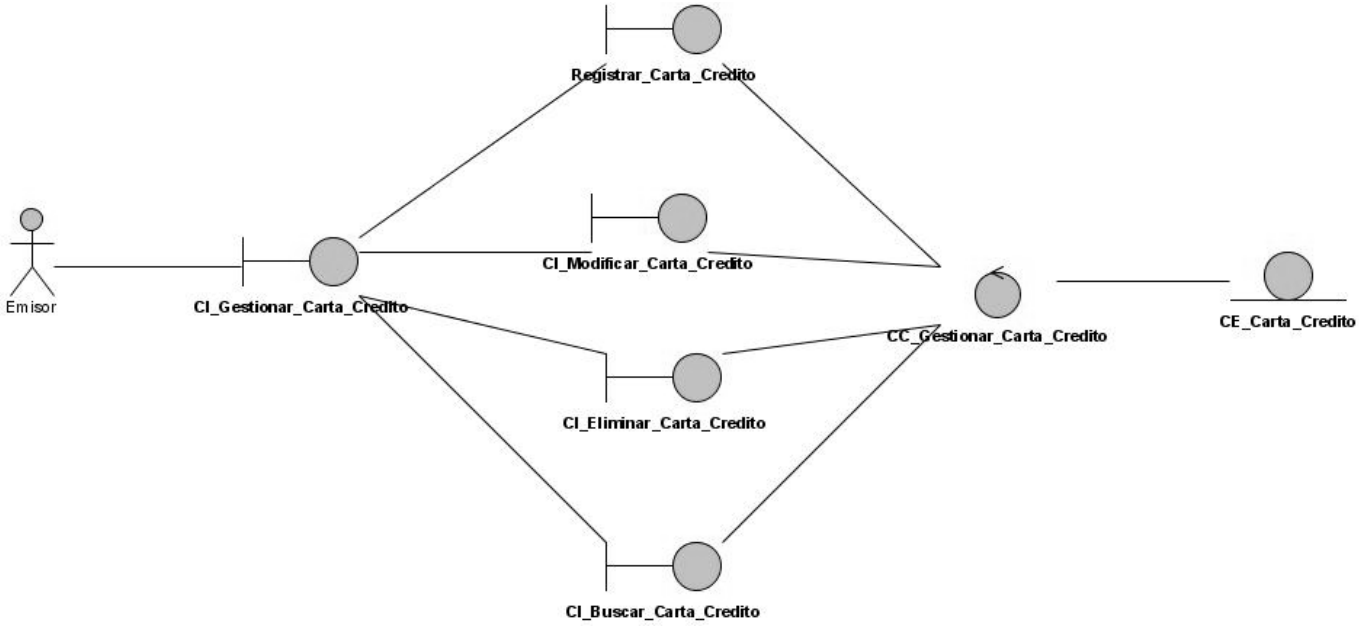


21 Contabilizar Pago

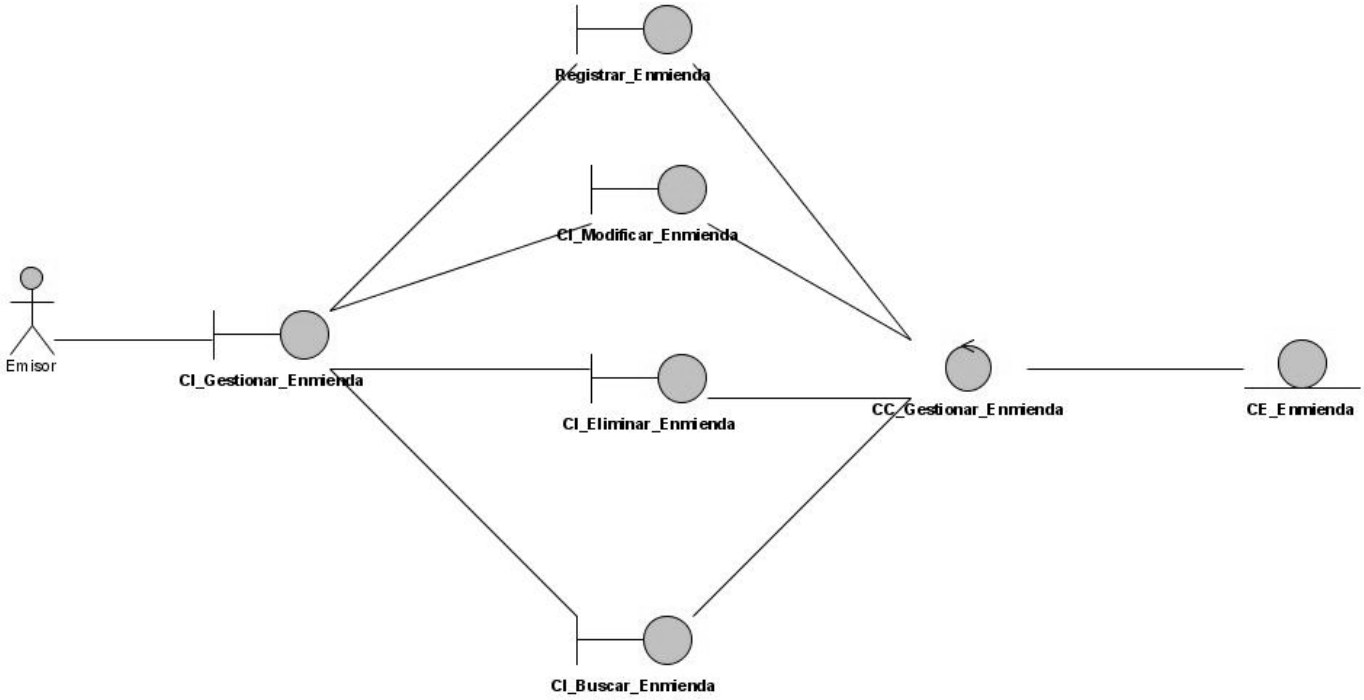
22 Contabilizar Comisiones



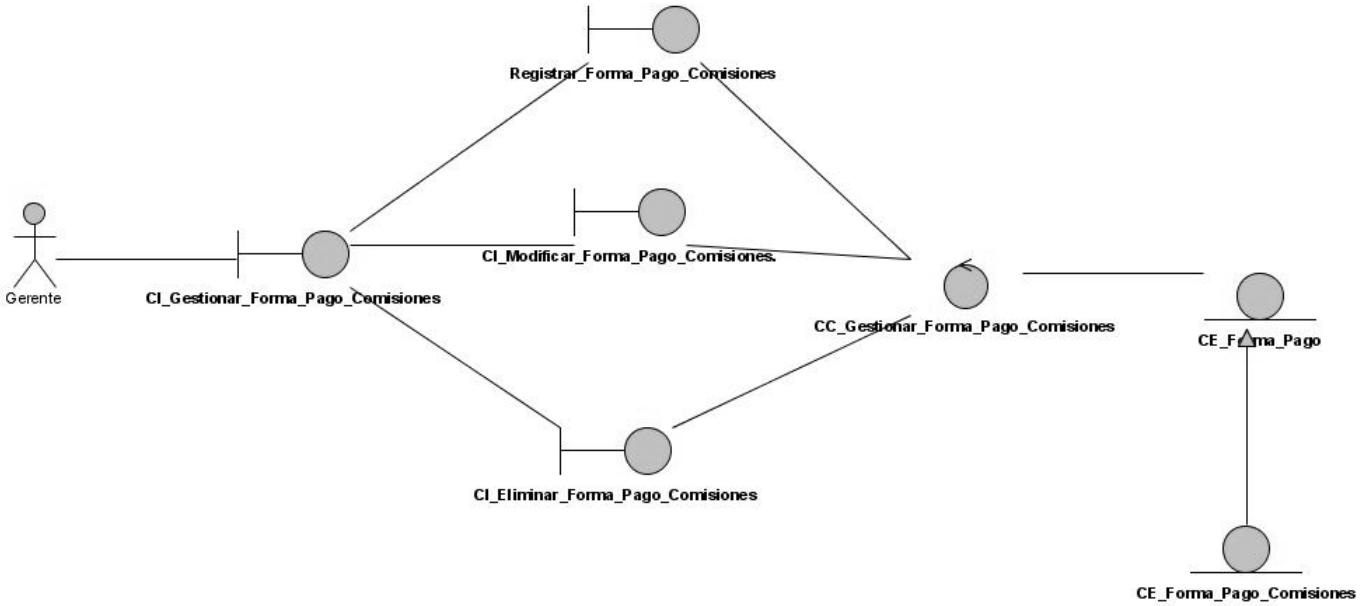
23 Gestionar Carta de Crédito



24 Gestionar Enmienda



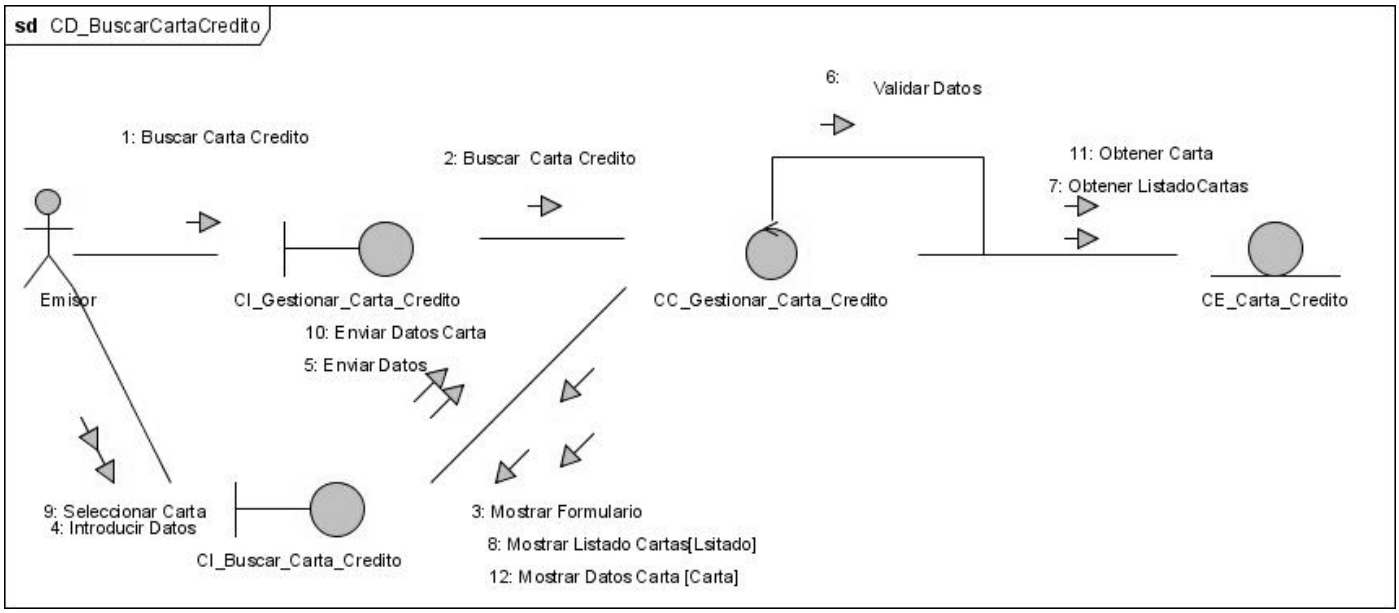
25 Gestionar Forma de Pago por Comisiones



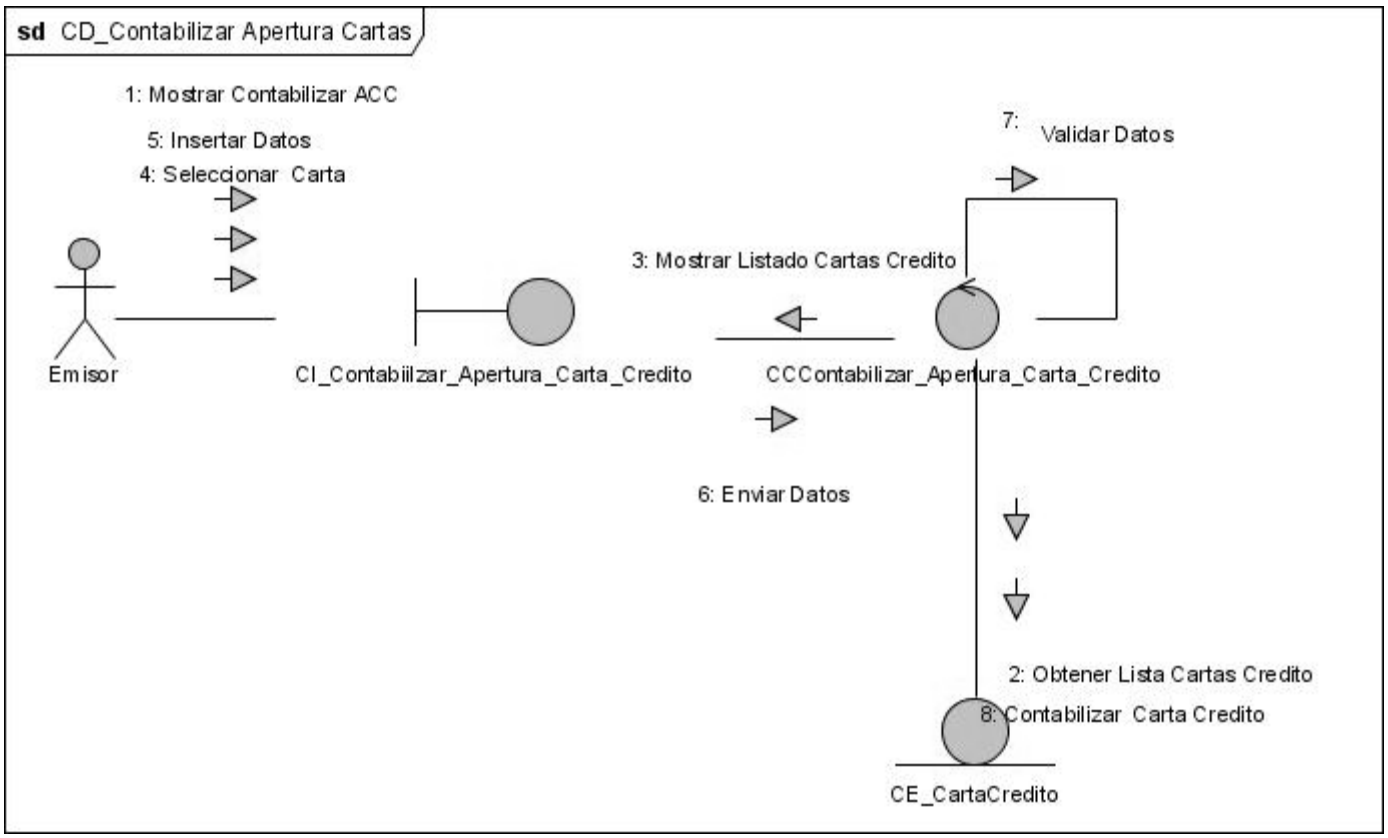
26 listar Cartas de Crédito por Vencer

7.1.1. Diagramas de Colaboración

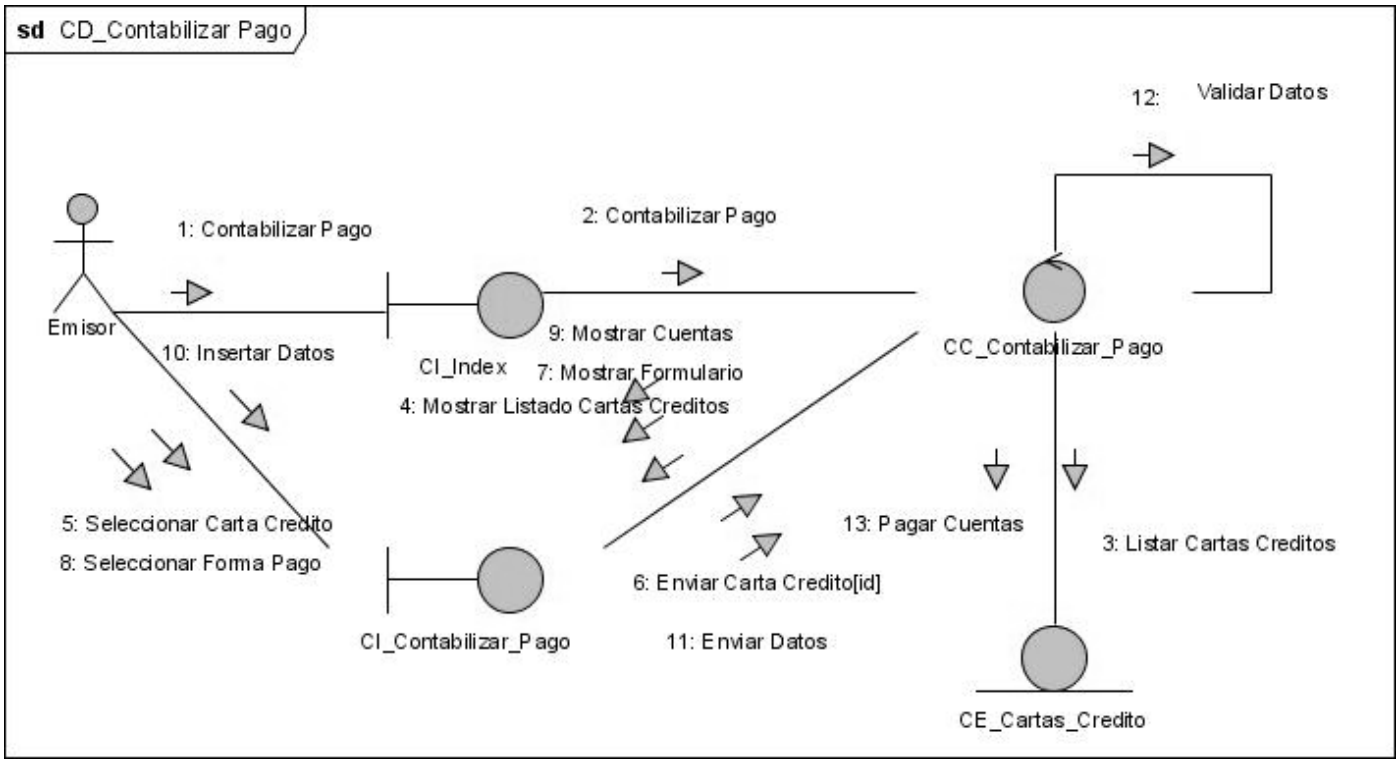
27 Gestionar Carta de Crédito



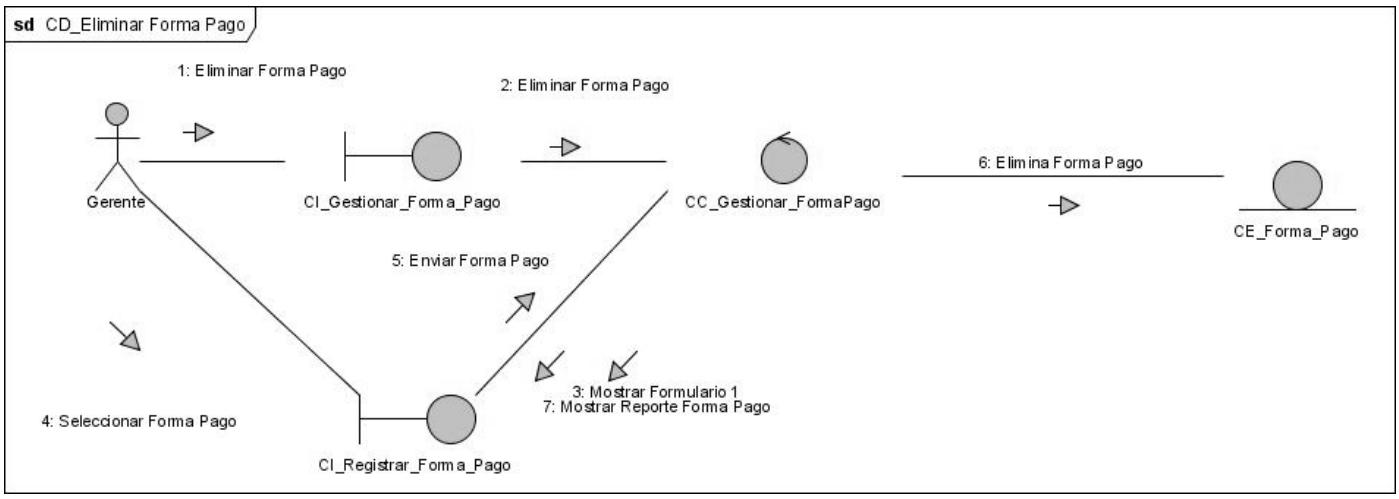
28 Contabilizar Apertura de Carta de Crédito



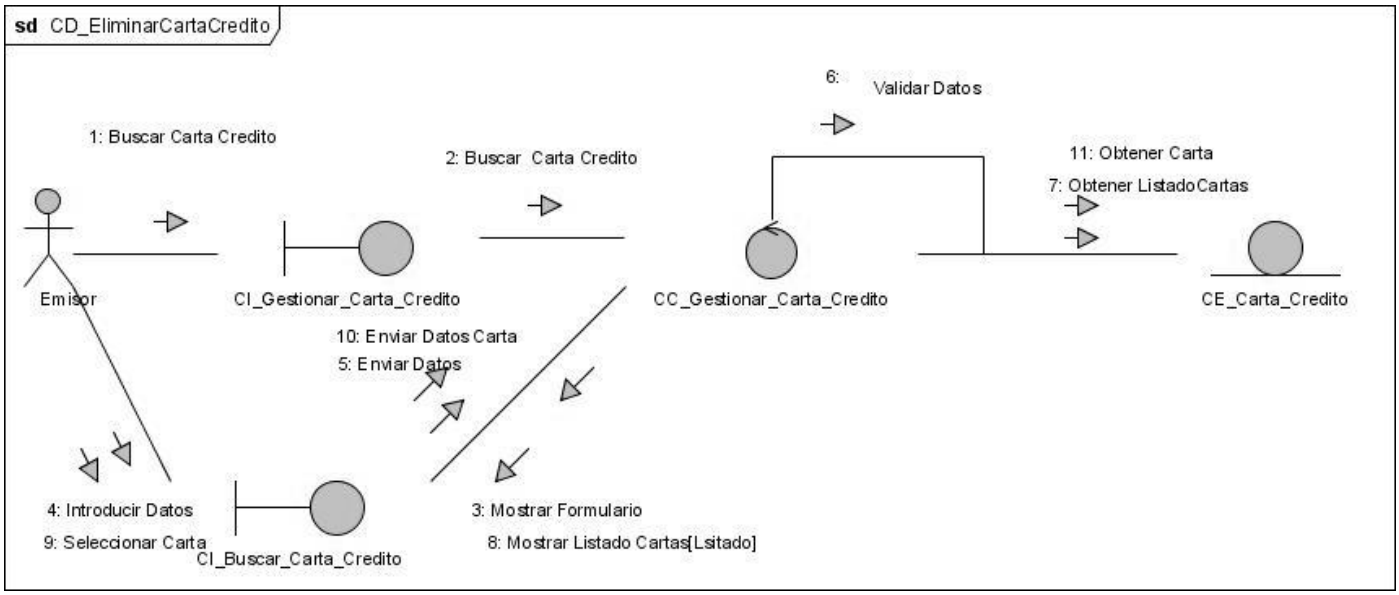
29 Contabilizar Pago



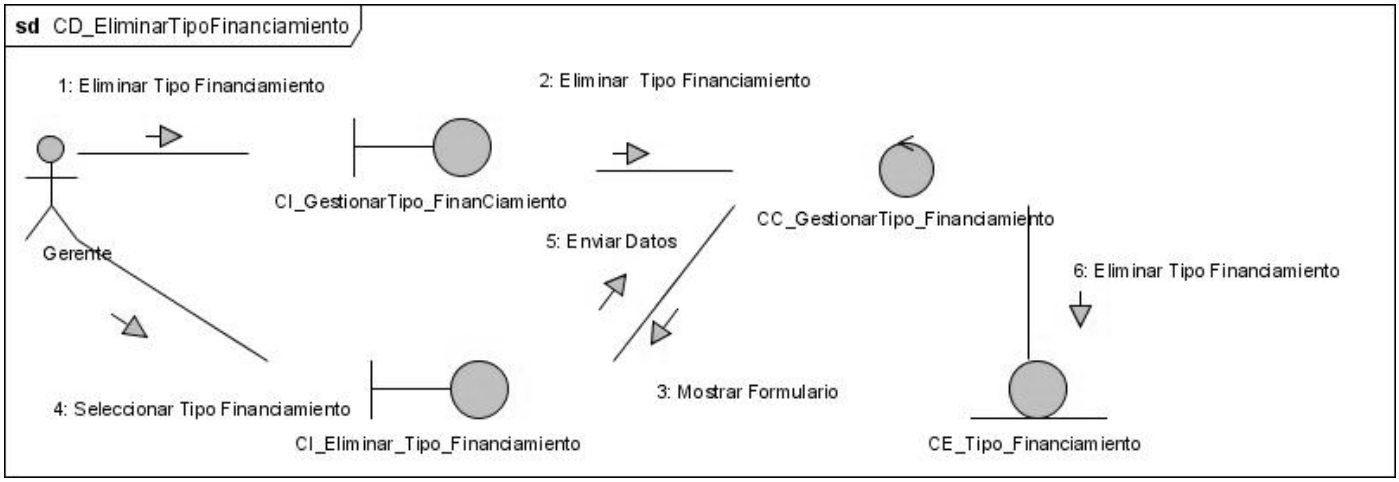
30 Eliminar Forma de Pago



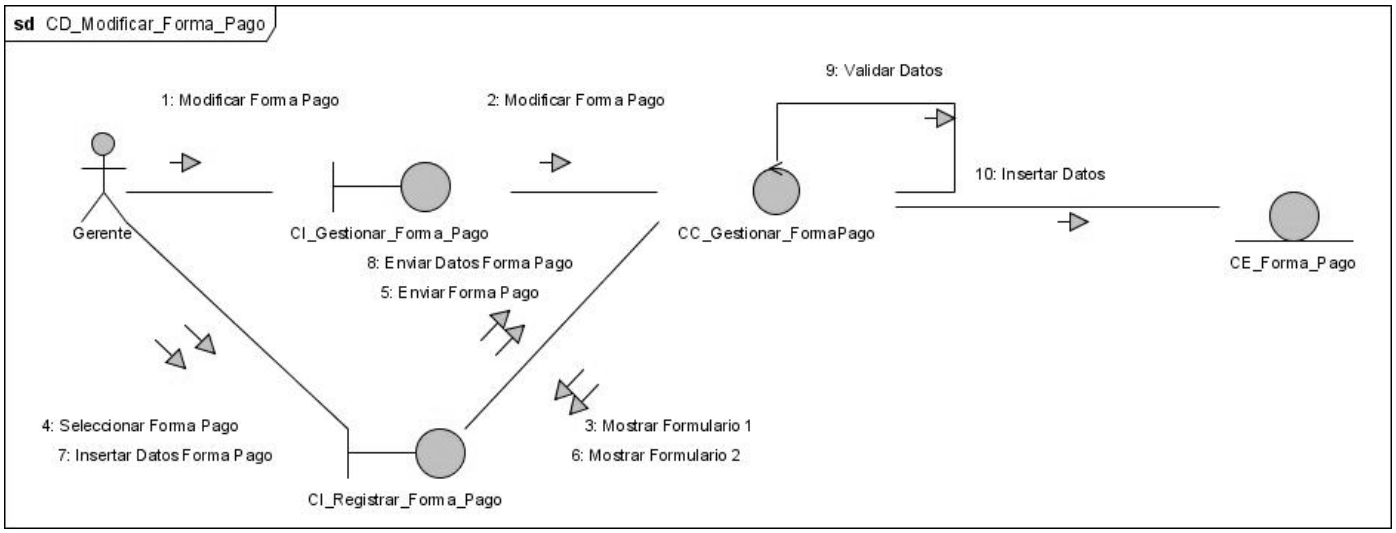
31 Eliminar Carta de Crédito



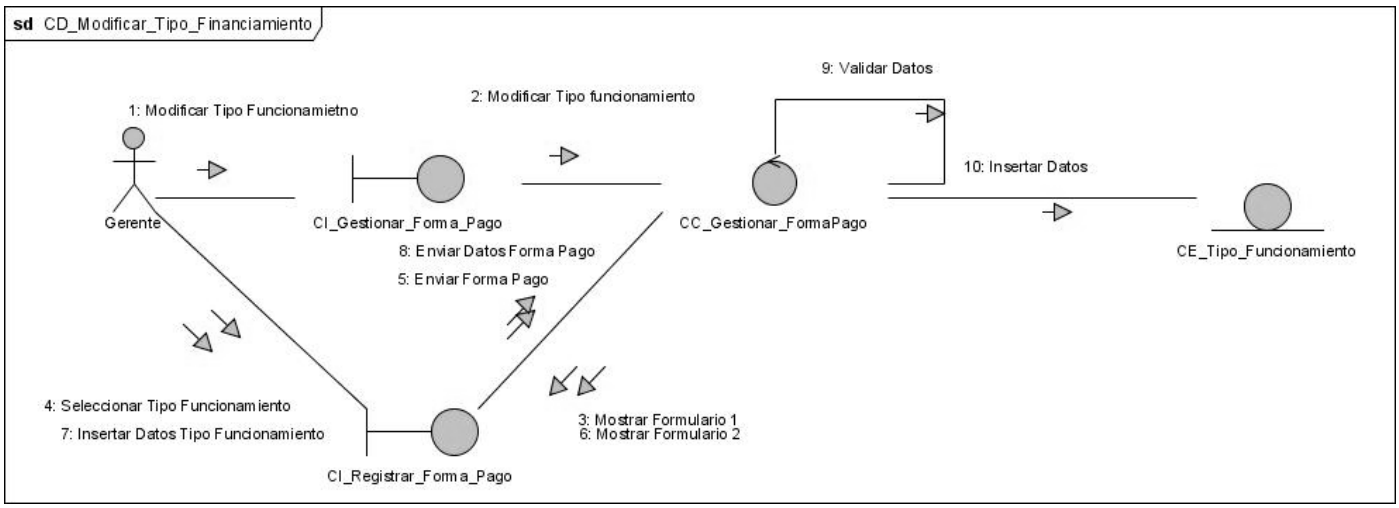
32 Eliminar Tipo de Financiamiento



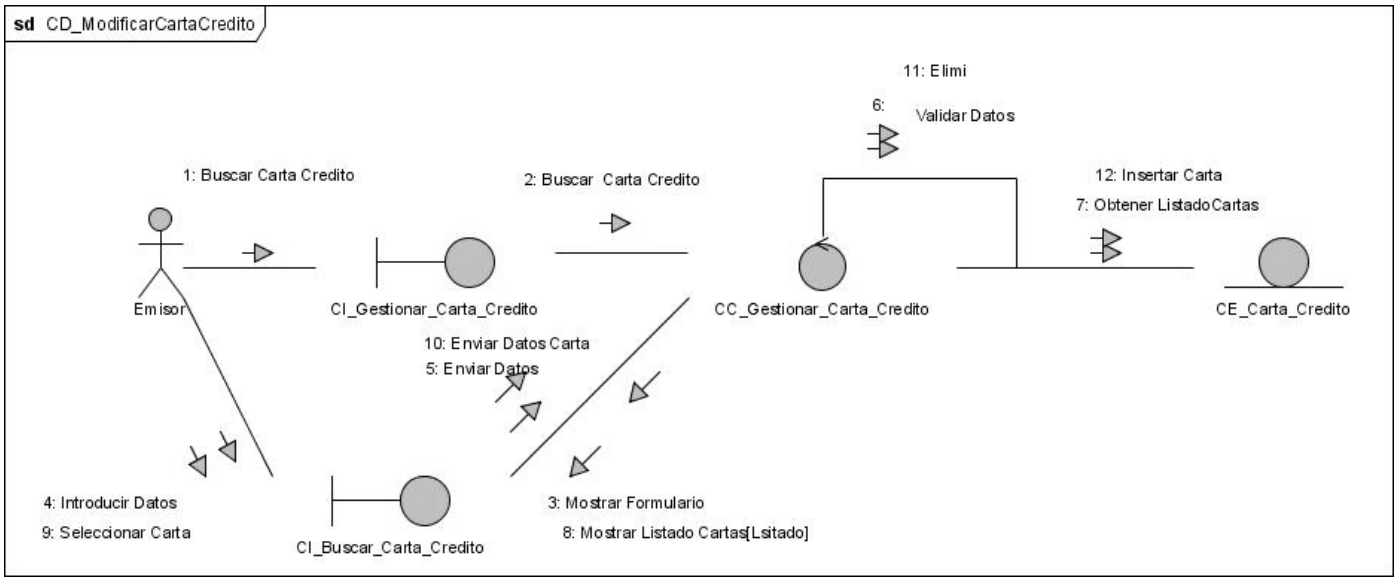
33 Modificar Forma de Pago



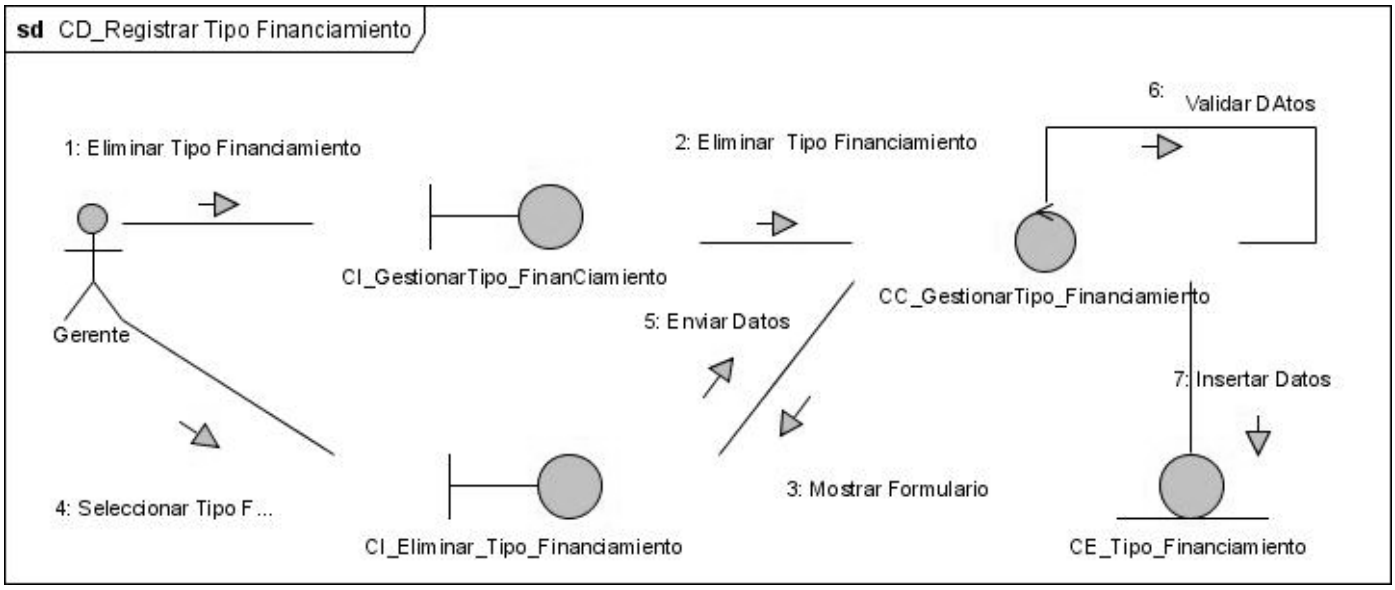
34 Modificar Tipo de Financiamiento



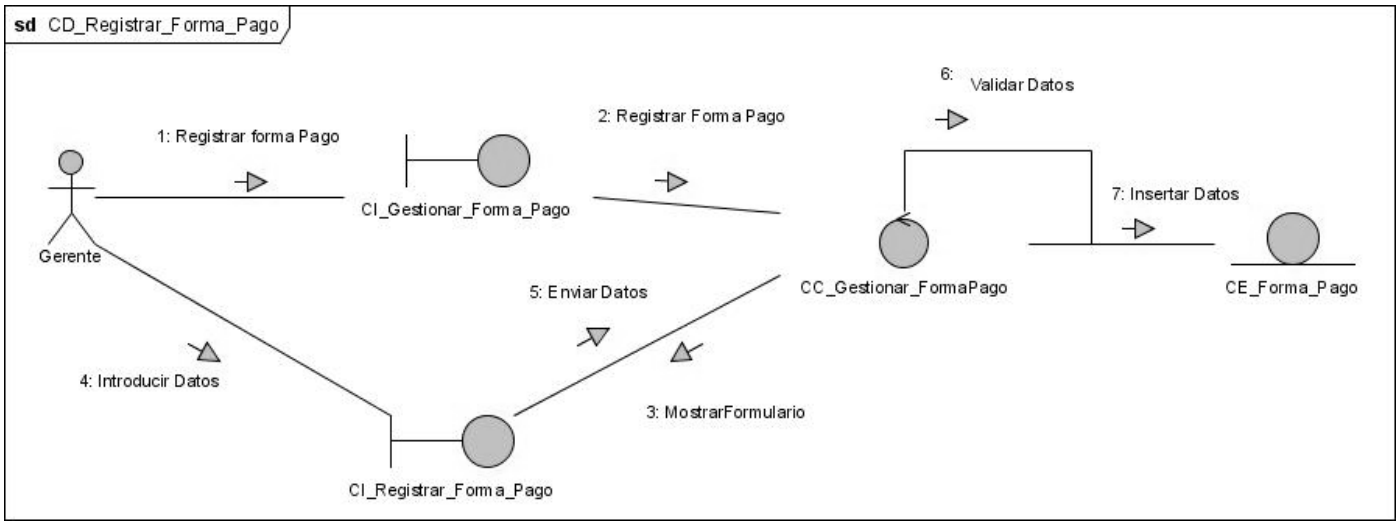
35 Modificar carta de Crédito



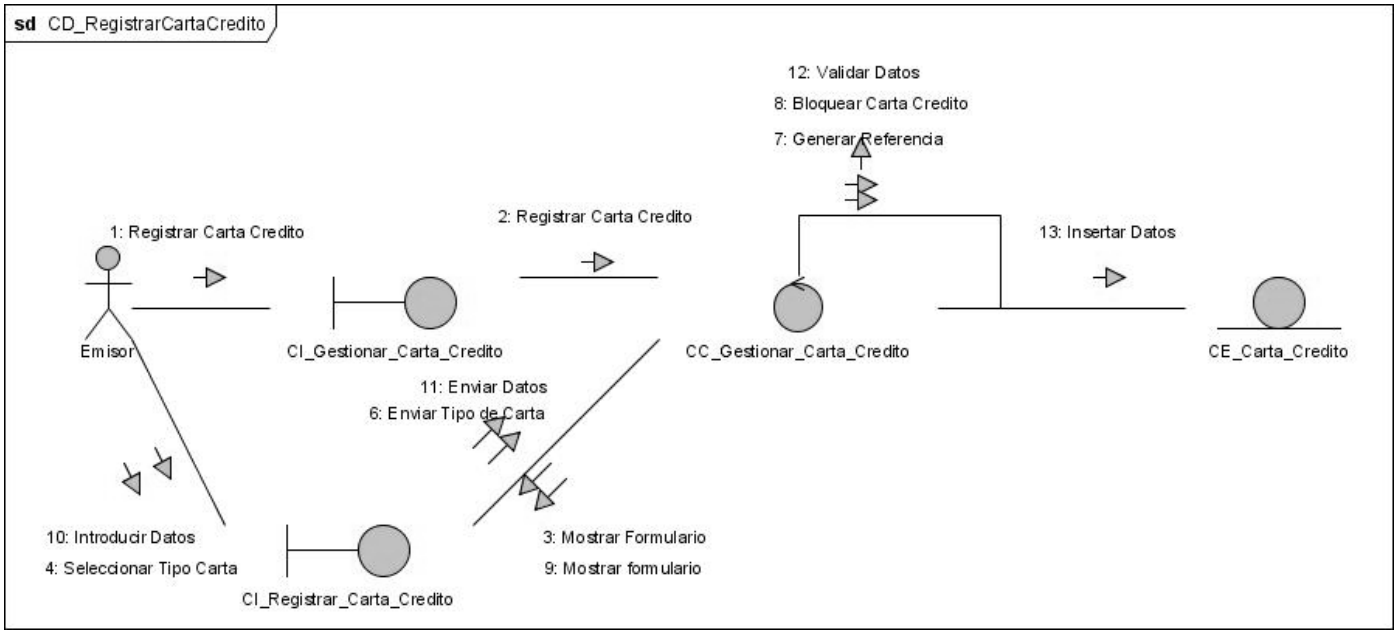
36 Registrar Tipo de Financiamiento



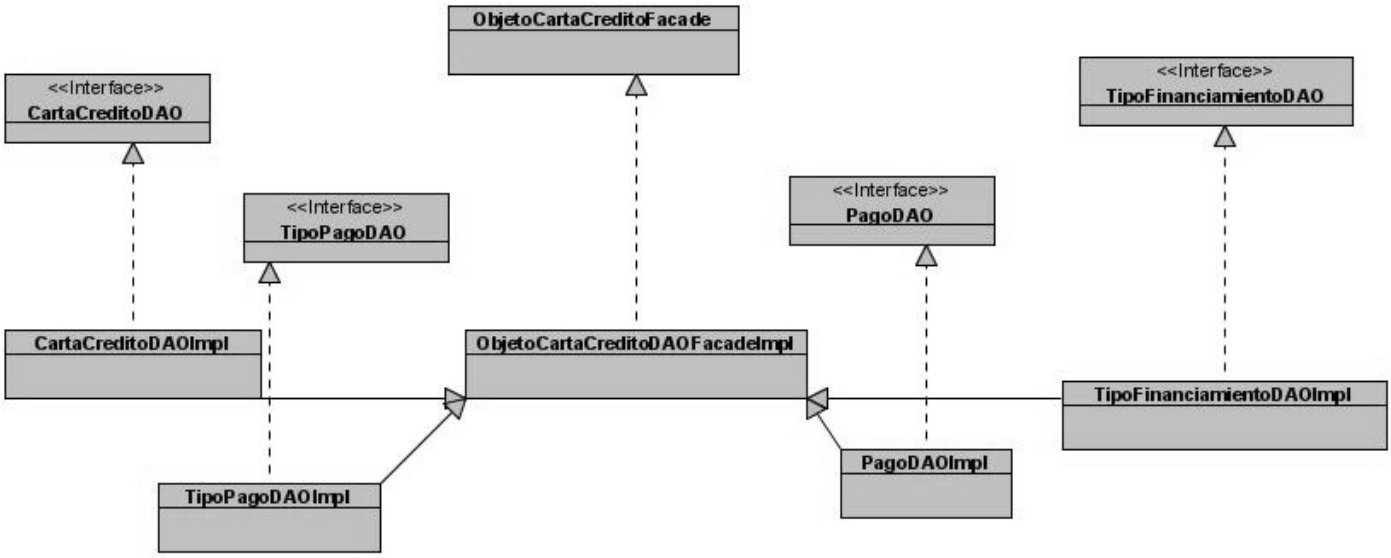
37 Registrar Forma de Pago



38 Registrar Carta de Crédito

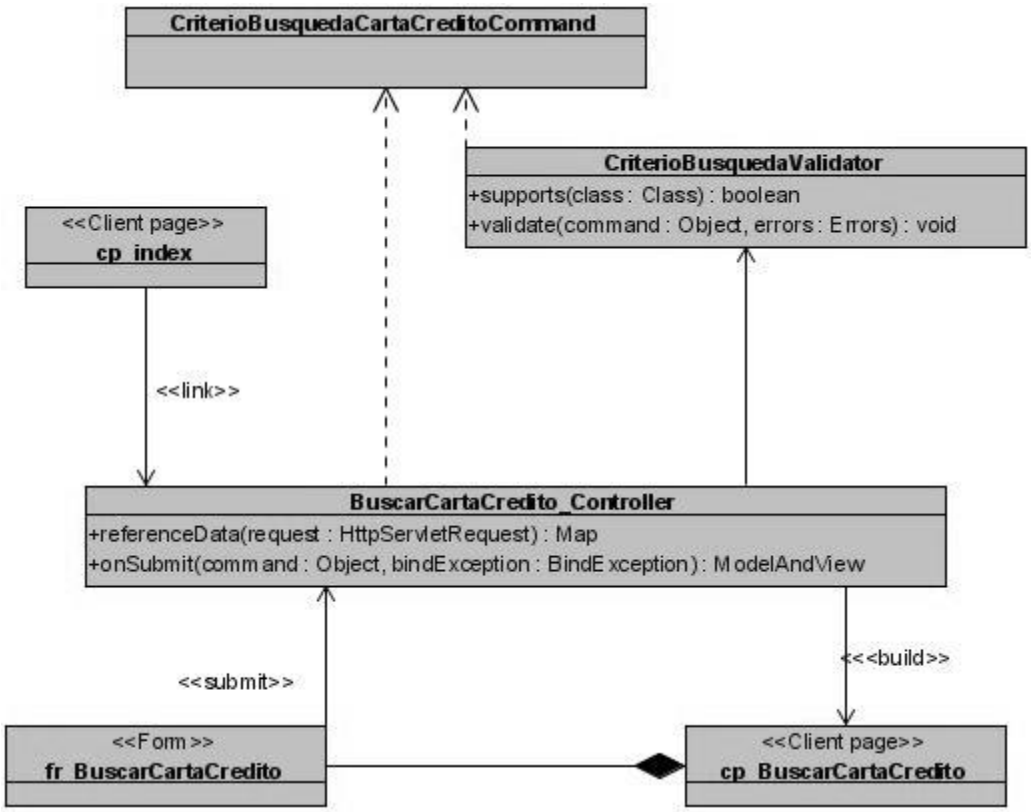


39 Clases de Acceso a Datos

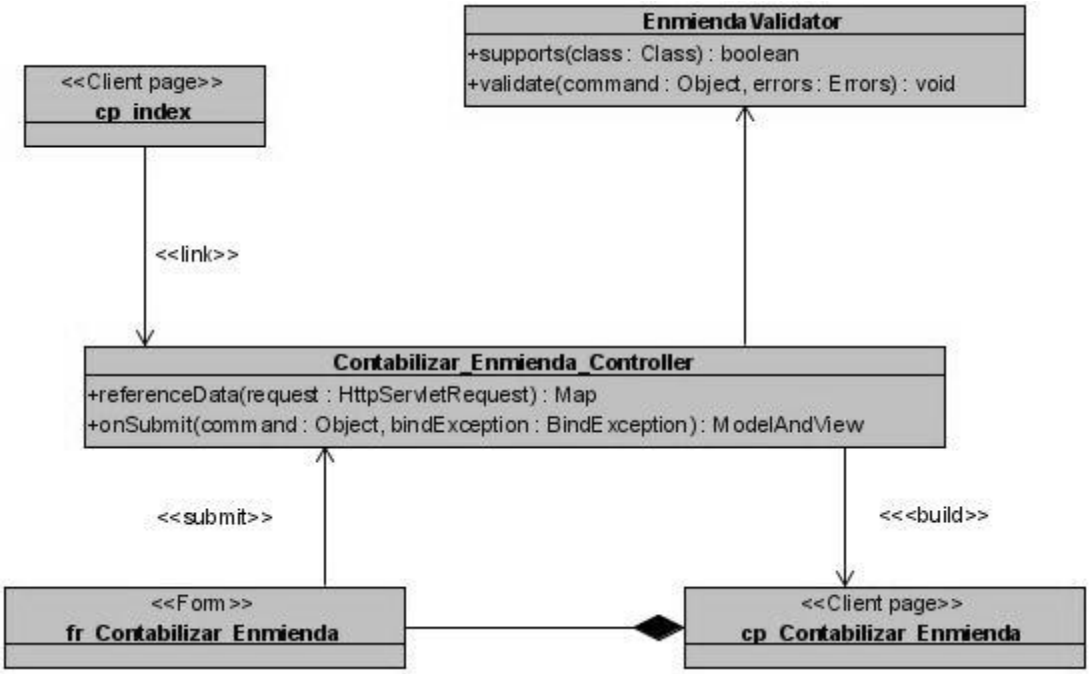


7.2. Diagramas de clases del diseño

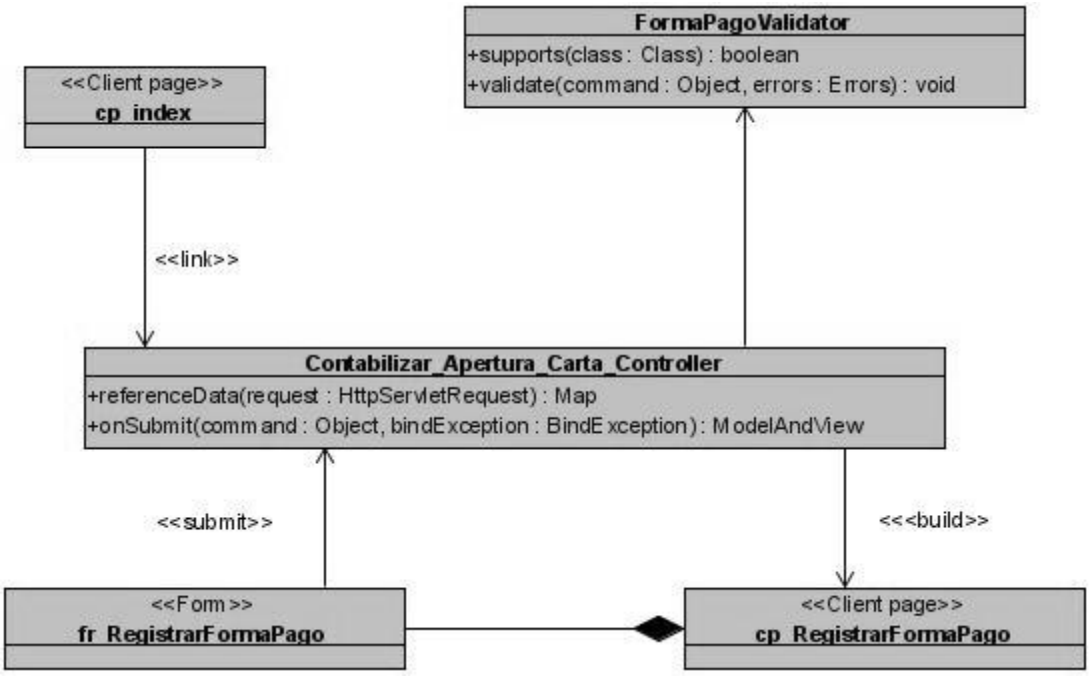
40 Buscar Carta de Crédito



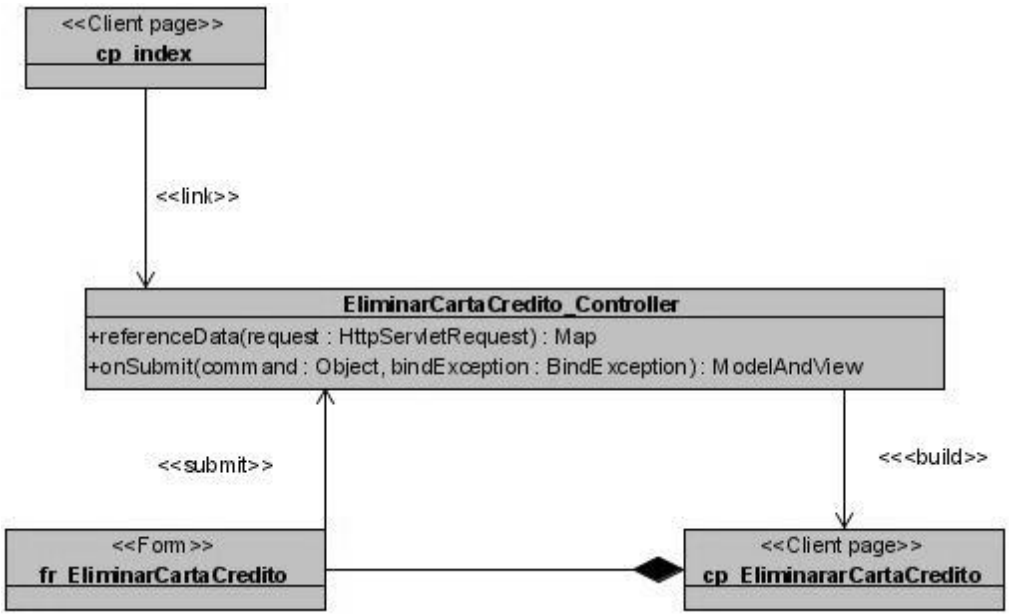
41 Contabilizar Enmienda



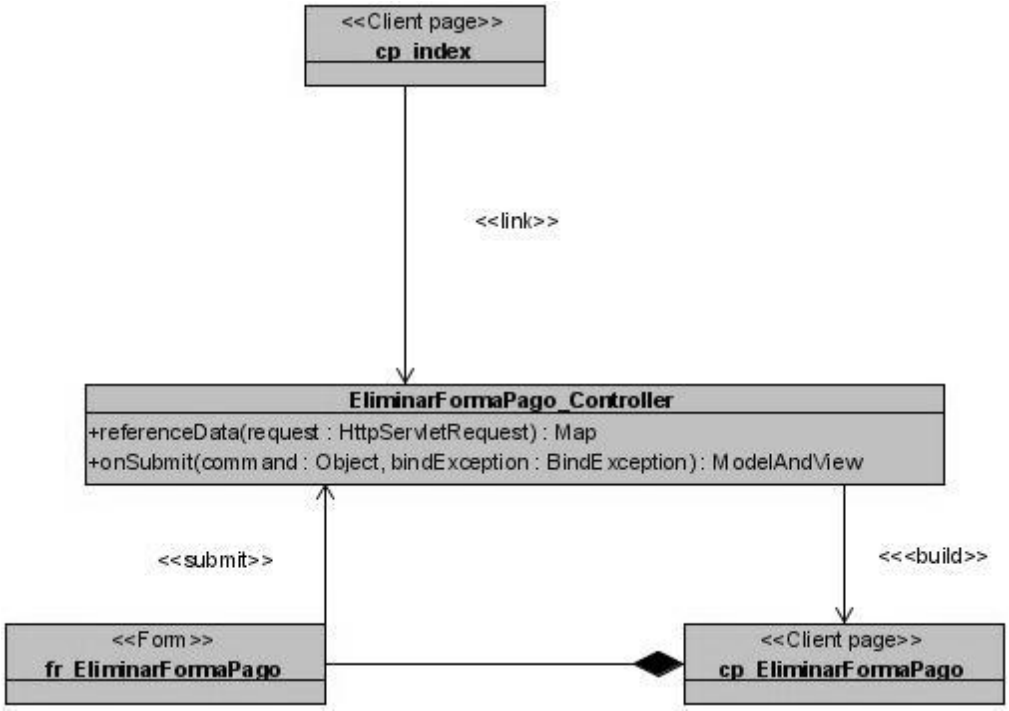
42 Contabilizar Apertura de Carta de Crédito



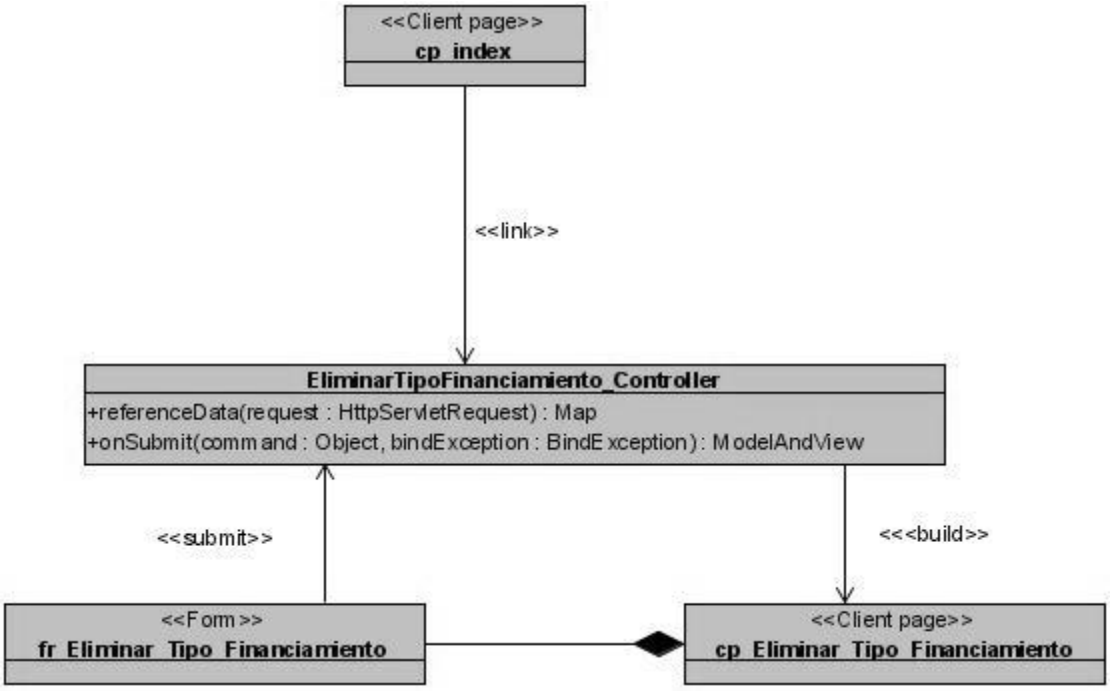
43 Eliminar Carta de Crédito



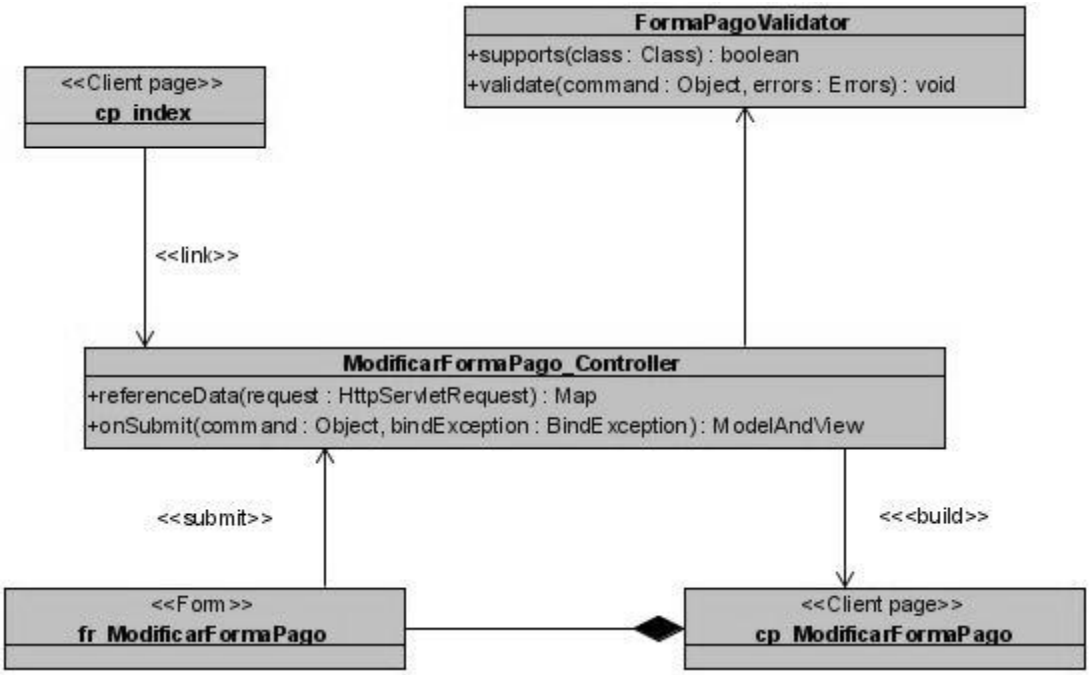
44 Eliminar Forma de Pago



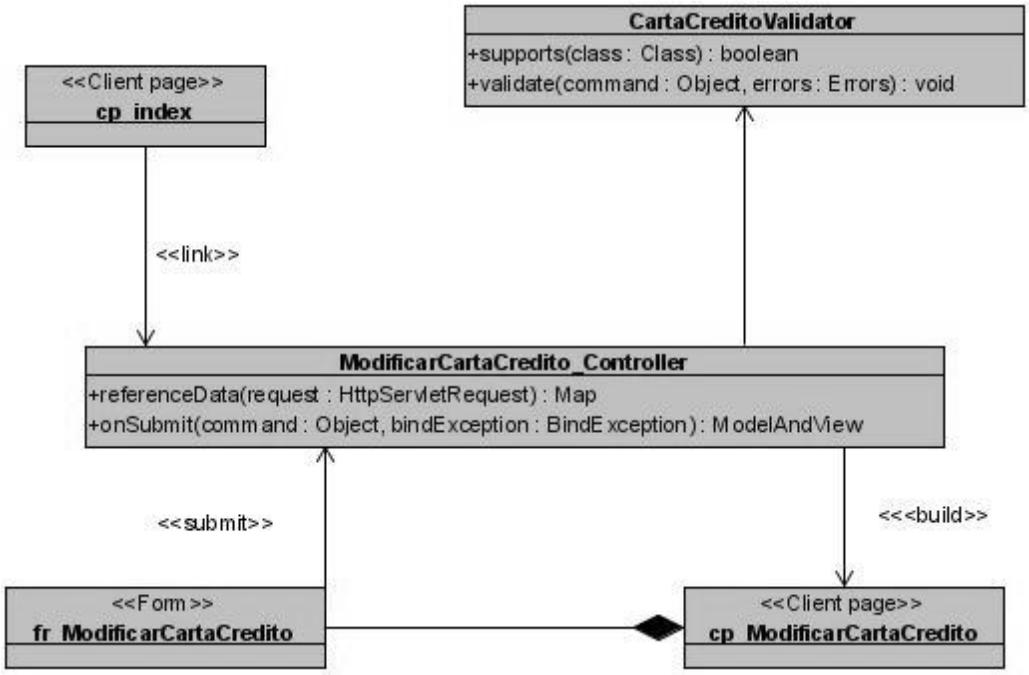
45 Eliminar Tipo de Financiamiento



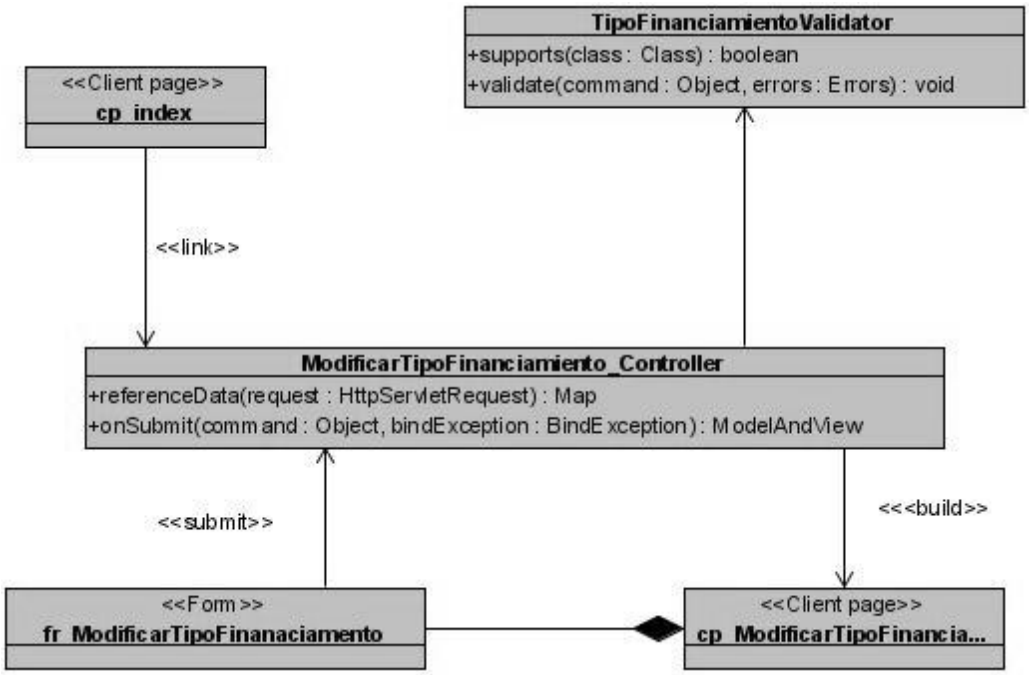
46 Modificar Forma de Pago



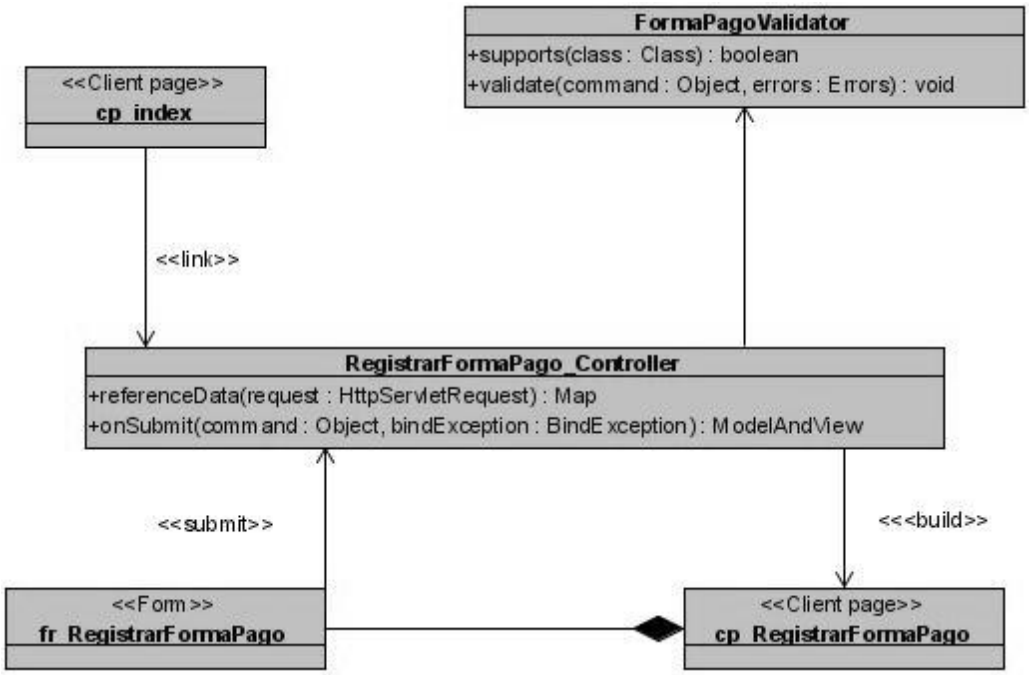
47 Modificar Carta de Crédito



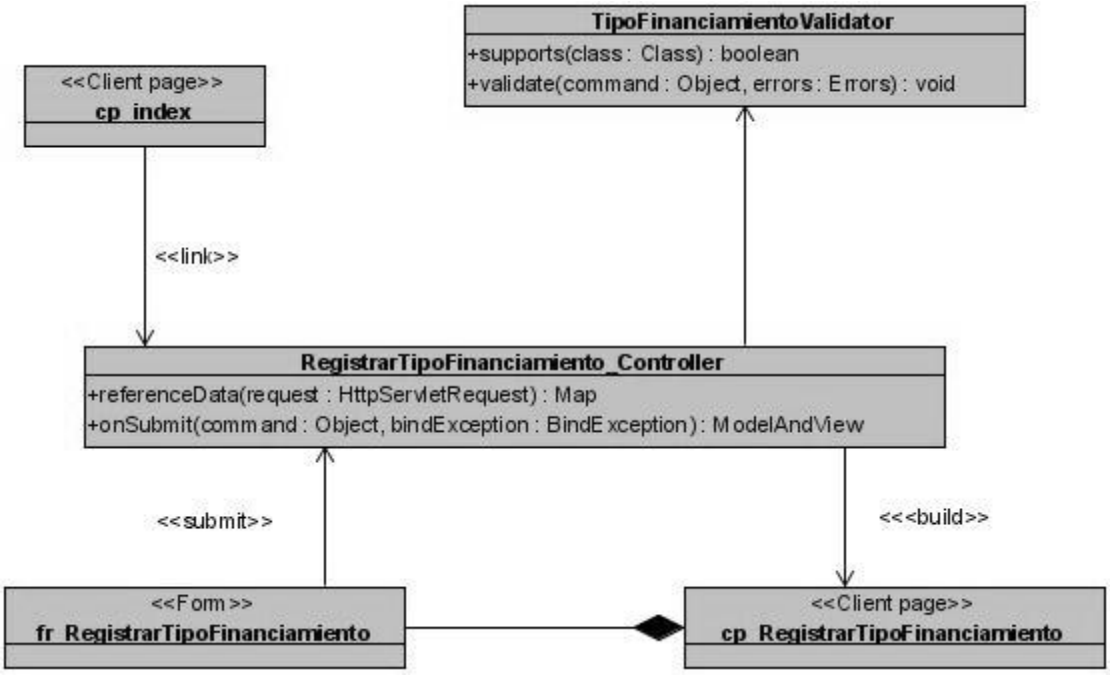
48 Modificar Tipo de Financiamiento



49 Registrar forma de Pago

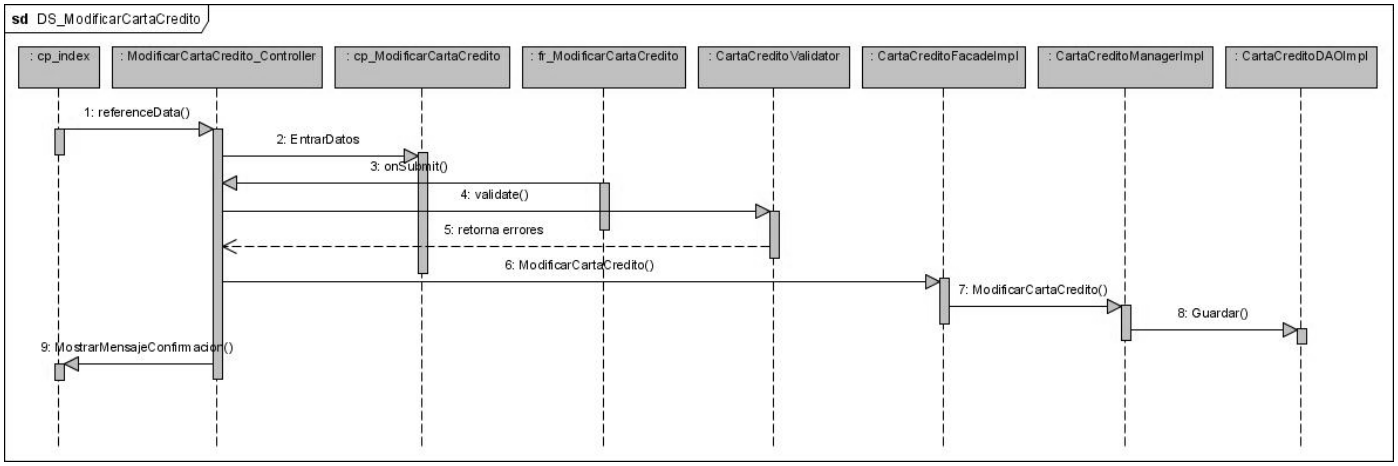


50 Registrar Tipo de Financiamiento

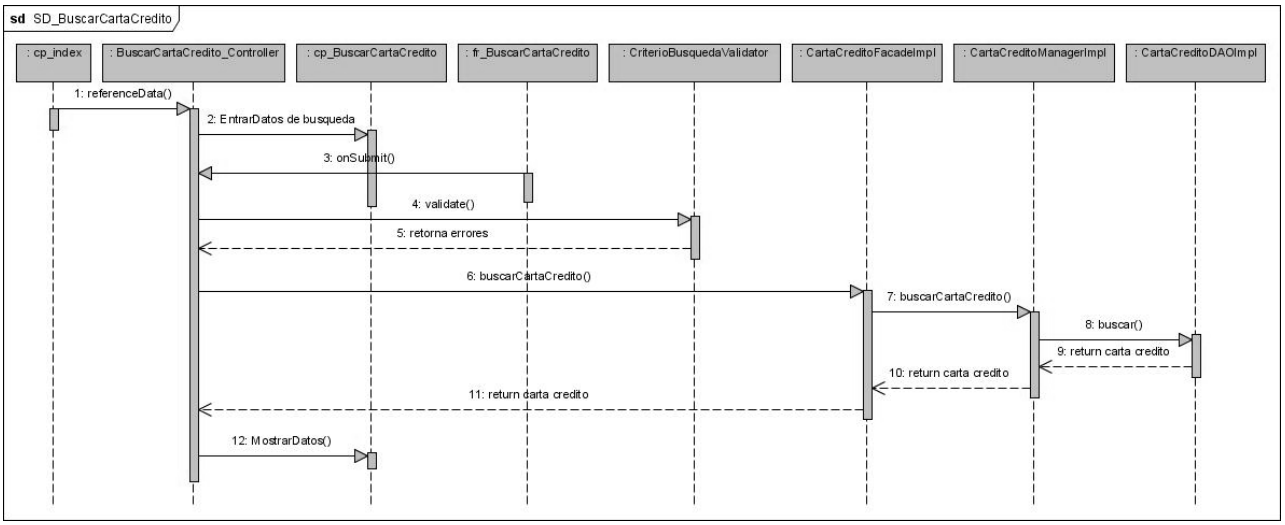


7.2.1. Diagramas de Secuencia

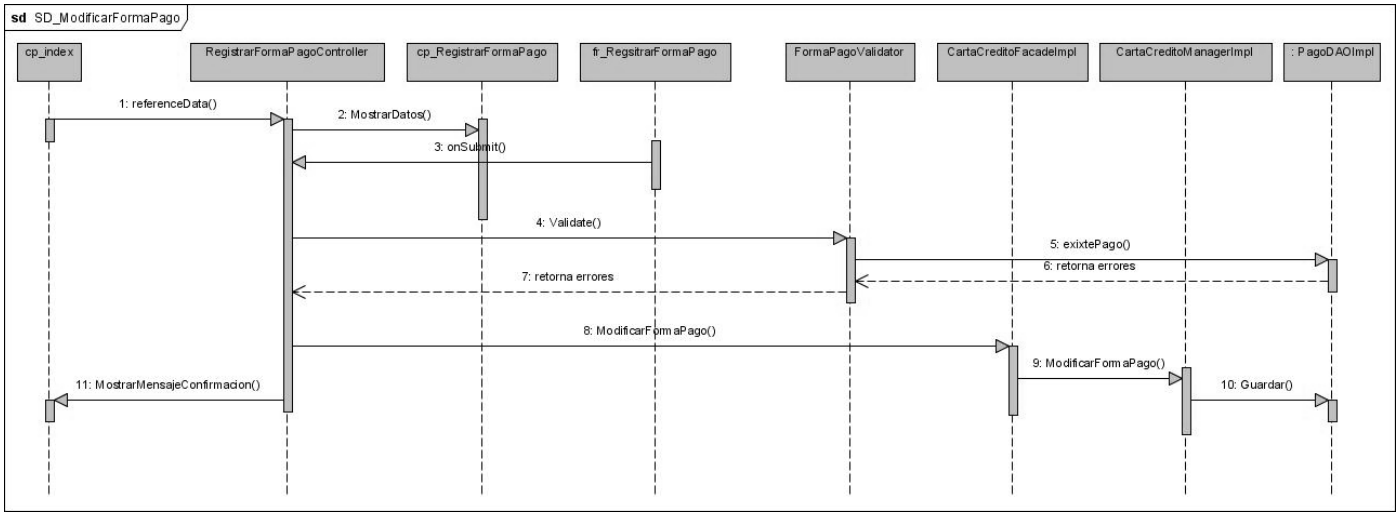
51 Registrar Carta de Crédito



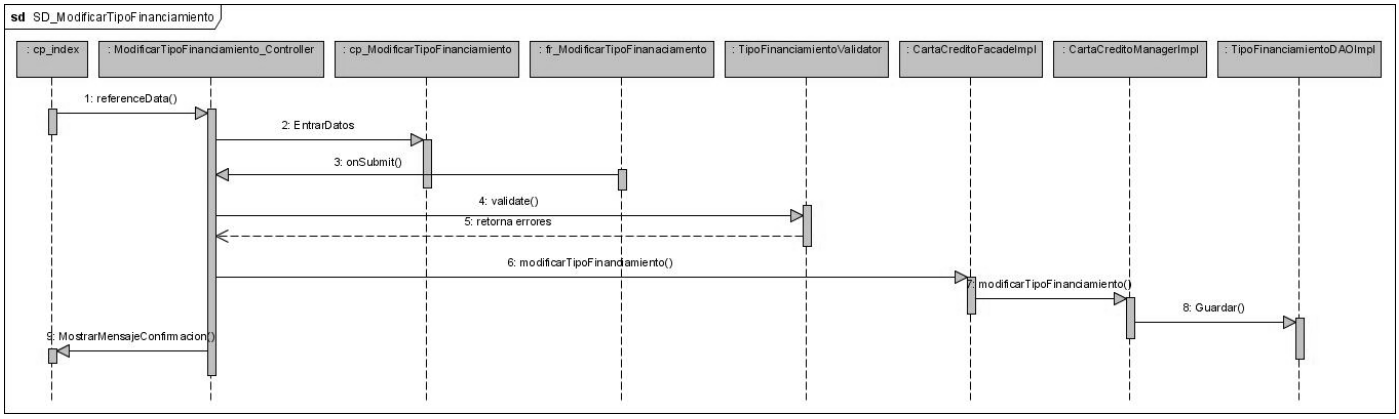
52 Buscar Carta de Crédito



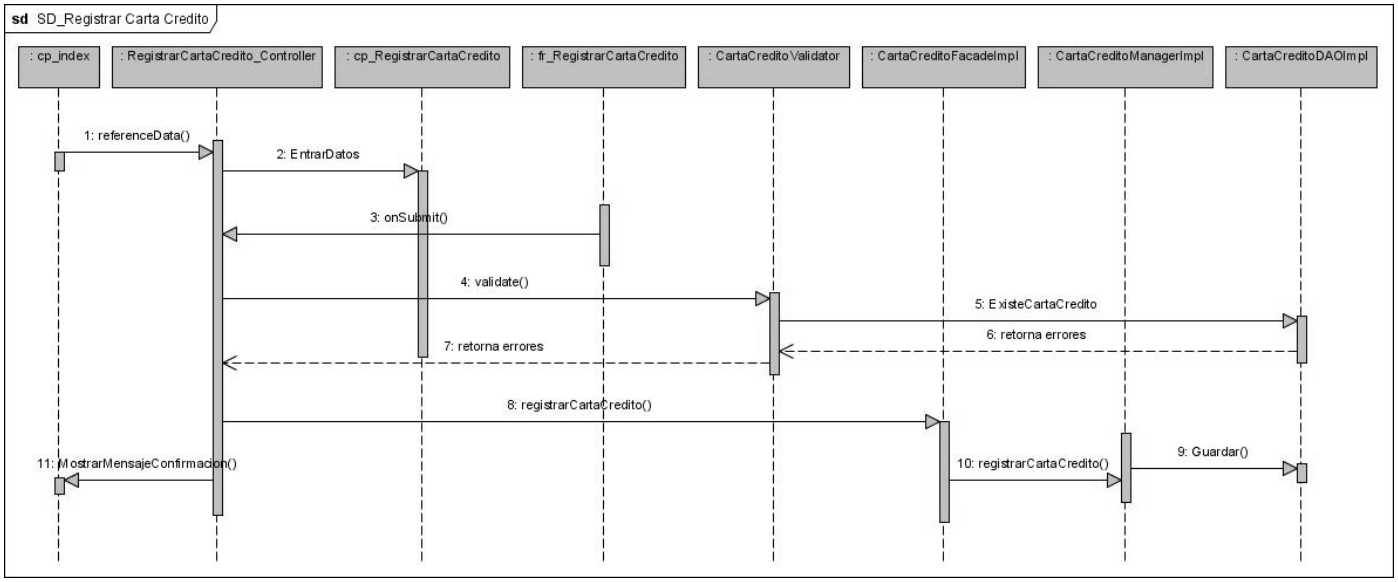
53 Modifica Forma de Pago



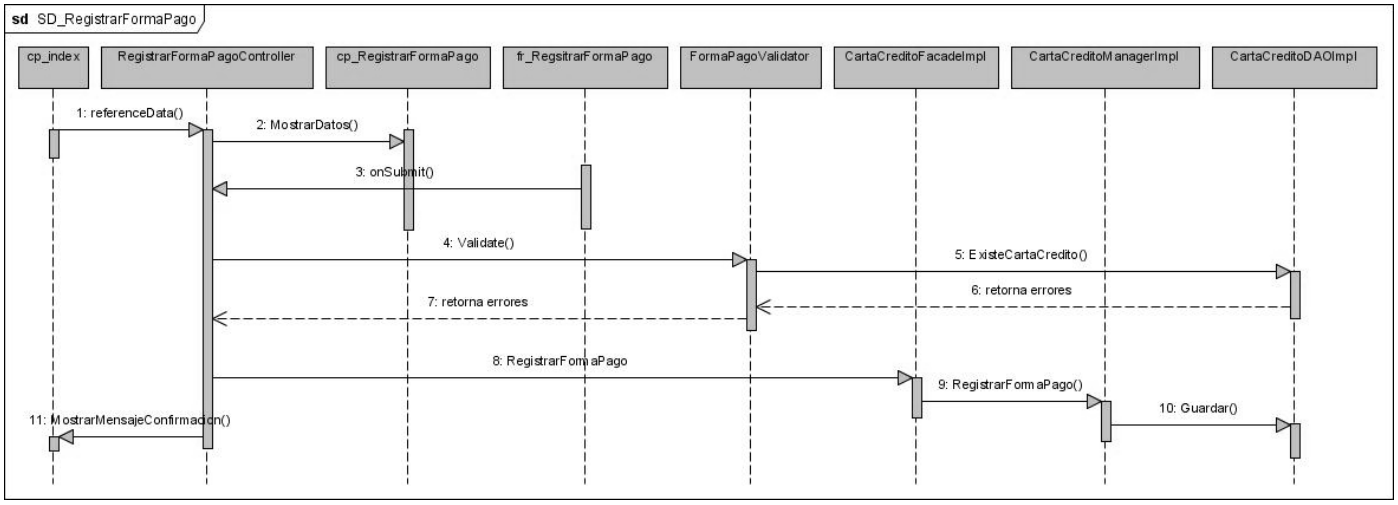
54 Modifica Tipo de Financiamiento



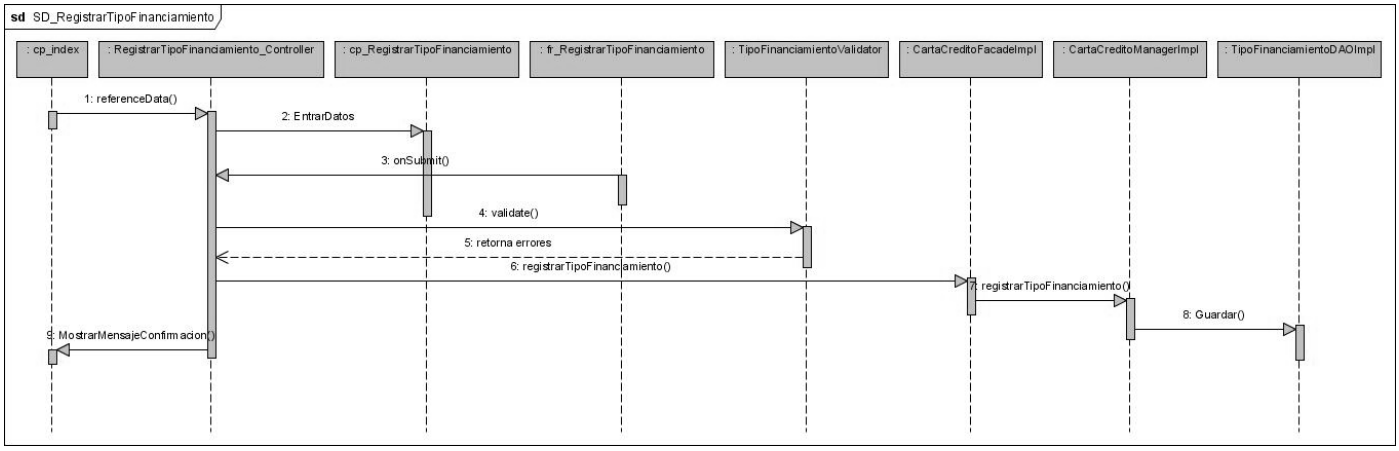
55 Registrar Carta de Crédito



56 Registrar Forma de Pago



57 Registrar Tipo de Financiamiento



8. GLOSARIO

AOP: Paradigma de Orientación a Aspectos.

APH: Árbol de Profundidad de Herencia.

BNC: Banco Nacional de Cuba.

CARCRED: Sistema desarrollado por la compañía venezolana LA. Sistemas, con más de 20 años de trascendencia en el mercado financiero de ese país, el cual permite mantener el control y seguimiento de todos los procesos asociados a las Cartas de Créditos tales como Negociaciones y Pago de las mismas.

CASE: Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador.

COBIS SCI: Sistema Integral de Comercio Internacional

CU: Caso de Uso.

DAO: Patrón de Acceso a Datos que permite acceder a la fuente de datos y encapsular los objetos clientes, ocultando tanto la fuente como el modo de acceder a ella.

FDD: Feature Driven Development. Desarrollo Manejado por Funcionalidades

HQL: Hibernate Query Language.

IDE: Entorno de Desarrollo Integrado

IEEE: Instituto de Ingenieros Eléctricos y Electrónicos. Asociación técnico-profesional mundial dedicada a la estandarización.

IS: Ingeniería de Software.

LoC: Técnica de Inversión de Control

MSF: MSF: Microsoft Solution Framework.

POO: Programación Orientada a Objetos.

RUP: Rational Unified Process.

SABIC: Sistema Automatizado para la Banca Internacional de Comercio

SCRUM: Metodología de desarrollo de software.

SISCOM: Sistema de mensajería Inter.-bancaria.

SPRING: Es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java.

SQL: Structured Query Language

TC: Tamaño de Clase.

UML: Unified Model Language. Lenguaje Unificado de Modelado.

SWIFT: Conducción interactiva de señales de datos entre terminales y computadoras, utilizando la técnica de conmutación de mensajes, conforme a las Recomendaciones del CCITT y de la ISO

(Organización Internacional para la Estandarización), proporcionado internacionalmente, en los términos establecidos en la Ley Federal de Derechos y en estas Condiciones.

CCITT: Comité Consultivo Internacional Telegráfico y Telefónico (Consultative Committee for International Telegraphy and Telephony