Universidad de las Ciencias Informáticas.

Facultad 3.

Ingeniería Informática.

Diseño e implementación del Módulo Fondos de Caja Chica en el Sistema de Administración Financiera del proyecto Registros y Notarías.

Trabajo de diploma para optar por el título de Ingeniero Informático.

Autor: Kenny Ahmed Quiñones Toledo.

Tutor: Ing. Johnny Pérez Acosta.

Ing. Armando Esteban Pacheco Iglesias.

Declaración de autoría.

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

Declaramos ser autores de la presente tesis y recomendamos 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 _____.

Kenny Ahmed Quiñones Toledo

_____

Firma del Autor.

Ing. Johnny Pérez Acosta

_____

Firma del Tutor.

Ing. Armando Esteban Pacheco Iglesias

_____

Firma del Tutor.

Agradecimientos.

A mi madre por ser madre y padre en tantos momentos de mi vida, gracias por tener ese corazón tan grande.

A mi hermano por existir, quiero que sepas que me siento muy orgulloso de tenerte.

A mi tío Rubén, son tantas las veces que has sido un padre para mí que sobran las palabras para decirte lo que siento por ti.

A todos mis familiares por confiar en mí y por darme tantos consejos cuando más lo necesitaba.

A esos dos tutores Johnny y Armando, ustedes son unos (p)(i)(n)(g)(ú)(c)(e)(s).

A mis compañeros de trabajo y de estudios, por soportarme, ayudarme y estar ahí en todo momento.

A todos aquellos que ya no se encuentran y que de una forma u otra han marcado la diferencia.

A Sandy, Norlen, Aurora, Yunelis, Niudis, Héctor, Osmel, Pedro.

A mis compañeros de proyecto, a Maikel (mano sin ti esto nunca hubiera salido, nunca olvidaré cuando fui a rescatarte del hospital).

A mis compañeros de vicio y al guild BloodSeeker.

Dedicatoria.

A mis familiares.

Resumen.

En el Ministerio del Poder Popular para las Relaciones Interiores y Justicia de Venezuela existe una Dirección para los Registros y Notarías, el cual es el órgano encargado de supervisar, documentar y controlar todos los procesos que se llevan a cabo en los diferentes registros: Mercantiles, Públicos, Principales y las Notarías.

Dicha Dirección se traza como una de sus principales actividades, la Administración Financiera de estas oficinas.

Como parte de los acuerdos de cooperación entre Cuba y la República Bolivariana de Venezuela surge el proyecto SAREN, el cual está conformado por varios subsistemas, siendo uno de ellos Administración Financiera. Este subsistema garantizará la gestión de procesos administrativos-financieros en los Registros y Notarías con el desarrollo de varios módulos entre ellos se encuentra Fondos de Caja Chica. El cual ya ha concluido su fase de inicio haciéndose necesario su diseño y posterior implementación para de esta manera dar continuidad al proceso de desarrollo.

Basado en el estudio de algunos estilos arquitectónicos, patrones de arquitectura y diseño, el presente trabajo pretende proponer una solución de diseño de software e implementación del módulo Fondos de Caja Chica comprendido en el sistema Administración Financiera del Producto SAREN[1].

La solución en cuestión se integra a los demás módulos comprendidos en el sistema de Administración Financiera, de esta manera se permitirá centralizar los procesos con los Fondos de Caja Chica a nivel nacional, así como facilitar además a la Dirección General de Registros y Notarías la obtención de las metas trazadas con la automatización de la actividad Administrativa Financiera a nivel nacional.

[1] Servicio Autónomo de Registros y Notarías.

Índice.

INTRODUCCIÓN.

En el Ministerio del Poder Popular para las Relaciones Interiores y Justicia de Venezuela existe una Dirección para los Registros y Notarías, el cual es el órgano encargado de supervisar, documentar y controlar todos los procesos que se llevan a cabo en los diferentes registros: Mercantiles, Públicos, Principales y las Notarías.

Actualmente, existen 210 registros de Notarías, 21 registros Principales, 47 registros Mercantiles y 213 registros Públicos, para un total de 491 oficinas adscritas a la Dirección General de Registros y Notarías. La primera fase del proyecto Registros y Notarías informatizó los registros Mercantiles y los registros Públicos, de los cuales están en producción 100 registros Públicos y 34 registros Mercantiles, para un total de 134 registros conectados y sincronizados con la Dirección SAREN.

Dicha Dirección se traza como una de sus principales actividades, la Administración Financiera de estas oficinas. Se entiende como Sistema de Administración Financiera Integrada como el conjunto de leyes, normas y procedimientos destinados a la obtención, asignación, uso, registro y evaluación de los recursos financieros del Estado, que tiene como propósito la eficiencia de la gestión de los mismos para satisfacer las necesidades colectivas.

Un modelo de Administración Financiera está integrado cuando las unidades organizativas y entes que lo conforman actúan en forma absolutamente interrelacionada bajo la dirección de un único órgano coordinador que debe tener la suficiente competencia para reglamentar el funcionamiento de ésta, y cuando el conjunto de principios, normas y procedimientos que están vigentes en el sistema son coherentes entre sí y permiten interrelacionar automáticamente sus actividades.

Las unidades organizativas que en el sector público integran la Administración Financiera, son las responsables de programar y evaluar el presupuesto, administrar el sistema de recaudación tributaria y aduanera, administrar el tesoro y contabilizar, tanto física como financieramente, las transacciones relacionadas con la captación y colocación de los fondos públicos. Los recursos humanos, materiales y financieros que demanden el funcionamiento de estas unidades, forman parte de la Administración Financiera.

Como parte de los acuerdos de cooperación entre Cuba y la República Bolivariana de Venezuela surge el proyecto SAREN, el cual está conformado por varios subsistemas, siendo uno de ellos Administración Financiera. Este subsistema garantizará la gestión de procesos administrativos-financieros en los Registros y Notarías con el desarrollo de varios módulos entre ellos se encuentra Fondos de Caja Chica. El cual ya ha concluido su fase de inicio haciéndose necesario su diseño y posterior implementación para de esta manera dar continuidad al proceso de desarrollo.

El Ministerio del Poder Popular para las Relaciones Interiores y Justicia de la República Bolivariana de Venezuela no posee un control centralizado de la Administración Financiera de los Registros y Notarías, los cuales funcionan con autonomía en los servicios que brindan, es decir funcionan independientemente. El organismo constitucional ha determinado crear un conjunto de acciones encaminadas a garantizar un Sistema de Administración Financiera Integrada, el cual contempla los siguientes módulos:

- Administración.
- Presupuesto.
- Contabilidad.
- Recaudación.
- Requisiciones.
- Compras y Servicios.
- Retenciones.
- Tesorería.
- Fondos en Anticipo.
- Fondos de Caja Chica.

El principal problema de la Dirección General de Registros y Notarías está dado a causa de que a cada una de sus oficinas registrales se les asigna de forma asidua cierta cantidad de efectivo para que puedan satisfacer sus necesidades más inminentes y en la actualidad

no existe una plataforma tecnológica única y centralizada para el control de las operaciones realizadas con este fondo, motivo por el cual estas son registradas de forma manual y bastante tediosa. Generando de esta manera un ambiente propicio para el desvío y la malversación.

Formulación del Problema:

¿Cómo lograr un sistema informático que permita gestionar los procesos de los Fondos de Caja Chica en las Oficinas Registrales de la República Bolivariana de Venezuela partiendo de los artefactos obtenidos durante el análisis?

Objeto de Estudio:

Proceso de Desarrollo de Software.

Objetivo General:

Desarrollar una solución informática que permita la automatización y centralización de los procesos que se llevan a cabo en los Fondos de Caja Chica de las Oficinas Registrales venezolanas.

De este se derivan los siguientes **objetivos específicos**:

1 Elaborar marco teórico para la justificación del trabajo y la formulación de hipótesis.

2 Diseñar la solución para los procesos de Fondos de Caja Chica en el Sistema de Administración Financiera.

3 Implementar la solución diseñada.

4 Evaluar a través de la aplicación de Métricas la calidad del diseño obtenido.

5 Evaluar a través de la aplicación de la técnica de Caja Blanca la solución obtenida.

Campo de Acción:

Diseño e implementación de los procesos identificados en los Fondos de Caja Chica de las Oficinas Registrales venezolanas.

Hipótesis:

Si se obtienen los artefactos correspondientes al diseño e implementación entonces se concluirá con el proceso de desarrollo obteniéndose así una solución que permita gestionar los procesos de los Fondos de Caja Chica.

Para el cumplimiento de los objetivos descritos anteriormente es necesaria la realización de las siguientes **tareas**:

1. Estudiar las leyes financieras que rigen a los Registros y Notarías de Venezuela.
2. Analizar los procesos en los Fondos de Caja Chica de los Registros y Notarías de Venezuela.
3. Explicar brevemente los principales procesos del negocio.
4. Desarrollar un estudio sobre las herramientas y tecnologías actuales para el desarrollo de software.
5. Llevar a cabo un estudio sobre patrones de Diseño y Arquitectura con el fin de determinar la factibilidad de su implementación en el sistema en cuestión.
6. Realizar el diseño e implementación de la aplicación para gestionar los Fondos de Caja Chica acorde con las necesidades de los Registros y Notarías de la República Bolivariana de Venezuela.
7. Determinar la calidad del diseño y la solución obtenidos con la utilización de métricas.

Estructura del Documento:

El presente documento ha sido estructurado de la siguiente forma:

Capítulo 1: Se realiza una breve explicación del negocio en cuestión. Aborda una breve caracterización sobre los Sistemas Financieros informáticos más usados en Cuba y el mundo. Resume un estudio de las principales herramientas CASE utilizadas para documentar y realizar el diseño de un proyecto software, plataformas de desarrollo, así como metodologías existente para el desarrollo de software.

Capítulo 2: Aborda la propuesta del sistema, para la cual se tienen en cuenta un estudio de los distintos tipos de estilos arquitectónicos, fundamentando el que se escogió por la dirección del proyecto para la solución del sistema, como también se abordan los patrones de diseño más utilizados en este. Se realiza un estudio de las principales tendencias y tecnologías que son empleadas en la actualidad en el mundo del desarrollo de software.

Capítulo 3: Se realiza el diseño de la solución informática para los Fondos de Caja Chica en el Sistema de Administración Financiera, obteniéndose los artefactos correspondientes al flujo de trabajo de Análisis y Diseño, específicamente los correspondientes a la etapa de diseño, mostrándose los diagramas más representativos de las actividades que se llevan a cabo. Se describe además la arquitectura establecida para la realización de este trabajo, así como el modelo de implementación, modelo de componentes y modelo de despliegue.

Capítulo 4: Se estudian algunas métricas destinadas a la medición de la calidad del diseño de software, se realizan pruebas de unidad al código implementado. Se analizan y evalúan finalmente los resultados obtenidos.

Capítulo 1: Fundamentación Teórica

En este capítulo se realiza una breve explicación de los procesos de los Fondos de Caja Chica. Inicialmente se aborda en qué consiste y luego se hace una exposición de los pasos que se realizan para llevarlo a cabo. Se hace una breve caracterización sobre los Sistemas Financieros Informáticos más usados en el mundo y en Cuba, enmarcados en los procesos de contabilidad y presupuesto, pues es el tema que ocupa el presente trabajo. Además se hace un estudio de las plataformas, metodologías y herramientas para el desarrollo de software, explicando sus características.

1.1 Fondos de Caja Chica.

Según (Artículo Nro 2, ASAMBLEA NACIONAL DE LA REPÚBLICA BOLIVARIANA DE VENEZUELA, 15 de Enero de 2007) la Administración Financiera comprende el conjunto de sistemas, órganos, normas y procedimientos que intervienen en la captación de ingresos públicos y en su aplicación para el cumplimiento de los fines del Estado, y estará regida por los principios constitucionales de legalidad, eficiencia, solvencia, transparencia, responsabilidad, equilibrio fiscal y coordinación macroeconómica.

El proceso de Fondos de Caja Chica dentro de la Administración Financiera consiste en asignar un valor monetario a una Oficina Registral para realizar compras o pagos de menor cuantía, que no pueden ser cubiertos a través del proceso normal de compras. (Saenz Fernández, 2005).

A continuación se muestra un esquema con la estructura financiera actual de la dirección nacional de Registros y Notarías para facilitar un mayor entendimiento al proceso.

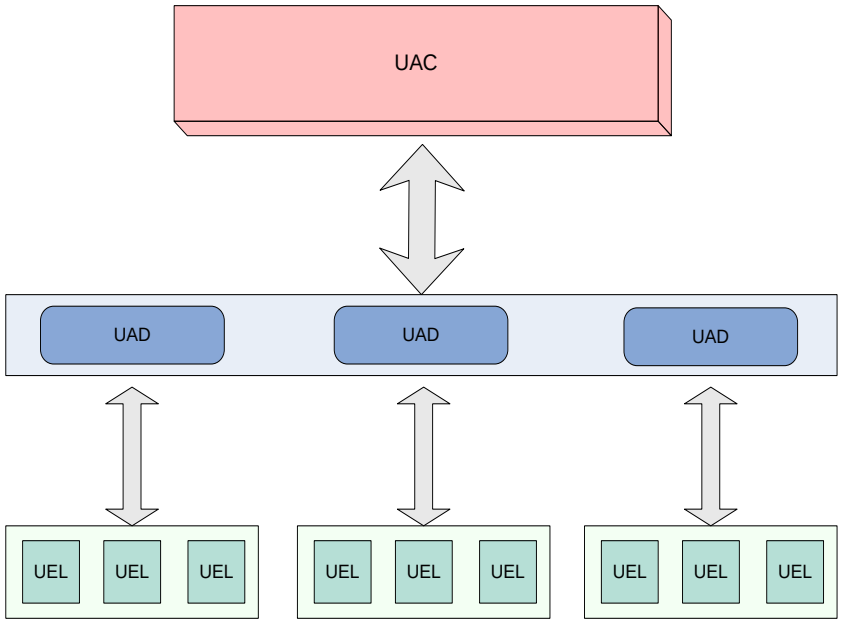

Figura 1 Estructura financiera de SAREN.

La solución de software está dirigida a la automatización de los procesos administrativos y financieros según la distribución adoptada actualmente a tal efecto en la dirección del SAREN.

Al haberse plasmado el estado actual de la estructura financiera de la dirección nacional de Registros y Notarías, así como haber visto en qué consiste el proceso de Fondos de Caja Chica podemos adentrarnos en el estudio de los principales procesos que se realizan con este fondo.

1.2 Procesos en los Fondos de Caja Chica.

Los procesos de la Caja Chica se inician cuando una Oficina, dígase Unidad Ejecutora Local (UEL) solicita a la Unidad Administradora[2] (UA) la apertura de un fondo a través de un memorándum[3], el cual utilizará para suplir sus gastos menores; esta Unidad Administradora se encargará de crear y constituir el fondo solicitado, seleccionando la

[2] Existen dos tipos de Unidad Administradora, la Unidad Administradora Central (UAC) y la Unidad Administradora Desconcentrada (UAD).

[3] Un **memorándum** no es más que un aviso, nota o escrito.

cuenta bancaria de la cual se retirará el capital para dar el fondo por constituido, se elegirá además un responsable perteneciente a la Oficina para la cual está destinado el Fondo de Caja Chica.

El Fondo de Caja Chica dispondrá de una serie de Categorías Presupuestarias sobre las cuales estarán encaminados los gastos menores de la Oficina Registral. Se entiende por gastos menores los siguientes (Chica, 2002):

- Gastos de transporte para el personal que se encuentra en comisión de servicios dentro del radio urbano, siempre que el trabajo a realizarse, por su urgencia así lo justifique.
- Gastos menores por la compra de material pequeño, para la refacción de muebles y equipos.
- Limpieza y material de limpieza para los centros de trabajo.
- Gastos menores por compra de útiles de escritorio de uso frecuente o extraordinario.
- Gastos de comunicación, servicios postal, telegráfico, fax y fotocopias.
- Gastos menores por refrigerio para reuniones oficiales.
- Otros gastos de servicio que tengan carácter de imprevisibles y urgentes.

Los pagos a efectuarse con Fondos de Caja Chica, deben ser al contado y por obligaciones que no excedan al 25% del monto asignado para Caja Chica. (Chica, 2002).

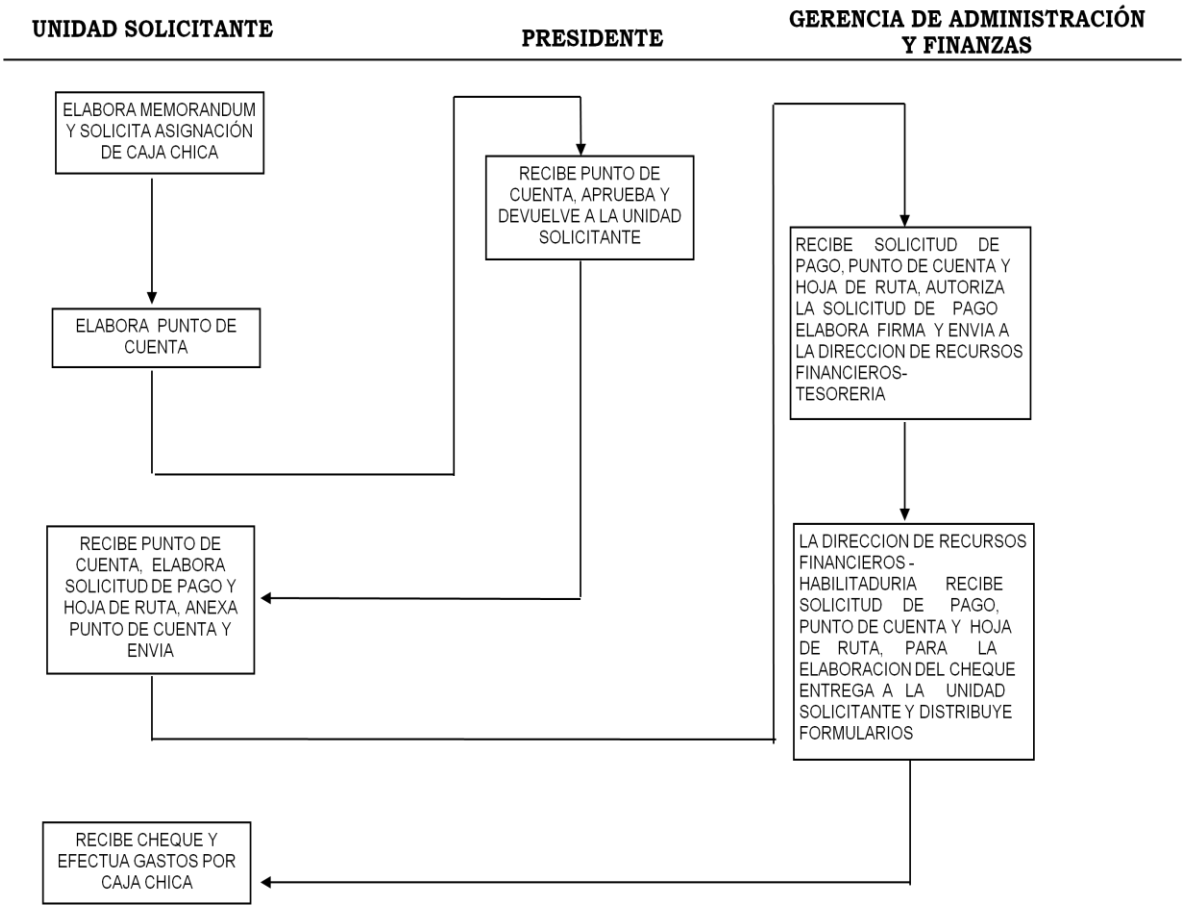

Figura 2 Flujo-grama del procedimiento apertura de Caja Chica.

Este fondo pasará a su estado de confirmación una vez que se obtenga el presupuesto en la Unidad Ejecutora Local que inició la solicitud, por lo cual se confirma a nivel de UEL.

Las operaciones con este fondo se realizarán a través de Vales de Caja Chica, Facturas o Recibos, Reposiciones y Modificaciones. Los vales se crearán asociando los bienes y servicios para los cuales estarán destinados, teniendo en cuenta que su monto total no debe exceder al monto en efectivo que tenga el Fondo de Caja Chica. Una vez aprobados los vales tendrán como contrapartida las facturas o recibos, las cuales contemplarán y justificarán la compra, manifestando la existencia o no de reembolsos. Los reembolsos se originarán en dependencia de si se utilizó o no el vale, completamente en la compra, la diferencia de efectivo entre el vale y la factura o recibo constituirán el reembolso, aclarar

que el monto de las facturas no excederá en ningún momento al monto del vale al cual está asociada.

Las UEL podrán además solicitar reposiciones para su Fondo de Caja Chica, en caso de comenzar un determinado mes sin suficiente efectivo para realizar sus operaciones, según (Chica., 2002):

Las reposiciones de Caja Chica, permiten aceptar rendiciones parciales cuando se haya gastado aproximadamente el 60% del Fondo de Caja Chica. En tanto se tramite la reposición se podrá disponer el 40% restante.

Por lo cual hacen su solicitud a la Unidad Administradora, presentando además las facturas que fueron generadas fomentando las compras para las cuales estuvo destinado el efectivo. La UA se encargará de aceptar la solicitud luego de un previo análisis de las facturas presentadas, las cuales deben ser legítimas y fidedignas; así como rechazar la solicitud en caso contrario.

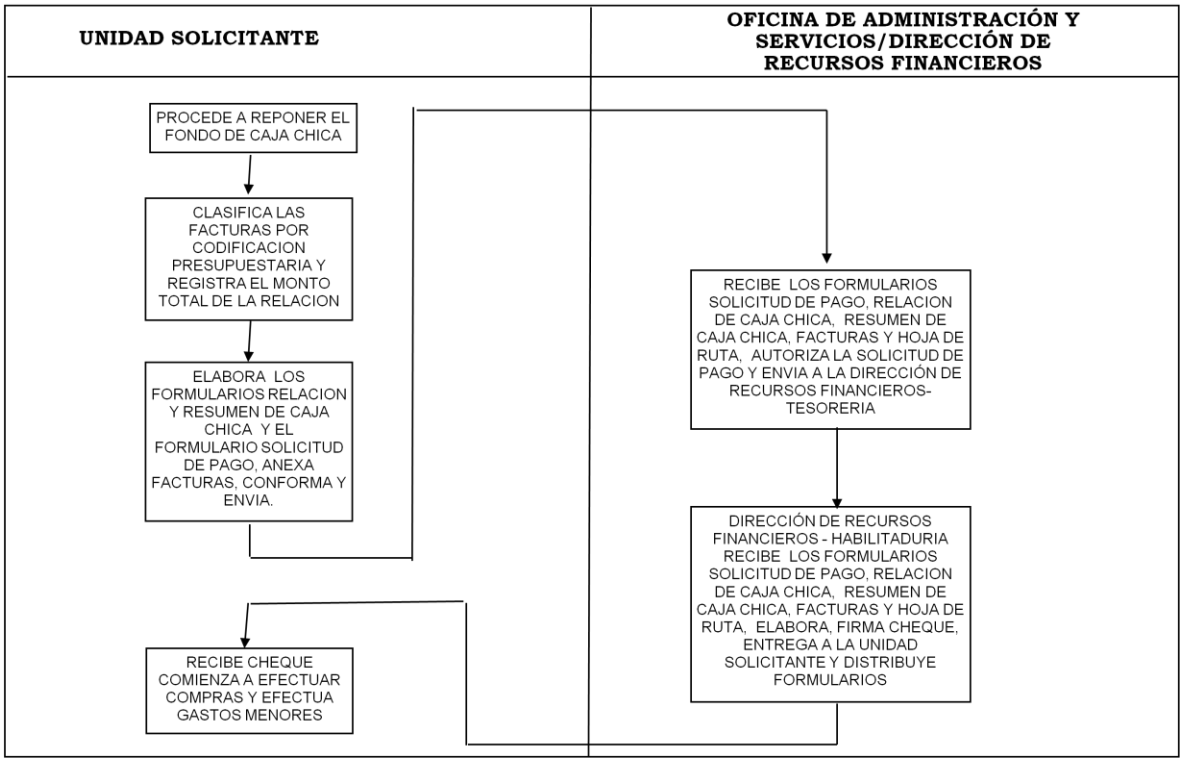

Figura 3 Flujo-grama del procedimiento de reposición de Caja Chica.

El proceso de cierre del Fondo de Caja Chica lo iniciará la Unidad Administradora, por determinados motivos, los cuales planteará mediante un memorándum enviado a la UEL, esta oficina tendrá que liquidar el fondo, reponiendo el fondo en efectivo que tenga la Caja Chica en ese momento. La UA concluirá dando por cerrado el Fondo de Caja Chica y depositando el efectivo en una cuenta bancaria.

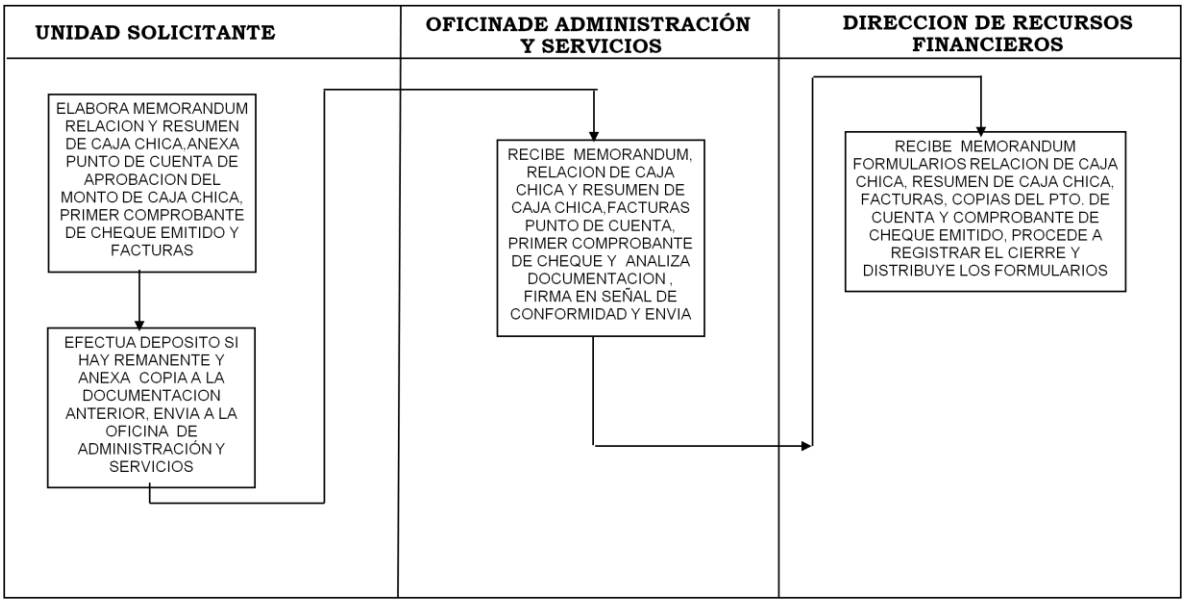

Figura 4 Flujo-grama del procedimiento cierre de Caja Chica.

Otra de las operaciones por la cual pudiera transitar el Fondo de Caja Chica son las modificaciones. Luego de un determinado período y en dependencia del uso que tenga cierto fondo, las Unidades Administradoras pueden realizar modificaciones de aumento o disminución a alguna o algunas UEL adscritas a estas, según sea necesario.

Todas estas operaciones con el Fondo de Caja Chica quedarán registradas en el Libro Auxiliar de Caja Chica, según se vayan realizando. Este libro facilitará el estricto control de las transiciones del efectivo del Fondo de Caja Chica. Al cerrar un determinado fondo, su correspondiente libro auxiliar no se eliminará, sino que pasará a estado inactivo, para tener una constancia del flujo de operaciones realizadas.

Una vez realizada una breve descripción de los principales procesos del negocio en cuestión, se realizará un estudio de algunos de los principales sistemas financieros existentes para de esta manera dar un mejor enfoque al nuestro.

1.3 Sistemas Financieros.

El sistema financiero se define como el conjunto de instituciones cuyo objetivo es canalizar el excedente que generan las unidades de gasto con superávit[4] para encauzarlos hacia las unidades que tienen déficit. (Moya Arjona, 2009).

El sistema financiero es un mecanismo dinámico en constante evolución, por lo que esta panorámica actual no constituye más que un fotograma de proyección continua, que es necesario descubrir asimismo para comprender en plenitud su funcionamiento. (Moya Arjona, 2009).

1.3.1 Sistemas Financieros en Venezuela.

Actualmente existen en la República Bolivariana de Venezuela diferentes aplicaciones informáticas que brindan servicios para controlar los procesos de Fondos de Caja Chica de los Registros y Notarías. La gran mayoría de estas aplicaciones son módulos separados entre los cuales no existen vínculos. Estas aplicaciones difieren unas de otras cuando se trata de diferentes registros. Es decir, cada oficina registral cuenta con uno o varios módulos diferentes de software que contribuyen a la realización de dichos procesos en estas oficinas. Se quiere llevar a cabo la estandarización y realización de una plataforma informática que responda a todos los procesos de la Administración Financiera Pública, y entre estos procesos se encuentran los mencionados en el epígrafe anterior, con un carácter centralizado y controlado, para menos malversación de los recursos del Estado.

1.3.2 Sistemas Financieros en el mundo.

La aparición de las computadoras, la generalización de las corporaciones, el surgimiento de grandes empresas multinacionales y la globalización de los comercios internacionales, dio un nuevo giro a la orientación de la información financiera, surgiendo la necesidad de crear Sistemas Financieros que sean más útiles en el desenvolvimiento de la gestión administrativa, y que dichas informaciones sean efectivas, confiables y oportunas; esta

[4] El **superávit** no es más que un sobrante o demasía.

necesidad fue lo que impulsó hacia la creación de los Sistemas de Finanzas Computarizados. (Smith, 1991).

Existe una amplia gama de Sistemas Informáticos que han formado parte de las herramientas de trabajo de muchas empresas desde hace más de 40 años, hasta tal punto que hoy en día son el motor de las operaciones de muchas de ellas. Lo que ha permitido a los Ingenieros en Sistemas buscar la forma de satisfacer de una manera más completa las necesidades en las que se ve envuelta la empresa, de acuerdo al volumen de las operaciones que esta maneja; por lo que han tratado crear Sistemas Computarizados de Finanzas que brinden los mismos beneficios que ofrecen los Sistemas Manuales. En su afán de crear este tipo de Software los expertos en la materia han diseñado diversidad de programas que tratan de suplir la necesidad de determinadas empresas de acuerdo a las actividades que realiza. Dentro de estos Software, los más conocidos son:

- **Dac Easy Accounting System**

Según (Smith, 1991) existen software comerciales donde el más usado y reconocido por todos es el DacEasy Accounting System, el cual desde los años 70 es conocido en el mercado como uno de los Software de Contabilidad más completo.

DacEasy Accounting System es un software que combina información financiera para un eficaz control de los negocios y una oportuna y acertada toma de decisiones. (Smith, 1991).

El DacEasy está compuesto por varios módulos, los cuales se tratarán más adelante, que pueden o no trabajar como un sistema totalmente integrado, ofreciendo así una ventaja más sobre los sistemas manuales de contabilidad que se caracterizan por su lentitud en el procesamiento de la información. (Smith, 1991).

Este software ha sido diseñado para su utilización en diversos tipos de negocios, como lo son los orientados a la prestación de servicios, a la fabricación y venta de productos, y a cualquier tipo de negocio que combine estas áreas.

El DacEasy es uno de los software de contabilidad de mayor venta en toda América, desde su lanzamiento en los años ’70 ha demostrado ser uno de los sistemas computarizados de

contabilidad más completo, obteniendo varios galardones en el área de la programación informática.

Según (Smith, 1991) Contabilidad DacEasy es un paquete que a pesar de tener múltiples características, es muy fácil de usar. Este programa de contabilidad contempla siete (7) módulos que son:

- Contabilidad General.
- Cuentas por Cobrar.
- Cuentas por Pagar.
- Efectivo (Caja y Banco).
- Facturación.
- Compras.
- Inventarios.

Estos módulos trabajan en conjunto como un sistema completamente integrado o pueden ser usados individualmente sin importar cual se escoja. Contabilidad DacEasy ofrece una interfaz rápida y fácil, reduciendo sus tareas contables a la simple entrada de datos. Las complejas características de los reportes permiten mirar y analizar la información facilitando una completa auditoria a medida que esta se procesa. (Smith, 1991).

Dada la importancia que adquiere el módulo de Efectivo (Caja y Banco) dentro de este trabajo, será abordado a continuación.

El módulo de Efectivo de Contabilidad DacEasy demostrará ser uno de los módulos más útil y más frecuentemente usado en el sistema. Se utiliza para entrar y rastrear todos los depósitos y cheques. DacEasy también ofrece una característica de reconciliación de un ilimitado número de cuentas. Este módulo permite consolidar todo el movimiento de efectivo que incluye tanto los pagos como las devoluciones de clientes y proveedores. (Smith, 1991).

- **PeachTree.**

Según (Hedtke, 2000) Peachtree es un sistema contable que ha tenido gran acogida en los Estados Unidos, y en Panamá, por sus grandes bondades. La facilidad de uso ha hecho bastante fácil la implantación de Peachtree en empresas pequeñas y medianas sin que sea necesario tener un gerente de cómputo dedicado a su mantenimiento. Y su amplitud le ha dado cabida en una diversa gama de empresas, desde restaurantes, arquitectos, corredores de seguros, abogados, hasta contadores y auditores.

Peachtree incluye, en un solo programa, los módulos de mayor general, planilla, cuentas por pagar, cuentas por cobrar, inventario, conciliación bancaria, costo de obras, y análisis financiero. También está incluido un módulo separado para el control de activos fijos. (Hedtke, 2000).

Peachtree es un programa totalmente integrado. Bajo un solo programa se encuentran todas las capacidades que en otros sistemas se considerarían módulos discretos. La organización de los módulos dentro del antes mencionado está más bien relacionada con la funcionalidad del paradigma de los documentos que usa el sistema. Lo que en otros sistemas se consideraría cuentas por pagar, en este software sería compras. (Hedtke, 2000).

Peachtree está compuesto por los siguientes módulos:

- Ventas.
- Compras.
- Inventario.
- Mayor General.

Peachtree es la mejor forma de llevar su contabilidad, administrar su negocio y su presencia en internet a través de las robustas características que usted quiere y del conocimiento que usted necesita. (Best, 2003).

Según (Best, 2003), peachtree es una confiable herramienta de administración para gente que:

- Ve la contabilidad como un elemento esencial en la toma de buenas decisiones.

- Quiere funcionalidad a través de una herramienta fácil de usar h que además genere reportes profesionales que le ayuden a entender su posición financiera.
- Necesita una solución de contabilidad completa, no sólo un paquete para estar organizado.

1.3.3 Sistemas Financieros en Cuba.

Sistema Económico Integrado Versat – Sarasola.

El **VERSAT - Sarasola** es un Sistema integrado de gestión económica, diseñado para ser utilizado por el sector empresarial Cubano, que se adecua a las características de cada entidad, es configurable por cada una de ellas en el momento de su instalación y tiene como objetivo fundamental ofrecerle a los usuarios la posibilidad de contar con un instrumento seguro, rápido, eficaz y de fácil manejo para la Planificación, Control y el Análisis de la Gestión Económica. (Cabrera González, y otros, 2004).

Permite llevar el control y el registro contable individual de todos los hechos económicos que se originan en las estructuras internas de las entidades y obtener los Estados Financieros y Análisis Económicos y Financieros en estos niveles. (Cabrera González, y otros, 2004).

Se estructura en un grupo de Subsistemas, donde se procesan y contabilizan documentos primarios y se anotan los movimientos de los recursos materiales, financieros y laborales que se utilizan en una entidad a partir de una configuración previa de los comprobantes que se originan.

A continuación se mencionan los principales subsistemas, detallando el más importante para nuestro trabajo.

- Configuración
- Contabilidad General.
- Costos y Procesos.
- Inventarios.
- Activos Fijos.

- Contratación y Facturación.
- Finanzas.

Finanzas.

Este Subsistema está concebido para que las administraciones dispongan de una información actualizada sobre el movimiento y la disponibilidad de efectivo a partir del procesamiento de todos los documentos, tanto de obligaciones como de pagos. Otro de los objetivos principales es dar la posibilidad a los usuarios de configurar, a través de conceptos, todos los movimientos que se procesan en el Subsistema, con lo que se logra que los comprobantes se asienten en la contabilidad bajo los criterios predeterminados, facilitando la obtención de la información deseada y agrupada según las necesidades de cada empresa. (Cabrera González, y otros, 2004).

Otro de los aspectos fundamentales a destacar para la realización de un sistema informático son las plataformas de desarrollos, la cuales se analizarán en el próximo epígrafe.

1.4 Breve análisis de las plataformas de desarrollo empresariales.

(Molpeceres Touris, y otros, 2002). Una plataforma de desarrollo empresarial ha de ofrecer un conjunto de servicios a los arquitectos y desarrolladores que ayuden a facilitar el desarrollo de aplicaciones empresariales, al tiempo que deben ofrecer la mayor cantidad posible de funcionalidades a los usuarios. Normalmente una plataforma de desarrollo empresarial tiene los siguientes requisitos:

- **Escalabilidad:** Ha de ofrecer una buena escalabilidad de modo que si aumenta la carga del sistema podamos añadir servidores o ampliar los existentes sin que sea necesario realizar muchas o ninguna modificación. (Molpeceres Touris, y otros, 2002).
- **Mantenibilidad:** Ha de permitir añadir modificar los componentes existentes sin que se modifique el comportamiento del sistema. (Molpeceres Touris, y otros, 2002).
- **Fiabilidad:** Ha de funcionar adecuadamente. (Molpeceres Touris, y otros, 2002).

- **Disponibilidad:** Ha de ofrecer soporte de arquitecturas tolerantes a fallos y sistemas de redundancia, que nos aseguren que nuestros sistemas estarán siempre disponibles. (Molpeceres Touris, y otros, 2002).

- **Extensibilidad:** Ha de ser posible añadir nuevos componentes y capacidades al sistema sin que se vean afectados el resto de componentes. (Molpeceres Touris, y otros, 2002).

- **Manejabilidad:** Los sistemas han de ser fácilmente manejables y configurables. (Molpeceres Touris, y otros, 2002).

- **Seguridad:** Se ha de tener buenos sistemas de seguridad tanto a nivel de autenticación, como de autorización y de transporte[5]. (Molpeceres Touris, y otros, 2002).

- **Rendimiento:** Se ha de ofrecer automáticamente soporte de clustering[6], balanceo de carga, piscina de objetos, piscina de conexiones, cachés, y en general mecanismos que permitan aumentar el rendimiento de manera transparente al usuario. (Molpeceres Touris, y otros, 2002).

La importancia de una plataforma empresarial es que todos estos componentes se nos ofrecen de manera automática de modo que aumenta la productividad de los desarrolladores. La diferencia entre utilizar una plataforma de desarrollo empresarial o no radica en que, en caso de no usarse, los desarrolladores perderían mucho tiempo realizando tareas de bajo nivel, no pudiéndose centrar en el desarrollo de aplicaciones y por consiguiente disminuyendo la productividad de los mismos. (Molpeceres Touris, y otros, 2002).

De las diferentes plataformas existentes en el mundo para el desarrollo de aplicaciones de escritorio se encuentran:

- **Java y su Máquina Virtual:** Una de las principales características que favoreció el

[5] El **nivel de transporte** es el cuarto nivel del modelo OSI encargado de la transferencia libre de errores de los datos entre el emisor y el receptor.

[6] Un C**lúster** es un grupo de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador.

crecimiento y difusión del lenguaje Java es su capacidad de que el código funcione sobre cualquier plataforma de software y hardware. Esto significa que un programa escrito para Linux puede ser ejecutado en Windows. Además es un lenguaje orientado a objetos que resuelve los problemas en la complejidad de los sistemas. El código que generan los compiladores de Java no es particular de una maquina física, sino de una máquina virtual. Aún cuando existen múltiples implantaciones de la Máquina Virtual Java, cada una específica de la plataforma sobre la cual subyace, existe una única especificación de la máquina virtual, que proporciona una vista independiente del hardware y del sistema operativo sobre el que se esté trabajando. (Pacheco Iglesias, y otros, 2008).

Las clases en Java pueden contener métodos que no estén implementados por códigos de operación (bytecode) Java, sino por algún otro lenguaje compilado en código nativo y almacenado en bibliotecas de enlace dinámico, como las DLL de Windows o las bibliotecas compartidas SO de Solaris[7]. El sistema de tiempo de ejecución incluye el código necesario para cargar dinámicamente y ejecutar el código nativo que implementa estos métodos. Una vez que se enlaza el módulo que contiene el código que implementa dicho método, el procesador virtual atrapa las llamadas a éste y se encarga de invocarlo. Este proceso incluye la modificación de los argumentos de la llamada, para adecuarlos al formato que requiere el código nativo, así como transferirle el control de la ejecución. Cuando el código nativo termina, el módulo de soporte para métodos nativos se encarga de recuperar lo resultados y de adecuarlos al formato de la Máquina Virtual Java. (Carballeira, 2000).

Las maquinas virtuales tienen algunas desventajas dentro de las que se encuentran, es que agregan gran complejidad al sistema en tiempo de ejecución. Por ejemplo, la MVJ espera que la computadora sobre la que subyace, soporte el estándar de IEEE para los números de punto flotante de 32 y 64 bits, así como enteros largos de 64 bits. La mayoría de las plataformas lo hacen pero hay algunas que no, lo que implica trabajo extra. (Carballeira, 2000).

[7] Es un sistema operativo desarrollado por Sun Microsystems. Es un sistema certificado como una versión de UNIX.

La principal desventaja de los lenguajes basados en máquina virtual, es que efectivamente son más lentos que los lenguajes completamente compilados, debido a la sobrecarga que genera tener una capa de software intermedia entre la aplicación y el hardware de la computadora.

Según (Carballeira, 2000), la Máquina Virtual de Java tiene tres (3) deficiencias, un conjunto de instrucciones no ortogonal[8], difícil de entender, ya que se utiliza un byte para codificar el código de operación de las instrucciones del procesador virtual Java y no posee un árbol de análisis sintáctico, o sea el código intermedio, usado en la MVJ, es simple y plano, es decir no incluye información acerca de la estructura del método original.

- **Plataforma Mono:** Es un proyecto de código abierto impulsado por Novell y creado por Miguel de Icaza como alternativa al Framework .NET para los programadores de software libre para brindar un grupo de herramientas, basadas en GNU/Linux y compatibles con .NET según lo especificado por el ECMA[9].

  Esta plataforma presenta importantes ventajas, es ideal para desarrollo de plataformas cruzadas y se ha posicionado como un entorno que permite ejecutar en Linux aplicaciones diseñadas para Microsoft .Net en entorno Windows, facilitando la migración de aplicaciones a Linux y aumentando su base de desarrolladores y usuarios. (Cano Ossa, Agosto 12, 2006).

  Según (Seoane, 2004), la plataforma Mono posee independencia de lenguajes, se puede usar clases escritas en cualquier lenguajes que este soporte (Python, C#, Mono Basic, Java, Nemerle, Boo); posee también independencia de plataforma, lo cual las aplicaciones son muy portables y la mayoría compatibles en binario entre plataformas; tienen un gran soporte para bases de datos MS SQL, MySQL, Postgres, OLE DB; posee una gestión automática de memoria, lo cual es una fuente inagotable de errores y se pierde una gran cantidad de tiempo programando esto, si se automatiza, se obtiene más tiempo que se puede dedicar a resolver el

[8] Un lenguaje de programación es ortogonal, si tiene el mismo número de instrucciones asociadas a cada uno de los tipos de datos.

[9] European Computer Manufacturers Association, ECMA Internacional es una organización basada en membrecías de estándares para la comunicación y la información. Creada en 1961.

verdadero problema; como otra característica presente está las aplicaciones GUI[10] multiplataforma, lo cual se pueden escribir aplicaciones con interfaz gráfica que se ejecutan invariantemente en multitud de plataformas.

Las aplicaciones se traducen a CIL[11], la ventaja fundamental de CIL frente los formatos usados anteriormente es claramente la independencia del lenguaje, por ejemplo, ya no tendrás que escribir tu código en Fortran o en C, simplemente porque las librerías que debes de usar están escritas en ese lenguaje, puedes, por ejemplo, escribir tus librerías en C# y usarlas desde Python sin ninguna dificultad, de hecho, no hay diferencia ente una librería escrita en Python, C# o Fortran si éstos generan CIL. (Seoane, 2004).

El concepto de máquina virtual fue usado por IBM en 1959 para describir uno de los primeros sistemas operativos que existieron en la historia de la computación. En 1970, el ambiente de programación de SmallTalk llevó la idea a un nuevo nivel y construyó una máquina virtual para soportar abstracciones orientadas a objetos de alto nivel, sobre las máquinas subyacentes.

A diferencia de los programas tradicionales que se ejecutan sobre el sistema directamente, los programas en la plataforma Mono se ejecutan sobre un entorno controlado de ejecución conocido como la máquina virtual. Este entorno proporciona numerosas ventajas sobre la ejecución tradicional directa: gestión de memoria automática (el sistema se encarga de recuperar automáticamente la memoria no usada por las aplicaciones simplificando la gestión a las aplicaciones), un entorno seguro de ejecución (donde podemos definir los recursos físicos y lógicos a los que la aplicación tiene acceso) y un sistema de control de errores y ejecución que permite una gestión de errores avanzada. (Hernández, 2004).

La principal desventaja que presenta esta plataforma es que es muy inmadura, todavía no ha madurado lo suficientemente como para implementar grandes proyectos de gestión, y debido a esto pues presentan errores como son en el

[10] Graphical User Interface, tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

[11] Common Intermediate Language, se conoció anteriormente como lenguaje intermedio de Microsoft (MSIL).

mecanismo de actualización, que maneja el redibujado de la aplicación, no funciona correctamente cuando ésta es ocultada por otra aplicación. Adicionalmente, el mecanismo de eventos presenta algunos problemas con algunos controles de interfaz.

- **.Net Framework:** Es una plataforma de software que conecta información, sistemas, personas y dispositivos. La plataforma .NET conecta una gran variedad de tecnologías de uso personal y de negocios, de teléfonos celulares a servidores corporativos, permitiendo el acceso a información importante, donde y cuando se necesiten. Desarrollado con base en los estándares de Servicios Web XML[12], .NET permite que los sistemas y aplicaciones, ya sea nuevos o existentes, conecten sus datos y transacciones independientemente del sistema operativo, tipo de computadora o dispositivo móvil que se utilice, o del lenguaje de programación empleados para crearlo. La idea fundamental de Microsoft .NET es un cambio de enfoque en lo que es la informática, pasando de un mundo de aplicaciones, sitios Web y dispositivos aislados a una infinidad de computadoras, dispositivos, transacciones y servicios que se conectan directamente y trabajan en conjunto para ofrecer soluciones más amplias y ricas en contenido. Esta plataforma cuenta con ciertas ventajas a tener en cuenta (MSDN, 2008):

    - **Código administrado:** El CLR[13] realiza un control automático del código para que este sea seguro, es decir controla los recursos del sistema para que la aplicación se ejecute correctamente.

    - **Interoperabilidad multilenguaje:** El código puede ser escrito en cualquier lenguaje compatible con .Net ya que siempre se compila en código intermedio (MSIL[14]).

    - **Compilación just-in-time**: El compilador JIT[15] incluido en el Framework

[12] Es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

13 Command Language Runtime, entorno común de ejecución para los diferentes lenguajes del .Net Framework.

[14] Es un código de bites que la Tecnología .NET de Microsoft utiliza para lograr independencia de la plataforma y seguridad en ejecución.

[15] Es una técnica para mejorar el rendimiento de sistemas de programación en las tecnologías .NET.

compila el código intermedio (MSIL) generando el código máquina propio de la plataforma. Se aumenta así el rendimiento de la aplicación al ser específico para cada plataforma.

- **Recolector de basura:** El CLR proporciona un sistema automático de administración de memoria denominado recolector de basura (Garbage Collector[16]). El CLR detecta cuándo el programa deja de utilizar la memoria y la libera automáticamente. De esta forma el programador no tiene por que liberar la memoria de forma explícita aunque también sea posible hacerlo manualmente.

- **Seguridad de acceso al código**: Se puede especificar que una pieza de código tenga permisos de lectura de archivos pero no de escritura. Es posible aplicar distintos niveles de seguridad al código, de forma que se puede ejecutar código procedente del Web sin tener que preocuparse si esto va a estropear el sistema.

- **Despliegue:** Por medio de los ensamblados resulta mucho más fácil el desarrollo de aplicaciones distribuidas y el mantenimiento de las mismas. El Framework realiza esta tarea de forma automática mejorando el rendimiento y asegurando el funcionamiento correcto de todas las aplicaciones.

Algunas desventajas que presenta esta plataforma es el mantenimiento en múltiples lenguajes. Mantener un proyecto en múltiples lenguajes es costoso. Si una aplicación está realizada en varios lenguajes se necesitan expertos en varios lenguajes para entenderla y mantenerla, aumentando los costos. Otros de las desventajas fundamentales es que no es multiplataforma, o sea ella solo está disponible para la familia de Windows. El tema de las licencias es también otras de las desventajas por lo que es un código cerrado y no hay licencias libres. La infraestructura para desarrollar en .NET representa un alto costo para las empresas. Y para nuestro país la desventaja principal es que esta plataforma está representada por la Microsoft, por lo que por razones políticas no se puede obtener

[16] Mecanismo implícito de gestión de memoria implementado en algunos lenguajes de programación.

para utilizarse en software.

La dirección del proyecto escogió para el desarrollo, la **plataforma .NET**. Hay varios elementos que se tuvieron en cuenta para esta decisión:

- No se tenía experiencia por el equipo de desarrollo en construcción de proyectos de complejidad similar.
- El número de personas que tenían dominio sobre la plataforma .NET en la universidad era superior a los de la plataforma Java o PHP.
- Existía un precedente por parte de otro equipo de desarrollo en la construcción de un proyecto también para la República Bolivariana de Venezuela, hecho en .NET.
- La plataforma Mono no estaba concluida y no se consideraba un Framework estable y maduro, además, los entornos de desarrollo no ofrecían las mismas ventajas y velocidad de trabajo que en el Visual Studio .NET.

En base al estudio de plataformas realizado y los factores que se mencionaron. La plataforma Java no era una buena opción, existía poco conocimiento sobre esta y para el desarrollo del proyecto en corto período de tiempo era vital que el equipo de desarrolladores estuviera preparado, lo mismo sucedía con la plataforma Mono que se veía agravado por el hecho de no contar con un entorno de desarrollo tan avanzado como Visual Studio .Net.

Por todas estas razones se considera la elección de la plataforma .NET una buena decisión, el entorno de desarrollo posibilitó dar una respuesta mucho más eficiente en cuanto a tiempo a lo que en otras plataformas hubiera significado más horas de construcción. Se contaba entonces con una plataforma estable que resolvía algunos temas como la administración de memoria (Recolector de basura), administración del código y seguridad de acceso a este. Además posibilitó la reutilización de componentes libres y otros desarrollados por proyectos de la Universidad.

Una vez escogida la plataforma comenzaba la tarea de decidir qué metodologías y herramientas se irían a seleccionar, lo cual veremos en el próximo epígrafe.

1.5 Metodologías y herramientas de modelado para el diseño de sistemas.

El modelado de sistemas informáticos de altos grados de complejidad requiere del uso de la Ingeniería de Software para llevar a cabo de una manera organizada y bien definida las tareas del Software en cuestión. Según (Medina Pasaje, Marzo 7, 2006) existen diferentes metodologías de desarrollo para llevar a cabo un sistema informático, y cada una de ellas contiene ciertos pasos que identifican el proceso de ingeniería de software:

- **Análisis de requisitos:** Extraer los requisitos de un producto de software es la primera etapa para crearlo. Mientras que los clientes piensan que ellos saben lo que el software tiene que hacer, se requiere de habilidad y experiencia en la ingeniería de software para reconocer requisitos incompletos, ambiguos o contradictorios. (Medina Pasaje, Marzo 7, 2006).

- **Especificación:** Es la tarea de describir detalladamente el software a ser escrito, en una forma matemáticamente rigurosa. En la realidad, la mayoría de las buenas especificaciones han sido escritas para entender y afinar aplicaciones que ya estaban desarrolladas. Las especificaciones son más importantes para las interfaces externas, que deben permanecer estables. (Medina Pasaje, Marzo 7, 2006).

- **Diseño y arquitectura:** Se refiere a determinar cómo funcionará de forma general sin entrar en detalles. Consiste en incorporar consideraciones de la implementación tecnológica, como el hardware, la red. (Medina Pasaje, Marzo 7, 2006).

- **Implementación:** Reducir un diseño a código puede ser la parte más obvia del trabajo de ingeniería de software, pero no es necesariamente la porción más larga. (Medina Pasaje, Marzo 7, 2006).

- **Prueba:** Consiste en comprobar que el software realice correctamente las tareas indicadas en la especificación. Una técnica de prueba es probar por separado cada módulo del software, y luego probarlo de forma integral. (Medina Pasaje, Marzo 7, 2006).

- **Documentación:** Realización del manual de usuario, y posiblemente un manual

técnico con el propósito de mantenimiento futuro y ampliaciones al sistema. (Medina Pasaje, Marzo 7, 2006).

- **Mantenimiento:** Mantener y mejorar el software para enfrentar errores descubiertos y nuevos requisitos. Esto puede llevar más tiempo incluso que el desarrollo inicial del software. Alrededor de 2/3 de toda la ingeniería de software tiene que ver con dar mantenimiento. Una pequeña parte de este trabajo consiste en arreglar errores. La mayor parte consiste en extender el sistema para hacer nuevas cosas. (Medina Pasaje, Marzo 7, 2006).

1.5.1 Metodologías.

Existen dos grupos en los cuáles se dividen estas metodologías, los métodos tradicionales y los procesos ágiles. Todas contribuyen a un buen y organizado desarrollo de software aunque tengan sus marcadas diferencias. A continuación una breve descripción de dos de estas metodologías:

- **Rational Unified Process (RUP).** (Jacobson, y otros, 2000)**.**

El Proceso Unificado Racional es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

RUP es uno de los procesos más generales de los existentes actualmente, ya que en general está pensado para adaptarse a cualquier proyecto, no tan solo de software. Sus autores destacan que el proceso de software propuesto por RUP tiene tres características esenciales: está dirigido por los casos de uso, está centrado en la arquitectura, y es iterativo e incremental. (Jacobson, y otros, 2000).

- Proceso dirigido por casos de uso.

Los casos de uso son una técnica de captura de requisitos que fuerza a pensar en términos de importancia para el usuario y no sólo en términos de funciones que sería bueno contemplar. Se define un caso de uso como un fragmento de funcionalidad del sistema que proporciona al usuario un valor añadido. Los casos de uso representan los requisitos funcionales del sistema. En RUP los casos de uso no son sólo una herramienta para especificar los requisitos del sistema. También guían su diseño,

implementación y prueba. Los Casos de Uso constituyen un elemento integrador y una guía del trabajo. Los Casos de Uso no sólo inician el proceso de desarrollo sino que proporcionan un hilo conductor, permitiendo establecer trazabilidad entre los artefactos que son generados en las diferentes actividades del proceso de desarrollo.

- Proceso centrado en la arquitectura.

La arquitectura de un sistema es la organización o estructura de sus partes más relevantes, es necesaria para controlar el desarrollo e involucra los aspectos estáticos y dinámicos más significativos del sistema. La definición de la arquitectura debe tomar en consideración elementos de calidad del sistema, rendimiento, reutilización y capacidad de evolución por lo que debe ser flexible durante todo el proceso de desarrollo. La arquitectura se ve influenciada por la plataforma software, sistema operativo, gestor de bases de datos, protocolos, consideraciones de desarrollo. RUP presta especial atención al establecimiento temprano de una buena arquitectura que no se vea fuertemente impactada ante cambios posteriores durante la construcción y el mantenimiento. Cada producto tiene tanto una función como una forma. La función corresponde a la funcionalidad reflejada en los casos de uso y la forma la proporciona la arquitectura. Existe una interacción entre los casos de uso y la arquitectura, los casos de uso deben encajar en la arquitectura cuando se llevan a cabo y la arquitectura debe permitir el desarrollo de todos los casos de uso requeridos, actualmente y en el futuro. Esto provoca que tanto arquitectura como casos de uso deban evolucionar en paralelo durante todo el proceso de desarrollo de software.

- Proceso iterativo e incremental.

  De esta forma el trabajo se divide en partes más pequeñas o mini proyectos. Permitiendo que el equilibrio entre casos de uso y arquitectura se vaya logrando durante cada mini proyecto, así durante todo el proceso de desarrollo. Cada mini proyecto se puede ver como una iteración (un recorrido más o menos completo a lo largo de todos los flujos de trabajo fundamentales) del cual se obtiene un incremento que produce un crecimiento en el producto.

  Una iteración puede realizarse por medio de una cascada. Se pasa por los flujos fundamentales (Requisitos, Análisis, Diseño, Implementación y Pruebas), también

existe una planificación de la iteración, un análisis de la iteración y algunas actividades específicas de la iteración. Al finalizar se realiza una integración de los resultados con lo obtenido de las iteraciones anteriores.

El proceso iterativo e incremental consta de una secuencia de iteraciones. Cada iteración aborda una parte de la funcionalidad total, pasando por todos los flujos de trabajo relevantes y refinando la arquitectura. Se puede determinar si han aparecido nuevos requisitos o han cambiado los existentes, afectando a las iteraciones siguientes. Durante la planificación de los detalles de la siguiente iteración, el equipo también examina cómo afectarán los riesgos que aún quedan al trabajo en curso. Toda la retroalimentación de la iteración pasada permite reajustar los objetivos para las siguientes iteraciones. Se continúa con esta dinámica hasta que se haya finalizado por completo con la versión actual del producto.

RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades. Estas fases son: inicio, elaboración, construcción y transición.

Las primeras iteraciones (en las fases de Inicio y Elaboración) se enfocan hacia la comprensión del problema y la tecnología, la delimitación del ámbito del proyecto, la eliminación de los riesgos críticos, y al establecimiento de una baseline[17] de la arquitectura. Durante la fase de inicio las iteraciones hacen ponen mayor énfasis en actividades modelado del negocio y de requisitos. En la fase de elaboración, las iteraciones se orientan al desarrollo de la baseline de la arquitectura, abarcan más los flujos de trabajo de requerimientos, modelo de negocios (refinamiento), análisis, diseño y una parte de implementación orientado a la baseline de la arquitectura. En la fase de construcción, se lleva a cabo la implementación del producto por medio de una serie de iteraciones. Para cada iteración se seleccionan algunos casos de uso, se refina su análisis y diseño y se procede a su implementación y pruebas. Se realiza una pequeña cascada para cada ciclo. Se realizan tantas iteraciones hasta que se termine la implementación de la nueva versión del producto. En la fase de

[17] Instantánea de todos los artefactos del proyecto, registrada para efectos de gestión de configuración y control de cambios.

transición se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios.

- **XP (Extreme Programming).**

La **programación extrema** o *eXtreme Programming* (XP) es una aproximación a la ingeniería de software formulada por Kent Beck, autor del primer libro sobre la materia, *Extreme Programming Explained: Embrace Change*. Se trata de un proceso ágil de desarrollo de software. Es una metodología que está basada en los principios del manifiesto ágil que es el punto de partida de XP y de otras metodologías ágiles que están apareciendo:

1. La metodología XP está diseñada para proyectos de corta duración.
2. Los individuos e interacciones son más importantes que los procesos y herramientas.
3. Software que funcione es más importante que documentación exhaustiva.
4. La colaboración con el cliente es más importante que la negociación de contratos.
5. La respuesta ante el cambio es más importante que el seguimiento de un plan.
6. XP se basa en 12 prácticas ya conocidas y que se refuerzan entre sí:
    - Retroalimentación a escala fina.
        - Desarrollo dirigido por pruebas.
        - El juego de la planificación.
        - Cliente in-situ.
        - Programación en parejas.
    - Proceso continuo en lugar de por lotes.
        - Entregas pequeñas.
        - Refactorización sin piedad.

- Integración continúa.

- Entendimiento compartido.
  - Diseño Simple.
  - Metáfora del sistema.
  - Propiedad colectiva del código.
  - Convenciones de código. Estándares de programación.

- Bienestar del programador.
  - 40 horas por semana.

Diseño Simple: Como fue mencionado anteriormente, el Diseño Simple o "Simple Design" establece que siempre deben ser realizadas las tareas de la forma más simple posible. Para lograr un diseño simple, nunca serán adicionadas características funcionales que no formen parte de la propuesta de requisitos analizada en la planificación del juego. En la Programación Extrema, un buen diseño es esencial para asegurar el éxito esperado. (Beck, 2005).

La metodología RUP, fue la seleccionada por la dirección del proyecto Registros y Notarias como la adecuada para llevar a cabo la confección de dicho proyecto, en el cuál se incluye el Subsistema de Administración Financiera. Esta metodología además de varias de sus características antes mencionadas, es identificada como perteneciente a los procesos de desarrollo del tipo pesados. Se le llama de esta forma por la extensión en tiempo de desarrollo, así como el gran número de especialistas necesarios para la confección del software. A diferencia de las metodologías ágiles, dentro de las cuáles se encuentra XP, que funcionan con un número inferior de iteraciones y artefactos[18], así como un menor número de especialistas y finalmente el tiempo de desarrollo es relativamente corto.

Se sigue dicha metodología de desarrollo debido a que independientemente de ser una metodología tradicional puede hacerse tan ágil como se desee, ya que en su misma concepción una de las características más fundamentales es que es muy configurable ya

[18] Herramientas organizativas identificadas para la confección organizada y correcta del software en cuestión.

que sus artefactos no son obligados a usarse, solamente los que se necesite por parte del equipo de desarrollo. Esta metodología permite evaluar constantemente la calidad del producto así como tener una idea bastante cercana sobre el estado actual del proyecto al poder contar con los artefactos generados en cada una de las iteraciones.

Se consideró que el uso de esta metodología para la confección de este proyecto fue el más indicado, según la magnitud y complejidad de todo su negocio, como también su proceso de desarrollo a distancia, no se determinó aplicar una metodología ágil y si una correcta y robusta captura de requisitos, lo cual es exigido como premisa fundamental por la metodología RUP.

1.5.2 Herramientas.

Las Herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) son las aplicaciones informáticas que tienen como tarea principal lograr una mayor productividad en el desarrollo de software y una reducción de costos de desarrollo. (Pacheco Iglesias, y otros, 2008). Estas herramientas contribuyen de manera directa en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, calculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras. Según (Pacheco Iglesias, y otros, 2008) cada una de estas herramientas persigue nueve Objetivos principales:

- 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.

A continuación se relacionan algunas de estas herramientas existentes en el mundo actualmente:

- **Rational Rose.**

  Rational Rose se considera una de las herramientas CASE para modelado más potente existente en el mercado mundial. Esto se basa principalmente en el nivel de integración que tiene esta con el resto de las herramientas que lo acompañan en la suite, donde aparece el Rational Clear CASE para el control de versiones, el Rational Clear Quest para el control de cambios, el Rational Model Integrator, entre otros. La posibilidad de generar y realizar ingeniería inversa en una buena cantidad de lenguajes de programación es otro aspecto a destacar así como también el número de framework que vienen predefinidos, entre los cuales se pueden citar j2ee, .net, Visual Basic 6, C++ entre otros. (Rational, 2002).

- **Visual Paradigm for 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 más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Presenta un conjunto de características, según (Pacheco Iglesias, y otros, 2008) las más fundamentales son:

- Modelado colaborativo con CVS[19].
- Interoperabilidad con modelos UML 2.0 a través de XML.
- Generación de código - Modelo a código, diagrama a código.
- Editor de Detalles de Casos de Uso - Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación.
- Generador de informes para generación de documentación.
- Distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML.

- **Enterprise Architect 7.0.**

Enterprise Architect (EA) combina el poder de la última especificación UML 2.1 con alto rendimiento, interfaz intuitiva, para traer modelado avanzado al escritorio, y para el equipo completo de desarrollo e implementación. EA puede equipar a su equipo entero, incluyendo analistas, evaluadores, administradores de proyectos, personal del control de calidad, equipo de desarrollo y más, por una fracción del costo de algunos productos competitivos. EA provee trazabilidad completa desde el análisis de requerimientos hasta los artefactos de análisis y diseño, a través de la implementación y el despliegue. Combinados con la ubicación de recursos y tareas incorporados, los equipos de Administradores de Proyectos y Calidad están equipados con la información que ellos necesitan para ayudarles a entregar proyectos en tiempo.

Enterprise Architect 7.0 es una herramienta de construcción y modelado de software de alto rendimiento basado en el estándar de UML 2.1, diseñada para ayudar a construir software robusto y fácil de mantener. Brinda una solución de

[19] Concurrent Versions Systems. Aplicación informática que implementa un sistema de control de versiones: mantiene el registro de todo el trabajo y los cambios en los ficheros.

> modelado verdaderamente ágil, fácil de usar, rápido, flexible y con una interfaz gráfica amigable. Soporta generación e ingeniería inversa de código fuente para muchos lenguajes populares, incluyendo C++, C#, Java, Delphi, VB.Net, Visual Basic y PHP. Es una herramienta multiusuario, con seguridad y administración de permisos incorporada. Soporta diferentes repositorios basados en DBMS (Sistemas Manejadores de BD), incluyendo Oracle, SQL Server, My SQL, PostgreSQL. Brinda soporte para control de versiones y posee bajos costos de licencias. (SPARX, 2000).

Atendiendo a la característica de utilizar el UML como lenguaje de modelado se selecciona la herramienta case **Enterprise Architect** 7.0. Esta herramienta está diseñada para ayudar a construir software robusto y fácil de mantener basada fundamentalmente en la utilización de UML 2.1 como lenguaje de modelado. Es una herramienta multiusuario con seguridad y administración de permisos incorporada. Fácilmente integrable con el Subversion y el VSS. Permite generación de código fuente e ingeniería inversa para el lenguaje a utilizar en las fases posteriores del ciclo de desarrollo.

1.6 Conclusiones del Capítulo.

El presente capítulo abordó un primer momento sobre los principales procesos del negocio (Fondos de Caja Chica), realizando una breve explicación de cómo se llevan a cabo. Además se analizaron los principales sistemas financieros existentes en Cuba y el mundo.

Partiendo de lo mencionado durante el presente capítulo se define la siguiente propuesta de solución técnica, la cual regirá el desarrollo de este trabajo de diploma en sus fases posteriores.

Se sigue la metodología de desarrollo de software **Rational Unified Process (RUP)** debido a que independientemente de ser una metodología tradicional puede hacerse tan ágil como se desee. Esta metodología permite evaluar constantemente la calidad del producto así como tener una idea bastante cercana sobre el estado actual del proyecto al poder contar con los artefactos generados en cada una de las iteraciones. Por último pero no por eso menos importante utiliza el UML como lenguaje de modelado. Coincidiendo con esta característica se encuentra la herramienta case seleccionada el **Enterprise Architect 7.0**. Esta herramienta está diseñada para ayudar a construir software robusto y

fácil de mantener basada fundamentalmente en la utilización de UML 2.1 como lenguaje de modelado. Es una herramienta multiusuario con seguridad y administración de permisos incorporada. Fácilmente integrable con el Subversion y el VSS. Permite generación de código fuente e ingeniería inversa para el lenguaje a utilizar en las fases posteriores del ciclo de desarrollo.

Entre las plataformas se selecciona **Microsoft.NET,** por las facilidades que brinda para la comunicación entre las aplicaciones. Además con el framework.NET se hace mucho más sencillo el desarrollo de aplicaciones al encontrarse integrado en él un conjunto de lenguajes, herramientas y servicios que facilitan en gran medida el trabajo. Perfectamente compatible con la plataforma seleccionada, **Microsoft Visual Studio**; es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) pensado para hacer rápida y fácilmente las aplicaciones de la nueva generación. Este IDE puede utilizarse para construir aplicaciones dirigidas a **Windows** (utilizando Windows Forms), **Web** (usando ASP.NET y Servicios Web) y **dispositivos portátiles** (utilizando .NET Compact Framework).

Capítulo 2: Propuesta del Sistema.

En este capítulo se hace una valoración de las arquitecturas existentes a escoger para la realización del sistema, de cada una se hace un análisis de por qué no es apropiada para dicha realización y se exponen los criterios o argumentos necesarios de la seleccionada. Se analizarán los patrones de diseño a utilizar en nuestro sistema y estándares de codificación seleccionados por la dirección del proyecto, así como un recorrido por las tendencias y tecnologías más conocidas en el mundo.

2.1 Arquitectura del Software.

Según (Clements, 1996), la arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se le percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.

Otras de las definiciones de arquitectura de software la enfatiza (Pacheco Iglesias, y otros, 2008) que plantea: “La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución”.

2.1.1 Estilo Arquitectónico.

No es objetivo de este epígrafe hacer un análisis de los distintos estilos arquitectónicos en su totalidad, sino solo aquellos que por su estructura y solución podrían considerarse para la construcción del sistema, analizando las ventajas y desventajas que estos poseen para determinar si debe ser o no aceptado para el desarrollo del software.

Según (Shaw, y otros, Abril 1996), identifican los estilos arquitectónicos como un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer en sistema o subsistema, junto con las restricciones locales o globales de la forma en que se lleva a cabo la composición. Es en gran medida la interacción entre los componentes, mediados por conectores, lo que confiere a los distintos estilos sus características distintivas.

2.1.1.1 Estilos de Flujo de Datos.

La familia de los estilos de Flujos de Datos, enfatiza la reutilización y la modificación, es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos, ejemplos de estas podemos encontrar las arquitecturas de proceso secuencial por lotes, red de flujos de datos, y tuberías y filtros.

Según (Reynoso, y otros, 2004), el sistema tubería-filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes. En el estilo secuencial por lotes, los componentes son programas independientes; el supuesto es que cada paso se ejecuta hasta completarse antes que se inicie el paso siguiente.

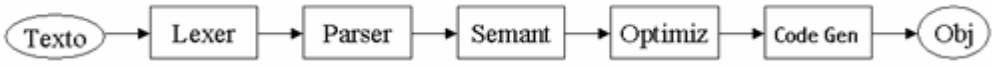

Figura 5 Compilador en tubería-filtro.

De las arquitecturas que ofrece esta familia de estilos no se recomienda ninguna de ellas para el desarrollo de un sistema con las características que se necesita. A pesar de que es simple de entender, implementar, al igual que enfatiza la reutilización y la modificación, en los procesos que complementan al proceso de Fondos de Caja Chica pasa por determinados pasos, no se escoge pues la modificación de la Caja Chica constituirá uno de los procesos a automatizar y en otros casos lo único que se le hace es trasmitir esos datos a diferentes departamentos pero sin modificaciones a este documento.

Además de las desventajas que traería este estilo para el sistema, se le integran las siguientes:

- El patrón puede resultar demasiado simplista, especialmente para la orquestación de los servicios que podrían ramificar la ejecución lógica de negocios de formas complicadas, (Reynoso, y otros, 2004).

- No se maneja con demasiada eficiencia construcciones condicionales, bucles y otras lógicas de control de flujo. Agregar un paso suplementario afecta el rendimiento de cada ejecución en la tubería. (Reynoso, y otros, 2004).

- Según (Reynoso, y otros, 2004), el estilo no es apto para manejar situaciones interactivas, sobre todo cuando se requieren actualizaciones incrementales de la representación en pantalla.

No se seleccionó ningún estilo de esta familia debido a su simplicidad, no son los más adecuados, pues los procesos que complementan al proceso de Fondos de Caja Chica (solicitudes de vales de caja chica, solicitudes de reposiciones y solicitudes de modificaciones) son complejos en su mayoría por lo cual esto traería desventajas en el sistema.

2.1.1.2 Estilos Centrados en Datos.

Otros de los estilos arquitectónicos es el Centrado en Datos, esta familia de estilos enfatiza la integridad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Entre los sub-estilos característicos están los Repositorios, Base de Datos y Arquitectura de Pizarra.

En la Arquitectura de Pizarra o Repositorio se tienen dos componentes principales, una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él.

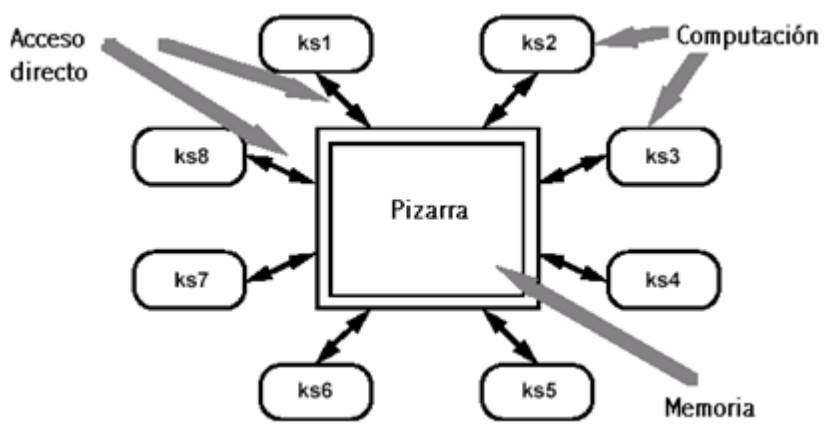

Figura 6 Arquitectura de Pizarra o Repositorio.

Según (Reynoso, y otros, 2004), este estilo arquitectónico se usa en aplicaciones que requieren complejas interpretaciones de procesos de señales (reconocimientos de patrones, reconocimientos de habla), o en sistemas que involucran acceso compartido a datos con agentes débilmente acoplados.

No se considera dicha arquitectura apropiada para la realización del sistema, pues ésta más bien enfatiza los datos y no el modelo, además no son objetivos de este tipo de estilo la reutilización y la flexibilidad del código. En el entorno del sistema o del negocio de los procesos de Fondo de Caja Chica no existen los componentes individuales que actúen sobre la pizarra o el repositorio.

2.1.1.3 Estilos Peer-to-Peer.

Los estilos Peer-to-Peer o también llamada de componentes independientes, enfatiza la modificación por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Algunos de los miembros de esta familia de estilos son las Arquitecturas Basadas en Eventos, Arquitectura Basadas en Servicios y la Arquitectura Basada en Recursos. (Reynoso, y otros, 2004).

A pesar de que los eventos se utilizan para manejar las interfaces de usuario,

desde el punto de vista arquitectónico suponen que los métodos y procedimientos no se llaman de forma directa sino que uno o más de estos métodos puede ser disparado por un evento, de esta forma el módulo ofrece, tanto una interfaz de métodos como una interfaz de eventos. Este estilo más bien se utiliza en ambientes de integración de herramientas, interfaces de usuario o cuando no se quiere que exista un acoplamiento entre el emisor y el receptor.

Entre las desventajas que presentan esta familia de estilos están las siguientes:

- El estilo no permite construir respuestas complejas a funciones de negocio. (Reynoso, y otros, 2004).
- Un componente no puede utilizar los datos o el estado de otro componente para efectuar su tarea. (Reynoso, y otros, 2004).
- Cuando un componente anuncia un evento, no tiene idea sobre qué otros componentes están interesados en él, ni el orden en que serán invocados, ni el momento en que finalizan lo que tienen que hacer. Pueden sugerir problemas de performance global y de manejo de recursos cuando se comparten un repositorio común para coordinar la interacción. (Reynoso, y otros, 2004).

No se selecciona los estilos Peer-to-Peer, porque el negocio de los procesos que se desean automatizar, tienen sus propias reglas, donde es muy importante el rendimiento del sistema, destacando la flexibilidad y la reusabilidad, además de que existen ciertas condiciones en las que el sistema no podría funcionar, rompiendo así con los esquema propuestos por dichos estilos de arquitectura.

2.1.1.4 Estilo de Llamada y Retorno.

Los estilos de Llamada y Retorno enfatizan en lo escalable y modificable, son los estilos más utilizados o más generalizados por los sistemas a gran escala. Dentro de esta familia se encuentran el Modelo-Vista-Controlador (MVC), la Arquitectura Orientada a Objetos y la Arquitectura en Capas.

El **Modelo-Vista-Controlador** (MVC), ha sido propio de las aplicaciones en *SmallTalk* por lo menos desde 1992, antes que se generalizaran las arquitecturas en capas múltiples, según (Reynoso, y otros, 2004), en ocasiones se define más bien como un patrón de diseño como práctica recurrente.

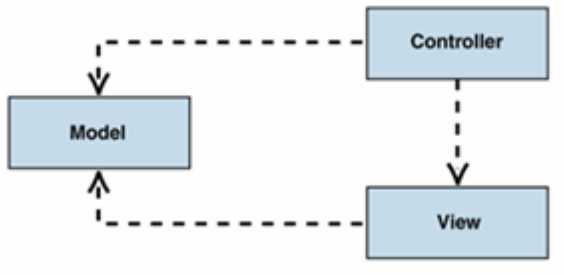

Figura 7 Modelo-Vista-Controlador.

La idea fundamental de este estilo es separar el modelo del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes, de manera tal que puedan ser reutilizados y desarrollados de forma paralela por diferentes personas.

Para el desarrollo de la capa de presentación o interface se escogió un Framework creado por el proyecto Identidad, en la Universidad de las Ciencias Informáticas, el cual se basa en el patrón MVC (Modelo-Vista-Controlador), de esta forma se aprovechan las ventajas del patrón y se deja abierta la posibilidad de escoger otra arquitectura diferente, por toda la complejidad del negocio de los procesos antes mencionados; aunque desde el punto de vista arquitectónico la selección de la arquitectura sería diferente.

Las **Arquitectura en Capas** constituye uno de los estilos que aparecen con mayor frecuencia mencionados como categorías mayores del catálogo. Según (Shaw, y otros, 1996), definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a cada capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.

Esta arquitectura puede decirse que su finalidad es abstraer las funcionalidades de una capa de manera tal que pueda ser totalmente remplazada. De esta arquitectura la más común es la compuesta por tres (3) capas, *presentación*, *modelo o reglas del negocio* y *acceso a datos*. De esta forma se puede remplazar cualquier capa sin afectar a las otras solamente cambiar las referencias de las implicadas en el cambio. Aunque sea la de tres (3) capas la más común, esto no significa que mientras crezca la complejidad, se mantienen estas tres capas, sino al contrario, si la complejidad crece y es necesario entonces pueden aparecer otras capas para descomponer las funcionalidades que en estas aparezcan. A su vez estas capas pueden estar compuestas por subcapas y una capa o subcapa puede estar compuesta por una o varias clases del diseño.

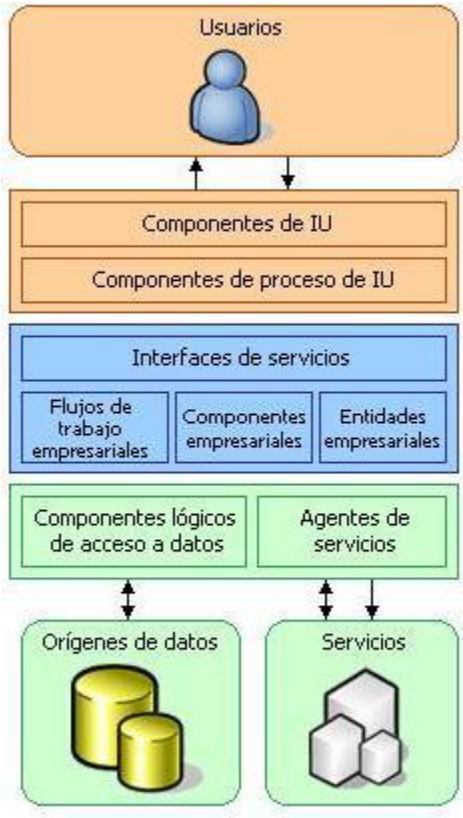

Figura 8 Arquitectura en Capas.

Esta característica de la arquitectura de capas permite implementar las reglas del negocio en una capa aparte, para que estas reglas puedan ser usadas por otros sistemas o por otros servicios que necesiten estas funcionalidades, como es la integración entre dos sistemas dentro del mismo Registro y Notarías, los cuales son los sistemas registrales y el sistema de Administración Financiera, todo esto evita la duplicación de código y una mejor organización.

Según (Trowbridge, 2003), las ventajas que presenta la arquitectura de capas son:

- El mantenimiento y las mejoras de la solución son fáciles debido al bajo acoplamiento entre las capas, la alta cohesión de las clases y la habilidad de cambiar su implementación sin cambiar las interfaces. Esto es muy importante para el desarrollo del sistema pues este debe permitir cambios ya sea de requisitos funcionales, agregaciones o mejoras en las funcionalidades.

- Otras soluciones pueden rehusar las funcionalidades expuestas por las diferentes capas, especialmente si las capas de las interfaces son diseñadas con la reutilización en mente.

- El desarrollo distribuido es fácil si este se puede dividir con las capas como fronteras. Esto lo que permite es el desarrollo de forma paralela y posibilitando la especialización de los desarrolladores e incrementando la productividad.

- Distribuir las capas a lo largo de múltiples capas físicas puede mejorar la escalabilidad, tolerancia a errores y rendimiento. Esto lo que quiere decir es que como una capa física puede estar compuesta por una o más computadoras, si se logra distribuir el rendimiento aumenta y de esta forma el desempeño, la durabilidad y la escalabilidad del sistema.

- Beneficios a la hora de realizar las pruebas teniendo bien definidas las interfaces de las capas por la habilidad de cambiar las implementaciones de estas capas manteniendo la interfaz.

También se señalan algunas desventajas en este estilo como son:

- La sobrecarga extra de pasar los mensajes a través de las capas en lugar de llamar los componentes directamente, puede impactar de forma negativa en el rendimiento. Lo cual se puede mitigar con el uso de un modo relajado.

- El desarrollo de las interfaces de usuario puede algunas veces tomar tiempo si la estructura de las capas evita el uso de componentes de interfaz de usuario que interactúan directamente con la Base de Datos. Esto ya esta mitigado por la plataforma de desarrollo que se escogió para el desarrollo del sistema, lo cual la mayoría de los componentes interactúan con DataSet[20].

- El uso de capas ayuda a controlar y encapsular la complejidad de aplicaciones grandes, pero agrega complejidad a las aplicaciones simples.

  Como se ha expuesto anteriormente, el sistema que se plantea no es una aplicación simple por lo que es necesaria esta encapsulación.

- Cambios en las interfaces de las capas inferiores tienden a propagarse a los altos niveles, especialmente si el modo relajado es usado.

El modo relajado es una de las formas de concebir la arquitectura en capas, se le llama modo relajado cuando las capas superiores pueden interactuar con las capas inferiores, esto aumenta el rendimiento pero implica menos flexibilidad en la aplicación, mientras que la otra forma de concebir esta arquitectura es que una capa este ligada únicamente con la capa inferior inmediata, esta mantiene un bajo acoplamiento pero puede implicar impactos negativos en el rendimiento. (Pacheco Iglesias, y otros, 2008).

Ésta fue la arquitectura que fue escogida para el desarrollo del sistema, por todas las ventajas que posee en comparación con otros estilos antes

[20] Según (Bipin), es una estructura de datos del Microsoft .NET framework que encapsula un conjunto de datos y soporta un modelo relacional de una o más tablas.

mencionados y el hecho de que se combine con el patrón Modelo-Vista-Controlador esto obtiene resultados muy satisfactorios en el desarrollo del sistema.

Resaltar que existen otros tipos de estilos arquitectónicos que aquí no se mencionaron porque no correspondían sus aplicaciones con el sistema que se pretende desarrollar, para estudiar estos estilos más a profundidad, y los que no se mencionan aquí se pueden remitir a (Trowbridge, 2003).

2.1.2 Framework para la Gestión de la Capa de Presentación.

En el desarrollo de software, un framework es una estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado (Pérez Acosta, y otros, 2008). Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje de scripting[21] entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Un framework representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio. Las plataformas de desarrollo orientadas a objetos, llámense simplemente framework son la piedra angular de la moderna ingeniería del software. El desarrollo del framework está ganando rápidamente la aceptación debido a su capacidad para promover la reutilización del código del diseño y el código fuente. Los framework son los Generadores de Aplicación que se relacionan directamente con un dominio específico, es decir, con una familia de problemas relacionados. (Pérez Acosta, y otros, 2008).

Una tendencia bastante reciente en el mundo de la programación es la utilización de framework como base para el desarrollo de las aplicaciones, en ellos se definen reglas y funcionalidades que serán utilizadas con gran frecuencia dentro de la aplicación. A continuación se muestran algunas de las ventajas que provee la utilización de framework. (Pérez Acosta, y otros, 2008).

- El programador no necesita plantearse una estructura global de la aplicación, sino que el framework le proporciona un esqueleto que hay que

[21] Un lenguaje de programación interpretado es aquel que está diseñado para ser ejecutado por medio de un intérprete, en contraste con los lenguajes compilados.

"rellenar".

- Facilita el desarrollo, como la estandarización de muchos aspectos dentro del sistema, se hace más ágil el desarrollo y la comunicación entre los miembros del equipo de trabajo.

- Es más fácil encontrar herramientas (utilidades, librerías) adaptadas al framework concreto para facilitar el desarrollo.

Básicamente un framework evita construir una aplicación desde cero (0), típicamente un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto, también se debe tener en cuenta que para comenzar a desarrollar un framework se debe conocer sus especificaciones.

El framework escogido como se mencionó antes fue el desarrollado por el proyecto de Identidad de la Universidad de las Ciencias Informáticas, el cual tiene elementos del patrón Modelo-Vista-Controlador (MVC). Estas ideas quedan encapsuladas en la capa de presentación por ser la que está compuesta por los elementos de este patrón. El uso de este framework no cambia para nada el estilo escogido para el desarrollo de la aplicación, el uso de éste es solo para la gestión de la capa de presentación.

Se analizarán las ventajas y desventajas del uso de este framework, algunas de ellas son propias del patrón MVC y otras de la implementación en concreta de este.

2.1.2.1 Ventajas.

Entre las ventajas del framework escogido están las siguientes:

- *Múltiples vistas usando el mismo modelo*. Dado que las vistas se encuentra separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente. (Reynoso, y otros, 2004). En el framework el modelo y el controlador se pueden unir, pero si los desarrolladores tienen esto en cuenta, separando los dos componentes esta ventaja se puede aprovechar.

- *Fácil para soportar nuevos tipos de clientes*. Para soportar nuevos tipos de clientes solo se debe escribir la vista y el controlador para este y se reutiliza el modelo ya existente. (Dass, 2009). Esto es muy importante si alguna vez se decide migrar el sistema a software libre.

- *Claridad de diseño*. Mirando la lista de métodos públicos, debe ser fácil entender cómo controlar el funcionamiento del modelo. (Dass, 2009). Esto depende además en lo complejo de la solución, a medida que el software crece es necesario ir desarrollando otros diagramas como apoyo.

- *Fácil de crecer*. Los controladores y las vistas pueden crecer según crece el modelo, viejas versiones de las vistas y los controladores pueden usarse mientras que la interfaz común es mantenida. (Dass, 2009).

- *Seguridad a nivel de aplicación*. El framework sienta las bases para brindar seguridad a nivel de aplicación, mediante un fichero XML de configuración donde se definen roles y en base a estos se gestiona el acceso a las acciones, las acciones constituyen en el framework de cierta forma los controladores, aunque pueden tener otras funciones.

- *Colores de interfaz configurable*. Se puede cambiar el color de toda la aplicación con solo unos pocos cambios en la configuración de éste.

- *Configuración del Menú de Opciones.* Mediante un fichero XML de configuración se puede cambiar la estructura del menú de opciones, donde cada opción puede contener opciones y así sucesivamente por varios niveles.

- *Múltiples controladores usando la misma vista.* Debido a que la vista está completamente separada del controlador y no se conoce, sino que al revés, la misma vista puede ser utilizada por distintos controladores, además permite que la vista sea modificada en su estructura y diseño sin afectar a los controladores.

2.1.2.2 Desventajas.

Entre las desventajas del framework escogido se pueden ver las siguientes:

- *Complejidad*. Incrementa la naturaleza basada en eventos del código de la interfaz de usuario, lo cual puede ser más fácil de depurar. (Trowbridge, 2003).

- *Acoplamiento de la Vista y el Controlador con el Modelo.* Cambios en la interfaz del modelo requieren cambios paralelos en la vista y cambios adicionales en el controlador, algunos de estos cambios del código pueden llegar a ser bastante difíciles.

- *Interfaz.* El framework posee una interfaz al estilo Web donde en la parte derecha se muestra el menú con todas las opciones, si se desea cambiar esta apariencia puede conllevar un gran esfuerzo por el equipo de desarrollo, partiendo desde el punto que se cuente con el código del mismo, de lo contrario es imposible.

"La capacidad de reutilización del código y del diseño de frameworks orientados al objeto permite una productividad mayor y un tiempo de mercado breve en el desarrollo de aplicaciones, en comparación con el desarrollo tradicional de los sistemas de software. La configuración flexible de frameworks, permite la reutilización del núcleo. El desarrollo del framework ha sido exitoso en muchos dominios." (Lucena, y otros).

2.2 Patrones de Diseño de Software.

Los Patrones de Diseño nos hablan de cómo construir software, de cómo utilizar las clases y los objetos de forma conocida.

Los precedentes a los patrones de diseño vienen del campo de la Arquitectura, Christopher Alexander a finales de los 70 escribe varios libros acerca de urbanismo y construcción de edificios, y se plantea reutilizar diseños ya aplicados en otras construcciones que cataloga como modelos a seguir. (Pestano Pino, y otros, 2007).

En 1987 Ward Cunningham y Kent Beck utilizan las ideas de Alexander para desarrollar un lenguaje de patrones como guía para los programadores de Smaltalk, dando lugar al libro "Using Pattern Languajes for Object-Oriented Programs". (Pestano Pino, y otros, 2007).

Entre 1990 y 1994, Erich Gamma, Richard Helm, Ralph Johnson y Hohn Vlissides (conocidos como el grupo de los cuatro) realizan el primer catálogo de patrones de diseño, que publican en el libro "Design Patterns: Elements of Reusable Object-Oriented Software" (Gang of Four).

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. (Alexander, 1997).

Un patrón de diseño, obviamente, debe encajar por un lado con otros tipos de patrones imaginables, y por el otro con la teoría, la práctica y los marcos que en general rigen el diseño. Los patrones de diseño de software buscan codificar y hacer reutilizables un conjunto de principios a fin de diseñar aplicaciones de alta calidad. Se aplican en principio sólo en la fase de diseño, aunque en los últimos tiempos se ha comenzado a definir y aplicar patrones en las otras etapas del proceso de desarrollo, desde la concepción arquitectónica inicial hasta la implementación del código. (Pérez Acosta, y otros, 2008).

Según (Pestano Pino, y otros, 2007), los patrones se clasifican según su propósito en:

1. **Patrones de Creación:** Tratan la creación de instancias.
2. **Patrones Estructurales:** Tratan la relación entre clases, la combinación de clases y la formación de estructuras de mayor complejidad.
3. **Patrones de Comportamiento:** Tratan la interacción y cooperación entre clases.

2.2.1 Fábrica Abstracta o Abstract Factory (Af).

El propósito de este patrón es crear una interfaz para la creación de familias de objetos u objetos independientes, sin tener la necesidad de especificar sus clases correspondientes.

Problema:

El problema que trata de resolver es proporcionar una interfaz para la creación de familias

de objetos interdependientes o interrelacionados, sin especificar sus clases concretas. (Unidad Docente de Ingeniería del Software, 2003).

Cuándo usarlo:

- Cuando el sistema debe ser independiente de como sus productos se crean, componen y representan.
- Cuando el sistema debe configurarse con una familia de productos de entre múltiples posibles.
- Cuando se quiere dar énfasis a la restricción de que una familia de productos ha sido diseñada para que actúen todos juntos.
- Cuando se quiere proporcionar una librería de clases de productos, de los que sólo se desea revelar su interfaz, no su implementación.

Ventajas:

- Se potencia el encapsulamiento, puesto que se aísla a los clientes de las implementaciones.
- Se incrementa la flexibilidad del diseño, resultando fácil cambiar de familia de productos (recordar que actúa toda junta).
- Se refuerza la consistencia, puesto que se restringe el uso a productos de una sola familia cada vez.

Inconvenientes:

Se dificulta la extensibilidad, puesto que no es fácil añadir nuevos tipos de productos.

2.2.2 Fachada o Facade.

Según (Ramirez, 2003), el patrón de diseño fachada sirve para proveer de una interfaz unificada sencilla que haga de intermediaria entre un cliente y una interfaz o grupo de interfaces más complejas.

Fachada puede:

- Hacer una biblioteca de software más fácil de usar y entender, ya que implementa métodos convenientes para tareas comunes.
- Hacer el código que usa la librería más legible, por la misma razón.
- Reducir la dependencia de código externo en los trabajos internos de una librería, pues la mayoría del código lo usa Fachada, permitiendo así más flexibilidad en el desarrollo de sistemas.
- Envolver una colección mal diseñada de API[22] con un solo API bien diseñado.

Problemas que soluciona: (Ramirez, 2003)**.**

Problema: Un cliente necesita acceder a parte de la funcionalidad de un sistema más complejo.

Solución: Definir una interfaz que permita acceder solamente a esa funcionalidad.

Problema: Existen grupos de tareas muy frecuentes para las que se puede crear código más sencillo y legible.

Solución: Definir una funcionalidad que agrupe estas tareas en funciones o métodos sencillos y claros.

Problema: Una biblioteca es difícilmente legible.

Solución: Crear un intermediario más legible.

Problema: Dependencia entre el código del cliente y la parte interna de una biblioteca.

Solución: Crear un intermediario y realizar llamadas a la biblioteca sólo o, sobre todo, a través de él.

Problema: Necesidad de acceder a un conjunto de APIs, que pueden, además, tener un diseño no muy bueno.

Solución: Crear una API intermedia, bien diseñada, que permita acceder a la funcionalidad de las demás.

[22] Interfaz de Programación de Aplicaciones.

2.2.3 Solitario o Singleton.

Según (Gamma, y otros, 1995), el objetivo de este patrón es garantizar que una clase sólo tenga una única instancia, proporcionando un punto de acceso global a la misma. Este patrón se utiliza cuando debe haber únicamente una instancia de una clase y debe ser claro su acceso para los clientes. Aunque también es utilizado en ocasiones en que la "Instancia Única" debe ser especializada mediante herencia y los clientes deben poder usar la instancia extendida sin modificar su código. Como es de esperar esto aporta muchas ventajas como son:

- El acceso a la "Instancia Única" está más controlado.
- Se reduce el espacio de nombres (frente al uso de variables globales).
- Permite refinamientos en las operaciones y en la representación, mediante la especialización por herencia de "Solitario".
- Es fácilmente modificable para permitir más de una instancia y, en general, para controlar el número de las mismas (incluso si es variable).

El método "Instancia" es el punto de acceso a la "Instancia Única" del "Solitario" para los clientes. Lo que ocurre en realidad con este método es que, si "Instancia Única" no ha sido aún creada, la crea (inicialización perezosa) llamando al constructor "Solitario", el cual solo puede ser accedido por el mismo Singleton debido a su nivel de visibilidad privado.

2.2.4 Proxy.

El patrón Proxy se utiliza como intermediario para acceder a un objeto, permitiendo controlar el acceso a él.

Este es un patrón estructural fundamental, muy general que ocurre frecuentemente en muchos otros patrones. El patrón obliga a que las llamadas a métodos de un objeto ocurran indirectamente a través de un objeto proxy, que actúa como sustituto del objeto original, delegando luego las llamadas a los métodos de los objetos respectivos. (Ramirez, 2003).

Un objeto proxy recibe llamados a métodos que pertenecen a otros. Los objetos clientes llaman a los métodos de los objetos proxy, este último no provee directamente el servicio

que el cliente espera, sino que invoca los métodos en el objeto específico que provee cada servicio.

Un objeto proxy provee alguna administración requerida para los servicios. Por lo general un objeto proxy comparte una interfaz o superclase común con el objeto que realmente provee el servicio. De esta manera, los objetos cliente no se dan cuenta que están invocando métodos en el proxy, y piensan que los están invocando directamente en los objetos respectivos. Esta transparencia en la administración de los servicios de otros objetos, es la razón fundamental para usar un proxy. (Ramirez, 2003).

2.2.5 State.

Según (Massot Puigserver, y otros, 2009), el patrón estado nos propicia cambiar el comportamiento dependiendo del estado, o sea, cuando queremos que un objeto cambie su comportamiento, según cambia su estado, se presenta el problema de la complejidad de código.

En código estructurado (Massot Puigserver, y otros, 2009):

- Variable para cada estado.
- Discriminación por switch.

Según (Massot Puigserver, y otros, 2009) el patrón state se utiliza:

- Cuando el objeto tiene diferentes estados y cambia su comportamiento para cada estado, como máquina de estados.
- Si el comportamiento de un objeto depende de un estado, y debe cambiar en tiempo de ejecución dependiendo del estado.
- Si las operaciones tienen largas sentencias con múltiples ramas que dependen del estado del objeto.

Consecuencias:

- Localiza el comportamiento dependiente del estado y divide dicho comportamiento en diferentes estados. Las transiciones entre estados no reside en sentencias if o switch monolíticas, sino que se reparte entre las subclases.

- Hace explícitas las transiciones entre estados.
- Los objetos Estado pueden compartirse.

2.3 Estándar de codificación.

Según (Pacheco Iglesias, y otros, 2008), un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas. Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento y mantenimiento del software.

En el caso de Registros y Notarías, el estándar de codificación fue definido por la dirección del proyecto en sus inicios, se define el formato para los siguientes puntos (ALBET, Ingeniería en Sistemas, Octubre 10, 2006):

- Organización de los ficheros.
- Indentación.
- Comentarios.
- Declaraciones.
- Espacios en blanco.
- Convenciones de declaración.
- Prácticas de programación.

Una vez visto y explicado las bases sobre las cuales se construyó el sistema se presentarán algunas de las tendencias y tecnologías más significativas.

2.4 Tendencias y Tecnologías.

Con el surgimiento de los sistemas de computo se hizo necesario comenzar a darle órdenes para que cumplieran cierta función, de esta forma comenzaron a surgir lenguajes de programaciones y técnicas de programación las cuales han estado evolucionando incluso en la actualidad. Las técnicas de programación son aplicables cuando se desea trasladar a la computadora el funcionamiento de un proceso, o la solución a un problema determinado, para ello se realiza una abstracción, o sea, un modelo simplificado de la realidad tomando los elementos más significativos y transformándolos en variables, de esta forma la computadora podrá entenderlo y se habrá obtenido el resultado que se estaba buscando. (Müller, 1997).

Según (Müller, 1997), lo que permite especificar, a modo de instrucciones, cuáles son los pasos que tendrá que seguir la computadora para resolver el problema se denomina lenguaje de programación, que no es otra cosa que una herramienta. La forma en que se especifique y se elabore la solución es a lo que se le llama técnica de programación.

2.4.1 Programación Estructurada.

Los programas computarizados pueden ser escritos con un alto grado de estructuración, lo cual les permite ser más comprensibles en actividades tales como pruebas, mantenimiento y modificación de los mismos. Mediante la Programación Estructurada todas las divisiones de control de un programa se encuentran estandarizadas, de forma tal que es posible leer la codificación del mismo desde su inicio hasta su terminación en forma continua, sin tener que saltar de un lugar a otro del programa siguiendo el rastro de la lógica establecida por el programador, como es la situación habitual con codificaciones desarrolladas bajo otras técnicas. (Müller, 1997).

Programación Estructurada: Según (Müller, 1997), es una técnica en la cual la estructura de un programa, la escritura de sus partes se realiza tan claramente cómo es posible mediante el uso de tres estructuras lógicas de control:

- **Secuencia:** Sucesión simple de dos o más operaciones.

- **Selección:** División condicional de una o más operaciones.
- **Interacción:** Repetición de una operación mientras se cumple una condición.

Estos tres tipos de estructuras lógicas de control pueden ser combinados para producir programas que manejen cualquier tarea de procesamiento de información.

2.4.2 Programación Procedimental.

La Programación Procedimental es un paradigma de programación basado en el concepto de "llamado de procedimientos", los procedimientos, también conocidos como rutinas, subrutinas, métodos o funciones, simplemente consisten en una series de pasos computacionales. La mayor parte de los lenguajes de alto nivel la soportan, permiten la creación de procedimientos, que son pequeños fragmentos de código que realizan una tarea determinada. (USR., 2004).

Un procedimiento podrá ser invocado muchas veces desde otras partes del programa con el fin de aislar la tarea en cuestión y, una vez que finaliza su ejecución, retorna al punto del programa desde donde se realizó la llamada. (Müller, 1997).

De esta forma, se puede dividir el problema en problemas más pequeños, y así, llevar la complejidad a un nivel manejable. Si un procedimiento ya es correcto, cada vez que es usado produce resultados correctos.

2.4.3 Programación Orientada a Objetos.

Según (Booch, 1998), el término de Programación Orientada a Objetos indica más una forma de diseño y una metodología de desarrollo de software que un lenguaje de programación, en realidad se puede aplicar el Diseño Orientado a Objetos (en inglés abreviado OOD[23]), a cualquier tipo de lenguaje de programación.

Básicamente la Programación Orientada a Objeto (OOP[24]) permite a los programadores escribir software, de forma que esté organizado en la misma manera que el problema que trata de modelar.

23 Object Oriented Design.

24 Object Oriented Programming.

Los lenguajes de programación convencionales son poco más que una lista de acciones a realizar sobre un conjunto de datos en una determinada secuencia. Si en algún punto del programa se modifica la estructura de los datos o la acción realizada sobre ellos, el programa cambia. (Booch, 1998).

La OOP aporta un enfoque nuevo, convirtiendo la estructura de datos en el centro sobre el que pivotan las operaciones. De esta forma, cualquier modificación de la estructura de datos tiene efecto inmediato sobre las acciones a realizar sobre ella, siendo esta una de las diferencias radicales respecto a la programación estructurada. (Müller, 1997).

Según (Booch, 1998), la OOP proporciona las siguientes ventajas sobre otros lenguajes de programación:

- **Uniformidad:** La representación de los objetos lleva implícita tanto el análisis como el diseño y la codificación de los mismos.
- **Comprensión:** Tanto los datos que componen los objetos, como los procedimientos que los manipulan, están agrupados en clases, que se corresponden con las estructuras de información que el programa trata.
- **Flexibilidad:** Al tener relacionados los procedimientos que manipulan los datos con los datos a tratar, cualquier cambio que se realice sobre ellos quedará reflejado automáticamente en cualquier lugar donde estos datos aparezcan.
- **Estabilidad:** Dado que permite un tratamiento diferenciado de aquellos objetos que permanecen constantes en el tiempo sobre aquellos que cambian con frecuencia permite aislar las partes del programa que permanecen inalterables en el tiempo.
- **Reusabilidad:** La noción de objeto permite que programas que traten las mismas estructuras de información reutilicen las definiciones de objetos empleadas en otros programas e incluso los procedimientos que los manipulan. De esta forma, el desarrollo de un programa puede llegar a ser una simple combinación de objetos ya definidos donde estos están relacionados de una manera particular.

2.4.4 Programación Orientada a Servicios.

De forma resumida se podría decir que un Servicio Web es un componente de software

que se comunica con otras aplicaciones codificando los mensajes en XML y enviando estos mensajes a través de protocolos estándares de Internet, intuitivamente es similar a un sitio web pero sin interfaz de usuario, brinda servicio a las aplicaciones en vez de a las personas. (González).

La Programación Orientada a Servicios constituye un complemento de la Programación Orientada a Objetos debido a que puede representarse como una capa adicional en el modelado de una solución, esta capa podría llamarse “Capa de Servicios” la cual se encargará de publicar aquellas funcionalidades que puedan resultar comunes para diferentes problemas, también ofrece facilidad para el desarrollo de aplicaciones distribuidas, que están dadas producto de la experiencia acumulada en la última década, sobre todo en las áreas de la computación distribuida, instalación de una solución e interoperabilidad entre sistemas heterogéneos. (Schwindt).

Una de las diferencias fundamentales entre esta técnica y la Programación Orientada a Objetos es la manera en la que ambas definen una “aplicación”. La Programación Orientada a Objetos determina que una aplicación está compuesta de clases interdependientes, mientras que la Programación Orientada a Servicios considera que una aplicación está compuesta por un conjunto de servicios autónomos.

Los sistemas orientados a servicios, en cambio, son diseñados con un bajo nivel de acoplamiento que facilita la implementación de cambios y estos servicios pueden ser desarrollados en cualquier lenguaje corriendo en diferentes plataformas. (Schwindt).

2.4.5 Programación Orientada a Aspectos.

La Programación Orientada a Aspectos (AOP), por las siglas de (Aspect-Oriented Programming) o AOSD, por (Aspect-Oriented Software Development) buscan resolver un problema identificado hace tiempo en el desarrollo de software. Es un paradigma de programación relativamente reciente cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos. Gracias a la AOP se pueden encapsular los diferentes conceptos que componen una aplicación en entidades bien definidas, eliminando las dependencias entre cada uno de los módulos. (Calle Congote, 2006).

De esta forma se consigue razonar mejor sobre los conceptos, se elimina la dispersión del

código y las implementaciones resultan más comprensibles, adaptables y reusables. Varias tecnologías con nombres diferentes se encaminan a la consecución de los mismos objetivos y así, el término AOP el cual es usado para referirse a varias tecnologías relacionadas como los métodos adaptivos, los filtros de composición, la programación orientada a sujetos o la separación multidimensional de competencias. (Calle Congote, 2006).

2.5 Conclusiones del Capítulo.

En este capítulo se hizo un análisis de varios de los estilos arquitectónicos más conocidos para la realización de sistemas similares. El resultado de este análisis comprobó que la Arquitectura en Capas era la más factible para la realización del sistema, por todas las cualidades que tenía el proyecto; como son las peticiones de cambios que vienen por parte de los clientes, por lo que el sistema debe tener mucha flexibilidad, para que estos cambios constantes no afecten el tiempo de entrega del producto, además de todos los argumentos presentados en la explicación en el epígrafe correspondiente.

También se realizó el análisis de una serie de patrones para que se pudiera definir un diseño apropiado para el sistema, donde se explicó cuáles eran los candidatos más favorables para el desarrollo constante de la aplicación.

Por último se dio un breve recorrido por los estándares de codificación y tendencias y tecnologías.

Una vez vistos estos aspectos se define en el siguiente capítulo el diseño e implementación de nuestra aplicación.

Capítulo 3: Diseño e Implementación.

El diseño de sistemas se ocupa de desarrollar las directrices propuestas durante el análisis con la intención de satisfacer los requisitos planteados tanto funcionales como no funcionales. El objetivo del presente capítulo es el diseño y la implementación del sistema para los Fondos de Caja Chica de la República de Venezuela, partiendo de las herramientas case, plataformas y frameworks, se facilita la obtención de los artefactos generados por este flujo de trabajo. Se utiliza la metodología *Rational Unified Process (RUP)* y Enterprise Arquitect como herramienta de modelado, el cual se basa en el uso del UML 2.1 como lenguaje de modelado. El proceso de diseño de un sistema complejo se suele realizar de forma descendente:

- Diseño de alto nivel (o descomposición del sistema a diseñar en subsistemas menos complejos).
- Diseño e implementación de cada uno de los subsistemas.
- Especificación consistente y completa del subsistema de acuerdo con los objetivos establecidos en el análisis.
- Desarrollo según la especificación.
- Prueba.
- Integración de todos los subsistemas.
- Validación del diseño.

En el desarrollo de este capítulo se exponen una serie de diagramas correspondientes al diseño del sistema para los Fondos de Caja Chica.

3.1 Modelo de Diseño.

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. (García Batista, y otros, 2008).

Como parte del modelo de diseño del los procesos en los Fondos de Caja Chica, se realizaron los casos de usos esenciales según su alto valor para la arquitectura, representados a través de los diagramas de interacción y diagramas de clases, estos diagramas fueron agrupados en pequeños grupos equivalentes a los casos de usos significativos arquitectónicamente identificados, pues constituyen las funciones fundamentales del sistema:

- Gestionar Responsables del Fondo de Caja Chica.
- Gestionar Fondos de Caja Chica.
- Gestionar Vales de Caja Chica.
- Gestionar Facturas o Recibos.
- Gestionar Reposiciones del Fondo de Caja Chica.
- Gestionar Modificaciones del Fondo de Caja Chica.

Para dar soporte al diseño de este sistema, existe una arquitectura definida, de la cual se expondrán algunos elementos importantes a tener en cuenta para la realización del diseño.

El módulo Fondos de Caja Chica está estructurado siguiendo el patrón arquitectónico en Capas ya que este facilita tanto la comprensión como la construcción del subsistema, reduce las dependencias entre las capas evitando así tener que realizar grandes cambios como consecuencia de pequeñas modificaciones y propicia excelentes condiciones para la reutilización al poder utilizar en las capas superiores las funcionalidades implementadas en capas inferiores. El módulo tiene como base un framework común que regula entre otras cosas la gestión de la presentación adoptando para esto una variante del Modelo Vista Controlador (MVC). Además maneja los temas relacionados con la seguridad, ya sea control de acceso, manejo de excepciones y navegabilidad dentro del módulo.

A continuación se hace referencia a cada una de las capas existentes en el módulo definiendo los tipos de clases que podemos encontrar y su relación con los recursos brindados por el framework común antes mencionado.

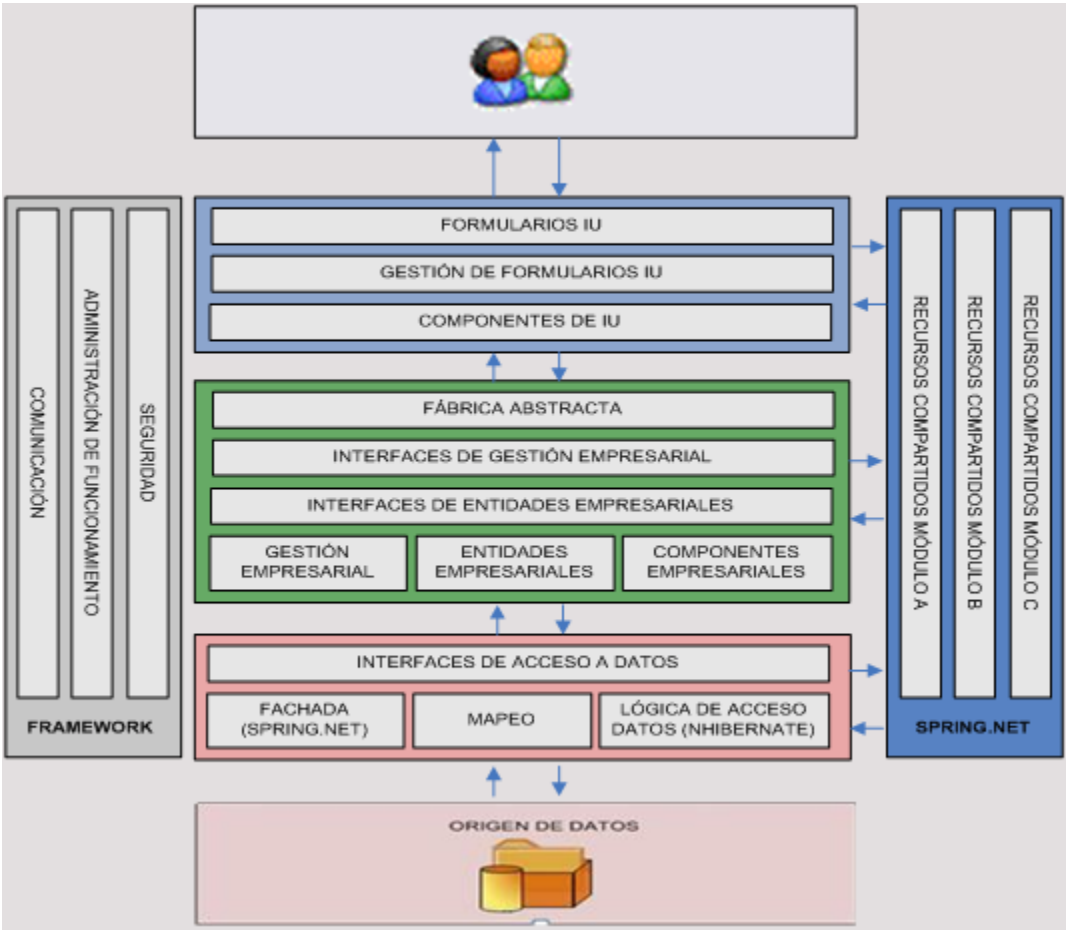

Figura 9 Vista de la arquitectura del módulo Fondos de Caja Chica.

3.1.1 Capa de Presentación.

En esta capa se encuentran todas las clases encargadas de una forma u otra de facilitar la interacción del usuario con el sistema. En ella se encuentran los formularios de interfaz de usuario los cuales responden a un diseño y comportamiento estándar en todos los módulos con vista a facilitar el trabajo con ellos y lograr la estandarización de la aplicación. Están basados fundamentalmente en dos tipos de formularios proveídos por el framework (Framework Común) dependiendo de sus características.

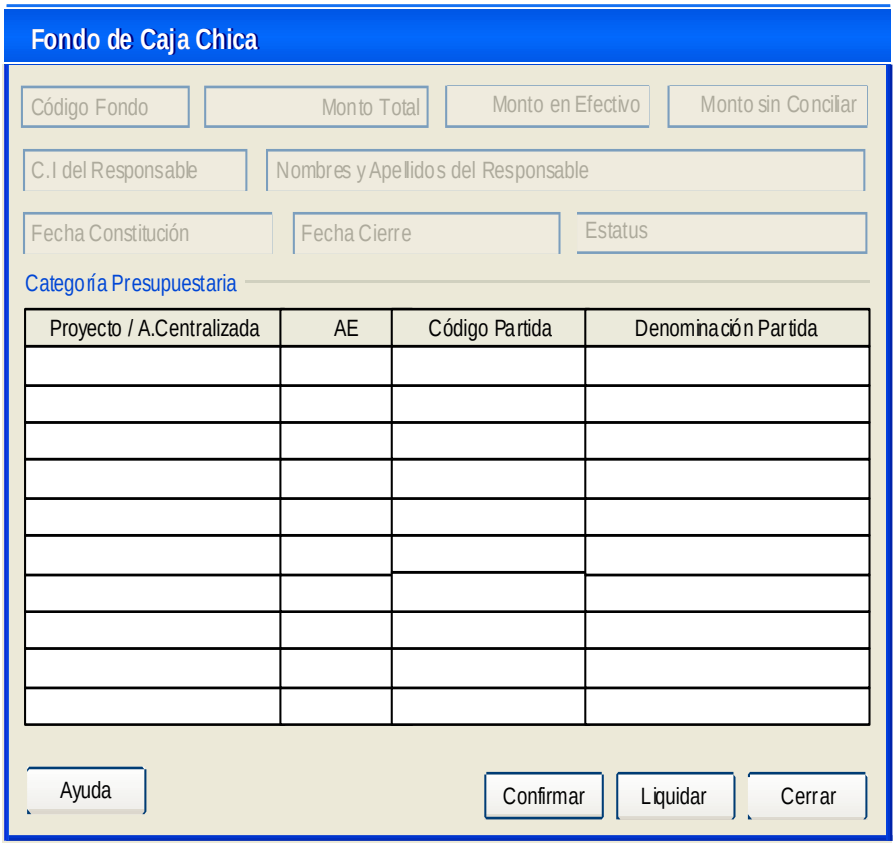

Figura 10 Formulario Estándar.

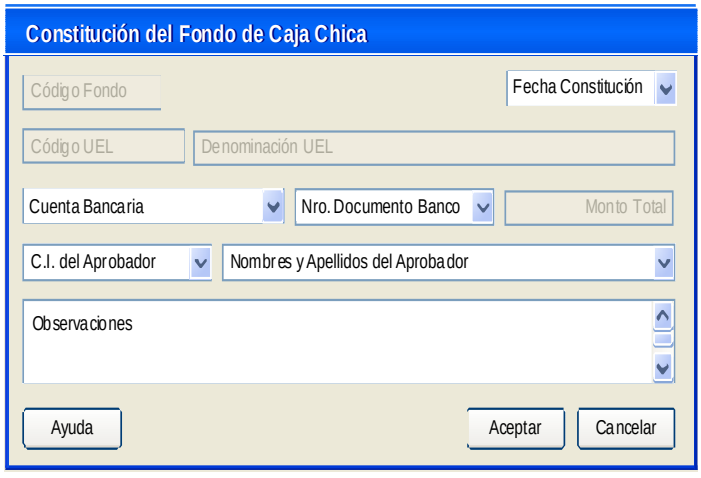

Figura 11 Formulario Flotante.

Frmmensaje (Figura 11 Formulario Flotante.) para las interfaces flotantes o FrmAreaDeTrabajo (Figura 10 Formulario Estándar.) para aquellas interfaces de tamaño grande que no son flotantes y se encuentran dentro del área de trabajo de la aplicación. El control de estos formularios esta dado por la utilización de gestores de interfaz de usuario, estos componentes son brindados por el framework, el cual facilita el desarrollo del sistema basándose en acciones.

¿Qué es una acción?

Cada acción debe tener sentido semántico propio y completo. Deben ser atómicas. Las acciones básicamente definen un bloque de código reusable por varias operaciones sobre el sistema. Las acciones pueden estar asociadas a vistas del sistema. Cada acción puede representar un estado del sistema que puede o no persistir. Una acción es una clase que representa precisamente la ejecución de alguna tarea concreta. Estas tareas no deben ser excesivamente complejas y la clase debe: Heredar de la clase “Acción” o “Acción Segura” (Framework Común).

En esta misma capa se pueden encontrar también los componentes de interfaz de usuario. Estos componentes tienen como objetivo fundamental encapsular una función determinada que será utilizada en más de una ocasión dentro del módulo. El hecho que se encuentren en la capa de presentación se debe precisamente a que trabajan directamente con los formularios, ya sea de forma visual o con los datos que se encuentran relacionados en el mismo a través de otros componentes.

3.1.2 Capa Lógica del Negocio.

Esta capa está estrechamente relacionada con las características del negocio en cuestión, en ella se modela, gestiona y procesa la información obtenida de la presentación y en caso de necesitarse se interactúa con la Capa de Acceso a Datos, la cual se explicará más adelante en este mismo epígrafe.

Entre las clases que se encuentran en la capa lógica del negocio se pueden ver las Entidades del Negocio (Figura 12), estas son clases objeto-valor que representan los datos con los que se va a trabajar en cada uno de los procesos que se están

automatizando. Cada entidad es un elemento auto sustentado, es decir, se encarga de procesar sus propios datos o valores sin interactuar con los demás elementos del negocio, con esto se garantiza la independencia y el encapsulamiento de la información.

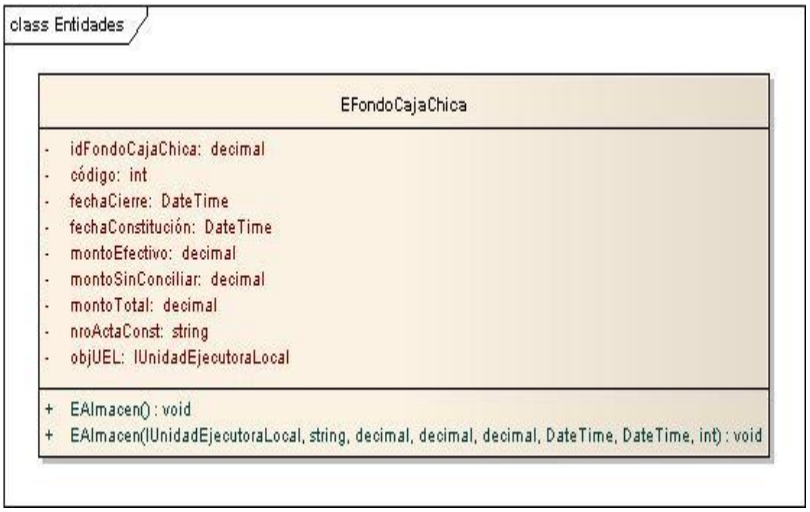

Figura 12 Clase entidad EFondoCajaChica.

También se encuentran las clases de Gestión del Negocio (Figura 13) estas son las encargadas de proporcionar las funcionalidades que van a interactuar sobre las entidades. En estas clases se propone la utilización del patrón Singleton, debido a que en ellas solo se encuentran funcionalidades y con una única instancia se posibilita su ejecución efectiva. Al igual que en la capa de presentación existen componentes que agrupan una serie de funcionalidades que serán utilizadas en distintas ocasiones y para contribuir en disímiles funcionalidades del módulo, estos son los Componentes del Negocio y se encuentran solo en este nivel a pesar de que puedan tener en ocasiones relación con el Acceso a datos.

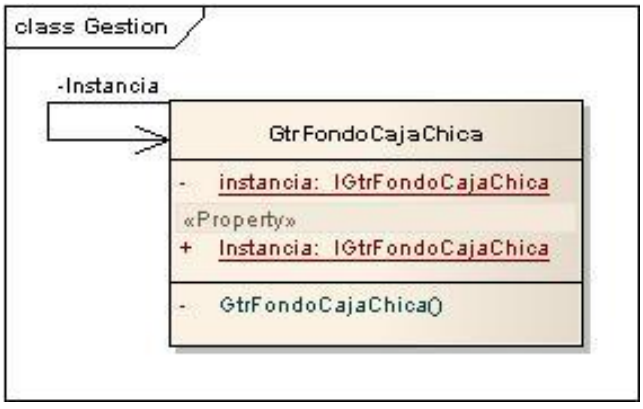

Figura 13 Clase Gestora GtrFondoCajaChica.

Una premisa de la estructura definida para el módulo, es la utilización de Interfaces (Figura 14). Estas son de gran importancia ya que se utilizan para brindar funcionalidades que más tarde serán implementadas. La mayoría de las interfaces de la aplicación van a estar precisamente en el Negocio, puesto que aquí se encuentran en su mayoría la implementación de estas funcionalidades, ya sea en las entidades, los gestores o los componentes de gestión. En esta capa se aplica el patrón de diseño Fábrica Abstracta definiéndose para eso una clase con la responsabilidad de instanciación ya sea de las entidades o los gestores de negocio. Otro de los patrones de diseño aplicados en esta capa es el patrón Estado, el cual se utiliza para permitir que el objeto cambie su comportamiento a medida que se modifica su estado.

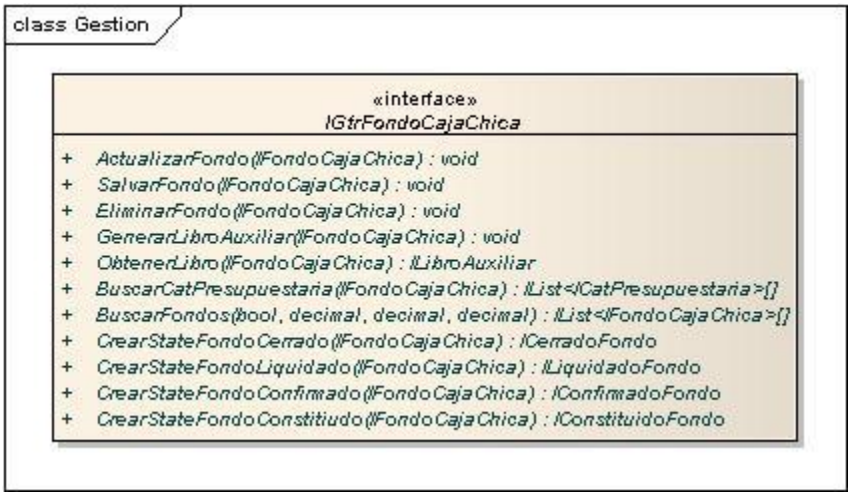

Figura 14 Interfaz IGtrFondoCajaChica.

3.1.3 Capa de Acceso a Datos.

Definir la estrategia de persistencia de una aplicación es una de las decisiones de arquitectura más importantes. En una aplicación estándar más del 50% del código generado está relacionado con lógica de persistencia. Partiendo de esto, el Acceso a Datos se considera la capa más crítica y sensible de la arquitectura pues controla todo lo concerniente a la información que se encuentra en la fuente de almacenamiento (Capa de Datos).

Al ser la capa inferior no conoce los niveles superiores, únicamente se limita al manejo de la información, ya sea para persistirla o proporcionarla para su procesamiento y propagación por la aplicación. Todo este manejo es responsabilidad de los Objetos de Acceso a Datos (DAOs por sus siglas en ingles). Aquí se propone el uso de Nhibernate versión 1.1, framework orientado a objetos para la persistencia de datos.

Lógica de Acceso a Datos.

Aquí recaen todas las funcionalidades del Acceso a Datos en los DAOs, éste ensamblado presenta la totalidad de las operaciones de persistencia y obtención de datos explotando los recursos que brinda NHibernate framework que cumple perfectamente con el objetivo de este nivel, dígase trabajo con procedimientos almacenados y métodos de persistencia o consultas.

Fachada.

La fachada brinda una interfaz de alto nivel con la cual se va a interactuar cada vez que se necesite comunicarse, en este caso con el Acceso a Datos logrando una completa enajenación ante cualquier modificación que pueda ocurrir en esta.

El módulo Fondos de Caja Chica forma parte de un gran subsistema, motivo por el cual debe interactuar de forma estable con el resto de los módulos definidos. La integración de estos módulos tiene como característica fundamental la implementación de una fachada en la cual se encontrarán todas las funcionalidades que se desean compartir con el resto del sistema. Con la adopción de esta modalidad de comunicación entre los módulos se gana en abstracción y se reduce la dependencia entre los mismos; con la utilización del framework Sprint.Net para su implementación se mantiene la uniformidad y estabilidad del módulo.

Mapeo.

Este elemento es de uso exclusivo de NHibernate, se decidió aislarlo en un ensamblado ya que básicamente son ficheros XML de configuración que son independientes de cualquier implementación, por tanto resulta muy útil tenerlos separados de todo código y así se evita recompilar toda una capa ante cualquier modificación en una de estas configuraciones.

Interfaces de Acceso a Datos.

Representan las funcionalidades que brinda este nivel, es decir las referidas a los DAOs. Su uso e importancia es la misma que las de la capa Lógica de Negocio.

3.1.4 Capa de Datos.

Esta capa corresponde a los almacenes de datos. A ella pertenecen las bases de datos disponibles en los servidores de bases de datos.

3.1.5 Capa Fachada.

La Capa de Fachada es una representación del patrón Proxy y Fachada a gran escala, este nivel es un recurso utilizado para mantener una representación de los demás módulos en el que se está implementando, siempre y cuando lo necesite. Es decir, cada módulo que se vaya a comunicar con el otro debe dar las funcionalidades que necesita, las cuales se van a agrupar aquí. Esta distribución tiene su utilidad en el enfoque horizontal de la Arquitectura. Al emplear esta forma de comunicación entre los módulos se garantiza la abstracción y enajenación gracias a Spring.Net, el cual ya se ha mencionado con anterioridad.

3.2 Artefactos involucrados en el Diseño.

A continuación se detallan algunos de los artefactos fundamentales generados en el diseño. Es válido resaltar que estos son los artefactos seleccionados por la dirección del proyecto como los fundamentales por lo cual son los que se generan durante el desarrollo del presente trabajo de diploma.

3.2.1 Modelo de diseño.

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de usos, centrándose en el impacto que puede tener los requisitos funcionales/no funcionales y otras restricciones relacionadas con el entorno de implementación. Las abstracciones del modelo de diseño tienen una correspondencia directa con los elementos físicos del ambiente de implementación.

3.2.1.1 Clase del diseño.

Según (Pérez Acosta, y otros, 2008), una clase de diseño es una abstracción de una clase de implementación, donde las operaciones, atributos, tipos,

visibilidad, se pueden especificar con la sintaxis del lenguaje elegido.

3.2.1.2 Diagrama de Transición de Estados.

Algunos objetos del diseño poseen estados controlados que determinan su comportamiento cuando reciben un mensaje. En casos como este es viable la creación de un diagrama de transición de estados para describir las distintas transiciones a las que puede verse sujeto dicho objeto del diseño.

3.2.1.3 Realización de caso de uso-diseño.

Un Caso de Uso (CU) es una secuencia de transacciones que son desarrolladas por un sistema en respuesta a un evento que inicia un actor sobre el propio sistema. El objetivo de la Realización de los Casos de Uso es conseguir separar la especificación del sistema (Modelo de Casos de Uso) del diseño del sistema, distribuyendo el comportamiento definido en los casos de uso entre las clases del modelo de diseño. La realización de un caso de uso ofrece una visión más cercana al diseño de la solución planteada para el escenario descrito por el caso de uso. Describe la manera en cada caso de uso es realizado y ejecutado en términos de clases y sus objetos. (Pérez Acosta, y otros, 2008).

El documento generado en esta fase puede contener varios elementos para representar la realización de un caso de uso como son los diagramas de clases del diseño, los diagramas de interacción, haciendo uso de los diagramas de secuencia por la necesidad de detallar las secuencias de interacciones y ordenarlas en tiempo, y también se obtiene un flujo de sucesos-diseño que básicamente consiste en una descripción textual del flujo de eventos, descripción que explica y complementa los diagramas.

- **Diagramas de Clases.**

Este diagrama muestra la especificación detallada de cada una de las clases modeladas, es decir sus atributos, operaciones que cubrirán las responsabilidades identificadas en el análisis, métodos que implementan dichas operaciones y el diseño preciso de las relaciones que se establecen entre las

clases, bien sea de agregación, asociación o generalización. Las clases de diseño a menudo participan en varias realizaciones de casos de uso.

- **Diagramas de Interacción.**

Los diagramas de interacción ilustran el flujo de interacciones entre las clases y subsistemas participantes. Estos diagramas de interacción son normalmente diagramas de secuencia y diagramas de colaboración, que permiten expresar fácilmente el comportamiento de los casos de uso en función de objetos que colaboran entre sí para realizar una operación. Estos diagramas contienen objetos, enlaces, mensajes y también pueden contener notas y restricciones.

  - **Diagramas de Secuencia.**

    Los diagramas de secuencias muestran las interacciones entre los objetos o subsistemas mediante transferencia de mensajes. Suele ser útil crear un diagrama por cada escenario aportando claridad en la realización del caso de uso, en ellos se destaca la ordenación temporal de los mensajes. Para su creación se comienza por el principio del flujo del caso de uso, y después seguir ese flujo detalladamente, decidiendo qué objetos del diseño y qué interacciones son necesarias para realizar cada paso.

3.2.1.4 Subsistema y Paquetes de diseño.

Los subsistemas de diseño son utilizados para organizar los artefactos del modelo de diseño en piezas más manejables, puede contener clases de diseño, realizaciones de casos de uso, interfaces y otros subsistemas. Los paquetes de diseño se utilizan para la recopilación de clases, relaciones, diagramas y otros paquetes. Paquetes que estructuran el modelo de diseño dividiéndolo en componentes más pequeños y organizados.

3.3 Realización de los Casos de Uso del módulo Fondos de Caja Chica.

Ya conocidos los artefactos involucrados en el diseño a continuación se presenta la realización de algunos de los casos de uso más significativos para el módulo Fondos de Caja Chica.

3.3.1 Realización del CU: Gestionar Responsables del Fondo de Caja Chica.

Para la realización del caso de uso Gestionar Responsables del Fondo de Caja Chica se hace una representación estática y dinámica de las clases, interfaces y objetos a través de los diagramas de clases de diseño y diagramas de secuencia. El CU a pesar de no ser muy complejo en su desarrollo es considerado de suma importancia pues de su realización dependerá el caso de uso posterior.

3.3.1.1 Diagrama de Clases CU: Gestionar Responsables del Fondo de Caja Chica.

En este diagrama se muestran las relaciones entre las clases de diseño identificadas para el CU: Gestionar Responsables del Fondo de Caja Chica. Entre las clases más relevantes se encuentra el DaoComun y DaoResponsableCajaChica, ambas con la funcionalidad de realizar el acceso a datos desde el negocio, también se encuentra la FabricaResponsableCajaChica encargándose de instanciar las entidades y clases gestoras. La clase GtrResponsableCajaChica constituye un puente para el flujo de información entre las capas de Presentación y Datos. Otra de las clases de gran importancia es AccResponsableCajaChica su funcionalidad es controlar los eventos que puedan ser ejecutados desde el formulario. Es importante resaltar la utilización de la clase RecursosCompartidos perteneciente a la Fachada para obtener instancias de clases perteneciente a otros módulos. (Anexo1).

3.3.1.2 Diagrama de Secuencia.

Con el objetivo de facilitar la comprensión del diagrama y así lograr un mayor entendimiento de la secuencia de acciones que se suceden se realiza un diagrama por cada escenario posible dentro del caso de uso:

- **Escenario: Buscar Responsables del Fondo de Caja Chica.**

Este escenario representa una búsqueda de Responsables de Fondos de Caja Chica y tiene como patrones de búsqueda la UEL y su estado correspondiente. (Figura 22 Diagrama de Secuencia del Escenario Buscar Responsables.).

- **Escenario: Agregar Responsables del Fondo de Caja Chica.**

Escenario que posibilita agregar Responsables de Fondos de Caja Chica. (Figura 23 Diagrama de Secuencia del Escenario Agregar Responsables.).

- **Escenario: Modificar Responsables del Fondo de Caja Chica.**

Este escenario permite modificar un Responsable de Fondos de Caja Chica existente. (Figura 24 Diagrama de Secuencia del Escenario Modificar Responsables.).

- **Escenario: Eliminar Responsables del Fondo de Caja Chica.**

Este escenario permite eliminar un Responsable de Fondos de Caja Chica seleccionado, es importante resaltar que esta acción solo es válida en caso que se refiera a un Responsable en estado Activo. (Figura 25 Diagrama de Secuencia del Escenario Eliminar Responsables.).

3.3.2 Realización del CU: Gestionar Fondos de Caja Chica.

Para la realización del caso de uso Gestionar Fondos de Caja Chica se hace una representación estática y dinámica de las clases, interfaces y objetos a través de los diagramas de clases de diseño y diagramas de secuencia. También se representa el conjunto de estados por los cuales pasa un fondo durante su vida y los eventos que provocan el cambio de su estado mediante un diagrama de Transición de Estados. Este es considerado el CU más importante dentro del módulo Fondos de Caja Chica pues se encargará de darle tratamiento a la entidad Fondo de Caja Chica la cual es primordial en este trabajo. Para ver los diagramas de su realización ver (Anexo2)

- **Escenario: Buscar Fondos de Caja Chica.**
- **Escenario: Crear Fondos de Caja Chica.**
- **Escenario: Modificar Fondos de Caja Chica.**
- **Escenario: Eliminar Fondos de Caja Chica.**
- **Escenario: Constituir Fondos de Caja Chica.**
- **Escenario: Confirmar Fondos de Caja Chica.**
- **Escenario: Liquidar Fondos de Caja Chica.**

- **Escenario: Cerrar Fondos de Caja Chica.**

3.3.3 Realización del CU: Gestionar Vales de Caja Chica.

Para la realización del caso de uso Gestionar Vales de Caja Chica se hace una representación estática y dinámica de las clases, interfaces y objetos a través de los diagramas de clases de diseño y diagramas de secuencia. También se representa el conjunto de estados por los cuales pasa un vale de caja chica durante su vida y los eventos que provocan el cambio de su estado mediante un diagrama de Transición de Estados. El CU es considerado de suma importancia pues permitirá mantener un control de los vales de caja chica lo cual es de vital importancia para el caso de uso posterior. Para ver los diagramas de su realización ver (Anexo3)

- **Escenario: Buscar Vales de Caja Chica.**
- **Escenario: Crear Vales de Caja Chica.**
- **Escenario: Modificar Vales de Caja Chica.**
- **Escenario: Eliminar Vales de Caja Chica.**
- **Escenario: Aprobar Vales de Caja Chica.**
- **Escenario: Anular Vales de Caja Chica.**

3.3.4 Realización del CU: Gestionar Facturas o Recibos.

Para la realización del caso de uso Gestionar Facturas o Recibos se hace una representación estática y dinámica de las clases, interfaces y objetos a través de los diagramas de clases de diseño y diagramas de secuencia. También se representa el conjunto de estados por los cuales pasa un recibo durante su vida y los eventos que provocan el cambio de su estado mediante un diagrama de Transición de Estados. Este CU es de vital importancia pues tiene la responsabilidad de mantener un control detallado de las facturas. Para ver los diagramas de su realización ver (Anexo4).

- **Escenario: Buscar Facturas o Recibos.**
- **Escenario: Crear Facturas o Recibos.**
- **Escenario: Modificar Facturas o Recibos.**
- **Escenario: Eliminar Facturas o Recibos.**
- **Escenario: Confirmar Facturas o Recibos.**
- **Escenario: Anular Facturas o Recibos.**

3.3.5 Realización del CU: Gestionar Modificaciones del Fondo de Caja Chica.

Para la realización del caso de uso Gestionar Modificaciones del Fondo de Caja Chica se hace una representación estática y dinámica de las clases, interfaces y objetos a través de los diagramas de clases de diseño y diagramas de secuencia. También se representa el conjunto de estados por los cuales pasa una modificación durante su vida y los eventos que provocan el cambio de su estado mediante un diagrama de Transición de Estados. Este CU es de vital importancia pues tiene la responsabilidad de mantener un control detallado de las modificaciones a realizar en un Fondo de Caja Chica determinado, la cual podrá ser de aumento o disminución. Para ver los diagramas de su realización ver (Anexo5).

- **Escenario: Buscar Modificaciones.**
- **Escenario: Buscar Modificaciones UEL.**
- **Escenario: Crear Modificaciones.**
- **Escenario: Modificar Modificaciones.**
- **Escenario: Eliminar Modificaciones.**
- **Escenario: Aprobar Modificaciones.**
- **Escenario: Confirmar Modificaciones.**

3.4 Modelo de Implementación.

En este modelo se implementa el sistema diseñado en términos de componentes, ficheros de código fuente, ficheros de códigos binarios, ejecutables, entre otros.

Según (Pacheco Iglesias, y otros, 2008) la implementación sigue varios propósitos consigo como se muestra a continuación:

- Planificar las integraciones necesarias del sistema en cada iteración. Se sigue un enfoque incremental dando lugar a un sistema que se implementa en una sucesión de pasos pequeños y manejables.
- Distribuir el sistema asignando componentes ejecutables a nodos en el Diagrama de Despliegue.
- Implementar las clases y subsistemas encontrados durante el diseño. Las clases se implementan como componentes de ficheros que contienen código fuente.
- Probar los componentes individualmente y a continuación integrarlos y enlazándolos en uno o más ejecutables, antes de ser enviados para ser integrados, donde seguidamente se llevan a cabo las comprobaciones o prueba de unidad del sistema.

3.5 Diagrama de Componentes.

Los diagramas de componentes muestran las dependencias del compilador y del "runtime" entre los componentes del software; por ejemplo, los archivos del código fuente y las DLL. (Larman, 1999) A continuación se muestra el diagrama de componentes generado para el sistema desarrollado.

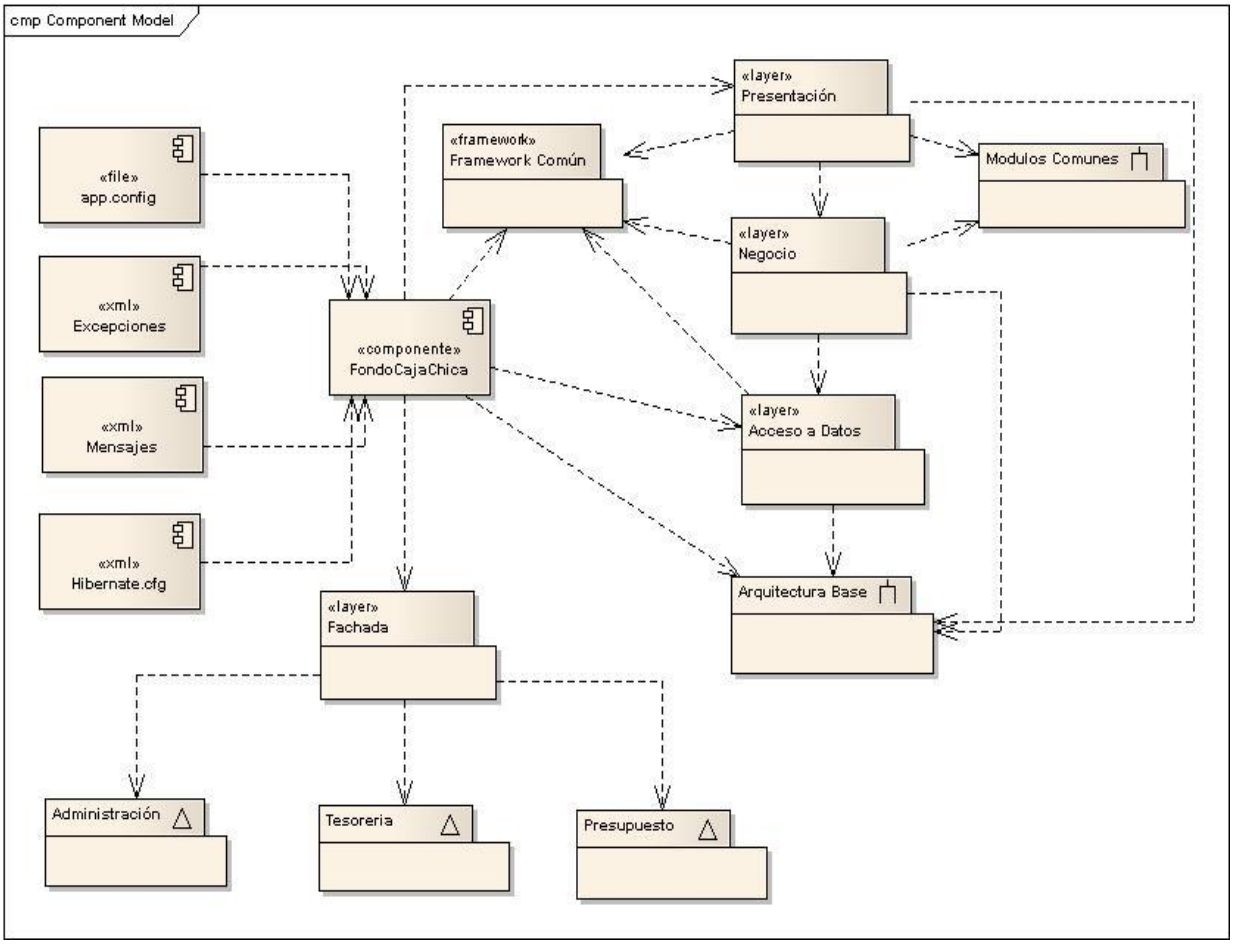

Figura 15 Diagrama de Componentes del subsistema Fondos de Caja Chica.

3.6 Diagrama de Despliegue.

Los diagramas de despliegue muestran a los nodos procesadores, la distribución de los procesos y de los componentes (Larman, 1999). A continuación se muestra el diagrama de despliegue generado para la implantación de este subsistema en los Registros y Notarías de la República Bolivariana de Venezuela.

En el siguiente diagrama las Unidades Ejecutoras Locales se conectan a una base de datos local que está en Oracle Standar Edition por el protocolo TCP/IP donde tienen una impresora compartida en la red la cual se conecta por el protocolo TCP/IP, las Unidades Administradoras Desconcentradas al igual que la UEL y la Unidad Administradora Central se conecta al Centro de Datos que está montada en Oracle Enterprise Edition 10 g y las

Bases de Datos se comunican mediante el protocolo TCP/IP replicando la información que se necesitan para las BD.

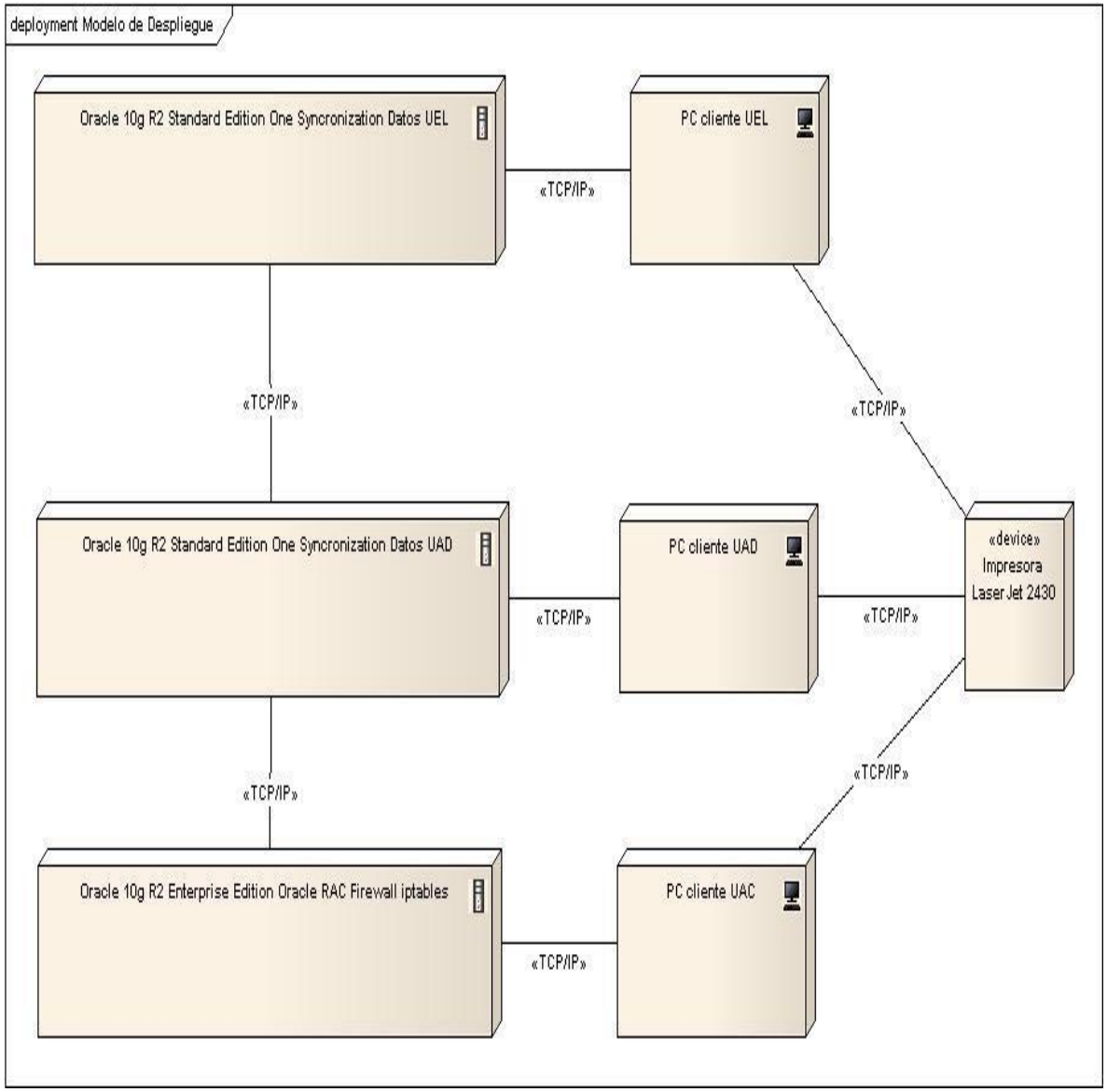

Figura 16 Diagrama de Despliegue subsistema Administración Financiera.

3.7 Conclusiones del Capítulo.

En este capítulo se definió una solución de diseño, que responde completamente a la arquitectura planteada en el capitulo anterior, la cual permite que el sistema sea auditable

y controlable para las situaciones que se plantean.

Se exponen los distintos tipos de clases que van a representar al diseño, se realizó el diseño de los diagramas de clases y diagramas de secuencia por cada uno de los procesos que competen al software que se presenta. Así como el Modelos de Componentes y Modelos de Despliegue fueron elaborados satisfactoriamente.

Capítulo 4: Análisis de los resultados.

En este capítulo se evalúan los resultados obtenidos con el desarrollo de este trabajo, teniendo por objeto la aplicación de algunas de las métricas empleadas internacionalmente para tal fin, específicamente las asociadas a las de la medición de la calidad del diseño de software. Es preciso señalar que las métricas de diseño para el software, como otras métricas del software, no son perfectas. En la actualidad continúa el debate relacionado de cómo deberían aplicarse y la eficacia de estas; muchos expertos plantean que se necesita más experimentación

4.1 Resultado de las métricas orientadas a clases.

A continuación se aplican un conjunto de métricas orientadas a clases con el objetivo de determinar el grado de calidad y fiabilidad del diseño propuesto en el capítulo anterior. Se seleccionaron las métricas a aplicar de dos grupos distintos para de esta manera aumentar el rigor de las pruebas realizadas. Estas métricas en la actualidad resultan ser unas de las más usadas.

4.1.1 Métricas propuestas por Lorenz y Kidd. Aplicación al modelo.

- **Tamaño de Clase (TC).**

Primeramente para conocer el tamaño de una clase es necesario conocer la cantidad de operaciones y números de atributos que poseen, para así poder determinar el valor del tamaño de dicha clase. También se puede determinar el TC, calculando el promedio de los atributos y operaciones encapsulados en una clase. En la Tabla 1 se muestran las medidas o umbrales para los parámetros de calidad que fueron aplicados al diseño obtenido, dichos umbrales constituyen una polémica en el diseño de sistemas a nivel mundial por la variedad de medidas que brindan diferentes especialistas.

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

Tabla 1 Umbrales para TC.

Los resultados obtenidos al aplicar la métrica TC al diseño propuesto fueron los siguientes:

Como se muestra en la Tabla 2 para un total de 71 clases existentes en el diseño se obtuvo un promedio de 9.9 operaciones y 3.8 atributos.

	Promedio de	
Total de Clases	**Operaciones**	**Atributos**
71	**9.9**	**3.8**

Tabla 2 Cantidad de clases de Diseño, operaciones y atributos promediados.

Según los umbrales propuestos se obtuvo que para un total de 71 clases presentes en el diseño, 66 son pequeñas, 3 de tamaño medio y solo 2 con un tamaño grande. Representándose en la Tabla 3

Cantidad de clases	Umbral	Tamaño
66	**<=20**	**Pequeño**
3	**>20 y <=30**	**Medio**
2	**>30**	**Grande**

Tabla 3 Cantidad de clases por tamaño.

Como se puede observar en la Figura 17 Representación de las clases según su TC. solo el 3 y el 4 por ciento constituyen clases grandes y medianas respectivamente, quedando la mayor cantidad de clases dentro del rango correspondiente a la clasificación de pequeñas, esto resulta ser positivo debido a que al obtener valores bajos para el TC aumenta la reutilización, se hace más fácil su implementación y la realización de pruebas en fases posteriores.

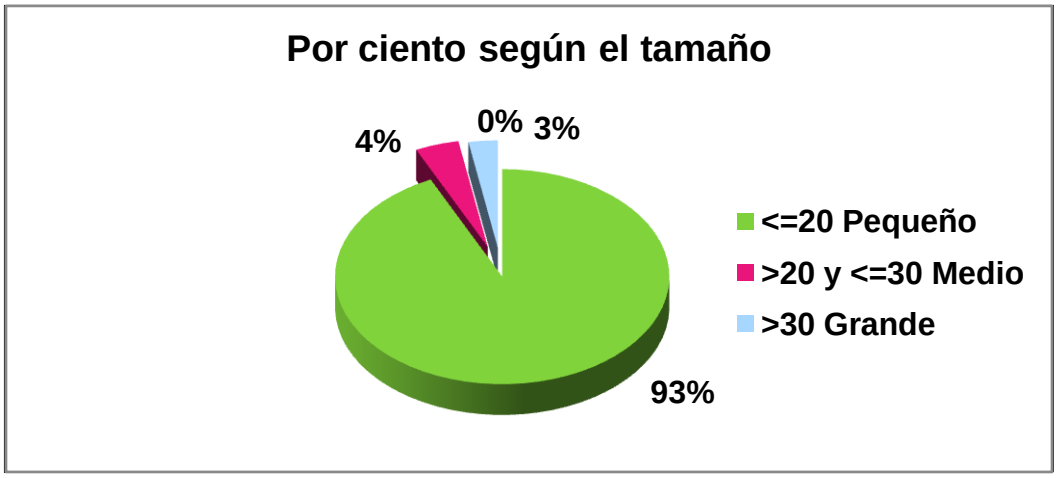

Figura 17 Representación de las clases según su TC.

- **Número de Operaciones redefinidas por una subclase (NOR).**

Para poder aplicar esta métrica es necesario conocer la cantidad de clases que redefinen métodos heredados de otras clases. La Tabla 4 Total de subclases que redefinen operaciones. muestra los valores obtenidos.

Cantidad de clases de diseño	Total de subclases que redefinen operaciones
71	**12**

Tabla 4 Total de subclases que redefinen operaciones.

Otra forma representativa es el por ciento de subclases que redefinen operaciones donde queda representado que el 17% de las clases existentes redefinen operaciones.

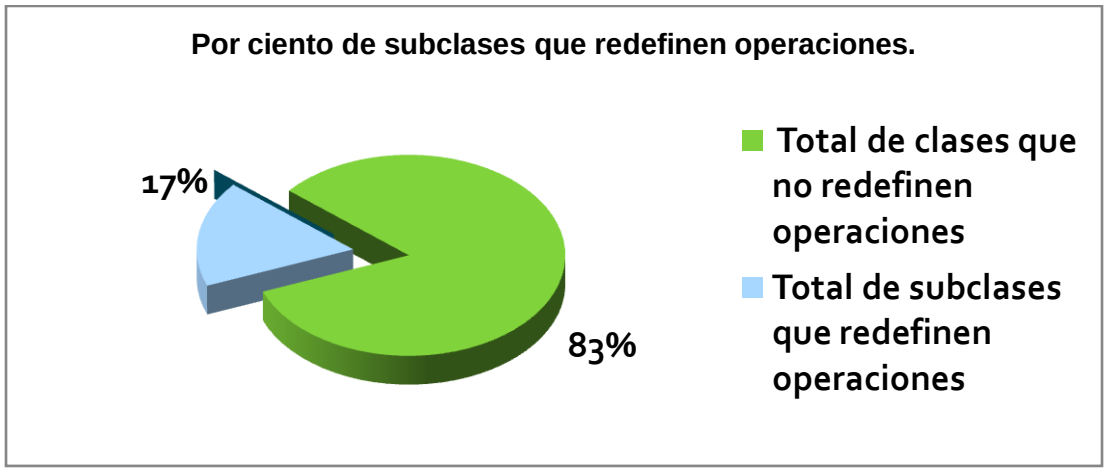

Figura 18 Por ciento de subclases que redefinen operaciones.

Como se puede observar el valor obtenido para la aplicación de la métrica NOR es relativamente bajo por lo que no se afecta la calidad del diseño propuesto, que puede ser modificado o llevado a pruebas sin dificultad.

4.1.2 Familia de métricas propuestas por Chidamber & Kemerer. Aplicación al modelo.

- **Árbol de profundidad de herencia (APH).**

Si se desea conocer el APH del diseño obtenido primeramente se debe tener conocimiento del nivel de jerarquía de clases que presenta dicho diseño.

A continuación en la Tabla 5 Umbral para la Métrica APH. se muestran los umbrales o medidas recomendadas por la documentación de Visual Studio .Net, estos umbrales brindan un conocimiento sobre la complejidad que puede alcanzar el diseño.

Nivel de jerarquía de clases	
APH	**Umbral**
Sencilla	**<= 5**
Compleja	**>5**

Tabla 5 Umbral para la Métrica APH.

Apoyándose en los umbrales definidos se obtiene el siguiente resultado:

Total de Clases	APH
71	**2**

Tabla 6 Resultados aplicando la métrica APH.

Como se puede observar el APH para la jerarquía de clases del diseño posee valor 2, al tener un indicador de nivel sencillo evita que exista problema si se desea predecir el comportamiento de una clase y también hace que el diseño no presente complejidad alguna.

- **Número de descendiente (NDD).**

Para obtener el NDD es necesario conocer el número de subclases inmediatamente subordinadas a una clase en la jerarquía existente del diseño propuesto. A continuación en la Tabla 7 Cantidad de clases por cantidad de descendientes. se muestra el número de clases presentes en el diseño que poseen descendientes.

Nro. de clases	Nro. de descendientes
7	**1**
2	**2**
1	**3**

Tabla 7 Cantidad de clases por cantidad de descendientes.

Como se puede observar para un total de 71 clases contenidas en el diseño propuesto, existen siete clases con NDD igual a 1, otras dos con valores iguales a 2 y solo una con 3 descendientes.

Los resultados alcanzados para el NDD del diseño propuesto son relativamente pequeños asegurándose con esto de que cada descendiente es realmente miembro de la clase predecesora y también al haber obtenido un valor pequeño del NDD se reduce la cantidad de pruebas necesarias a para ejercitar cada uno de los miembros de la jerarquía.

- **Acoplamiento entre clases objeto (ACO).**

Si se desea conocer el valor de ACO para el diseño propuesto primeramente se debe listar

para una clase el número de colaboraciones que posee. Para aplicar esta métrica solo se medirán los valores de las colaboraciones existentes en las entidades del negocio. A continuación en la Tabla 8 Colaboraciones por clases. se muestra la cantidad de clases que poseen dichas colaboraciones.

Nro. de clases	Nro. de Colaboraciones
9	**1**
11	**2**
5	**3**
4	**4**
1	**5**
2	**6**
1	**7**

Tabla 8 Colaboraciones por clases.

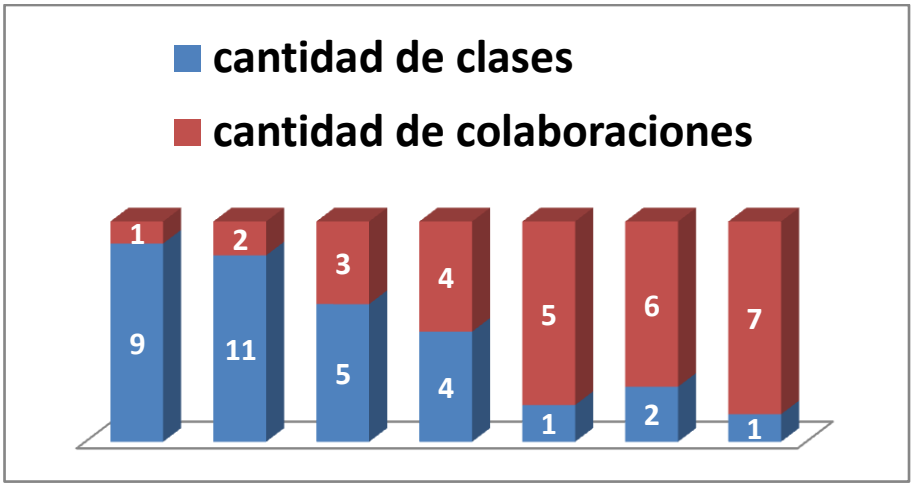

Figura 19 Colaboraciones por clases.

Conocido los valores para las colaboraciones existentes en las entidades del negocio se puede dar el siguiente resultado:

El diseño propuesto no presenta complejidad alguna, como se puede observar en la Figura

19 Colaboraciones por clases. el valor que indica el ACO según los datos que se muestran son relativamente bajos, es decir existe un bajo acoplamiento permitiendo que aumente el grado de reutilización de las clases existentes, también este resultado ayuda para que las pruebas o modificaciones necesarias a realizar sobre el diseño resulten fáciles de ejecutar.

4.2 Resultados obtenidos en pruebas de caja blanca.

La prueba de caja blanca, denominada a veces *prueba de caja de cristal* es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca, el ingeniero del software puede obtener casos de prueba que garanticen que se ejercite por lo menos una vez todos los caminos independientes de cada módulo; ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa; ejecuten todos los bucles en sus límites y con sus límites operacionales; y ejerciten las estructuras internas de datos para asegurar su validez. (Pressman, 1998).

Para la evaluación de la calidad en la implementación del subsistema de Fondos de Caja Chica se realizaron pruebas de caja blanca a algunos escenarios fundamentales de los Casos de Uso, así como a algunos de los componentes de software utilizados, para ello se tomó como herramienta de validación NUnit.

4.2.1 Resultados obtenidos de las pruebas aplicadas a los componentes.

Método que declara el punto de inicio de la prueba.

```
[SetUp]
public void Init()
{
    Assembly assembly = Assembly.GetExecutingAssembly();
    System.IO.Stream stream = assembly.GetManifestResourceStream(
        assembly.FullName.Substring(0, assembly.FullName.IndexOf(',')) + ".hibernate.cfg.xml");
    NHibernateUtil.Iniciar(stream);
    ArrayList lista = GtrTest.CrearClase();
    lvwTest.Objetos = lista;
}
```

Con este método se valida la veracidad de la inserción en el componente ListView. Después de la inserción en el método Init(), se verifica que la cantidad de elementos contenidos en el ListView no sea igual a 0, en caso de serlo se mostraría el mensaje “Valor no esperado”.

```
[Test]
public void AdicionarListView()
{
    Assert.AreNotEqual(0, lvwTest.Objetos.Count, "Valor no esperado");
}
```

Con este método se valida que el objeto previamente insertado en el componente mantiene su integridad. En este caso se verifica que el valor de la fecha no haya sido alterado.

```
[Test]
public void ValorEsperado()
{
    Assert.AreEqual(new DateTime(2009, 5, 5),
        (lvwTest.Objetos[0] as ETest).Fecha, "Valor no esperado");
}
```

4.2.2 Resultados obtenidos de las pruebas aplicadas a los gestores del negocio.

Con los parámetros especificados se esperan resultados, este método valida la existencia de los mismos.

```
[Test]
public IFondo ObtenerFondo()
{
    objDaoComun.IniciarTransaction();
    IFondoCajaChica obj = (IFondoCajaChica) objDaoFondo.FondoCajaChicaUEL();
    objDaoComun.ReafirmarTransaction();
    return obj;

}
```

Con los parámetros especificados se esperan resultados, este método valida la existencia de los mismos.

```
[Test]
public void SolicitarReposicion()
{
    objDaoComun.IniciarTransaction();
    ArrayList array = objDaoReposiciones.SolicitarReposiciones
        (idFondo, idUEL, desde, hasta, idEstado);
    objDaoComun.ReafirmarTransaction();
}
```

Este método verifica que las operaciones con el monto son realizadas satisfactoriamente.

```
[Test]
public void OperacionesSobreElMonto()
{
    objDaoComun.IniciarTransaction();
    objDaoLibroAuxiliar.OperacionesSobreMontoFondo(montoTotal,
        montoEfectivo);
    objDaoComun.ReafirmarTransaction();
}
```

Como resultado de las pruebas realizadas a los componentes y gestores especificados, a continuación se puede observar la interfaz de NUnit, donde muestra que las pruebas fueron realizadas satisfactoriamente.

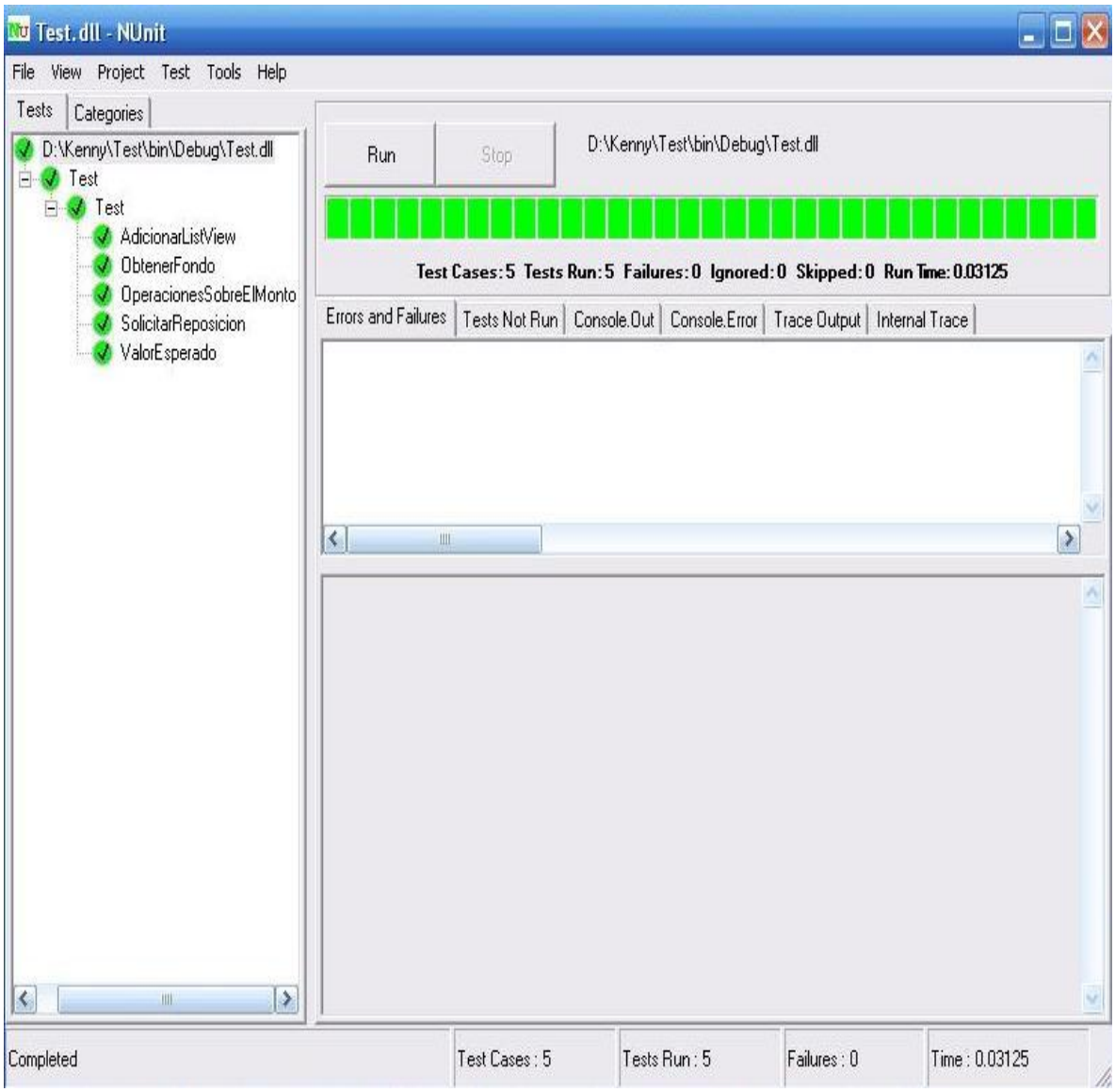

Figura 20 Interfaz NUnit.

Leyenda:

Satisfactorio
Fallido
Ignorado

4.3 Conclusiones del Capítulo.

En este capítulo se logró aplicar una serie de métricas para la validación del diseño

propuesto, en el cual se determinó que este no presenta una alta complejidad estructural, de datos, ni del sistema en general, permitiendo que las pruebas no sean complejas y afecten en mínimo de tiempo algún cambio.

La métrica de Tamaño de Clases evidencia que la mayoría de las clases son reutilizables, la implementación no es complicada y las pruebas no son complejas, al igual que la profundidad de los niveles de herencia están acorde con los umbrales definidos por autores consultados permitiendo este un bajo acoplamiento y que el sistema no sea complejo.

Por último se logró aplicar las pruebas de unidad con el NUnit Framework en las cuales se demostró haber implementado los requisitos primarios, donde los resultados fueron satisfactorios, llevando esto a que el producto responda a todas las necesidades y propósitos de los clientes.

Conclusiones.

Con la realización del presente trabajo se arriba a las siguientes conclusiones.

- Fueron analizados algunos sistemas financieros usados en el mundo y en Cuba que sirvieron de ejemplo en la confección de este trabajo.
- Se definió la estrategia de diseño basada en el uso de patrones de arquitectura y diseño abordados, en conjunto con el análisis de algunas tendencias de la programación, en este punto las herramientas CASE lograron un papel decisivo en la automatización de todos los procesos llevados a cabo para obtener las metas propuestas para el diseño del subsistema desarrollado.
- De acuerdo a las condiciones en que se desarrolló este subsistema, el cual sigue algunos requerimientos establecidos por los sistemas ya implementados en la solución SAREN, se seleccionó la plataforma de desarrollo, apoyándose además en un estudio breve de todas las plataformas existentes.
- Los resultados obtenidos a partir de los análisis del diseño y la implementación del sistema son positivos, tomando como argumento la aplicación de métricas y pruebas de caja blanca que permitieron evaluar el nivel de calidad que presenta el software.

Recomendaciones.

Los resultados obtenidos a partir de los análisis del diseño y la implementación del sistema son positivos, tomando como argumento la aplicación de métricas y pruebas de caja blanca que permitieron evaluar el nivel de calidad que presenta el software.

- Continuar en la investigación de sistemas con la misma finalidad que el presente, que aporten soluciones novedosas a la dada en este trabajo.
- Documentar el framework empleado para facilitar el uso de otros desarrolladores.
- Realizar pruebas de las antes mencionadas a todos los componentes y código en sentido general, ya que se garantiza un alto porciento de calidad del mismo.
- Integrar finalmente el módulo Fondos de Caja Chica a los restantes módulos del Sistema de Administración Financiera para corroborar su buen funcionamiento e integración total.

Bibliografía

ALBET, Ingeniería en Sistemas. Octubre 10, 2006. *Estándares de codificación.* Ciudad Habana, Cuba : s.n., Octubre 10, 2006.

Alexander, S. I. K. 1997. *A Pattern Language.* New York : s.n., 1997.

Artículo Nro 2, ASAMBLEA NACIONAL DE LA REPÚBLICA BOLIVARIANA DE VENEZUELA. 15 de Enero de 2007. *NUEVA LEY DE REGISTRO PÚBLICO Y DEL NOTARIADO DE VENEZUELA.* Caracas, Venezuela : s.n., 15 de Enero de 2007.

Beck, Kent. 2005. *Extreme Programming Explained: Embrace Change.* 2005.

Best, Software. 2003. Best Software Inc. *PeachTree.* [En línea] 2003. [Citado el: 21 de Febrero de 2009.] http://www.best.com.pa/productos/peachtree.php.

Booch, Grady. 1998. *OBJECT-ORIENTED ANALYSIS AND DESIGN.* Santa Clara, California : ADDISON-WESLEY, 1998. 1998. 0-8053-5340-2.

Cabrera González, M., y otros. 2004. *SISTEMA ECONÓMICO INTEGRADO.* 2004.

Calle Congote, John Edgar. 2006. Programación Orientada a Aspectos. *Slideshare.* [En línea] 2006. [Citado el: 28 de Marzo de 2009.] http://www.slideshare.net/jcongote/programacion-orientada-a-aspectos..

Cano Ossa, Mauricio. Agosto 12, 2006. *EL DIOS DE LOS MONOS. Una guía para el desarrollador de GUIs en windows con el mono.* Medellín : s.n., Agosto 12, 2006.

Carballeira, Dr. Félix García. 2000. Revista Digital Universitaria. [En línea] 01 de Octubre de 2000. http://www.revista.unam.mx/vol.1/num2/art4/..

Chica, Reglamento Interno de Caja. 2002. *CAPITULO IV: Administración del Fondo Fijo de Caja Chica.* Venezuela : s.n., 2002. Artículo 17.

—. 2002. *CAPITULO IV: Administración del Fondo Fijo de Caja Chica.* Venezuela : s.n., 2002. Artículo 18.

Chica., Reglamento Interno de Caja. 2002. *CAPITULO III: Asignación de Fondos.* Venezuela : s.n., 2002. Artículo 11.

Clements, Paul. 1996. *A Survey of Architecture Description Languages.* Alemania : s.n., 1996.

Dass, Karen. 2009. Model-View-Controller (MVC) Architecture. [En línea] 2009. http://www.indiawebdevelopers.com/technology/java/mvcarchitecture.asp…

Gamma, E. y otros. 1995. *Design Patterns.* s.l. : Addison-Wesley, 1995.

García Batista, Maikel Yonelis y Abiague Napoles, Alejandro. 2008. *Diseño e Implementación de la Solución Informática del proceso de Recaudación en la Administración Financiera de los Registros y Notarías de la República Bolivariana de Venezuela.* 2008.

González, Benjamín. Introducción al entorno de desarrollo de Microsoft .NET. *Desarrolloweb.* [En línea] [Citado el: 15 de Marzo de 2009.] http://www.desarrolloweb.com/articulos/1680.php..

Hedtke, John V. 2000. *Easy, P. M.* 2000.

Hernández, Jordi. 2004. *Mono: mucho más que una implementación libre de .NET.* 2004.

Jacobson, Ivar, Rumbaugh, James y Booch, Grady. 2000. *El proceso unificado de desarrollo de software.* 2000.

Larman, C. 1999. *UML y Patrones Introducción al Análisis y Diseño orientados a objetos.* 1999.

Lucena, M.E.M. y C.J., P.d. *El Desarrollo del Framework Orientado al Objeto.*

Massot Puigserver, Biel y Herraiz Aparicio, Edu. 2009. APOO 2007/2008. *APOO 2007/2008.* [En línea] 01 de Agosto de 2009. http://dmi.uib.es/~yuhua/APOO07-08/Presentation/Bridge-State.pdf.

Medina Pasaje, Julio Luis. Marzo 7, 2006. *Metodología y herramientas UML para el modelado y análisis de sistemas de tiempo real orientados a objetos.* Madrid, España : s.n., Marzo 7, 2006.

—. Marzo 7, 2006. *Metodología y herramientas UML para el modelado y análisis de sistemas de tiempo real orientados a objetos.* Madrid, España : s.n., Marzo 7, 2006.

Molpeceres Touris, Alberto y Pérez Mariñán, Martín. 2002. Arquitectura empresarial y software libre, J2EE. [En línea] 01 de Noviembre de 2002. [Citado el: 20 de Enero de 2009.] http://www.javahispano.org/articles.article.action?id=70..

Moya Arjona, Francisco. 2009. Monografías.com. [En línea] 2009. www.monografias.com/trabajos/sistfinanciero/sistfinanciero.shtml.

MSDN. 2008. .NET Framework Developer Center. [En línea] 2008. http://msdn.microsoft.com/netframework/..

Müller, Peter. 1997. Introducción a la Programación Orientada a Objetos Empleando C++. *Globewide Network Academy.* [En línea] 31 de Agosto de 1997. [Citado el: 11 de Marzo de 2009.] http://www.gnacademy.org/text/cc/Tutorial/Spanish/tutorial.html..

Pacheco Iglesias, Armando Esteban y Casanova Mutis, Alejandro. 2008. *Diseño e implementación de funcionalidades que se llevan a cabo en los registros mercantiles: solicitudes de expedientes, copias de documentos y sellado de libros.* Cuba : s.n., 2008.

Pérez Acosta, Johnny y Cardoso, Julio. 2008. *Diseño de la solución informática para el control de Bienes en los Registros Públicos y Notarías de la República Bolivariana de Venezuela.* Cuba : s.n., 2008.

Pestano Pino, Henrik y Montané Izaguirre, Juan Carlos. 2007. *Diseño del módulo para la Administración Contable y Financiera de los Registros y Notarías de la República Bolivariana de Venezuela.* Cuba : s.n., 2007.

Pressman, R. S. 1998. *Ingenieria de software. Un enfoque práctico.* 1998. Vol. Vol. 1..

Ramirez, A. O. 2003. *Patrones Estructurales.* 2003.

Rational, Software Corporation. 2002. *Rational Software Corporation.* 2002.

Reynoso, Carlos y Kiccillof, Nicolás. 2004. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. .* Universidad de Buenos Aires : s.n., 2004.

Saenz Fernández, Jose Antonino. 2005. *REGLAMENTO PARA LA ADMINISTRACION DEL FONDO DE CAJA CHICA PACIFICTEL S.A.* Venezuela : s.n., 2005.

Schwindt, Ariel. Construcción de Sistemas Multiplataforma basados en Servicios. [En línea] [Citado el: 11 de Marzo de 2009.] http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_3505.asp..

Seoane, Fabian. 2004. LUGCIX. [En línea] Diciembre de 2004. http://www.lugcix.org/tutoriales/mono/queesmono.php..

Shaw, Mary y Clements, Paul. Abril 1996. *A field guide to Boxology: Preliminary classification of architectural styles for software systems. Pennsylvania, Estados Unidos de América : Computer Science Department and Software Engineering Institute.* Carnegie Mellon University : s.n., Abril 1996.

Shaw, Mary y Garlan, David. 1996. *Software Architecture: Perspectives on an emerging discipline. Upper Saddle River : Prentice Hall.* 1996.

SlideShare. 2009. SlideShare. *SlideShare.* [En línea] 2009. [Citado el: 5 de Mayo de 2009.] http://www.slideshare.net/FARIDROJAS/compilador-presentation.

Smith, B. E. 1991. *Dac Easy Made Easy.* 1991.

SPARX, Systems. 2000. Modeling & Design Tools for your Enterprise. *Modeling & Design Tools for your Enterprise.* [En línea] 2000. http://www.sparxsystems.com.au/.

Trowbridge, David. 2003. *Enterprise Solution Patterns using Microsoft .NET. .* s.l. : Microsoft Corporation, 2003.

Unidad Docente de Ingeniería del Software, Facultad de informática. 2003. *Patrones del "Gang of Four".* Universidad Politécnica de Madrid. : s.n., 2003.

USR., CODE. 2004. *Programacion Orientada a Objetos.* s.l. : USR.CODE, 2004.

Glosario de Términos.

A.

Activos: Recursos económicos de propiedad de una empresa que se espera beneficie operaciones futuras.

Activos Fijos: Los activos fijos son aquellos que no varían durante el ciclo de explotación de la empresa.

B.

Bytecode: El bytecode es un código intermedio más abstracto que el código máquina.

C.

Categorías Presupuestarias: Término usado en el área del gobierno, ingresos tributarios y administración presupuestaria. Es el elemento presupuestario que se utiliza para identificar las plazas del personal de una unidad administrativa de acuerdo a la clave específica contemplada en el Catálogo de Empleos de la Federación, la cual se le asigna a cada uno de los individuos que laboran en la Administración Pública Central.

Compilador: Es un traductor que convierte un texto escrito en un lenguaje fuente de alto nivel en un programa objeto en código máquina.

D.

DLL: Dynamic Link Library (“Biblioteca de vínculos dinámicos" es un archivo que contiene funciones que se pueden llamar desde aplicaciones u otras DLL).

F.

Framework: Es una estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado. 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.

H.

Hardware: Corresponde a todas las partes físicas y tangibles de una computadora.

J.

Java: La plataforma Java es el nombre de un entorno o plataforma de computación, capaz de ejecutar aplicaciones desarrolladas usando el Lenguaje de programación Java u otros lenguajes que compilen a bytecode y un conjunto de herramientas de desarrollo.

L.

Libro Auxiliar: Término utilizado en el área de la contabilidad, auditoría y contabilidad financiera. Es un libro complementario a los principales libros de contabilidad. Su función es registrar todas las operaciones que le son propias y centralizarlas en el Libro Diario mediante un solo asiento contable.

Lenguaje Orientado a Objetos: Se le llama así a cualquier lenguaje de programación que implemente los conceptos definidos por la programación orientada a objetos.

Lenguaje de Programación: Un lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

M.

Máquina Virtual: Es un programa nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el Java bytecode), el cual es generado por el compilador del lenguaje Java.

P.

Plataforma: En informática, una plataforma es precisamente el principio, en el cual se constituye un hardware, sobre el cual un software puede ejecutarse/desarrollarse.

Programación Estructurada: La programación estructurada es una forma de escribir programas de ordenador de forma clara.

Programación Orientada a Objetos: es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora.

R.

Reutilización: Es la acción de volver a utilizar.

Runtime: Se denomina Tiempo de ejecución (Runtime en inglés) al intervalo de tiempo en el que un programa de computadora se ejecuta en un sistema.

S.

Sistema Operativo: Un sistema operativo es un software de sistema, es decir un conjunto de programas de computación destinados a realizar muchas tareas entre las que destaca la administración eficaz de sus recursos.

SmallTalk: Programa nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el Java bytecode), el cual es generado por el compilador del lenguaje Java.

U.

Unidades Administradoras (UA): Están constituidas por las organizaciones administrativas de los organismos, que tienen la responsabilidad de ejecutar los créditos presupuestarios manejados mediante órdenes de pago directo al proveedor o beneficiario, avances o adelantos de fondos a funcionarios.

Unidad Administradora Central (UAC): Es Dirección Administración u otra de igual competencia de cada organismo, manejará créditos centralizados y créditos propios mediante órdenes de pago directo, e igualmente girará órdenes de avances o adelanto de fondos a las Unidades Administradoras Desconcentradas y manejará el fondo de anticipo que se le asigne caja chica y fondos en avance.

Unidad Administradora Desconcentrada (UAD): Son los que con tal carácter determine la máxima autoridad del organismo ordenador. Estas Unidades deben tener una organización administrativa que les permita manejar fondos en anticipo por un monto igual o superior a 2 500 unidades tributarias y, podrán manejar además, fondos en avance y

cajas chicas, así como ordenar pagos directos contra el Tesoro Nacional, previa autorización de la máxima autoridad del organismo.

Unidad Ejecutora Local (UEL): Es la Unidad Operadora de menor nivel responsable de la ejecución física de una actividad de un proyecto.

UML: Lenguaje Unificado de Modelado, es un lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

Unidades Tributarias: Simplemente es una magnitud aritmética en la que se encuentran expresados ciertos montos establecidos en las leyes tributarias, constituyendo así una variable que permite la actualización constante de dichas leyes, con base en criterios objetivos y de simplificación en su aplicación.

Anexos.

Anexo 1. RCU Gestionar Responsables del Fondo de Caja Chica.

Diagrama de Clases del diseño y diagrama de Interacción para el CU Gestionar Responsables del Fondo de Caja Chica. Este CU posibilita buscar, incorporar, modificar y eliminar responsables.

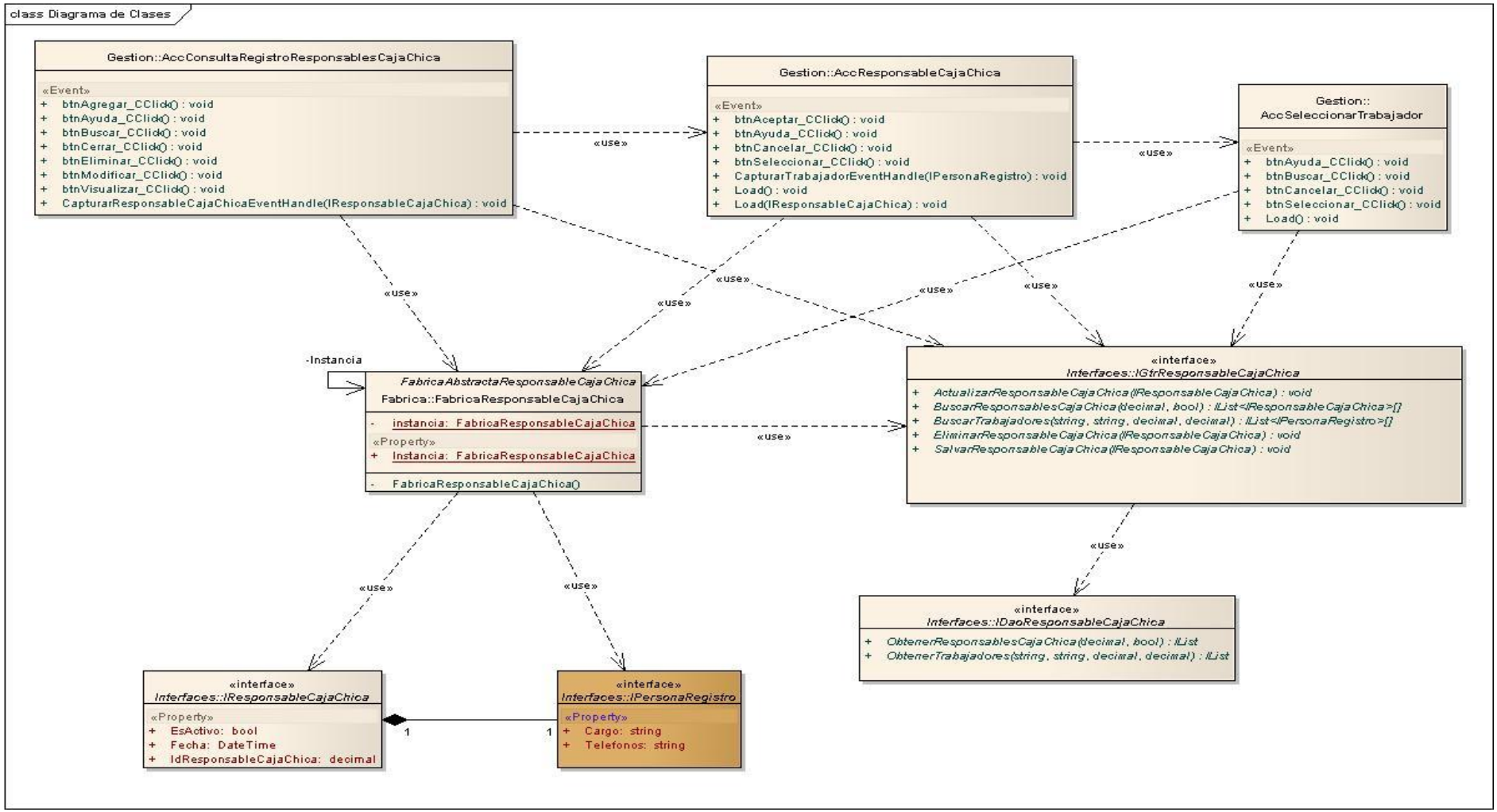

Figura 21 Diagrama de Clases de diseño CU: Gestionar Responsables del Fondo de Caja Chica.

A continuación se presentan los diagramas de secuencia correspondientes al CU: Gestionar Responsables del Fondo de Caja Chica. Estos se muestran divididos por escenarios para facilitar su comprensión.

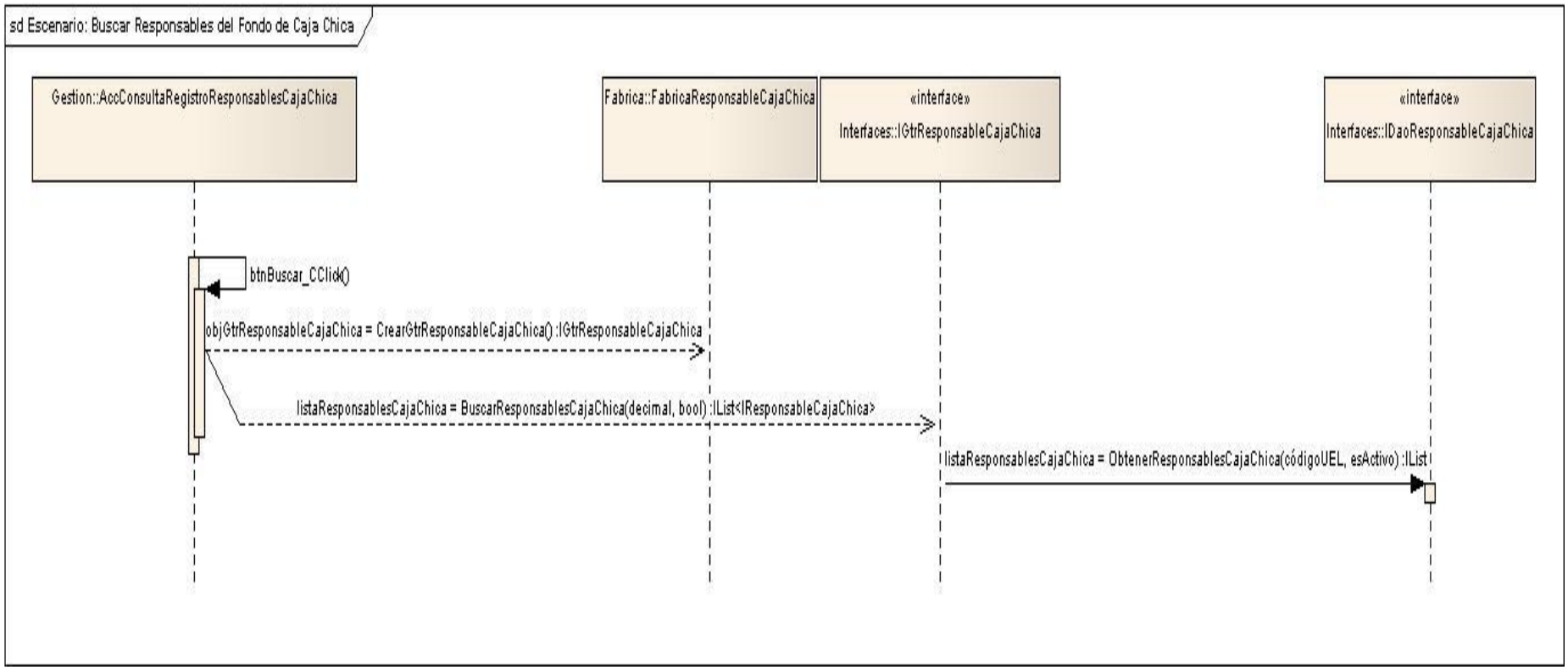

Figura 22 Diagrama de Secuencia del Escenario Buscar Responsables.

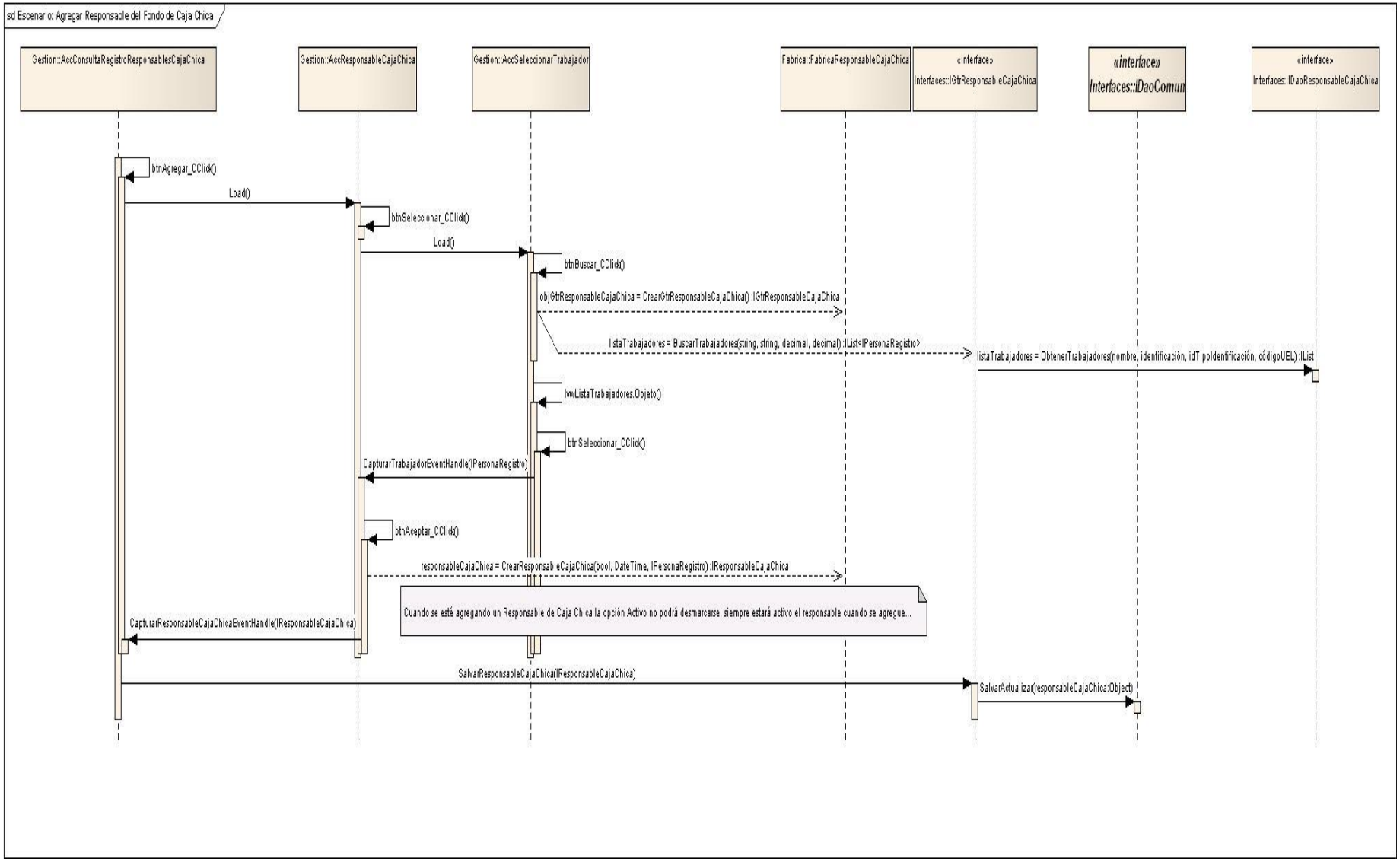

Figura 23 Diagrama de Secuencia del Escenario Agregar Responsables.

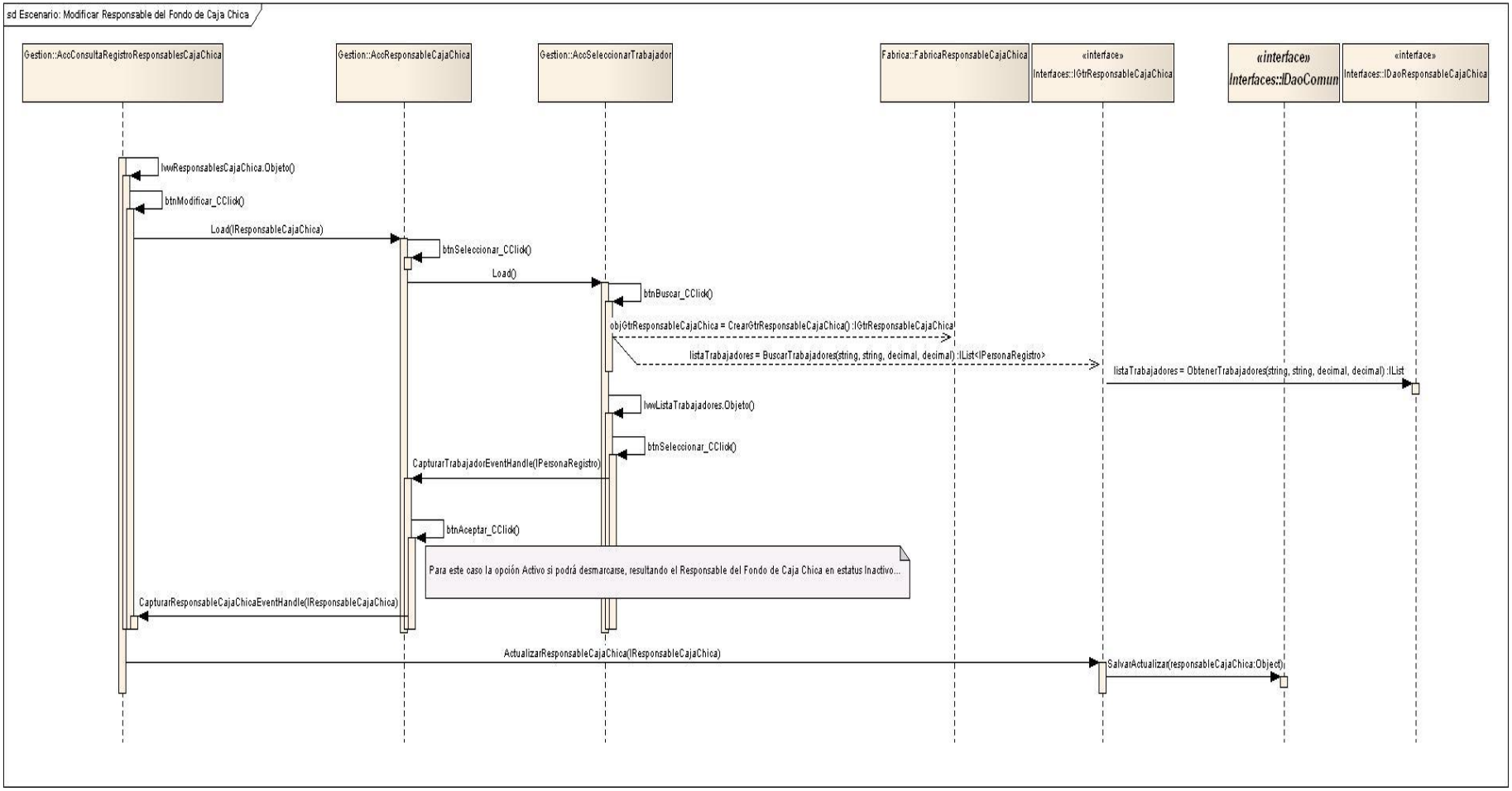

Figura 24 Diagrama de Secuencia del Escenario Modificar Responsables.

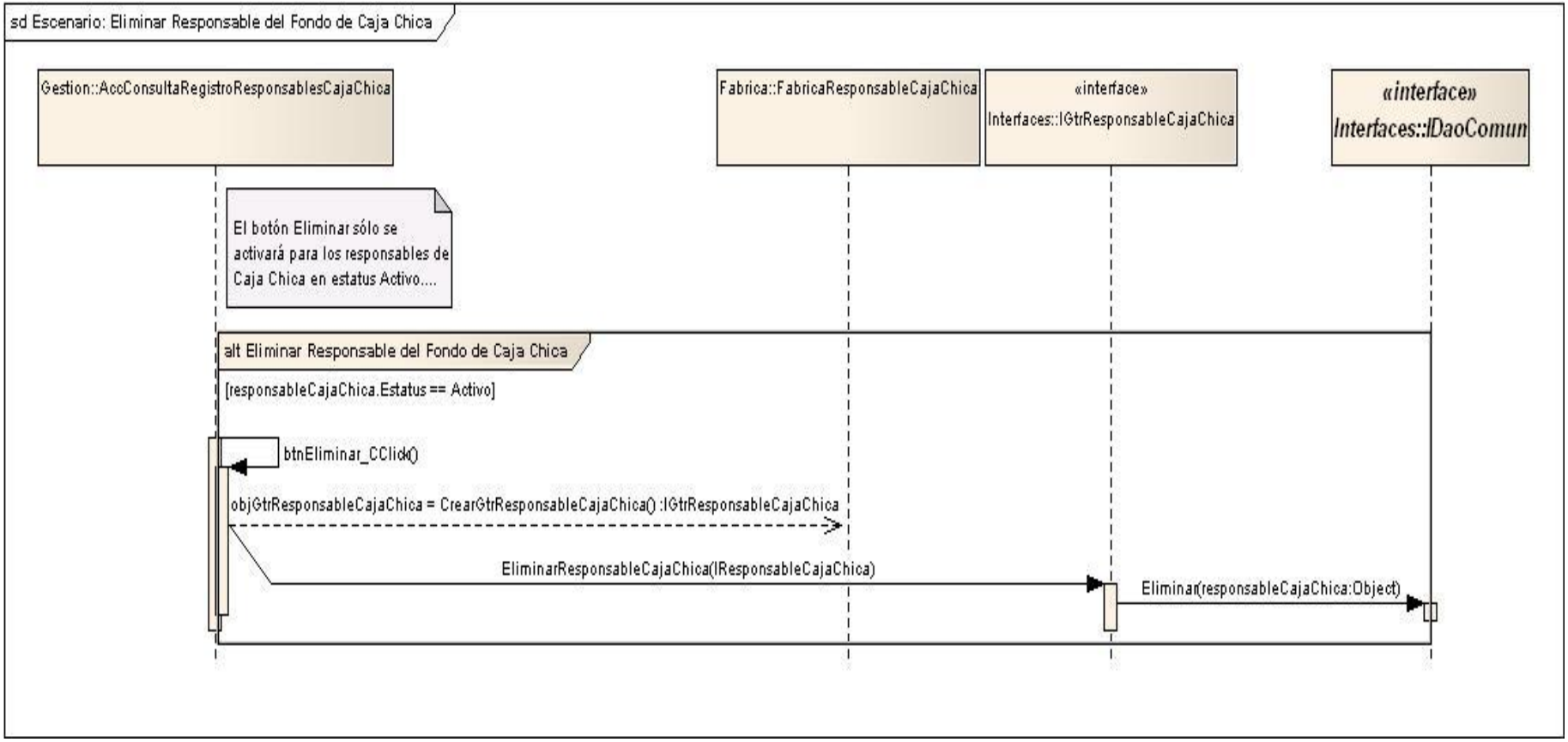

Figura 25 Diagrama de Secuencia del Escenario Eliminar Responsables.

Anexo 2. RCU Gestionar Fondos de Caja Chica.

Diagrama de Clases del diseño, diagrama de transición de estados y diagrama de Interacción para el CU Gestionar Fondos de Caja Chica. Este CU posibilita buscar, incorporar, modificar, eliminar, constituir, confirmar, liquidar y cerrar fondos. Este caso de uso es de suma importancia pues de él dependerán los posteriores casos de uso.

A continuación se presenta el diagrama de transición de estado correspondiente al CU: Gestionar Fondos de Caja Chica. Este muestra los diferentes estados por los cuales puede pasar la entidad Fondo de Caja Chica.

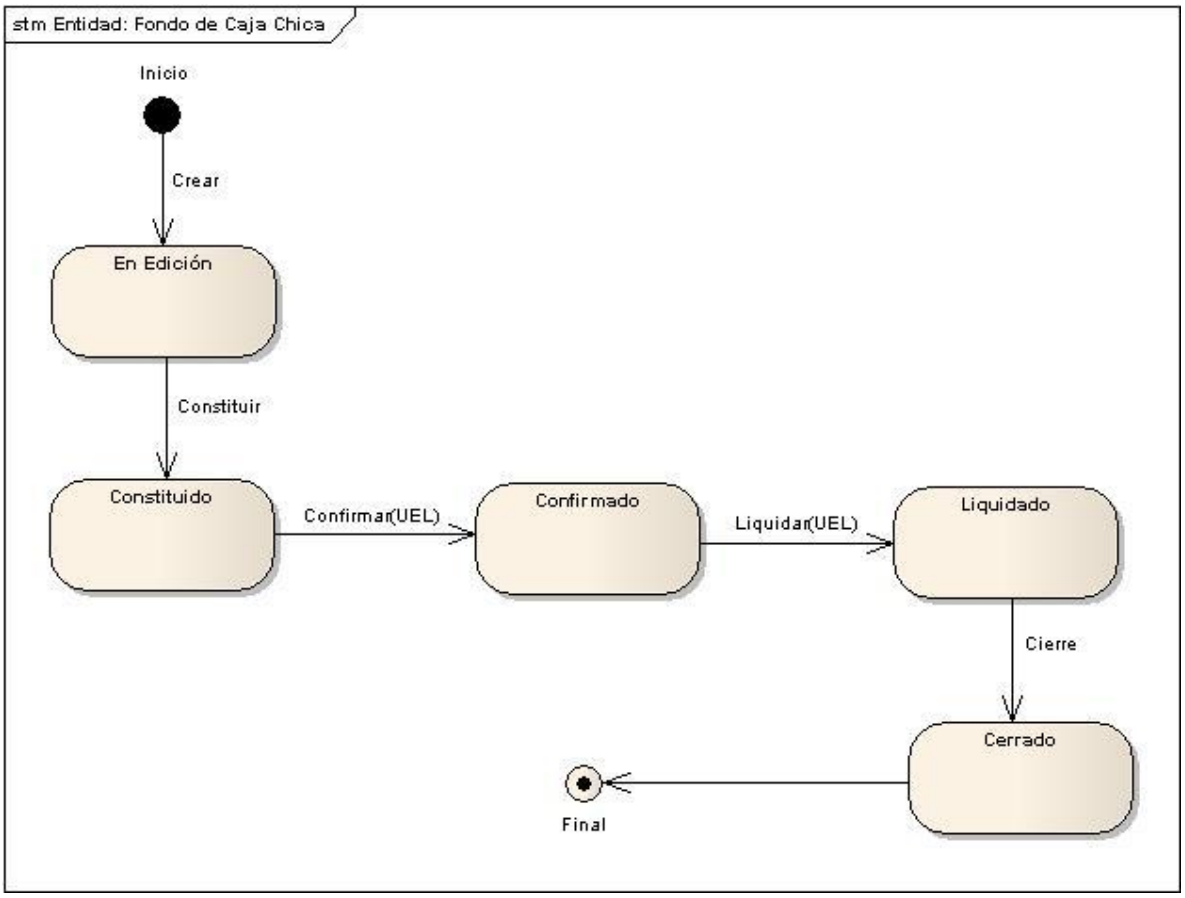

Figura 26 Diagrama de Transición de Estados de la Entidad Fondo de Caja Chica.

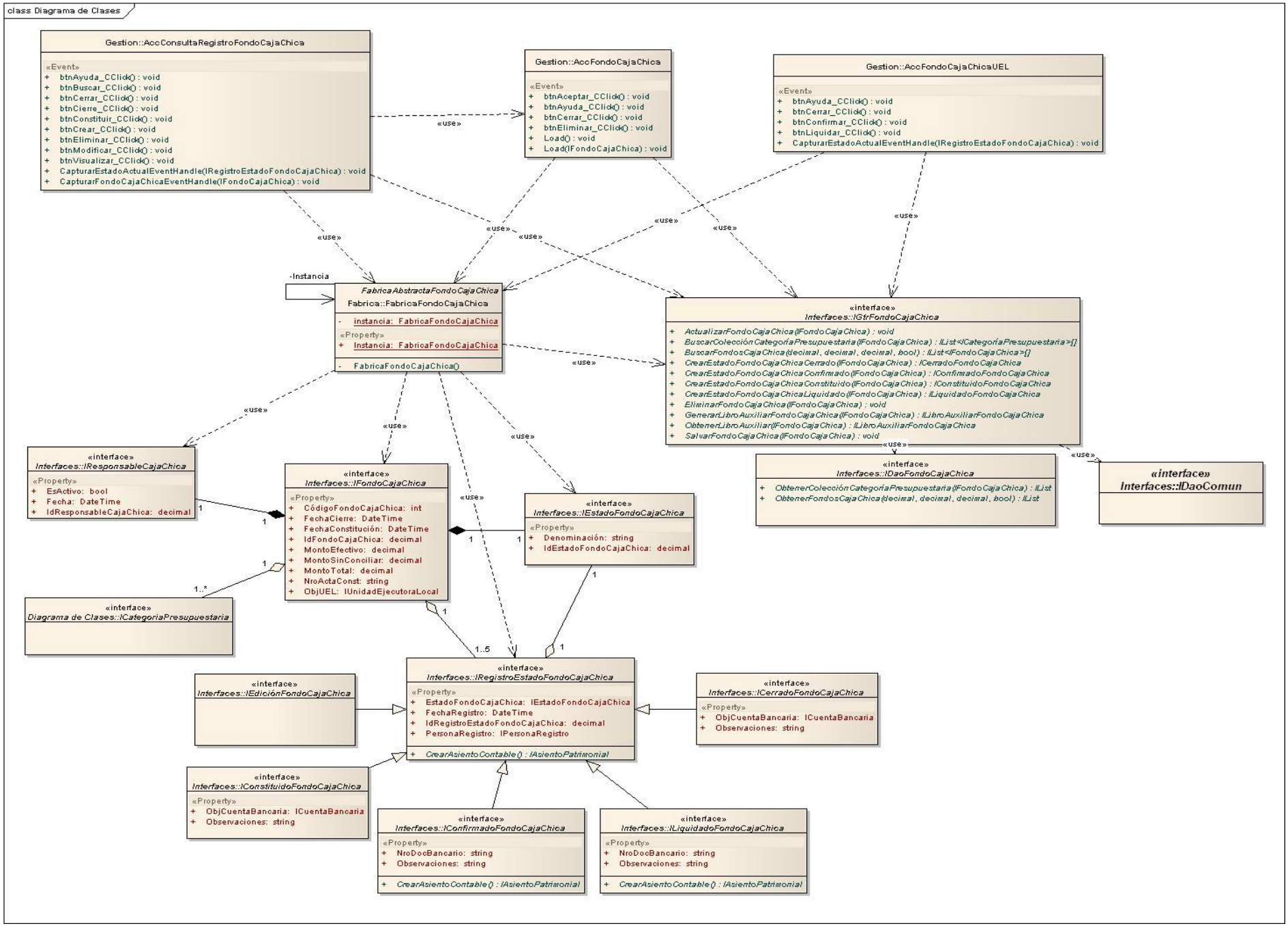

Figura 27 Diagrama de Clases de diseño CU: Gestionar Fondos de Caja Chica.

A continuación se presentan los diagramas de secuencia correspondientes al CU: Gestionar Fondos de Caja Chica. Estos se muestran divididos por escenarios para facilitar su comprensión.

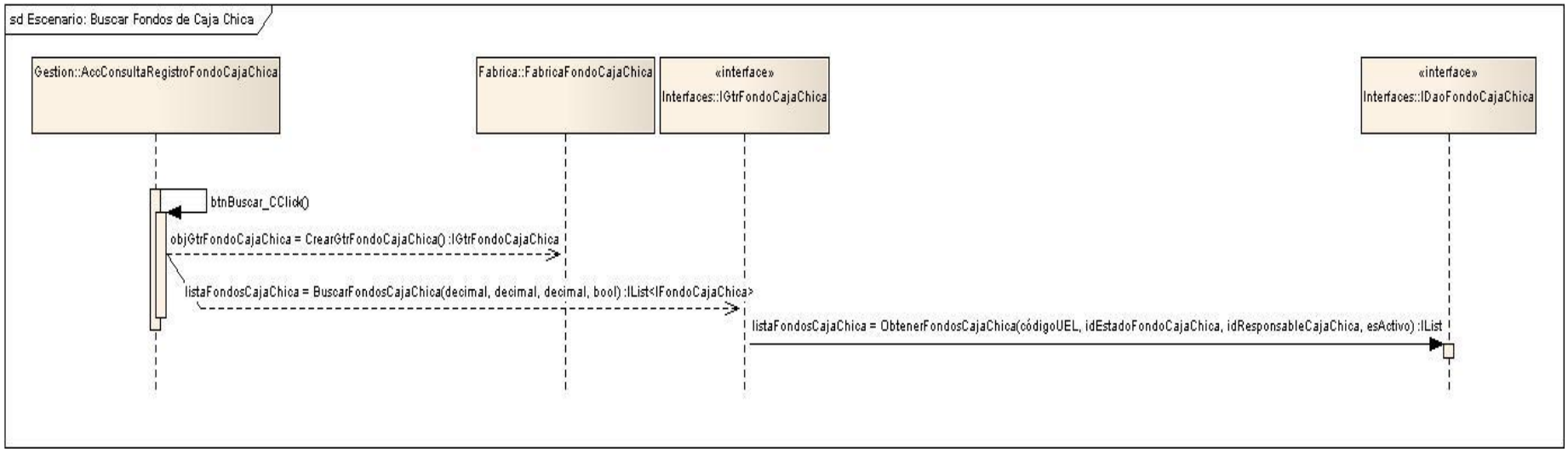

Figura 28 Diagrama de Secuencia del Escenario Buscar Fondos.

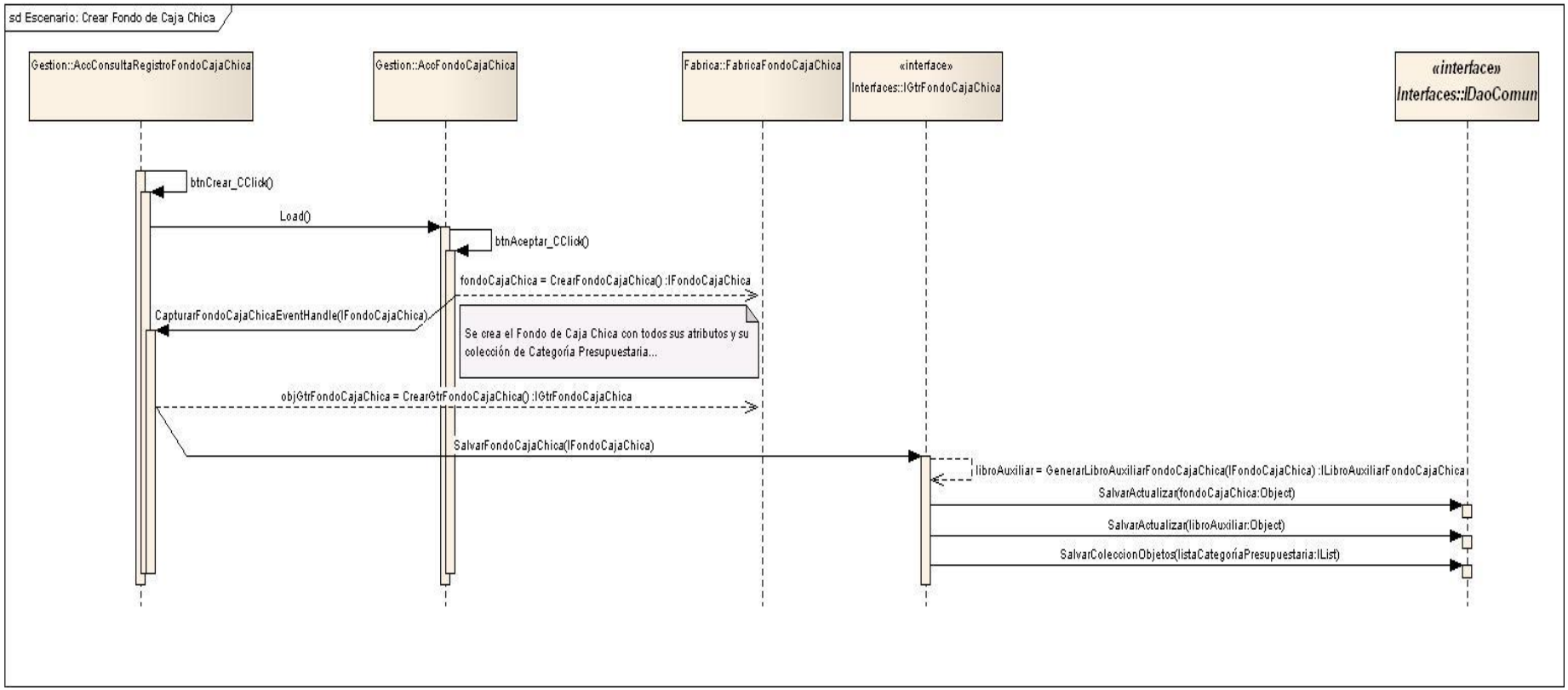

Figura 29 Diagrama de Secuencia del Escenario Crear Fondos.

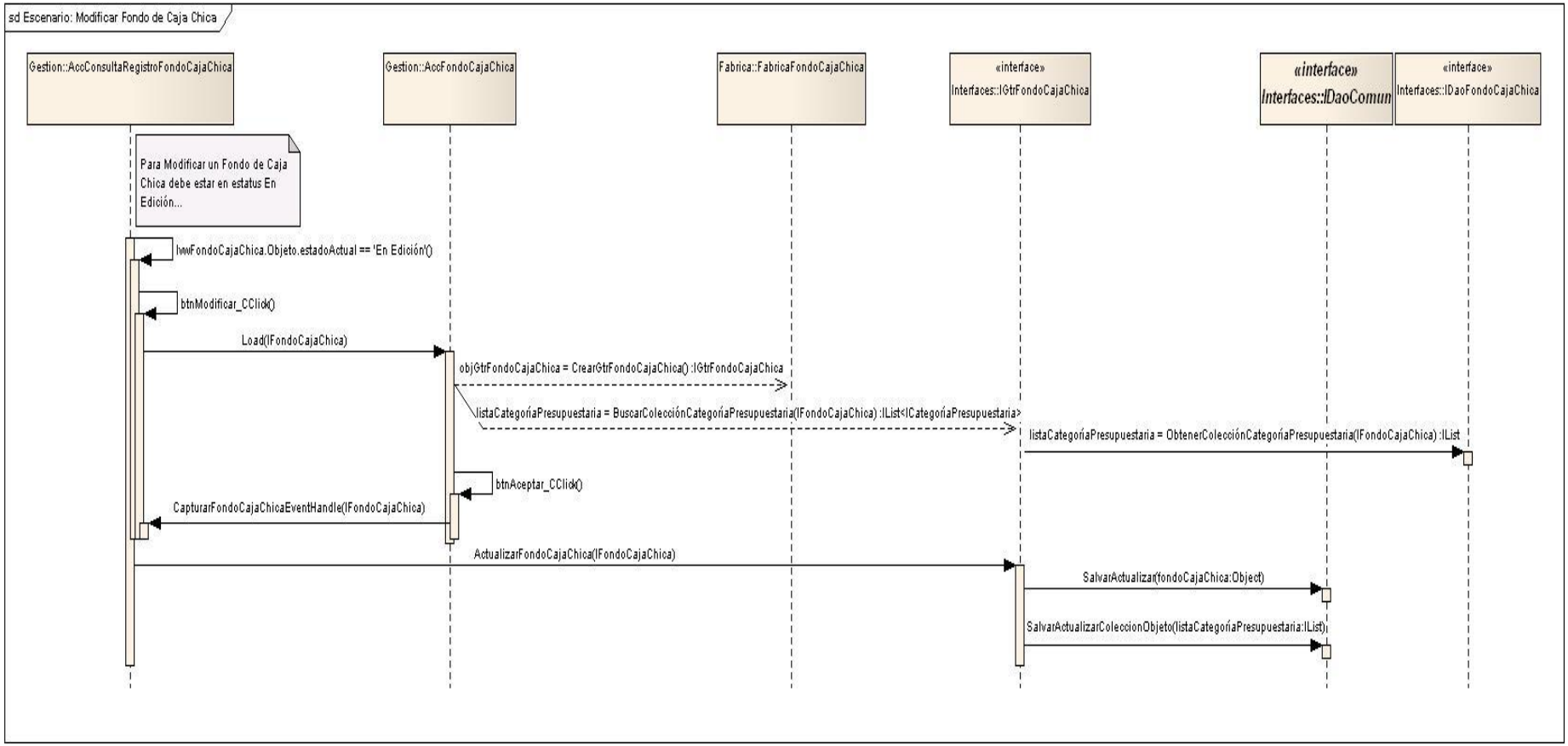

Figura 30 Diagrama de Secuencia del Escenario Modificar Fondos.

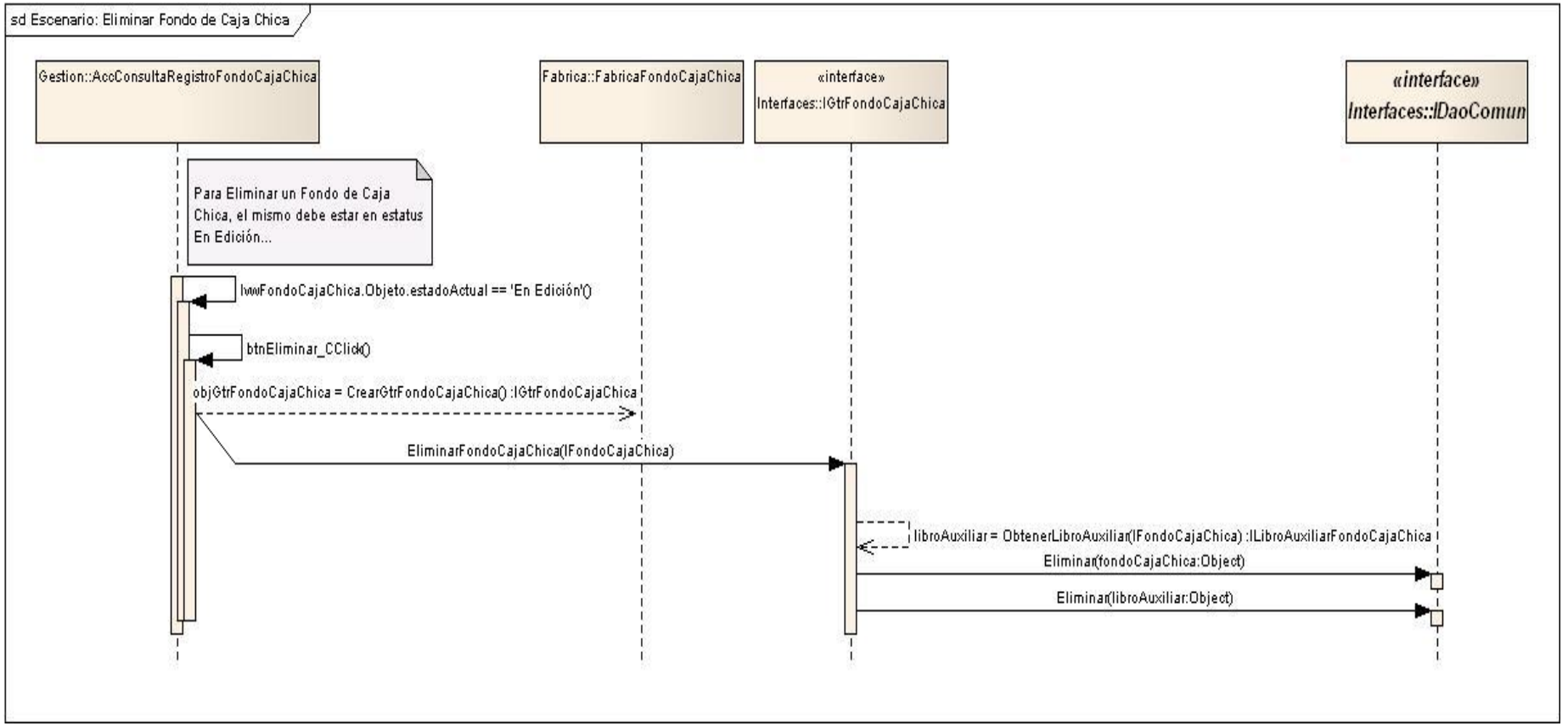

Figura 31 Diagrama de Secuencia del Escenario Eliminar Fondos.

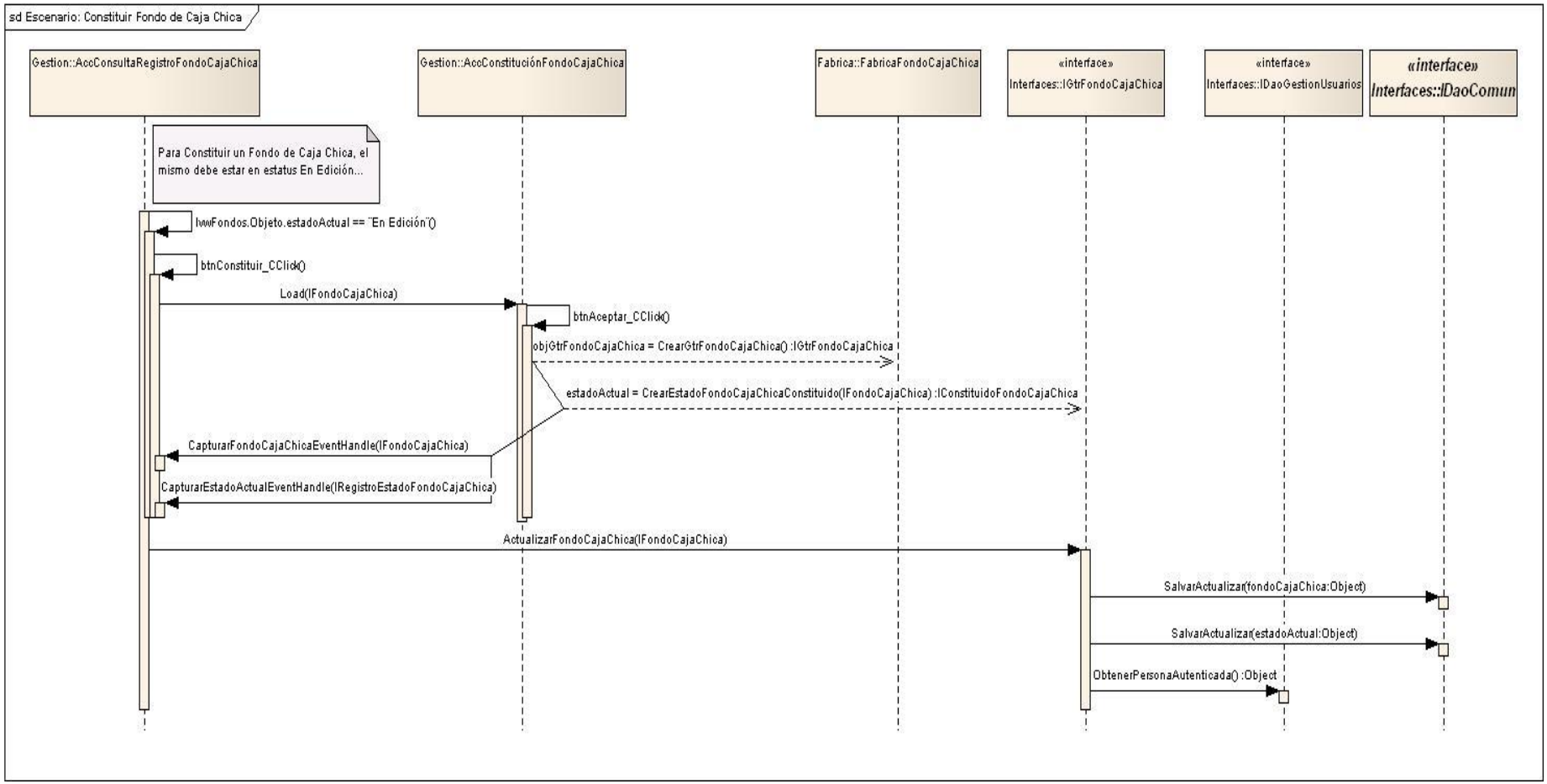

Figura 32 Diagrama de Secuencia del Escenario Constituir Fondos.

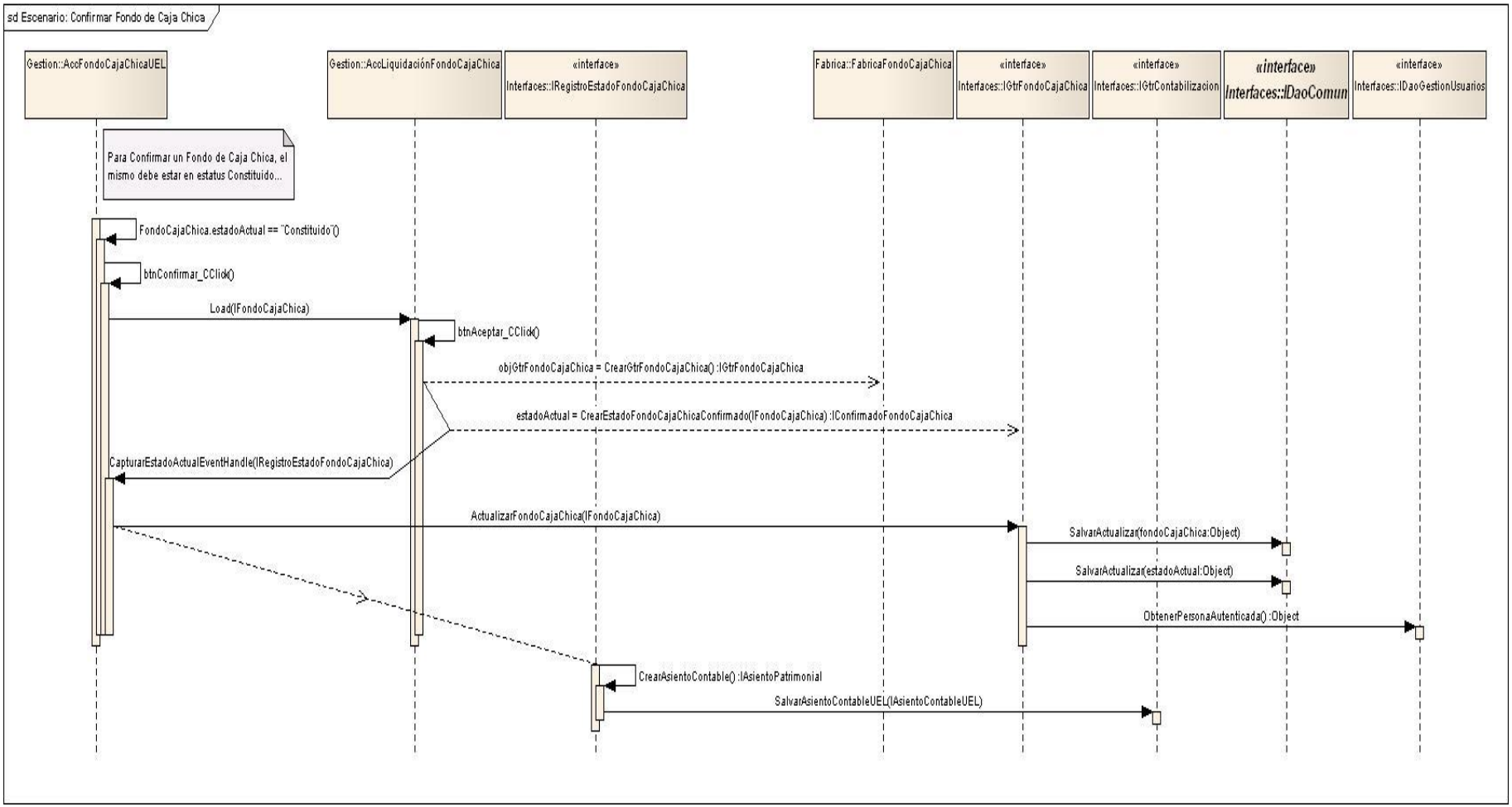

Figura 33 Diagrama de Secuencia del Escenario Confirmar Fondos.

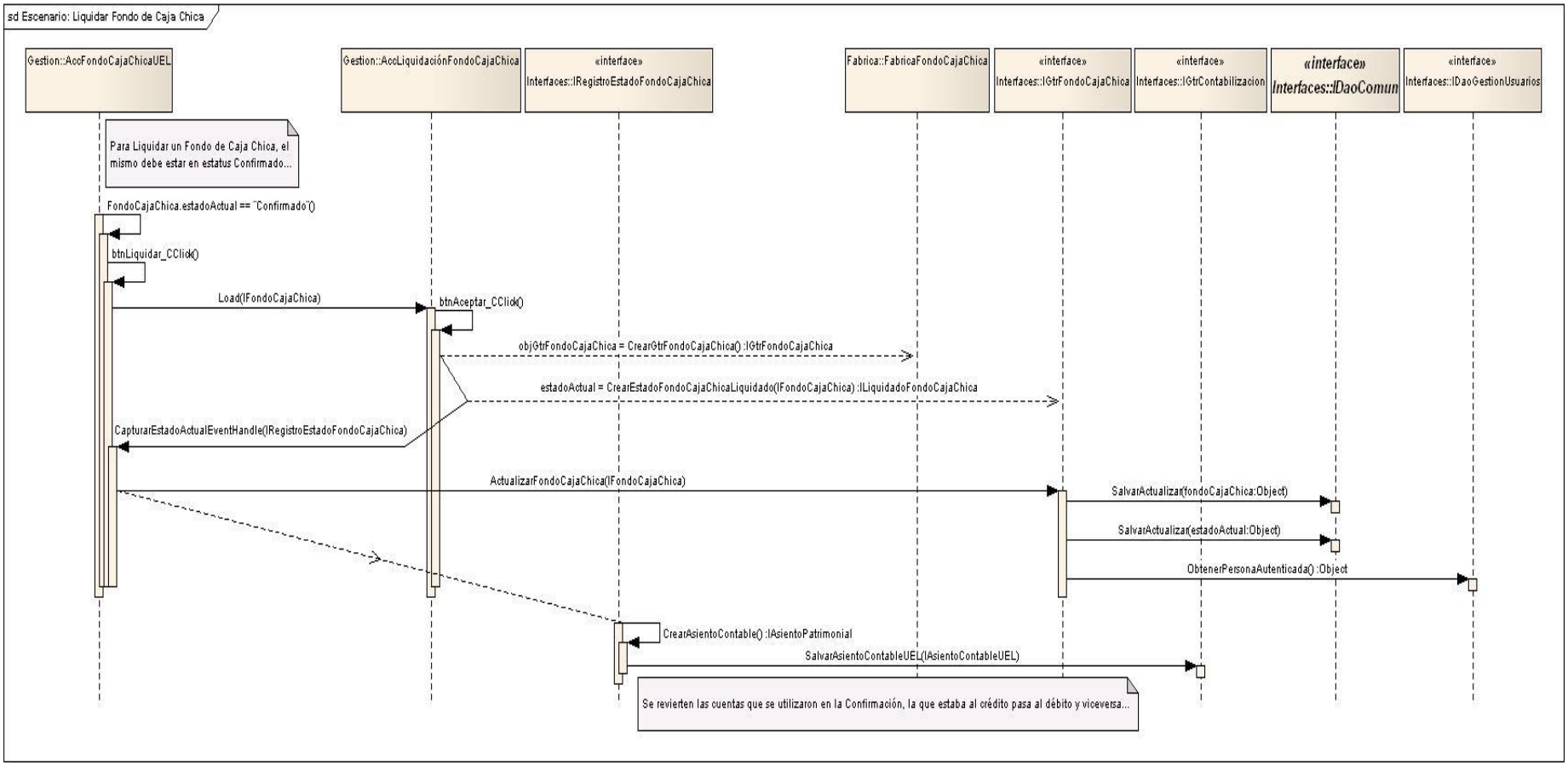

Figura 34 Diagrama de Secuencia del Escenario Liquidar Fondos.

Anexo 3. RCU Gestionar Vales de Caja Chica.

Diagrama de Clases del diseño, diagrama de transición de estados y diagrama de Interacción para el CU Gestionar Vales de Caja Chica. Este CU posibilita buscar, incorporar, modificar, eliminar, aprobar y anular vales de caja chica.

A continuación se presenta el diagrama de transición de estado correspondiente al CU: Gestionar Vales de Caja Chica. Este muestra los diferentes estados por los cuales puede pasar la entidad Fondo de Caja Chica.

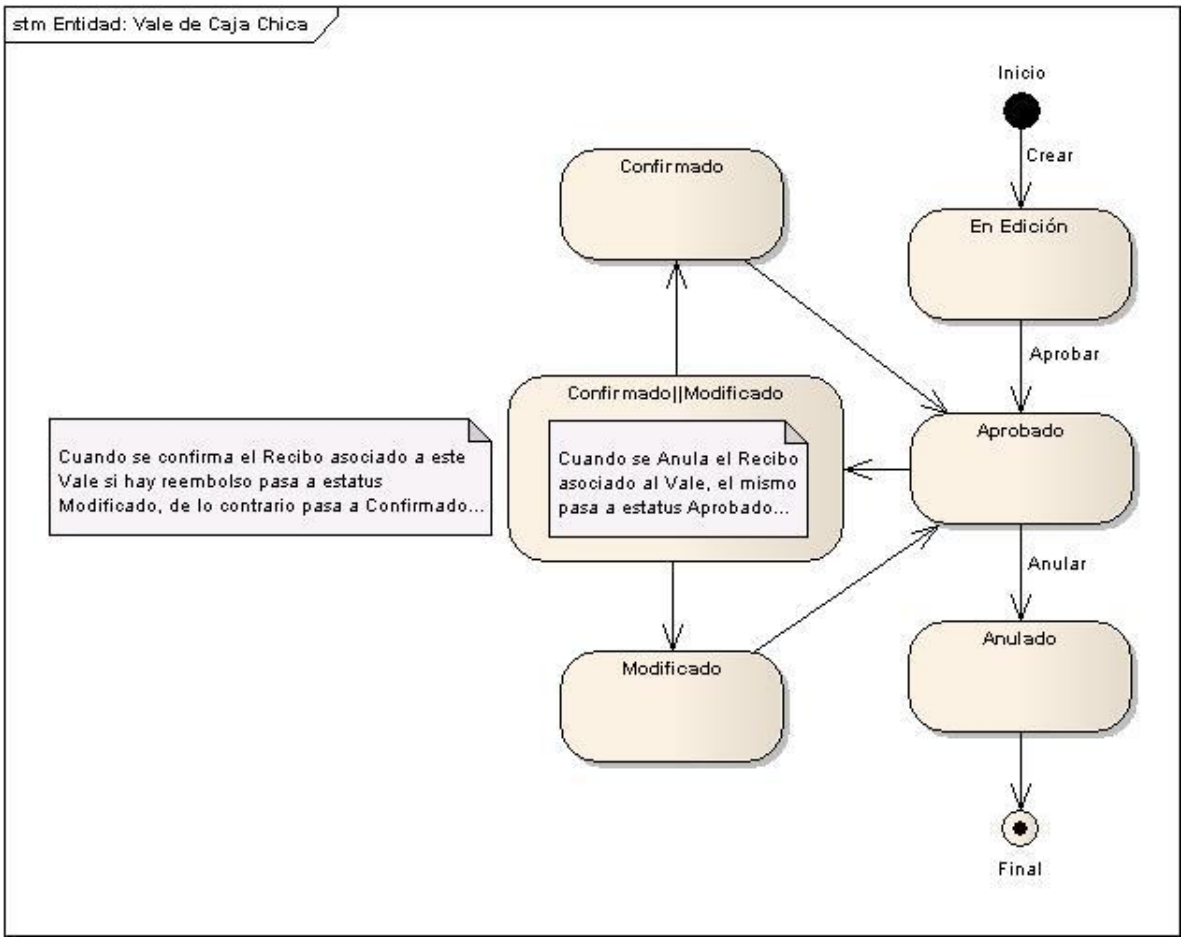

Figura 35 Diagrama de Transición de Estados de la Entidad Vale de Caja Chica.

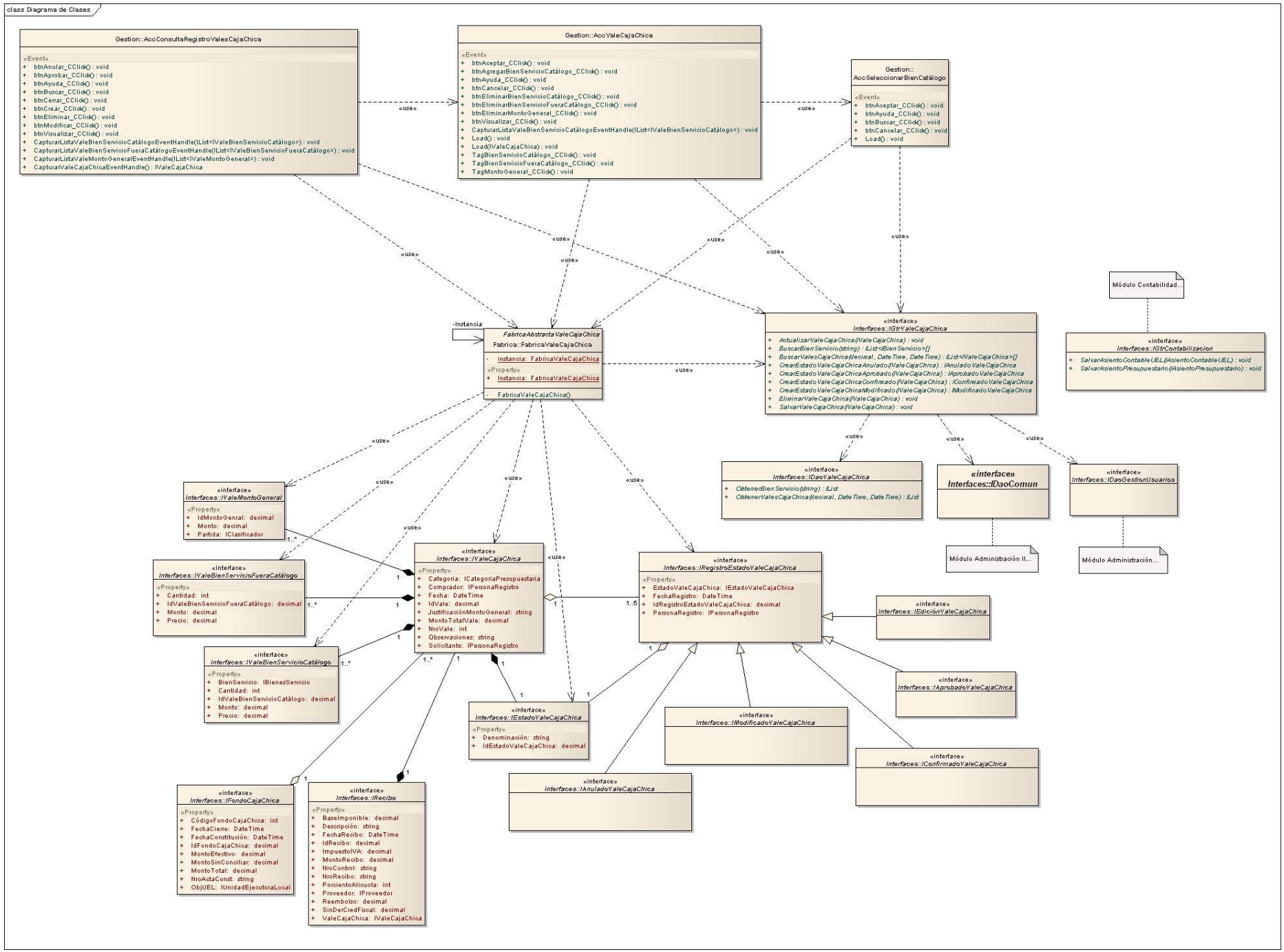

Figura 36 Diagrama de Clases de diseño CU: Gestionar Vales de Caja Chica.

A continuación se presentan los diagramas de secuencia correspondientes al CU: Gestionar Fondos de Caja Chica. Estos se muestran divididos por escenarios para facilitar su comprensión.

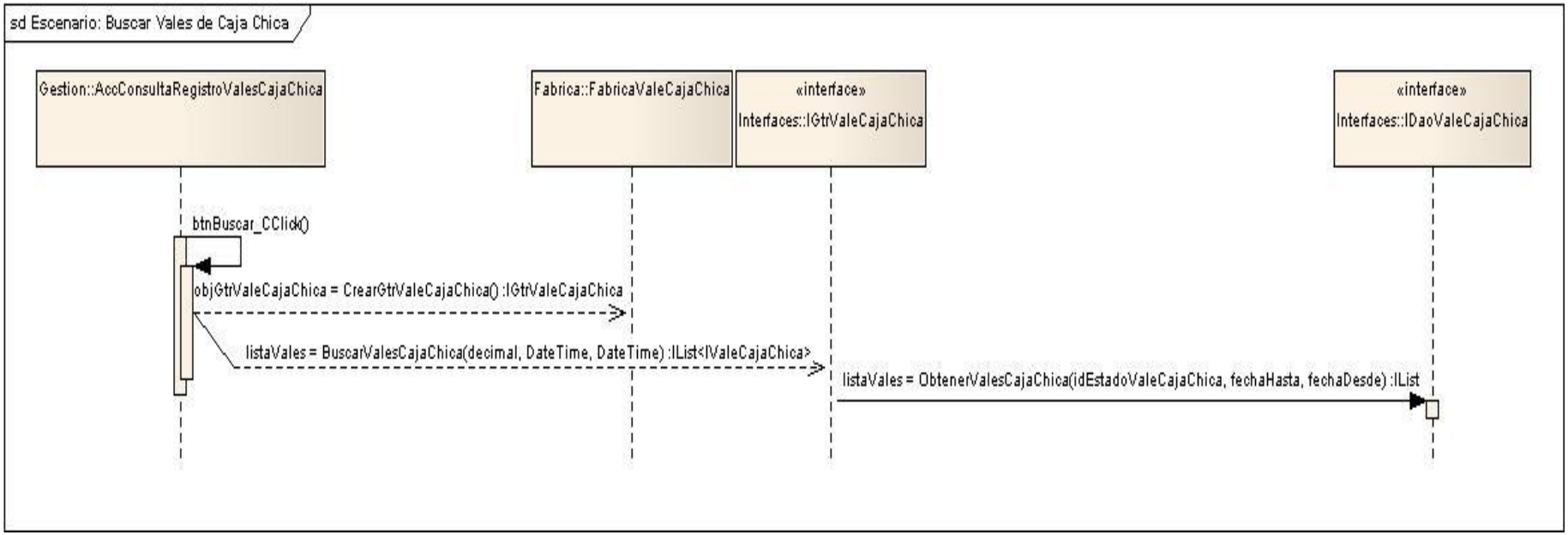

Figura 37 Diagrama de Secuencia del Escenario Buscar Vales.

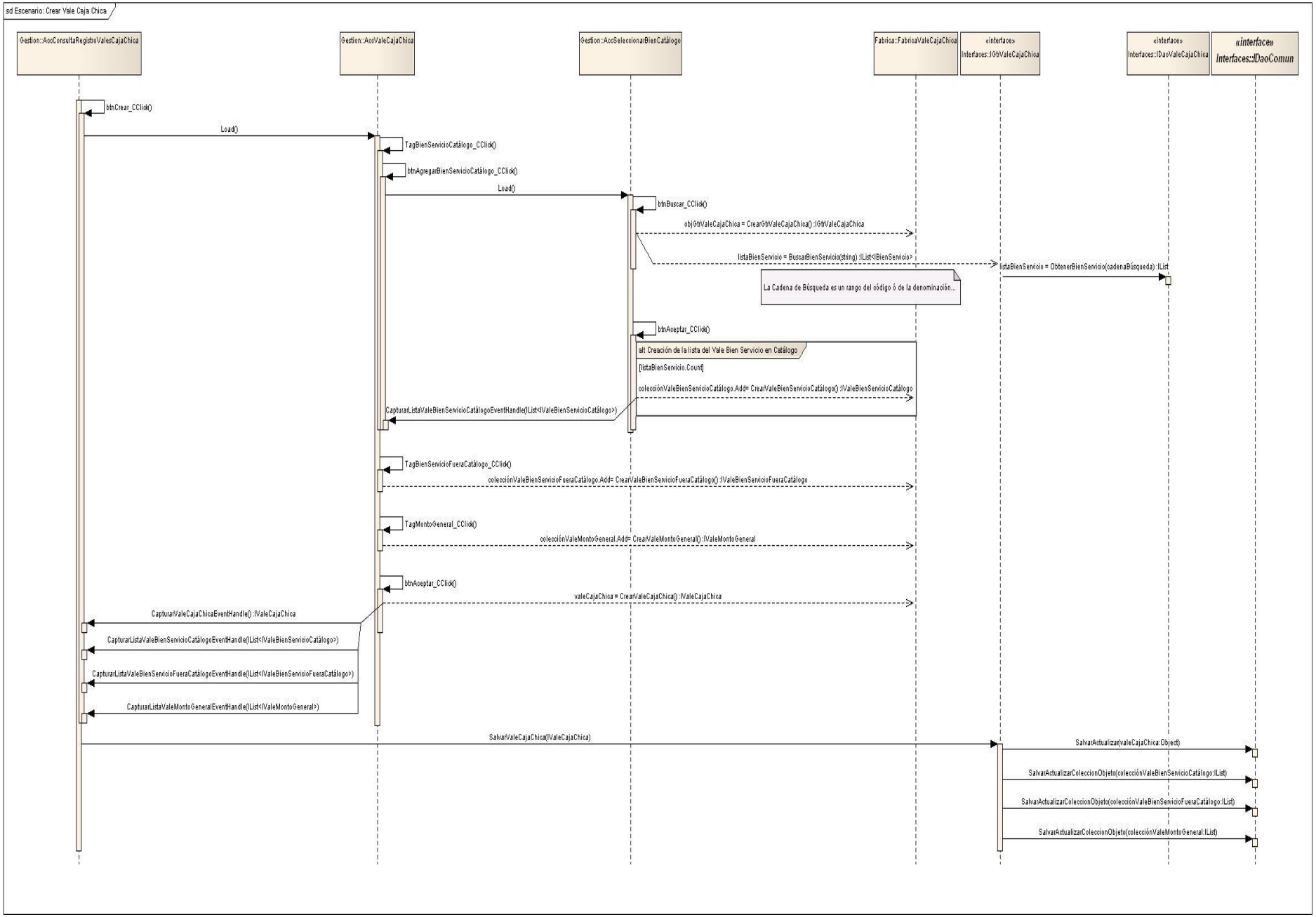

Figura 38 Diagrama de Secuencia del Escenario Crear Vales.

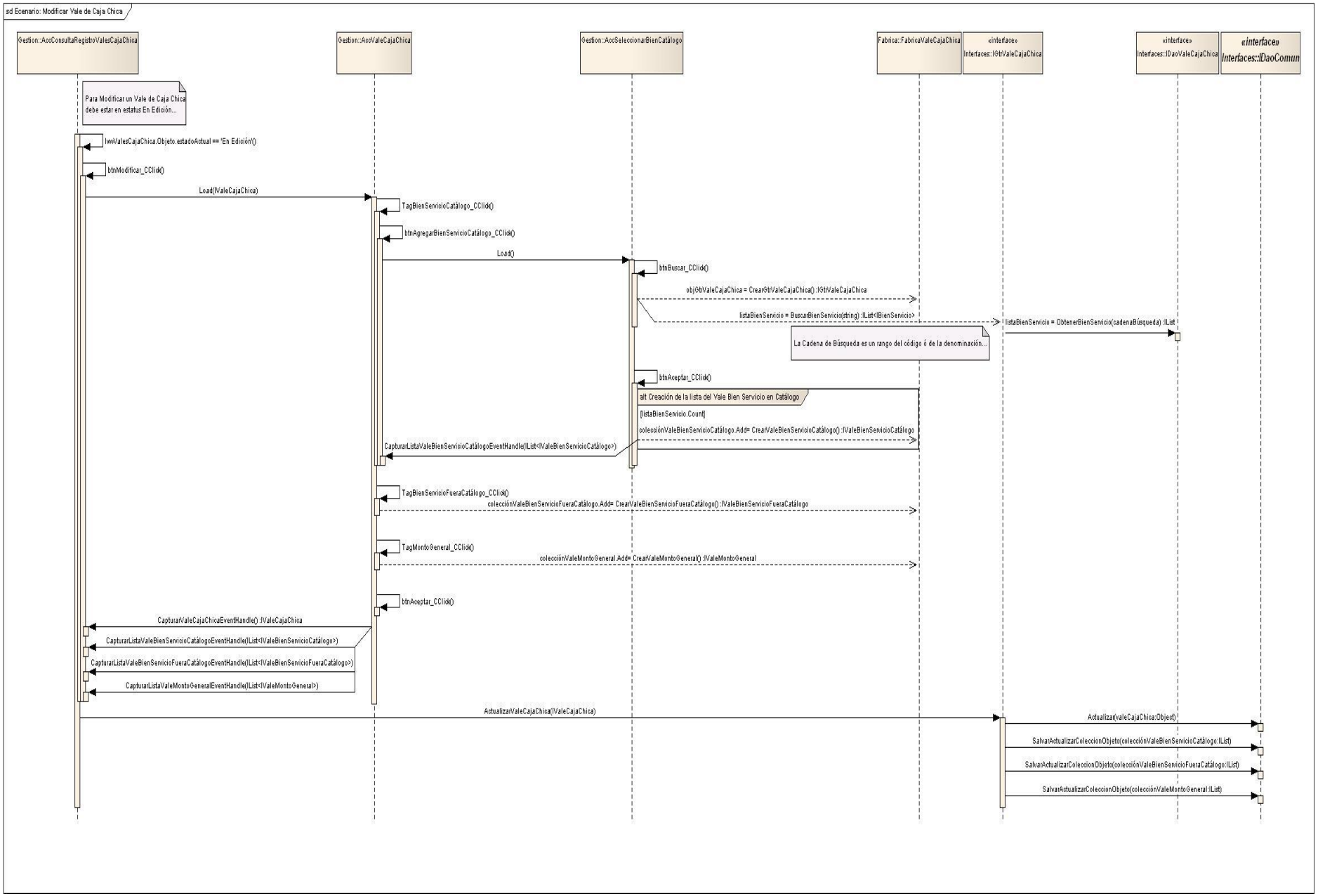

Figura 39 Diagrama de Secuencia del Escenario Modificar Vales.

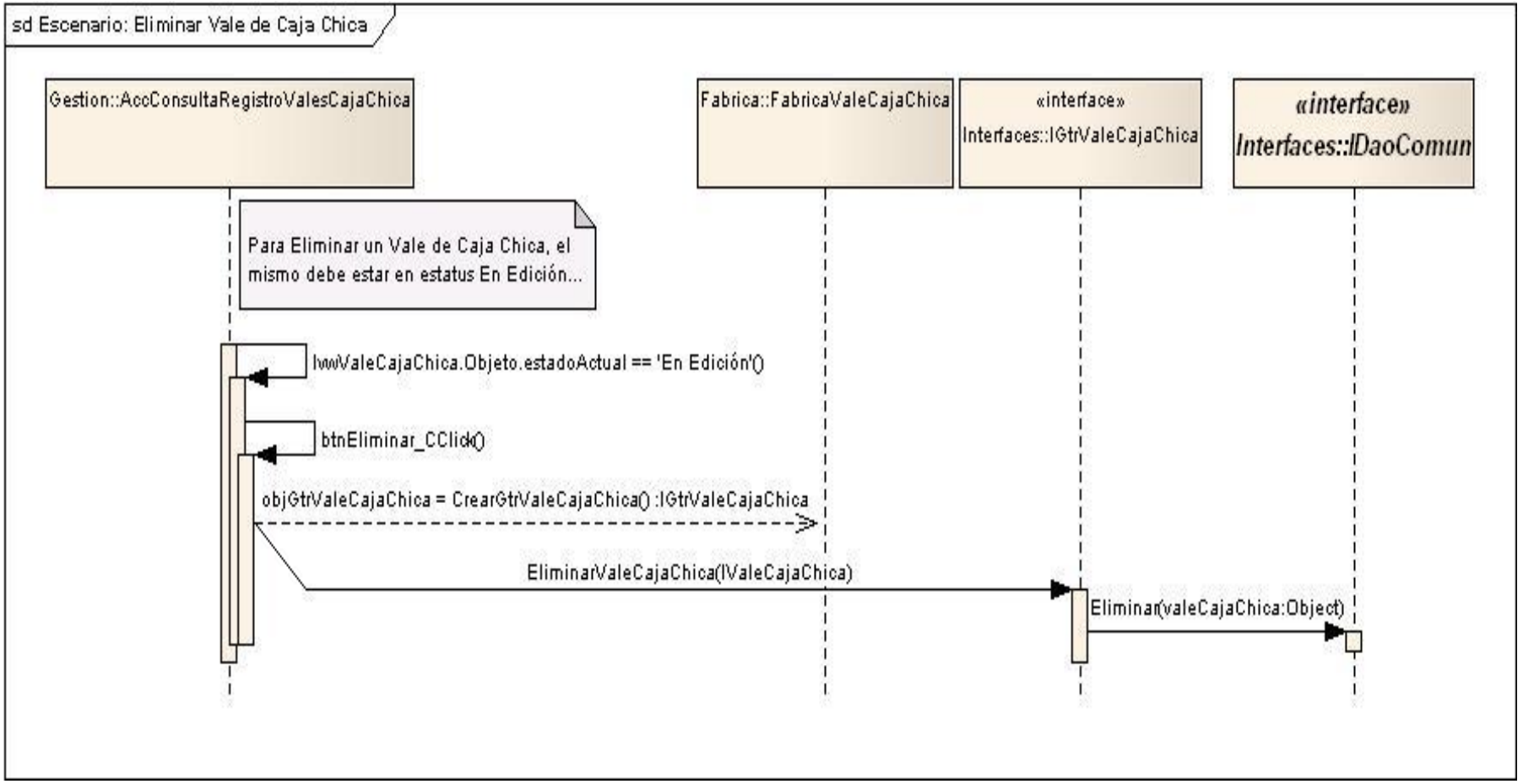

Figura 40 Diagrama de Secuencia del Escenario Eliminar Vales.

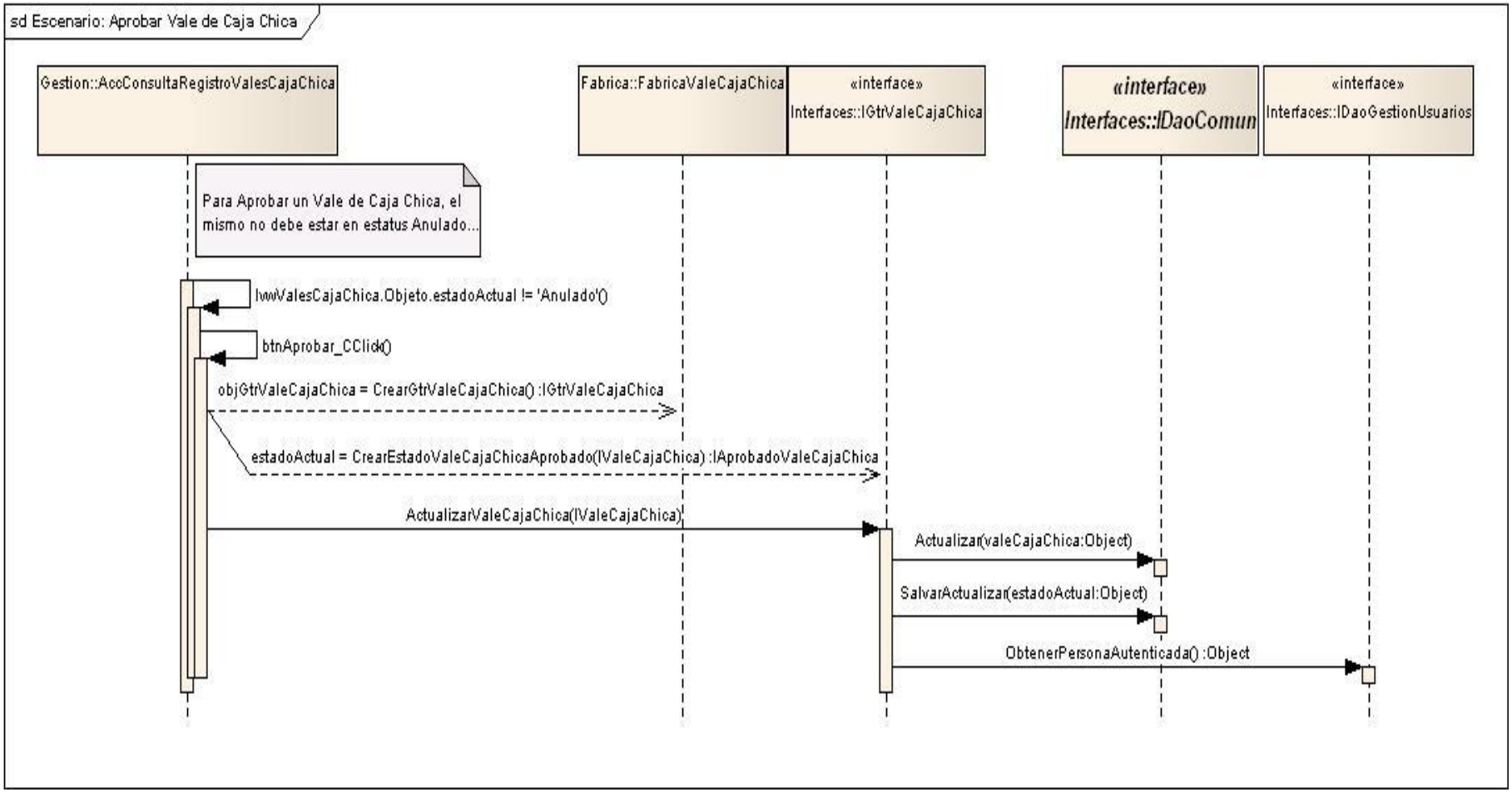

Figura 41 Diagrama de Secuencia del Escenario Aprobar Vales.

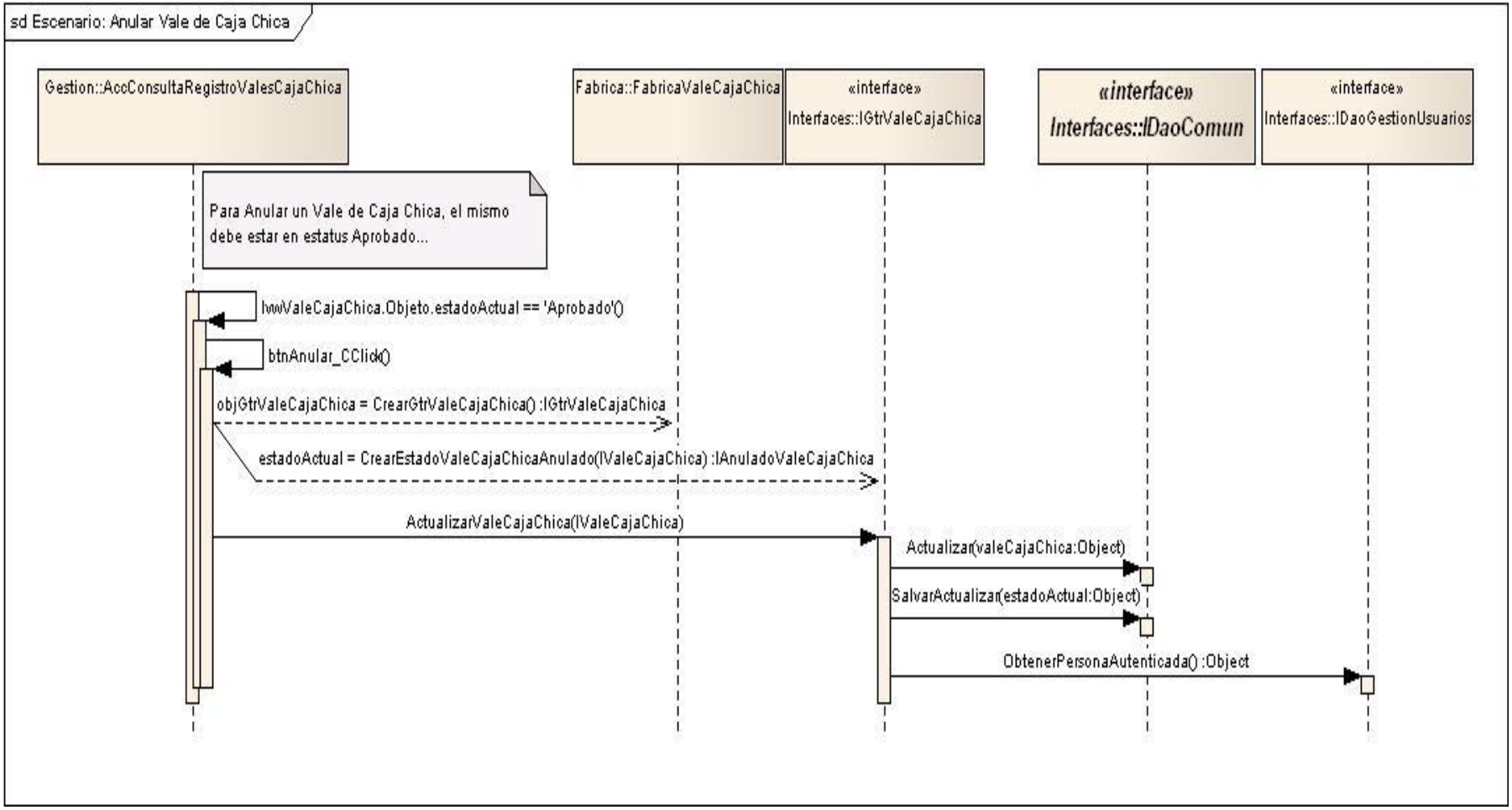

Figura 42 Diagrama de Secuencia del Escenario Anular Vales.

Anexo 4. RCU Gestionar Facturas o Recibos.

Diagrama de Clases del diseño, diagrama de transición de estados y diagrama de Interacción para el CU Gestionar Facturas o Recibos. Este CU posibilita buscar, incorporar, modificar, eliminar, confirmar y anular recibos.

A continuación se presenta el diagrama de transición de estado correspondiente al CU: Gestionar Facturas o Recibos. Este muestra los diferentes estados por los cuales puede pasar la entidad Recibo.

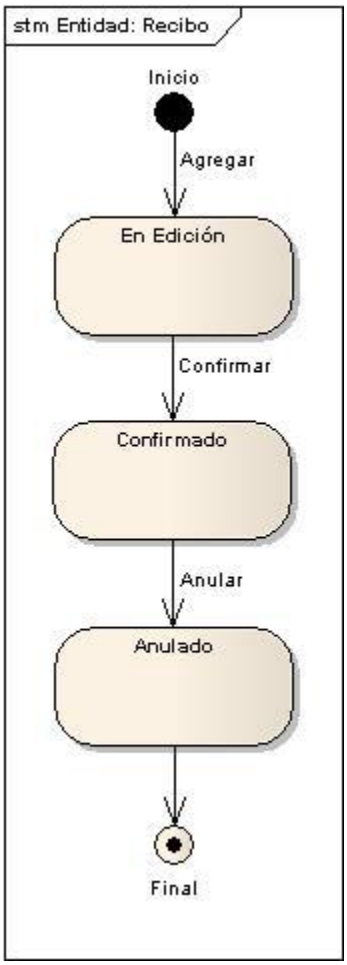

Figura 43 Diagrama de Transición de Estados de la Entidad Recibo.

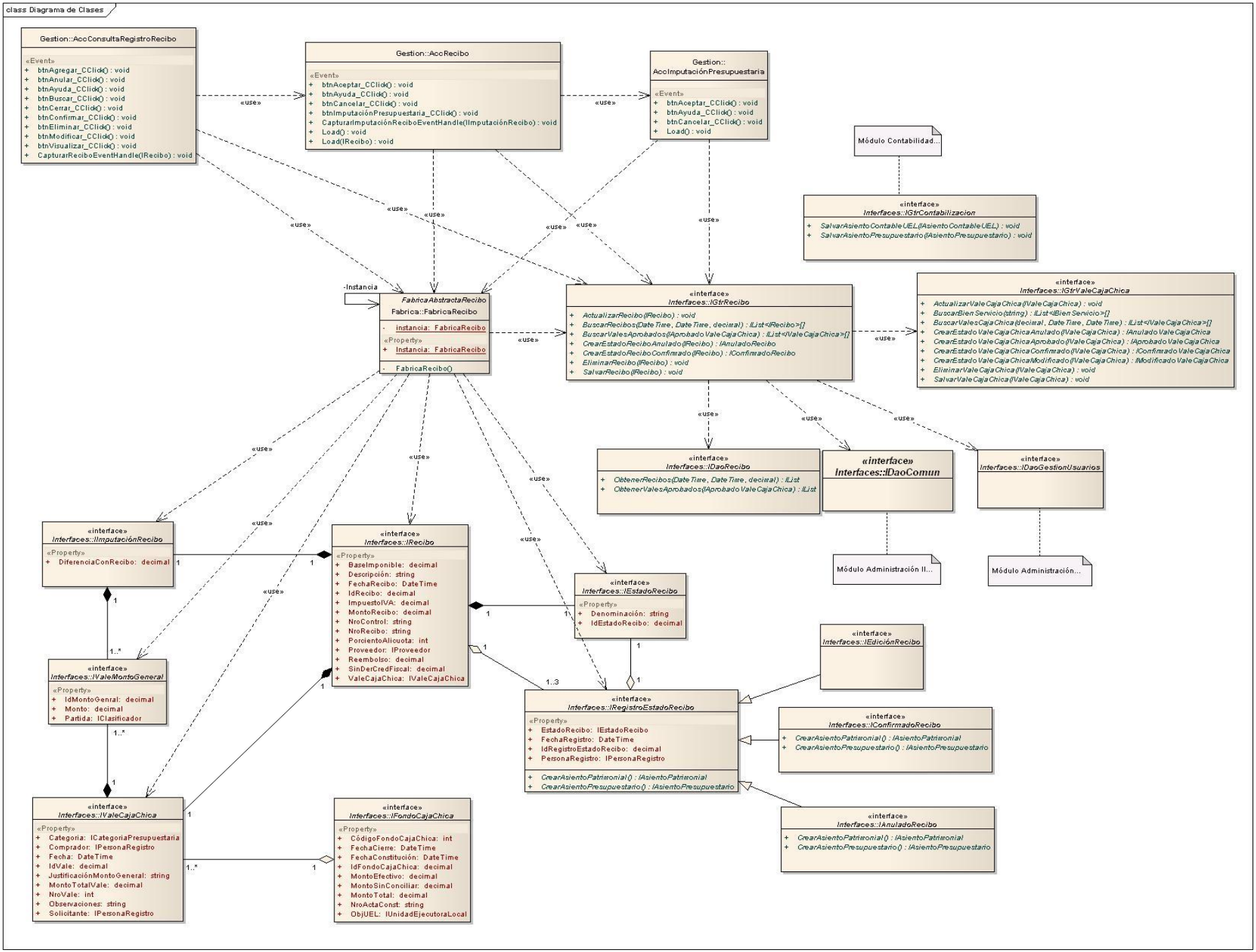

Figura 44 Diagrama de Clases de diseño CU: Gestionar Facturas o Recibos.

A continuación se presentan los diagramas de secuencia correspondientes al CU: Gestionar Facturas o Recibos. Estos se muestran divididos por escenarios para facilitar su comprensión.

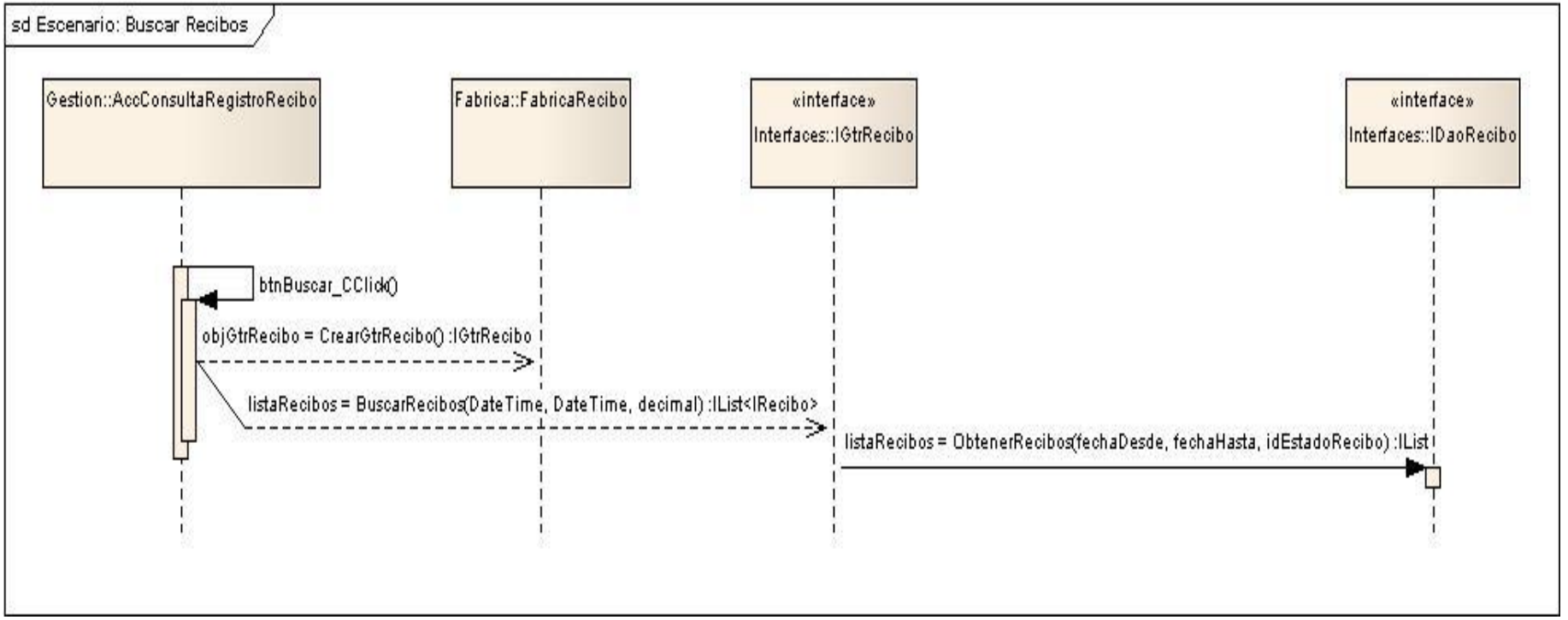

Figura 45 Diagrama de Secuencia del Escenario Buscar Recibos.

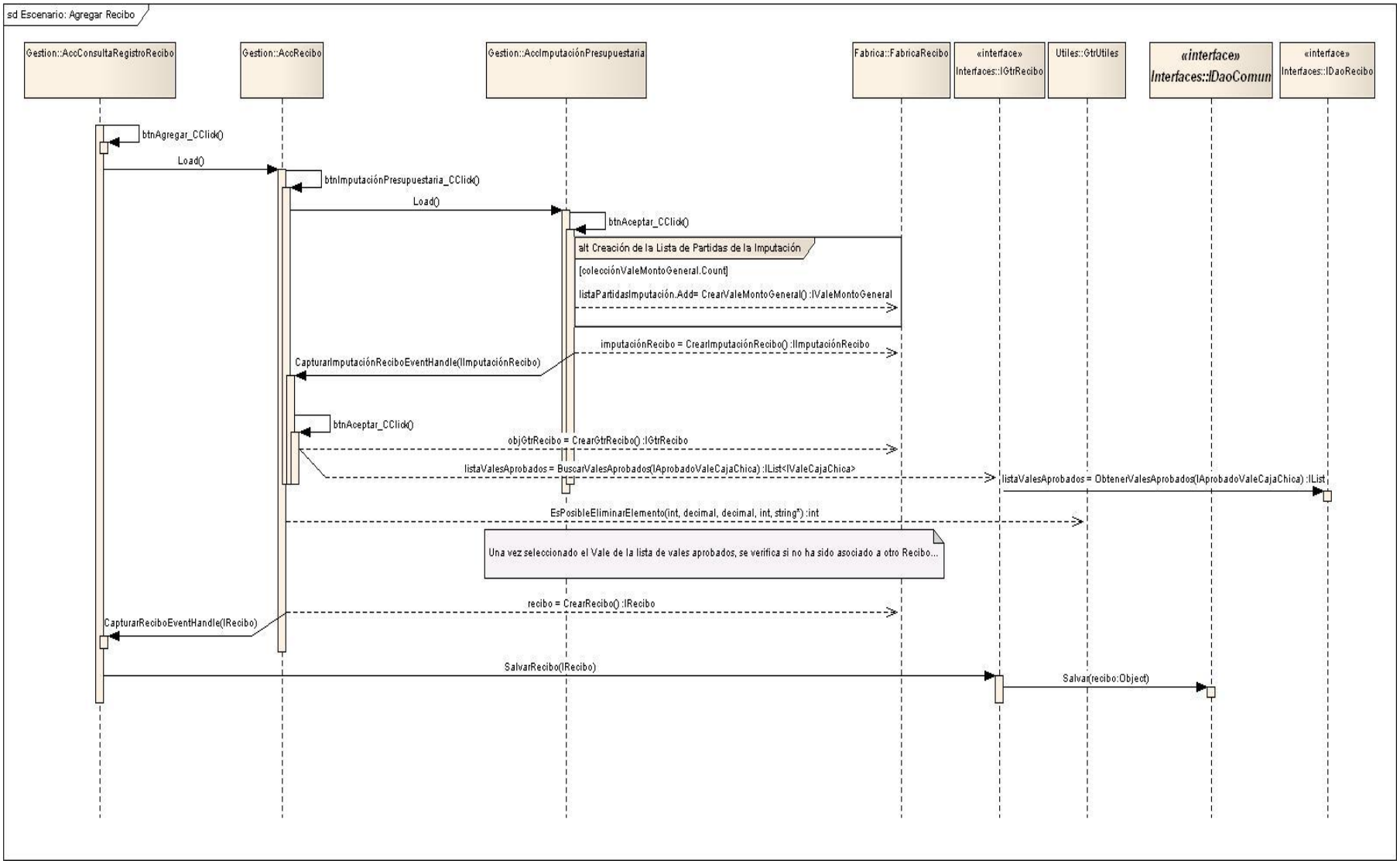

Figura 46 Diagrama de Secuencia del Escenario Crear Recibos.

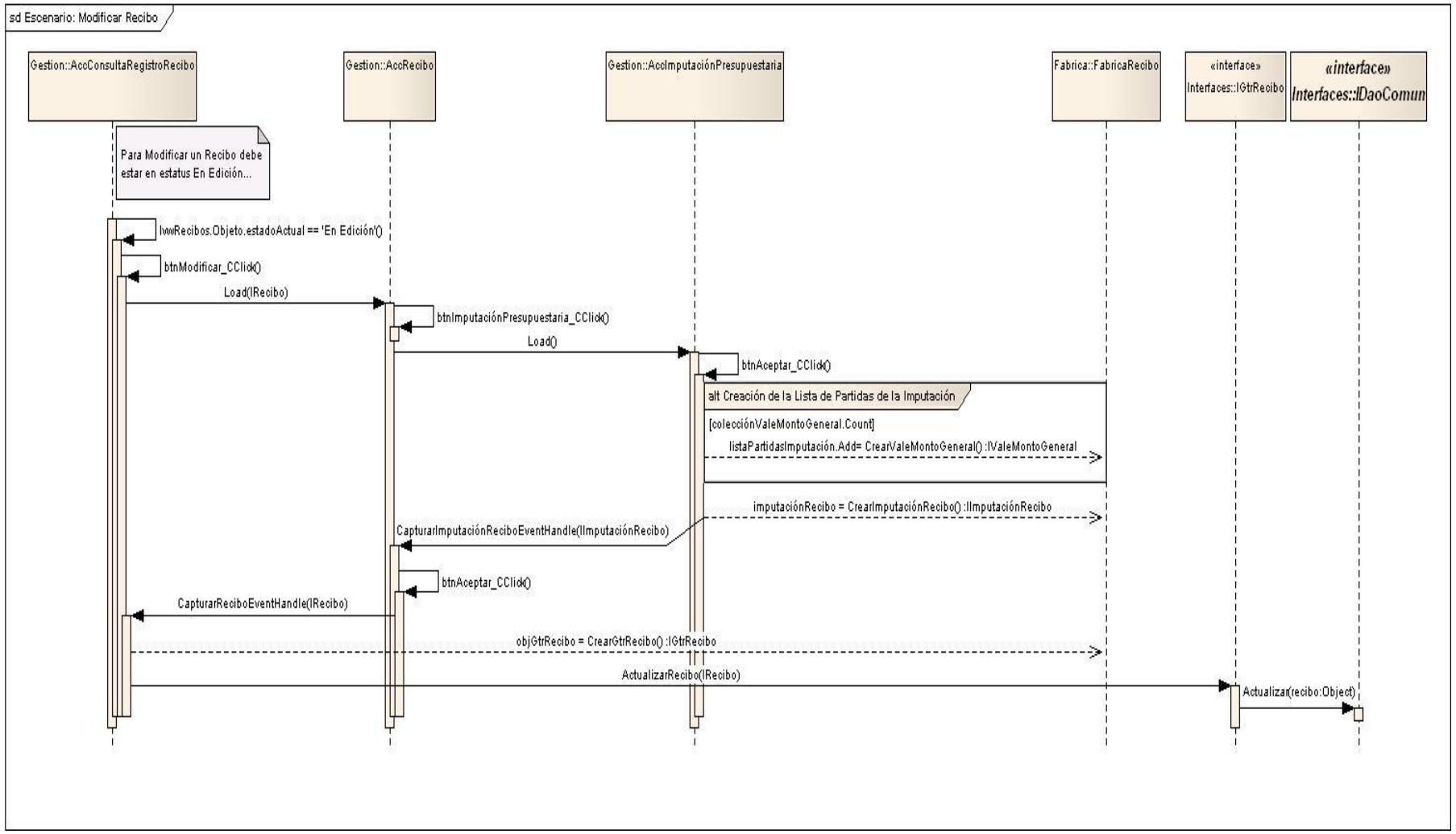

Figura 47 Diagrama de Secuencia del Escenario Modificar Recibos.

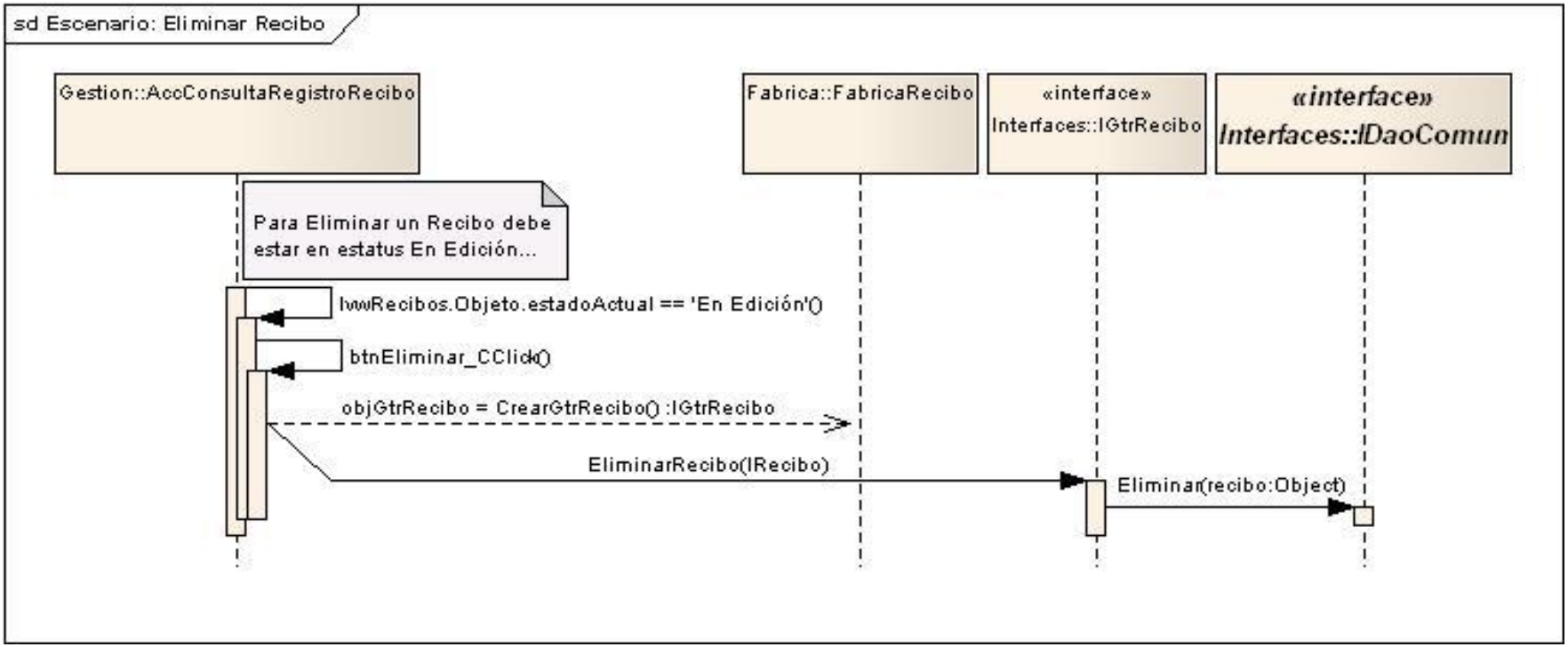

Figura 48 Diagrama de Secuencia del Escenario Eliminar Recibos.

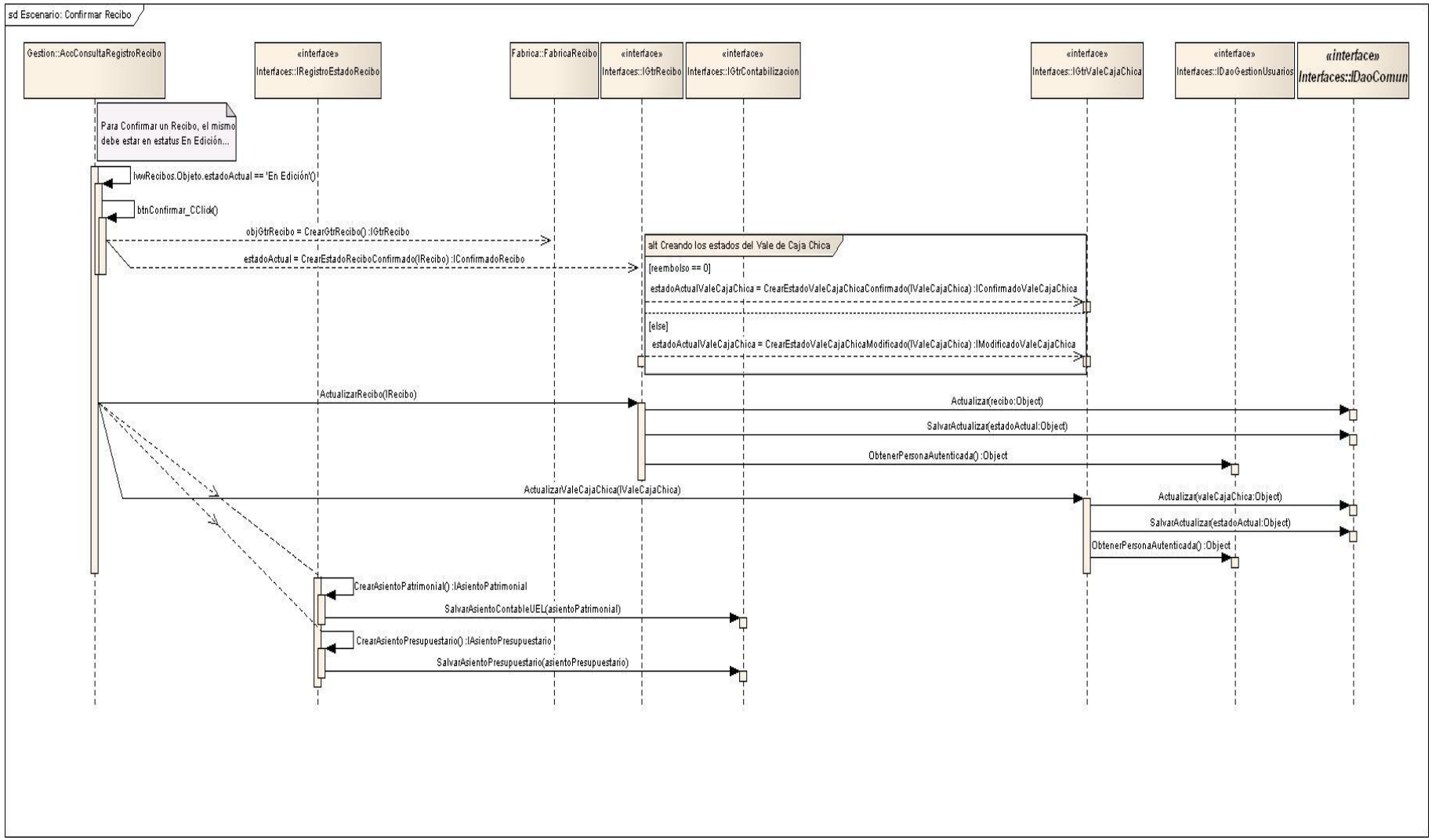

Figura 49 Diagrama de Secuencia del Escenario Confirmar Recibos.

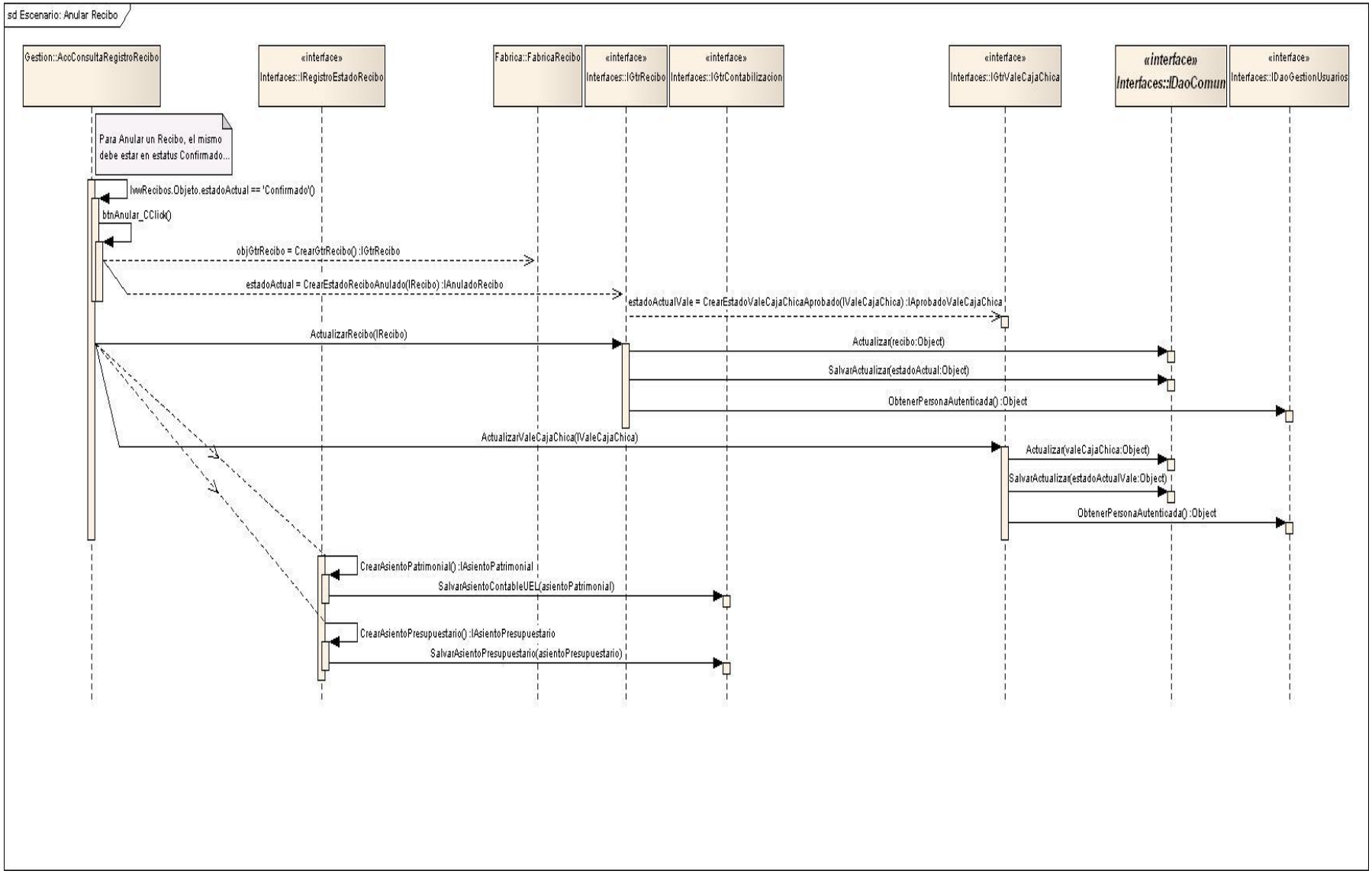

Figura 50 Diagrama de Secuencia del Escenario Anular Recibos.

Anexo 5. RCU Gestionar Modificaciones del Fondo de Caja Chica.

Diagrama de Clases del diseño, diagrama de transición de estados y diagrama de Interacción para el CU Gestionar Modificaciones del Fondo de Caja Chica. Este CU posibilita buscar, incorporar, modificar, eliminar, aprobar y confirmar modificaciones del fondo de caja chica.

A continuación se presenta el diagrama de transición de estado correspondiente al CU: Gestionar Modificaciones del Fondo de Caja Chica. Este muestra los diferentes estados por los cuales puede pasar la entidad Modificación.

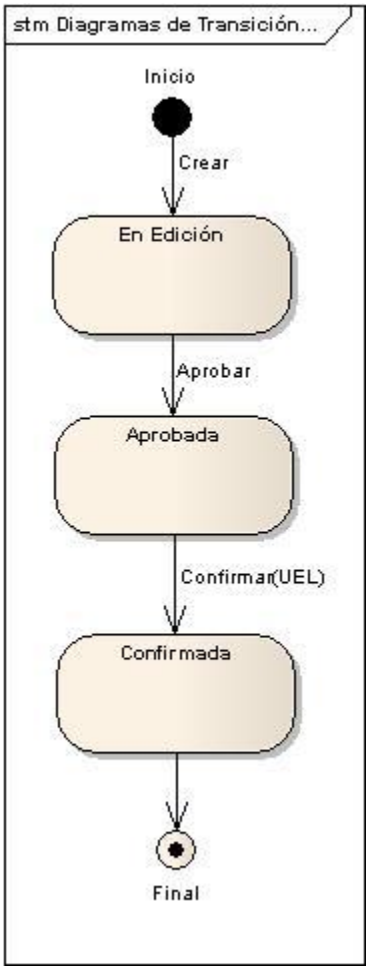

Figura 51 Diagrama de Transición de Estados de la Entidad Modificación.

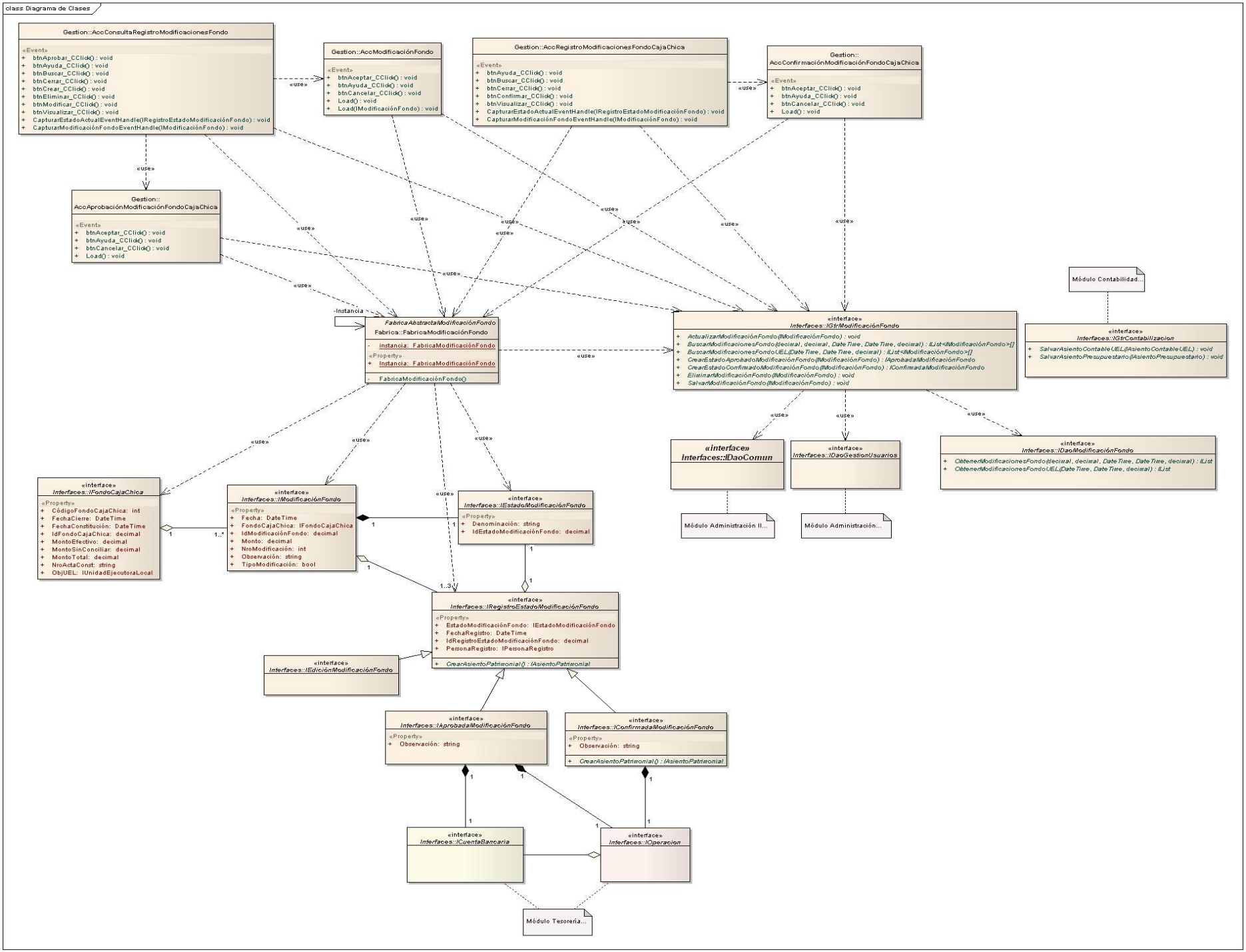

Figura 52 Diagrama de Clases de diseño CU: Gestionar Modificaciones del Fondo de Caja Chica.

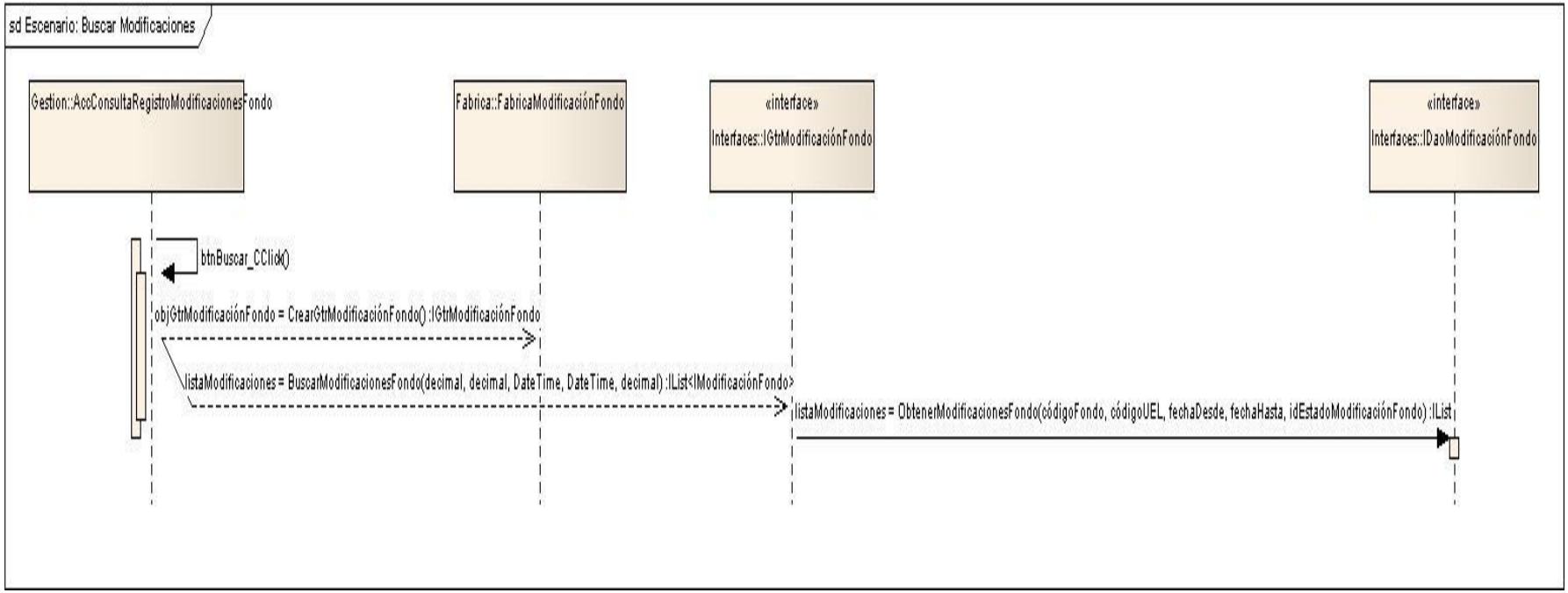

Figura 53 Diagrama de Secuencia del Escenario Buscar Modificaciones del Fondo de Caja Chica.

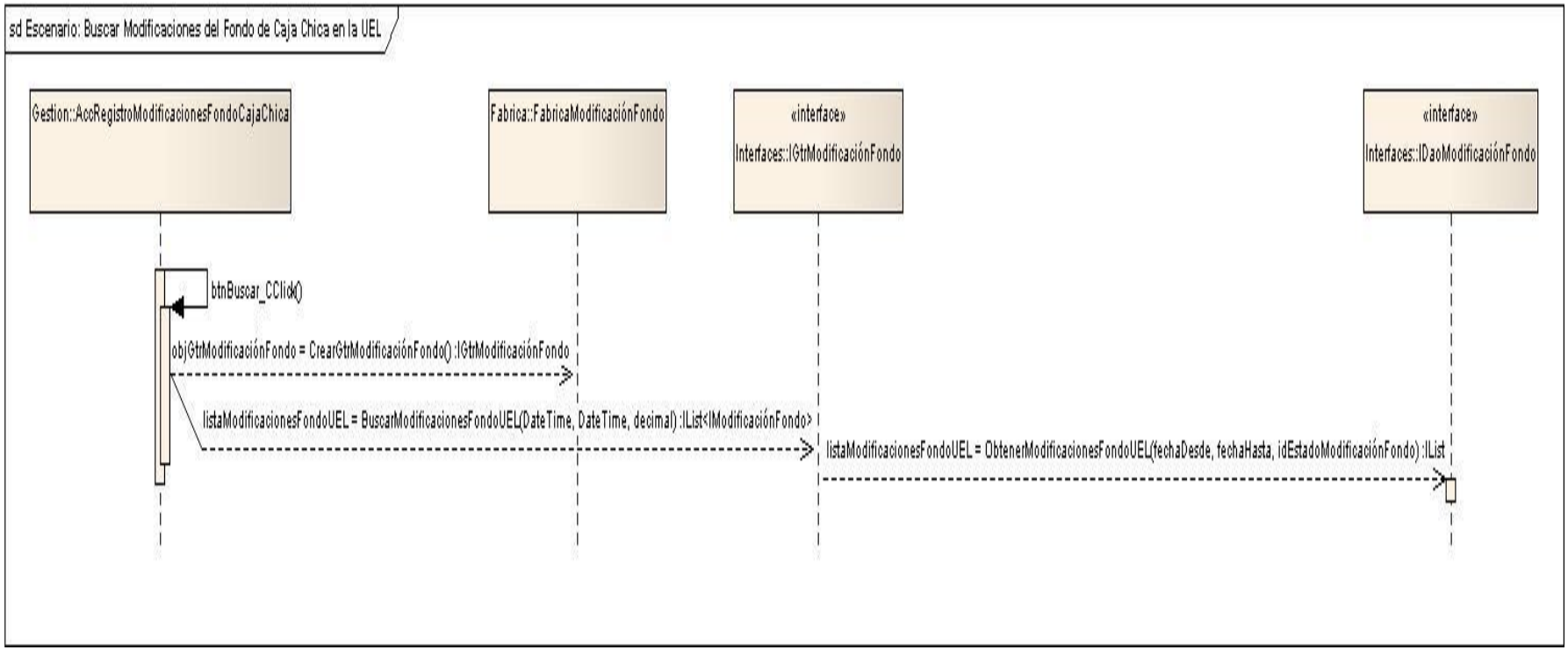

Figura 54 Diagrama de Secuencia del Escenario Buscar Modificaciones del Fondo de Caja Chica UEL.

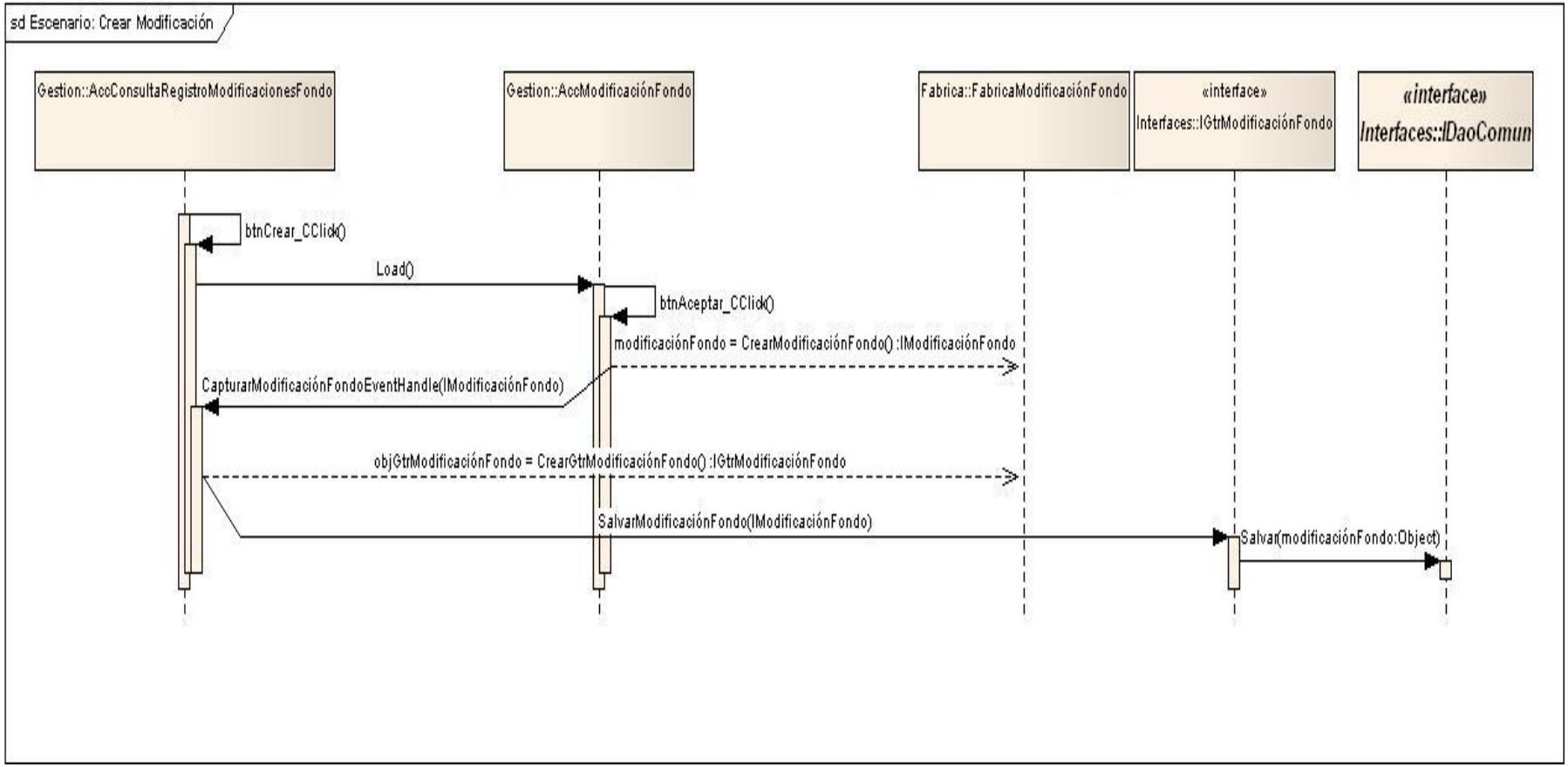

Figura 55 Diagrama de Secuencia del Escenario Crear Modificaciones del Fondo de Caja Chica.

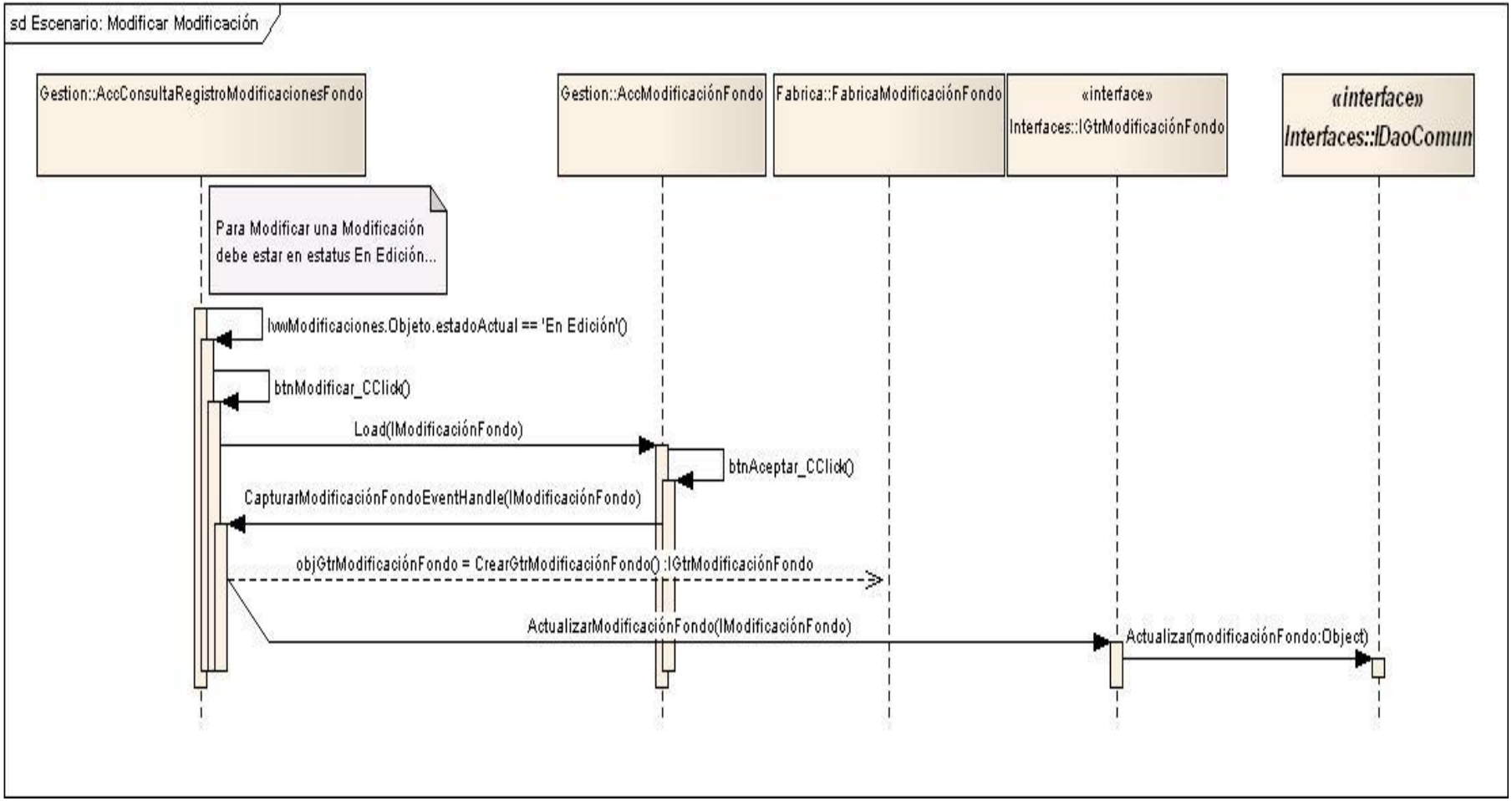

Figura 56 Diagrama de Secuencia del Escenario Modificar Modificaciones del Fondo de Caja Chica.

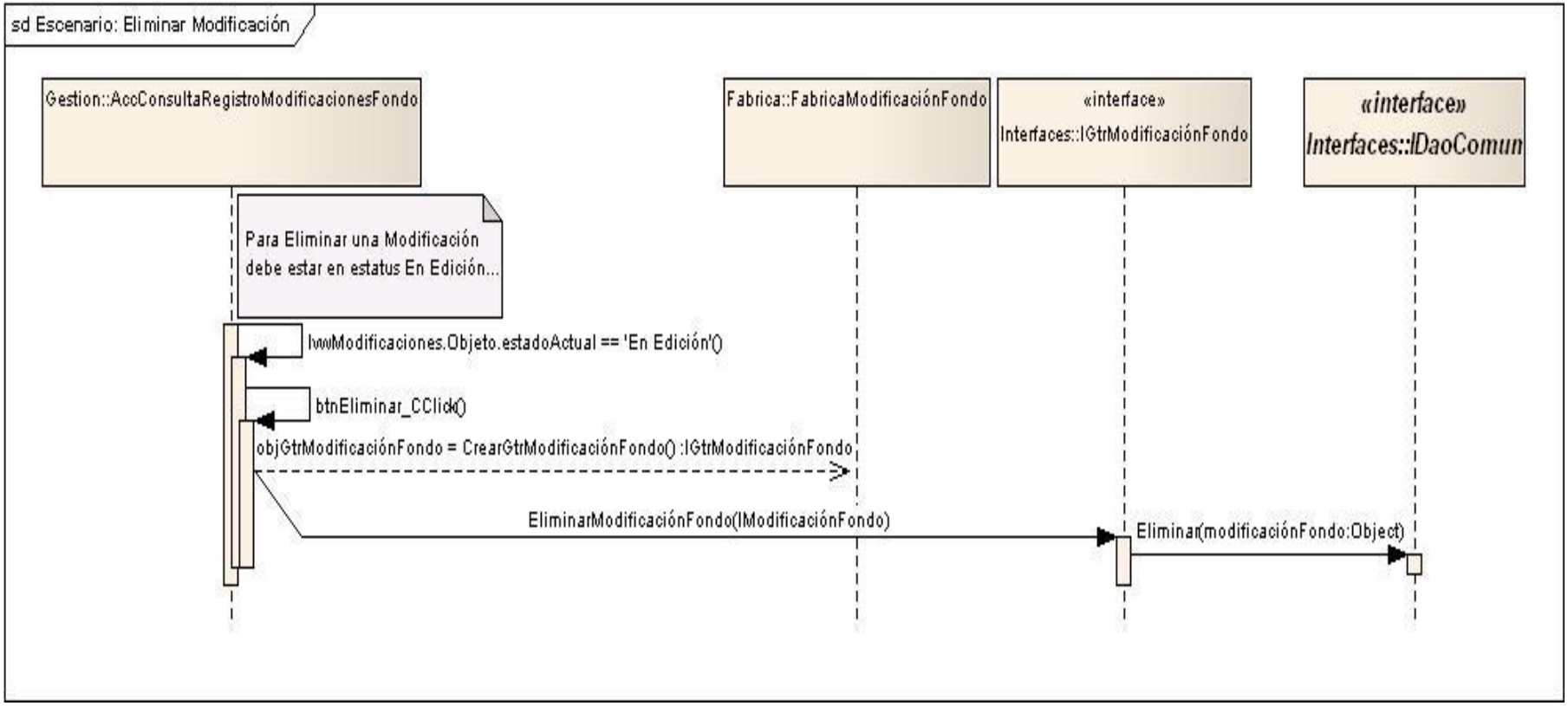

Figura 57 Diagrama de Secuencia del Escenario Eliminar Modificaciones del Fondo de Caja Chica.

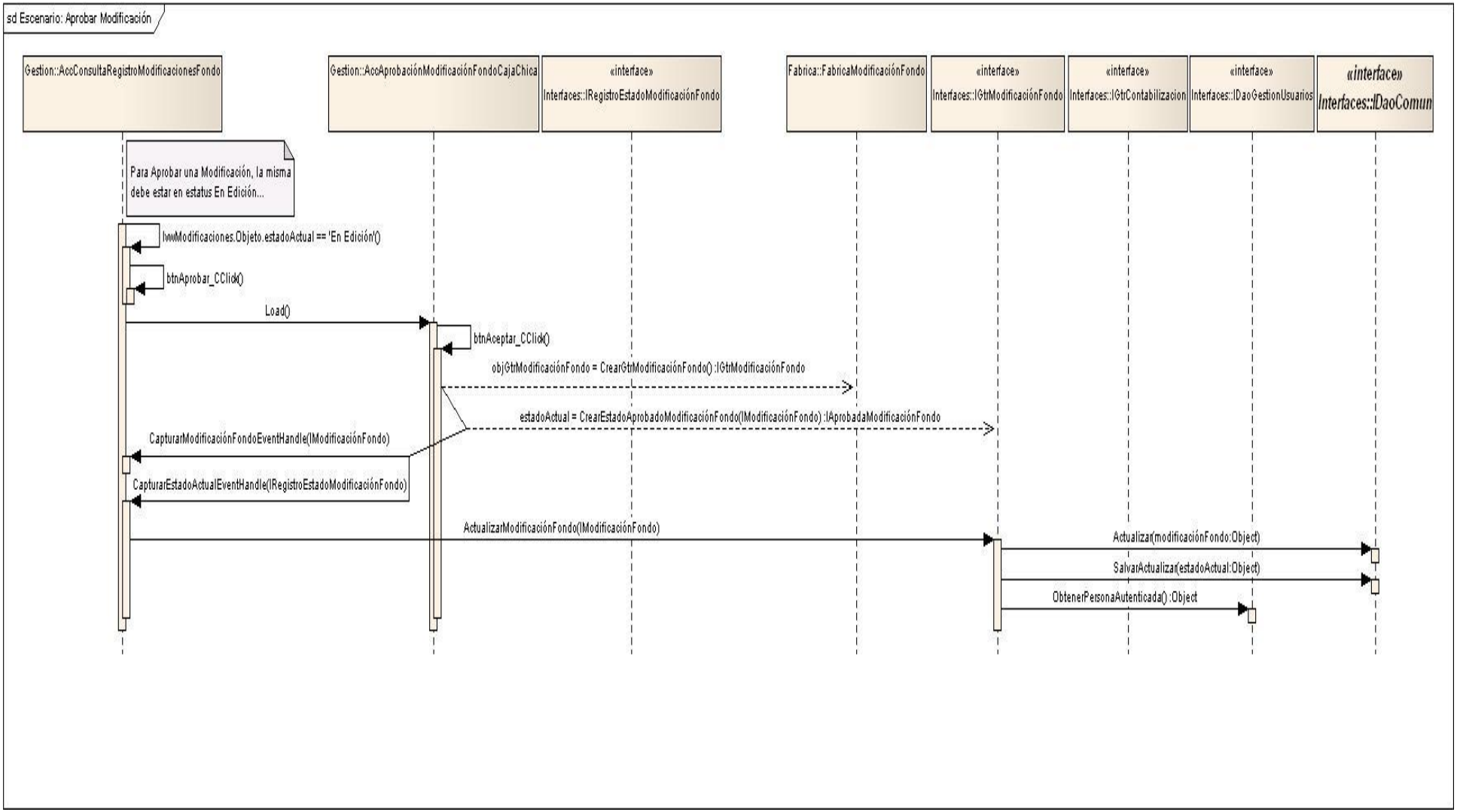

Figura 58 Diagrama de Secuencia del Escenario Aprobar Modificaciones del Fondo de Caja Chica.

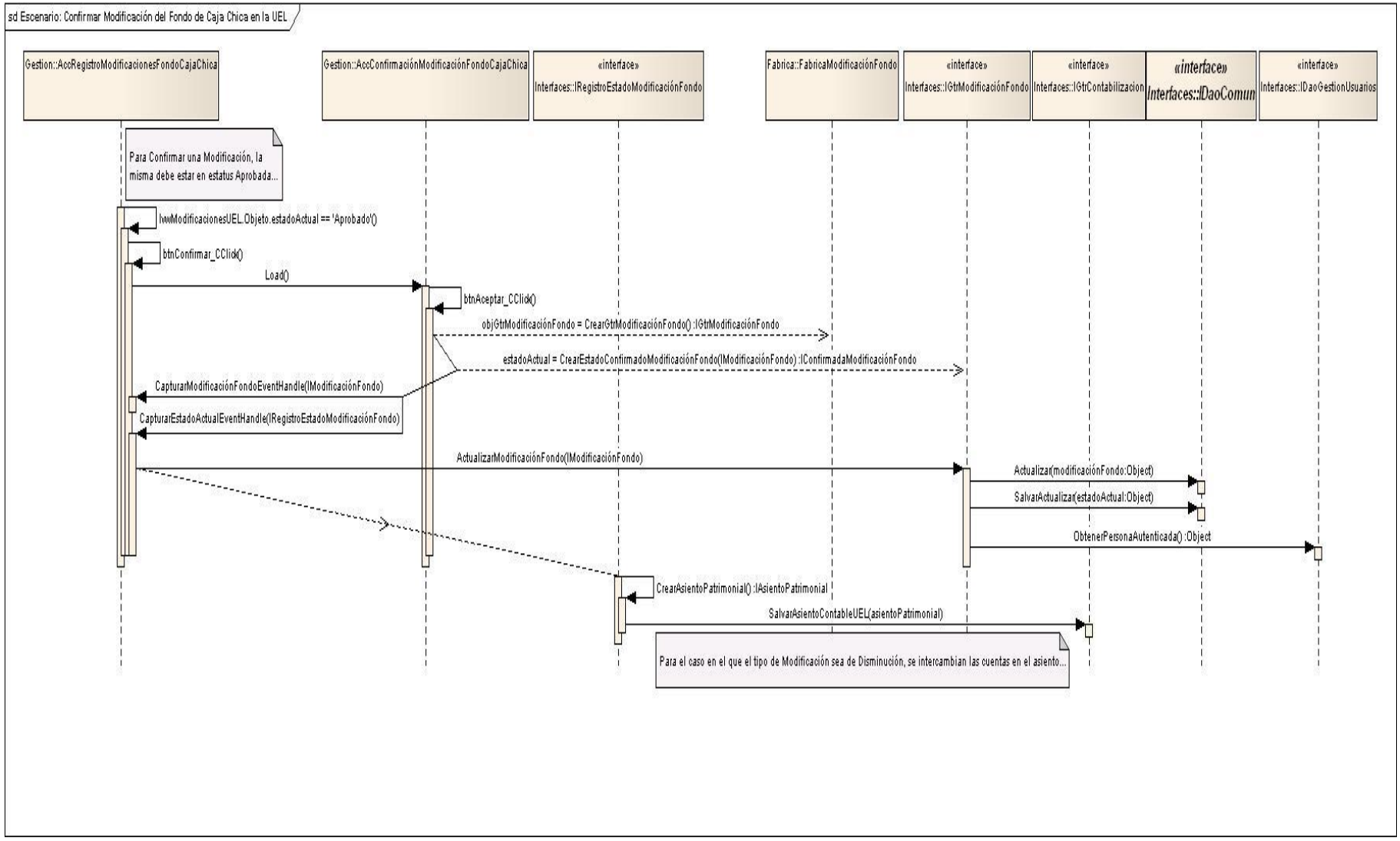

Figura 59 Diagrama de Secuencia del Escenario Confirmar Modificaciones del Fondo de Caja Chica.