UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 3

DISEÑO DEL SISTEMA INTEGRAL PARA EL CÁLCULO DE ÍNDICES DE PRECIOS AL CONSUMIDOR

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

AUTOR: REINEL MOLINA LUIS

TUTOR: LIC. MERLYN AVILÉS ESPINOSA

Ciudad de La Habana. Cuba

Junio de 2007


Hay ciertamente
dos cosas diferentes.
saber y creer que se sabe.
La ciencia consiste en saber.
en creer que se sabe
está la ignorancia
Hipócrates

Declaración de autoría.

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Reinel Molina Luis	Lic. Merlyn Avilés Espinosa
_____	_____
Autor	Tutora

Datos de contacto.

Datos del **tutor**.

Nombre: Merlyn Avilés Espinosa.
Área: Universidad de las Ciencias Informáticas. Facultad 3.
Cargo: Profesor.
Apartamento: 113102
Teléfono: 8372688
Correo: merlyna@uci.cu

Datos del **autor**.

Nombre: Reinel Molina Luis.
Área: Universidad de las Ciencias Informáticas. Facultad 3.
Apartamento: 11210
Teléfono: 8358707
Correo: rmolina@estudiantes.uci.cu

Agradecimientos

A **mis padres,** por su confianza y haberme guiado siempre por el buen camino, por todos sus consejos, por su cariño constante, por siempre estar “ahí” cuando los he necesitado. Ustedes son los mejores, a ustedes se lo debo todo.

A mi hermana **Ronaysi**, por ser lo que más quiero. Estoy seguro que algún día tú vas a logar este sueño también porque tienes todo el talento que se requiere.

A **mis abuelos,** no todos presentes físicamente. Gracias por el esfuerzo que han hecho para ayudarme y haberme enseñado tantas cosas buenas desde pequeño.

A **los amigos de mi tierra**, gracias por darme todo su apoyo, a pesar de la distancia en estos cinco años, siempre estuvieron conmigo en las buenas y en las malas; esos son los verdaderos amigos. A **los amigos de la universidad**, por compartir conmigo todo este tiempo y pasar momentos juntos que nunca voy a olvidar. Gracias por ayudarme cuando los necesité, y aunque nos separemos ahora, siempre los voy a recordar. A todos los que han sido **compañeros de estudio**, por la ayuda ofrecida y por haber tenido la suerte de compartir con ellos.

A mis primos, mis tíos, en fin, a toda **mi familia**, por preocuparse por mi y enseñarme el valor que tiene la familia.

A **mi tutora**, gracias por apoyarme y ayudarme en todo momento.

A **todos**, de corazón, muchas gracias.

Dedicatoria

A mís padres.

A mí famílía, por su confianza en mí.

A mís amígos.

A los que me han ayudado.

A los seres más queridos que han hecho

posíble la escritura de estas palabras.

Resumen

La existencia en Cuba de cuatro tipos de mercados: formal, informal, agropecuario y en divisas; hacen que la medición del crecimiento económico sea la más compleja en América Latina. La Oficina Nacional de Estadística (ONE) es la encargada de calcular los índices de precios de consumo, pero para ello cuenta con un software que no posee el grado de integración requerido para esta operación; no permite gestionar los productos de la canasta básica y no tiene sistema de seguridad. Cualquier persona puede entrar al sistema y obtener información confidencial.

El trabajo surge a partir de la necesidad de la ONE de realizar un nuevo sistema que sea seguro y que gestione la información relacionada con los productos en los diferentes mercados para calcular los índices de precios, dando solución a un problema de primer orden, que es la obtención de valores más reales de índices de precios con respecto a la actualidad en los mercados cubanos.

Este trabajo de diploma presenta el diseño para el Sistema Integral de Cálculo de Índices de Precios al Consumidor (IPC-ONE). Este sistema permitirá, además de calcular índices de precios, emitir tablas de precios y realizar operaciones de cambio sobre la canasta básica de productos. Con el desarrollo del modelo de diseño, se crearon las entradas apropiadas y un punto de partida para las actividades de implementación subsiguientes, capturando los requisitos o subsistemas individuales, interfaces y clases. Para el modelado visual de los artefactos que componen el modelo de diseño, la herramienta CASE utilizada fue Rational Rose en su versión Enterprise Edition 2003.

Palabras claves.

Diseño de sistemas, modelo de diseño, subsistema de diseño, clases de diseño, patrones de diseño, Rational Rose, artefactos, índice de precios, IPC.

Abstract

There are in Cuba four types of markets: formal, informal, agricultural and foreign currencies. They make that the measure of the economic growth becomes the most complex in Latin American. The National Office of Statistic (NOS), it's the one in charge of calculating the indexes of consumption prices, but they use a software that don't have the integration grade required for this operation; it allow operating the products of the basic basket, it didn't also have a system of security. Any person could enter to the system and to obtain confidential information.

This investigation work arises starting from the necessity of carrying out a new system that negotiates the information related with the products in the different markets to calculate the indexes of prices, giving solution to a problem of first order, that is the obtaining of real values of indexes of prices with regard to the present time in the Cuban markets.

The objective of this diploma work is to carry out the design of the Calculator System of Indexes of Prices to the Consumer. This system will allow, besides calculating indexes of prices, to emit charts of prices and to carry out operations of change on the basic basket of products. If the design model is developed, one will be able to create an appropriate entrance and a starting point for the subsequent implementation activities, capturing the requirements or individual subsystems, interfaces and classes. For the visual modeling of devices that compose the design model, was used the CASE tool Rational Rose with UML support, in its version Enterprise Edition 2003.

Tabla de contenidos.

Introducción

El perfeccionamiento empresarial y el desarrollo de las tecnologías de la información y las comunicaciones (TICs), son dos procesos que han avanzado juntos de la mano en los últimos años. Las empresas han tenido la necesidad de automatizar los procesos que realizan y los servicios que brindan para agilizar el tiempo de desarrollo y de esta forma aumentar la productividad y la calidad de los resultados. Nuestro país está inmerso en esta revolución de la información y ha optado por el desarrollo de la informática.

La creación de la Universidad de las Ciencias Informáticas (UCI), ha sido uno de los mayores pasos que se han dado en el desarrollo tecnológico para la informatización de la sociedad cubana. Esta institución tiene la alta responsabilidad de expandir el proceso de automatización empresarial a todo el país, además de la formación de profesionales altamente calificados para la producción de aplicaciones informáticas. Para ello cuenta con tecnología de última generación y un sin número de servicios telemáticos que facilitan el trabajo docente y productivo.

La Oficina Nacional de Estadísticas de la República de Cuba (ONE) es una de las empresas que ha necesitado el apoyo de la UCI para mejorar y automatizar sus servicios y operaciones con vista a responder eficazmente a la toma de decisiones.

La **situación problemática** viene dada por la ineficiencia que presenta el sistema informático que se usa en la ONE para calcular los índices de precios de consumo, el cual es ineficaz cuando trabajaba con grandes volúmenes de información y no cuenta con las funcionalidades básicas para poder manipular de manera eficiente los datos requeridos para calcular dichos índices. Es por eso que nos sumamos a la tarea, un pequeño grupo de desarrollo de la UCI, de realizar un nuevo sistema para calcular los índices de precios que cumpla con las necesidades del cliente y que elimine los problemas anteriormente mencionados.

A partir de la problemática analizada se puede formular el siguiente **problema científico:** ¿Cómo realizar un diseño cohesivo, poco complejo y que cumpla los requisitos funcionales para el sistema de cálculo de índices de precios, y que brinde información detallada para facilitar los trabajos de implementación?

El **objeto de estudio** de la presente investigación es el proceso de diseño de sistemas informáticos.

El **objetivo general** de este trabajo de diploma es realizar el diseño del sistema que de cumplimiento a los requisitos del problema para el Sistema de Cálculo de Índices de Precio al Consumidor. Para poder lograr este objetivo, es necesario dar cumplimiento a los **objetivos específicos** siguientes: a) dividir el modelo de diseño en piezas más manejables para su mejor entendimiento y organización, haciendo uso de patrones de diseño para garantizar un bajo acoplamiento, una alta cohesión y una mayor flexibilidad del diseño propuesto; b) generar los diferentes artefactos definidos para el modelo de diseño y, c) aplicar sistemas de medición al diseño para determinar en cierto grado la eficiencia del mismo.

El **campo de acción** que abarca este trabajo de diploma es la elaboración del diseño del sistema de cálculo de índices de precios al consumidor.

La **hipótesis** planteada parte de que con la realización un diseño poco complejo, cohesivo y bien detallado del sistema de cálculo de índices de precios de consumo, se podrá disminuir la complejidad de los trabajos de implementación del sistema y las pruebas.

Para lograr el cumplimiento de la hipótesis anteriormente planteada, se hizo necesario llevar a cabo un conjunto de **tareas específicas de la investigación**, las cuales se exponen a continuación:

a) Realizar un estudio de sistemas similares para analizar su estructura, tomar ideas de los mismos que puedan ser útiles para nuestro diseño.

b) Realizar un análisis de los requisitos funcionales y estudio profundo de la descripción de los casos de uso definidos por el analista.

c) Realizar un estudio de patrones de diseño para seleccionar los que puedan aplicarse en la solución propuesta.

d) Estudio y valoración de la arquitectura propuesta para el desarrollo del sistema.

e) Estudio profundo de las distintas métricas que existen para medir la calidad del diseño de software y selección de las que se van a aplicar para medir nuestro diseño.

f) Análisis de las potencialidades de la herramienta seleccionada para la elaboración del modelo de diseño.

Para la realización del diseño del sistema IPC-ONE se requiere previamente adquirir una gran comprensión minuciosa y detallada de los aspectos relacionados con los requisitos funcionales y con las especificaciones de los casos de uso. Es por eso que fue necesario poner en práctica el uso de **métodos y procedimientos de investigación científica,** para llegar a comprender el desarrollo del objeto.

Se han puesto en práctica varios métodos teóricos de investigación científica, entre ellos el método histórico, donde se analizó toda la trayectoria concreta del diseño de sistemas software, viendo cada uno de los componentes y herramientas que se relacionan con el mismo, como por ejemplo el diseño orientado a objetos y sus propósitos, los patrones de diseño, UML y la herramienta Rational Rose. A su vez se estudiaron algunos sistemas informáticos con características similares al que se diseña en este trabajo de diploma, con el fin de conocer su estructura, funcionamiento, y otros aspectos importantes.

Se empleó además el método sistémico, considerado uno de los más importantes, con el fin de poder dividir e interpretar todos los elementos contenidos dentro del diseño del sistema, así como la relación jerárquica que existe entre ellos, como son clases, objetos, colaboraciones, etc.

El **resultado esperado** con el desarrollo de este trabajo es el diseño del sistema ONE-IPC. Este diseño debe ser capaz de brindar información detallada a los demás desarrolladores del sistema con el fin de hacer más fáciles las tareas de implementación, y debe cumplir con los requisitos funcionales.

El contenido de este trabajo de diploma está estructurado de la siguiente manera:

Capítulo 1: "Fundamentación teórica". En este capítulo se realiza un estudio del estado del arte acerca de la actualidad de los sistemas informáticos estadísticos. Se hace una introducción al proceso de diseño de software. Se hace una panorámica sobre el tema de patrones de diseño y se fundamenta la utilización de la herramienta que fue seleccionada para elaborar la solución propuesta.

Capítulo 2: "El modelo de diseño propuesto". En este capítulo es donde se expone la solución propuesta. Primeramente se hace una valoración de la arquitectura definida para el desarrollo, y luego se plantean y analizan los requisitos funcionales, así como los casos de uso definidos por el analista. Finalmente se exponen cada uno de los artefactos generados en la etapa de diseño. El primero de estos artefactos son los subsistemas de diseño, donde se explica claramente la funcionalidad que tienen dentro del sistema y qué se logra con cada uno de ellos. El segundo artefacto son las clases del diseño, donde se expone una descripción detallada de cada clase, especificando sus atributos y métodos. El tercer artefacto corresponde a las realizaciones de los casos de uso-diseño, las cuales engloban los diagramas de clases y los diagramas de interacción, sumados a ellos una breve descripción textual que brinda un seguimiento del flujo de información a seguir en cada caso.

Capítulo 3: "Evaluación del modelo de diseño propuesto". En este capítulo final, como parte primera, se explican cada una de las métricas técnicas que se van a usar para medir la calidad del diseño propuesto. Como segundo paso, se aplican estas métricas y se analizan los resultados obtenidos durante la medición.

Capítulo 1 “Fundamentación teórica”.

1.1 Introducción

Este capítulo constituye una presentación del concepto de diseño de sistemas, y de la amplia serie de principios y propósitos que lo caracterizan como un proceso. Al mismo tiempo se expone un poco de historia del arte y datos relacionados con el uso y aplicaciones de los patrones de diseño, el lenguaje unificado de modelado y el proceso de diseño orientado a objetos, ya que constituyen elementos sobre los cuales se va a desarrollar nuestro diseño. Además, se valoran algunos ejemplos de sistemas funcionalmente similares al que se va a diseñar y se realiza un breve estudio sobre las potencialidades de la herramienta informática Rational Rose, propuesta para la realización de nuestro diseño.

1.2 Índice de Precios al Consumidor.

En esta sección se realiza un análisis sobre cómo ha evolucionado el IPC en Cuba y de la situación actual de algunos sistemas informáticos estadísticos utilizados para calcular índices de precios, a partir de los cuales se realiza una valoración y se toman ideas para el desarrollo de nuestro diseño.

IPC es la abreviatura de *Índice de Precios al Consumidor*, *Índice de Precios de Consumo* o *Índice de Precios al Consumo* (CPI en inglés).

Es un índice de precios el valor numérico que representa una medida relativa de diferencia de precios de bienes y servicios en diferentes situaciones temporales. Esto es, medir las variaciones de precios de una muestra de productos, (conocida como "canasta" o "cesta") con respecto a un período base determinado. De esta forma se pretende medir, mensualmente, la evolución del nivel de precios de bienes y servicios de consumo en un país. El objetivo es medir las diferencias de nivel de precios de una canasta con respecto a una región determinada. (ONE 2000)

1.2.1 Índice de Precios al Consumidor en Cuba.

En Cuba, una serie de sucesos trajeron consigo un cambio brusco en la utilización del IPC hace ya más de diez años. Uno de ellos fue la circulación de dos monedas, (el peso cubano y el dólar). Fue necesario establecer una taza de cambio para realizar estimaciones y entonces surgió un nuevo mercado, el Mercado en Divisas. Otro suceso que impactó en la medición económica fue la crítica situación económica que atravesó Cuba en la década del 90, donde, como estrategia del gobierno para aliviar esta situación, se creó el Mercado Informal, formado por los llamados trabajadores por cuenta propia. En la actualidad, la canasta de bienes y servicios data del año 1999, y Cuba cuenta hoy con cuatro tipos de mercado, incluyendo una taza de cambio que varía de CUC a Moneda Nacional. Estas características, hacen que la medición del crecimiento económico en Cuba sea una de las más complejas en toda América Latina. (ONE 2000)

1.2.2 Software estadístico. Tendencias actuales.

La creación de software para cálculos estadísticos ha sufrido un gran aumento en los últimos años. Este crecimiento se debe al incremento proporcional de las empresas en el mundo. El perfeccionamiento empresarial a nivel mundial constituye una constante evolución, y de ese modo, las organizaciones necesitan de una automática evaluación de todos los aspectos implicados en sus procesos. Precisamente los sistemas estadísticos permiten acceder a un mejor conocimiento de la información contenida en los datos mediante metodologías y procesos de recogida, análisis e interpretación. La evolución del software estadístico en los últimos años ha significado un importante ahorro en tiempo, en precisión y en calidad de los resultados en los procesos de las empresas.

En la actualidad, la mayoría de los sistemas estadísticos presentan funcionalidades que permiten el trabajo con indicadores visuales capaces de mostrar al usuario resultados en gráficas, de una forma más amigable. Esta característica la presentan con mayor frecuencia los sistemas que abarcan la estadística descriptiva, o sea, los que trabajan con modelos matemáticos, los que calculan estimaciones, etc.

Sin embargo, muchos de los sistemas que se usan para calcular IPC han evolucionado poco dentro del campo estadístico. Esto se debe a que la metodología del IPC no ha variado ni ha sufrido cambios en los últimos 15 años, pero eso si, cada país tiene una forma independiente de recolectar los datos, de procesar las encuestas, entre otros procesos. A continuación se analiza uno de los sistemas utilizados para calcular IPC.

Procesamiento y Difusión del IPC, Perú. (ALFARO 2001).

El sistema fue desarrollado en lenguaje FoxPro en 1998, actualmente se está migrando a Visual Studio .NET. Consta de los siguientes subsistemas:

1. SUB-SISTEMA DE ENTRADA DE DATOS INTELIGENTE

2. SUB-SISTEMA DE CONSISTENCIA ANALÍTICA

3. SUB-SISTEMA DE CÁLCULO

4. SUB-SISTEMA DE RESULTADOS.

Para la elaboración del IPC, se utilizan los cuatro subsistemas mencionados anteriormente. Pero, específicamente para el procesamiento automático se usa el Subsistema de Cálculo.

Ejecuta el algoritmo matemático del cálculo del índice, produciendo aproximadamente 350 mil operaciones matemáticas.

Para el proceso de cálculo el sistema usa aproximadamente 150 Mega Bytes de almacenamiento en el servidor de la base de datos.

Esta etapa es realizada, cuando se completa el período de captación y el conjunto de información que componen la canasta.

El cálculo se realiza con la información de tres sub-bases de datos: Base de Datos de Mercados, Base de Datos de Casas Comerciales y Base de Datos de Vivienda y para los diferentes niveles de desagregación del Índice General.

El Sistema de Cálculo se realiza mediante procedimientos que en el caso del IPC implican 800 instrucciones.

Este cálculo permite la generación de las siguientes sub-bases de datos o tablas:

1. Sub-Base de Precios Promedio Mensuales de Productos Alimenticios, adquiridos en los mercados y automercados.

2. Sub-Base de Precios Promedio Mensuales de Precios de Productos de Establecimientos Comerciales.

3. Sub-Base de Precios Promedio Mensuales de Precios de Alquiler de Vivienda.

4. Sub-Base de Números Índices.

5. Sub-Base de Variaciones Porcentuales de Precios.

El subsistema de resultados es el encargado de emitir todas las tablas con los cálculos realizados.

La estructura de estos subsistemas es adecuada, teniendo en cuenta las funcionalidades que debe tener el sistema. Cada uno de ellos abarca un conjunto de características que permiten de manera independiente dar solución a determinadas tareas. Podemos tomar de este sistema algunas ideas que pueden ser útiles para la elaboración de nuestro diseño, ya que existe cierta semejanza entre la funcionalidad de ambos, específicamente en el subsistema de resultados.

Haciendo un análisis más profundo de este ejemplo, podemos decir que carece de un subsistema de seguridad. Esta misma característica la presenta el sistema que se usa en la ONE. Otra deficiencia que

presenta este sistema, es que no permite a través de ninguno de los subsistemas anteriormente mencionados manipular los datos de la canasta fija. Esta deficiencia trae consigo no poder actualizar datos de manera eficiente, y también la presenta el sistema usado en la ONE. Estas funcionalidades constituyen una necesidad en el diseño que resolveremos en el próximo capítulo, ya que el sistema informático utilizado en la ONE carece de las mismas. Por otra parte, no sería conveniente para nuestro sistema la división en sub-bases de datos como lo hace este software, para nuestro proyecto es más adecuado usar una base de datos global, que abarque todas las entidades y sus relaciones, lo cual haría mucho más rápidos los procesos de consulta y manipulación de información persistente.

Sistema para Calcular Índices de Precios al Consumidor. Módulo Nacional. Cuba.

El sistema fue programado en Microsoft FoxPro versión 2.6 para Windows 98 en el año 1999.

Es importante aclarar que este el sistema que actualmente se usa en la ONE, será reemplazado por el nuevo sistema IPC-ONE que diseñaremos en este trabajo de diploma. Esto trae consigo una crítica y valoración sobre este sistema para capturar deficiencias y otros aspectos que pueden ser útiles para la construcción de nuestro diseño.

Para realizar un análisis crítico de este sistema, nos apoyaremos en la descripción que hace (FERNANDEZ 1999) sobre este sistema.

El sistema automatizado, está compuesto por un módulo para las provincias y otro nacional, teniendo en cuenta que las tareas a realizar son diferentes para cada módulo. En la ONE se fusionan los datos de provincia, se procesan, se emiten las tablas de precios, posteriormente se calculan los índices y se emiten las tablas de índices. El sistema trabaja a base de ficheros .dbf, lo cual viene ya constituyendo una deficiencia porque el esfuerzo que se requiere para cargar y salvar en ficheros de este tipo es mucho mayor que si trabajara conectado directamente a una base de datos local. De aquí se deduce también el problema de la seguridad de la información, los ficheros se guardan en el directorio donde se encuentra el sistema, y cualquier individuo puede acceder a ese directorio y obtener información confidencial.

Los procesos que se llevan a cabo en el módulo nacional son:

1. Fusión Nacional y Chequeo de Estacionalidad.

Este proceso siempre es el primero en ejecutarse. Se fusionan todos los ficheros recibidos de provincia, por cada tipo de mercado, en un fichero nacional. Este proceso se vuelve muy complicado; el sistema tiene que cargar al menos unos cuatro ficheros por cada provincia y fusionarlos en uno solo. Ese fichero resultante es muy probable que contenga información repetida, o sea, puede ser que el producto X de la provincia Y también exista en la provincia F y con el mismo precio que tenía en Y, y esto trae consigo que esa información esté en el nuevo fichero tantas veces aparezca en cada provincia. Este problema de redundancia de información puede ser eliminada si se implementa una base de datos.

2. Cálculo de las Variables.

El proceso consiste en calcular mensualmente por tipo de mercado, las variables: media aritmética, precio modal, mediana, entre otras, por cada producto. La deficiencia de este proceso está en que por cada mercado que se trabaje, se genera un fichero nuevo que amacena los datos calculados, y a los efectos de la información, se almacenan en un directorio de la aplicación de manera persistente, y es probable que el espacio ocupado en disco aumente considerablemente cada mes.

3. Ajuste de los Datos.

Este proceso se basa en las tablas de ajuste, que son tablas seriadas de precios de los últimos doce meses que permiten hacer comparaciones de precios. El problema está en que “esas comparaciones” se tienen que hacer de forma manual, con la vista y en realidad el personal que trabaja el sistema pasa mucho trabajo para realizar este proceso.

4. Emisión de las Tablas de Precios.

En este proceso se visualizan las tablas de precios en pantalla, las cuales contienen los resultados de los cálculos de las variables. Un servicio bueno que brinda este proceso y que podemos tomar en consideración para nuestro diseño es la opción que tiene de imprimir y poder salvar las tablas en diferentes tipos de formato, como .pdf, y .doc

5. Cálculo de los Índices de Precios.

En el proceso se realiza mensualmente el cálculo de los índices de precios para los diferentes mercados. Este proceso posiblemente sea el más complicado de todos por la cantidad de ficheros que utiliza. Se necesita de siete ficheros extremadamente grandes para esta operación, y al final, se generan cuatro ficheros nuevos con los resultados obtenidos. Este proceso puede mejorar para nuestro sistema si calculamos los índices en tiempo de ejecución y no los tomamos como datos persistentes.

6. Emisión de las tablas de Índices.

Este proceso es muy similar al de emitir tablas de precios, la diferencia está en el tipo de datos que se emiten en las tablas. También brinda las posibilidades de impresión y salvamento anteriormente mencionadas.

7. Procesamiento de la ESEH.

Este proceso es el encargado de capturar los datos de la encuesta base del año 1999, la cual se encuentra ubicada en uno de los ficheros del sistema. Estos datos se usan para el cálculo del IPC. El proceso no es complejo, pero sí debería brindar funcionalidades para poder insertar datos o eliminar datos de la encuesta, siempre teniendo en cuenta que la encuesta del 1999 no será eterna; en cualquier momento se necesitará trabajar con una nueva encuesta. Creo que este último aspecto lo podemos considerar para nuestro diseño.

8. Nomencladores.

Este proceso brinda las funcionalidades de listar y de ordenar los diferentes clasificadores que existen. Los nomencladores son: para mercado, para provincia y para producto. Para nuestro diseño podemos tener en cuenta las funcionalidades básicas de gestionar los clasificadores, o sea, poder insertar, eliminar, modificar clasificadores de cada tipo de nomenclador. Esta funcionalidad es muy importante, ya que uno de los problemas más graves que tiene el personal de la ONE es que no pueden eliminar de la canasta básica algunos productos que ya dejaron de circular en los mercados cubanos, y otros que no han podido ser agregados a la canasta.

1.3 Introducción al proceso de diseño de software.

En el contexto de la Ingeniería de Software el diseño se centra en cuatro áreas importantes de interés: datos, arquitectura, interfaces y componentes. En estas cuatro áreas se aplican los principios y conceptos que se abarcan en esta sección.

El diseño de software se encuentra en el núcleo técnico de la ingeniería de software y se aplica independientemente del modelo de diseño de software que se utilice. Una vez que se analizan y especifican los requisitos del software, el diseño es la primera de las tres actividades técnicas, (diseño, implementación y pruebas), que se requieren para construir y verificar el software. (PRESSMAN 2005)

El diseño de software es un proceso iterativo mediante el cual los requisitos se traducen en un plano para construir el software. Inicialmente, el plano representa una visión holística del software. El diseño se representa a un nivel alto de abstracción, un nivel que puede rastrearse directamente hasta conseguir el objetivo del sistema específico y según unos requisitos más detallados de comportamiento, funcionalidades y de datos. A medida que ocurren las iteraciones del diseño, el refinamiento subsiguiente conduce a representaciones de diseño a niveles de abstracción mucho más bajos. Estos niveles se podrán rastrear aún según los requisitos, pero la conexión es más sutil. (JACOBSON 2004)

1.3.1 Propósitos del diseño de software.

En el diseño se modela el sistema y se encuentra la forma que soporte todos los requisitos, incluyendo los requisitos no funcionales. Una entrada esencial en el diseño es el resultado del análisis, esto es, el modelo de análisis. El modelo de análisis proporciona una comprensión detallada de los requisitos. Y lo que es más importante, impone una estructura del sistema que se debe conservar lo más fielmente posible a la vez que el sistema adquiera una forma.

Jacobson y sus colaboradores (JACOBSON 2004), definen un conjunto general de propósitos del diseño:

- Adquirir una comprensión en profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, tecnologías de interfaz de usuario, etc.

- Crear una entrada apropiada y un punto de partida para actividades de implementación subsiguientes capturando los requisitos o subsistemas individuales, interfaces y clases.

- Descomponer los trabajos de implementación en partes más manejables que pueden ser llevadas a cabo por diferentes equipos de desarrollo, teniendo en cuenta la posible concurrencia.

- Crear una abstracción sin costuras de la implementación del sistema, en el sentido de que la implementación es un refinamiento directo del diseño que rellena lo existente sin cambiar la estructura. Esto permite la utilización de tecnologías como la generación de código y la ingeniería de ida y vuelta entre el diseño y la implementación.

1.3.2 Los principios del diseño de software.

Los principios básicos de diseño hacen posible que el ingeniero del software navegue por el proceso de diseño. Se han sugerido un conjunto de principios para el diseño del software en los últimos 10 años,

estos principios han sido adaptados y ampliados en otros trabajos, (PRESSMAN 2005), y se muestran en la lista siguiente:

- *En el proceso de diseño no deberá utilizarse orejeras.* Un buen diseñador deberá tener en cuenta enfoques alternativos, juzgando todos los que se basan en los requisitos del problema, los recursos disponibles para realizar el trabajo y los conceptos de diseño presentados en la próxima sección.

- *El diseño deberá poderse rastrear.* Es necesario tener un medio de rastrear cómo se han satisfecho los requisitos por el modelo de diseño.

- *El diseño no deberá inventar nada que ya esté inventado.* Los sistemas se construyen utilizando un conjunto de patrones de diseño. Estos patrones deberán elegirse siempre como una alternativa para reinventar.

- *El diseño deberá minimizar la distancia intelectual entre el software y el problema como si de la misma vida real se tratara.* Es decir, la estructura del diseño del software (siempre que sea posible) imita la estructura del dominio del problema.

- *El diseño no es escribir código, y escribir código no es diseñar.* Incluso cuando se crean diseños procedimentales para componentes de programas, el nivel de abstracción del modelo de diseño es mayor que el código fuente. Las únicas decisiones de diseño realizadas a nivel de codificación se enfrentan con pequeños datos de implementación que posibilitan codificar el diseño procedimental.

- *El diseño deberá evaluarse en función de la calidad mientras se va creando, no después de terminarlo.* Para ayudar al diseñador en la evaluación de la calidad se dispone de conceptos de diseño.

- *El diseño deberá revisarse para minimizar los errores conceptuales (semánticos).* A veces existe la tendencia de centrarse en minucias cuando se revisa el diseño, olvidándose del bosque por culpa

de los árboles. Un equipo de diseñadores deberá asegurarse de haber afrontado los elementos conceptuales principales antes de preocuparse por la sintaxis del modelo del diseño.

1.3.3 El papel del diseño en el ciclo de vida del software.

El diseño es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción. Esto contribuye a una arquitectura estable y sólida y a crear un plano del modelo de implementación.

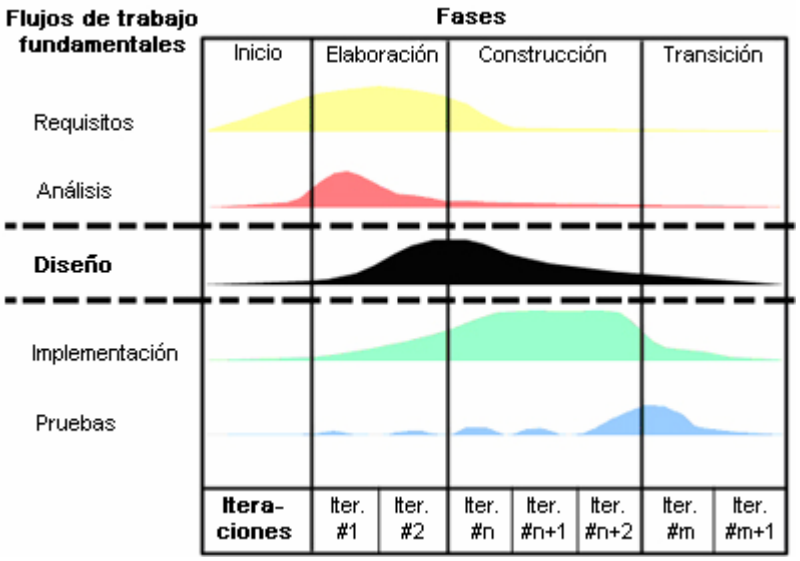

Figura 1. El Flujo de Trabajo Diseño.

Más tarde, durante la fase de construcción, cuando la arquitectura es estable y los requisitos están bien entendidos, el centro de atención se desplaza a la implementación.

No obstante, el modelo de diseño está muy cercano al de implementación, lo que es natural para guardar y mantener el modelo de diseño a través del ciclo de vida completo del software. Esto es especialmente cierto en la ingeniería de ida y vuelta, donde el modelo de diseño se puede utilizar para visualizar la implementación y para soportar técnicas de programación gráfica. (JACOBSON 2004)

1.3.4 Conceptos básicos del diseño de software.

Durante las últimas cuatro décadas se ha experimentado la evolución de un conjunto de conceptos fundamentales de diseño de software. Aunque el grado de interés en cada concepto ha variado con los años, (PRESSMAN 2005) hace una redefinición de cada uno de ellos con el objetivo de ayudar al ingeniero del software en la partición de componentes.

Abstracción.

Cuando se tiene en consideración una solución modular a cualquier problema, se pueden exponer muchos *niveles de abstracción.* En el nivel más alto de abstracción, la solución se pone como una medida extensa empleando el lenguaje del entorno del problema. En niveles inferiores de abstracción, se toma una orientación más procedimental. La terminología orientada a problemas va emparejada con la terminología orientada a la implementación en un esfuerzo por solucionar el problema. Finalmente, en el nivel más bajo de abstracción, se establece la solución para poder implementarse directamente. (PRESSMAN 2005).

***Refinamiento*.**

El refinamiento verdaderamente es un proceso de *elaboración.* Se comienza con una sentencia de función (o descripción de información) que se define a un nivel alto de abstracción. Esto es, la sentencia describe la función o información conceptualmente, pero no proporciona información sobre el funcionamiento interno de la información. El refinamiento hace que el diseñador trabaje sobre la sentencia original, proporcionando cada vez más detalles a medida que van teniendo lugar sucesivamente todos y cada uno de los refinamientos (elaboración).

***Modularidad*.**

La arquitectura de computadora expresa la modularidad; es decir, el software se divide en componentes nombrados y abordados por separado, llamados frecuentemente módulos, que se integran para satisfacer

los requisitos del problema. Se ha afirmado que la modularidad es el único atributo del software que permite gestionar un programa intelectualmente, (PRESSMAN 2005). El software monolítico (es decir, un programa grande formado por un único módulo) no puede ser entendido fácilmente por el lector.

Finalmente, es importante destacar que un sistema se puede diseñar modularmente, incluso aunque su implementación deba ser monolítica. Existen situaciones, (por ejemplo, software en tiempo real, software empotrado), en donde no es admisible que los subprogramas introduzcan sobrecargas de memoria y de velocidad por mínimos que sean (por ejemplo, subrutinas, procedimientos). En tales situaciones el software podrá y deberá diseñarse con modularidad como filosofía predominante. El código se puede desarrollar “en línea”. Aunque el código fuente del programa puede no tener un aspecto modular a primera vista, se ha mantenido la filosofía y el programa proporcionará los beneficios de un sistema modular.

1.4 El diseño orientado a objetos.

El diseño de software orientado a objetos (DOO) requiere la definición de una arquitectura de software multicapa, la especificación de subsistemas que realizan funciones necesarias y proveen soporte de infraestructura, una descripción de objetos (clases), que son los bloques de construcción del sistema, y una descripción de los mecanismos de comunicación, que permiten que los datos fluyan entre las capas, subsistemas y objetos. El DOO, cumple todos estos requisitos.

El *diseño de objetos* se centra en la descripción de objetos y sus interacciones con los otros. Una especificación detallada de las estructuras de datos de los atributos y diseño procedural de todas las operaciones, se crea durante el diseño de objetos. La *visibilidad* para todos los atributos de clase se define, y las interfaces entre objetos se elaboran para definir los detalles de un modelo completo de mensajes.

El diseño de sistemas y objetos en UML se extiende para considerar el diseño de interfaces, administración de datos con el sistema que se va a construir y administración de tareas para los subsistemas que se han especificado. La visión del modelo de usuario maneja el proceso del diseño de la

interfaz de usuario, proporcionando una situación que se elabora iterativamente, para volverse un conjunto de clases de interfaz. La administración de datos establece un conjunto de clases y colaboraciones, que permiten al sistema (producto) manejar datos persistentes (por ejemplo, archivos y bases de datos). El diseño de la administración de tareas establece la infraestructura que organiza subsistemas en tareas, y administra la concurrencia de tareas.

1.4.1 Descripción de objetos.

Una descripción del diseño de un objeto (instancia de clase o subclase) puede tomar una o dos formas (LARMAN 2004): (1) *Una descripción de protocolo* que establece la interfaz de un objeto, definiendo cada mensaje que el objeto puede recibir y las operaciones que el objeto lleva a cabo cuando recibe un mensaje, ó (2) *Una descripción de implementación* que muestra detalles de implementación para cada operación implicada por un mensaje pasado a un objeto. Los detalles de implementación incluyen información acerca de la parte privada del objeto; esto significa, detalles internos acerca de la estructura de datos, que describen los atributos del objeto, y detalles de procedimientos, que describen las operaciones.

Una descripción de la implementación se compone de la siguiente información: (1) una especificación del nombre del objeto y referencia a la clase; (2) una especificación de las estructuras de datos privadas, con indicación de los datos y sus respectivos tipos; (3) una descripción de procedimientos de cada operación o, alternativamente, indicadores a dichas descripciones de procedimientos. La descripción de implementación debe contener información suficiente, para el manejo adecuado de todos los mensajes descritos en la descripción de protocolo. (LARMAN 2004).

1.5 Métricas técnicas del software.

Un elemento clave de cualquier proceso de ingeniería es la medición. Empleamos medidas para entender mejor los atributos de los modelos que crearnos. Pero, fundamentalmente, empleamos las medidas para valorar la calidad de los productos de ingeniería o de los sistemas que construimos. A diferencia de otras disciplinas, la ingeniería del software no está basada en leyes cuantitativas básicas de la Física. Las

medidas absolutas, tales como el voltaje, la masa, la velocidad o la temperatura no son comunes en el mundo del software. En su lugar, intentamos obtener un conjunto de medidas indirectas que dan lugar a métricas que proporcionan una indicación de la calidad de algún tipo de representación del software. Como las medidas y métricas del software no son absolutas, están abiertas a debate. (PRESSMAN 2005).

Los ingenieros del software usan las métricas técnicas para ayudarse en el desarrollo de software de mayor calidad. Aunque las métricas técnicas para el software de computadora no son absolutas, nos proporcionan una manera sistemática de valorar la calidad basándose en un conjunto de reglas definidas. También le proporcionan al ingeniero del software una visión interna en el acto, en vez de a posteriori. Esto permite al ingeniero descubrir y corregir problemas potenciales antes de que se conviertan en defectos catastróficos.

1.5.1 Métricas del Modelo de Diseño OO. Tendencias actuales.

Existen muchas características del diseño OO que son subjetivas. Generalmente, un diseñador experimentado sabe cómo realizar un sistema OO que implemente los requerimientos del cliente. Pero a veces, cuando un modelo de diseño OO crece en tamaño y complejidad, se necesita tener una visión más objetiva de las características del diseño para poder medirlo en cuanto a calidad se refiere.

Como parte de un tratamiento detallado de las métricas de software para sistemas OO, (PRESSMAN 2005) redefine algunas características distintas y medibles de un diseño OO:

- ✓ **Tamaño:** El tamaño se define en términos de población, volumen, longitud y funcionalidad. *La población* se mide haciendo un recuento de las entidades OO, como las clases u operaciones. Las medidas de *volumen* son iguales pero se realizan dinámicamente, en un instante de tiempo dado. *La longitud* es la medida de una cadena de elementos de diseño interconectados, ejemplo la profundidad de un árbol de herencia. Las métricas de *funcionalidad* proporcionan una indicación indirecta del valor entregado al cliente por una aplicación OO.

- ✓ **Complejidad:** La complejidad se define en términos estructurales, examinando cómo se relacionan las clases de un diseño OO con otras.

- ✓ **Acoplamiento:** Son las conexiones físicas entre los elementos del diseño OO, ejemplo, el número de colaboraciones entre clases o el número de mensajes intercambiados entre objetos.

- ✓ **Cohesión:** Un componente OO debe diseñarse de manera que posea todas las operaciones trabajando conjuntamente para alcanzar un propósito único y bien definido. La cohesión de una clase se determina examinando el grado en que el conjunto de propiedades que posee sea parte del diseño o dominio del problema.

Sobre las propiedades mencionadas anteriormente es que se basan gran número de métricas utilizadas para medir el diseño OO. Entre estas métricas podemos mencionar:

Acoplamiento entre clases objeto (ACO). ACO es el número de colaboraciones listadas para una clase. Plantea que a medida que ACO aumenta, es más probable que el grado de reutilización de una clase disminuya, y viceversa.

Tamaño de clase (TC). El tamaño de una clase puede medirse determinando el número total de operaciones y atributos encapsulados por la clase. Los valores grandes para TC indican que la clase debe tener bastante responsabilidad, por lo que reducirá la reutilización de la clase y complicará la implementación y las pruebas.

Respuesta para una clase (RPC). El conjunto de RPC es una serie de métodos que pueden ser ejecutados en respuesta. A medida que la RPC aumenta, el esfuerzo requerido para la comprobación también se incrementa.

Carencia de cohesión en los métodos (CCM). Cada método dentro de una clase accede a uno o varios atributos. CCM es el número de métodos que accede a uno más de los mismos atributos. Si CCM es alto,

los métodos deben acoplarse a otro, por medio de los atributos. Esto incrementa la complejidad del diseño de clases.

Es importante destacar la necesidad que existe actualmente sobre sistemas formales de métricas para proyectos software orientados a objetos. No es un secreto para nadie, que algunos de los sistemas de métricas actuales, a pesar de su apariencia, carecen de formalismo, cuando aplicamos sobre ellos algunos conceptos básicos de la teoría de la medición y de la medida. Veamos esto a partir de algunos ejemplos.

La métrica **métodos ponderados por clase (MPC)**, (PRESSMAN 2005), pretende calcular la complejidad de una clase como la sumatoria de las complejidades de cada uno de sus métodos. Para el cálculo de las complejidades se usa el método de complejidad ciclomática, uno de los más famosos y debatidos. Sin embargo, ¿Cómo se puede estar seguro de que la relación es lineal?, ¿No puede ser una simplificación excesiva reducirlo a una suma? Por otro lado, la medida debe ser establecida sobre propiedades claramente definidas. ¿Es la complejidad una propiedad claramente definida? Es evidente que no, pues no existe una definición aceptada de este concepto, por lo cual decir que una clase tiene una complejidad de 40 no es significativo. Otro concepto que no está claro aquí es el de la escala. Decir que la complejidad de una clase es 40 no da información alguna. Por tanto, no hay manera de comparar dos complejidades; si dos clases tienen valores de complejidades de 20 y 40 respectivamente, ¿quiere ello decir que la segunda es el doble de compleja que la primera?

Estas razones hacen que las métricas del software sean un conflicto en la actualidad. Todavía no contamos con un sistema formal de métricas universalmente aceptadas. Mientras tanto estaremos desarrollando métricas de dudosa interpretación y extrapolación entre sistemas y que medirán conceptos que no conocemos con la suficiente profundidad.

En el capítulo 3, para realizar la evaluación de la solución propuesta en el capítulo 2, se realizarán algunas de las mediciones anteriormente mencionadas, redefinidas en (PRESSMAN 2005).

1.6 Los patrones de diseño.

Los mejores diseñadores en cualquier campo tienen una habilidad extraña para reconocer patrones que caracterizan un problema y los patrones correspondientes, que pueden combinarse para crear una solución. Existen patrones repetidos de clases y objetos de comunicación, en muchos sistemas orientados a objetos. Estos patrones resuelven problemas específicos de diseño, y vuelven al diseño orientado a objetos más flexible, elegante y extremadamente reutilizable. Ayudan a los diseñadores a reutilizar diseños exitosos basando nuevos diseños en experiencia previa.

Según (PRESSMAN 2005), un patrón es una solución probada que se puede aplicar con éxito a un determinado tipo de problemas que aparecen repetidamente en algún campo. El establecimiento de estos patrones comunes es lo que posibilita el aprovechamiento de la experiencia acumulada en el diseño de aplicaciones. Un diseñador experimentado producirá diseños más simples, robustos y generales, y más fácilmente adaptables al cambio. Por otra parte, un buen diseño no debe ser específico de una aplicación concreta, sino que debe basarse en soluciones que han funcionado bien en otras ocasiones.

La idea que subyace a estos patrones es la constatación de que, aunque la orientación a objetos facilita la reutilización de código, la reutilización efectiva solo se produce a partir de un buen diseño, basado en el uso de soluciones (los patrones) que han probado su utilidad en situaciones similares. Los patrones no son bibliotecas de clases, sino más bien un esqueleto básico que cada diseñador adapta a las peculiaridades de su aplicación. Los patrones se describen en forma textual, acompañada de diagramas y seudo código.

1.6.1 Descripción de los patrones de diseño.

En general, la descripción de un patrón como parte de una taxonomía debe proporcionar (LARMAN 2004):

- Nombre del patrón. Por ejemplo *Filtro.*
- Intención del patrón. Por ejemplo, un patrón debe tener la intención de facilitar el mantenimiento, pues puede acomodar un número de diferentes tipos de objeto.

- Los problemas de diseño que motivan el patrón. Por ejemplo, un patrón debe desarrollarse, para que un número de transformaciones diferentes de datos puedan ser aplicadas a un objeto, muchas de las cuales son desconocidas, cuando el patrón es desarrollado originalmente.
- La solución que resuelve estos problemas.
- Las clases que se requieren para implementar la solución.
- Las responsabilidades y colaboraciones entre las clases de solución. Por ejemplo, una clase debe ser responsable de la construcción de un objeto, cuyo comportamiento varía al tiempo de ejecución.
- Lineamientos que conduzcan a una implementación efectiva. Generalmente, existirá un número de formas diferentes de programar un patrón; por ejemplo, el procesamiento de errores debe ser tratado de diferentes formas.
- Ejemplos de código fuente.
- Referencias a otros patrones de diseño, o patrones que puedan usarse en conjunto con el patrón descrito.

1.6.2 Objetivos de los patrones de diseño.

Los patrones de diseño pretenden proporcionar catálogos de elementos reusables en el diseño de sistemas software, evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente y formalizar un vocabulario común entre diseñadores y estandarizan el modo en que se realiza el diseño. Asimismo, no pretenden imponer ciertas alternativas de diseño frente a otras ni eliminar la creatividad inherente al proceso de diseño. No es obligatorio utilizar siempre los patrones, solo en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable. Abusar o forzar el uso de los patrones puede ser un error. (LARMAN 2004)

1.6.3 Clasificación de los patrones de diseño.

Los Patrones de Diseño son soluciones a problemas que se presentan repetidamente en el desarrollo de aplicaciones orientado a objeto. Según GOF (Gang of four, o "El Grupo de los 4"), los patrones se clasifican en tres niveles; de Creación, de Comportamiento y de Estructura. (LARMAN 2004).

Los patrones de creación están dirigidos al proceso de creación de un objeto, la forma en que se crean los objetos, los de estructura conciernen a la composición de la clase o los objetos, como éstos estarán compuestos, y los de comportamiento tienen que ver con la forma en que los objetos o clases interactúan y se les asignan responsabilidades.

Los patrones de comportamientos están relacionados con los algoritmos y con la asignación de responsabilidades a los objetos, es decir, que designan la forma de colaboración y comunicación entre objetos.

Los patrones de Estructura tienen que ver con la forma en que las clases son agrupadas para formar grandes estructuras, un ejemplo clásico sería una estructura inicial llamada animales, luego otras estructuras que deriva de la anterior llamadas Animales Herbívoros y Animales Carnívoros, estas dos ultimas pertenecen a la clase inicial animales heredando así sus características y también poseen características particulares.

1.6.4 El futuro de los patrones de diseño.

En la actualidad, se ha desarrollado y catalogado un número relativamente pequeño de patrones. De cualquier manera, los últimos cinco años han sido testigos de una tremenda explosión, en términos de interés industrial. Los patrones, junto con los componentes, ofrecen la esperanza de que la ingeniería de software, eventualmente, se parezca a otras disciplinas de ingeniería, con clases volviéndose el análogo en software de componentes electrónicos, y los patrones volviéndose el análogo en software de pequeños circuitos hechos de componentes. Antes de que esto suceda, existe aún una ingente cantidad de trabajo que llevar a cabo al identificar patrones y catalogarlos; también, se producirá esta situación, cuando el

número de patrones existentes se vuelva tan grande, que se necesite una forma de indexado automática o semiautomática. (JACOBSON 2004)

1.7 El caso UML.

El desarrollo real de UML comenzó en octubre de 1994 cuando Grady Booch y Jim Rumbaugh de Rational Software comenzaron a trabajar en la unificación de los lenguajes de modelado Booch y OMT, es entonces cuando fueron reconocidos mundialmente a la cabeza del desarrollo de metodologías orientadas a objetos. Fue entonces cuando terminaron su trabajo de unificación obteniendo el borrador de la versión 0.8 del denominado *Unified Method* en octubre de 1995. Tras esto también en 1995, Ivar Jacobson, padre de la metodología OOSE se unió con Rational Software para obtener finalmente UML 0.9 y 0.91 en junio y octubre de 1996.

El UML (Lenguaje Unificado para la Construcción de Modelos) se define como un “lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software…” (JACOBSON 2000). Es un sistema rotacional (que, entre otras cosas, incluye el significado de sus notaciones) destinado a los sistemas de modelado que utilizan conceptos orientados a objetos.

El lenguaje UML estandariza los artefactos y la notación, pero no define un proceso oficial de desarrollo. (JACOBSON 2000) explica algunas de las razones:

- Aumentar las probabilidades de una aceptación generalizada de la notación estándar del modelado, sin la obligación de adoptar un proceso oficial.

- La esencia de un proceso apropiado admite mucha variación y depende de las habilidades del personal, de la razón investigación-desarrollo, de la naturaleza del problema, de las herramientas y de muchos otros factores.

1.7.1 Ventajas de UML.

Una de las grandes ventajas de UML es que es un lenguaje extensible que permite adaptarse a diferentes tipos de aplicaciones. Los mecanismos de extensibilidad de UML incluyen:

- Estereotipos: Permiten crear nuevos tipos de bloques de construcción a partir de los existentes pero que sean específicos para un determinado problema.

- Valores etiquetados: Extienden las propiedades de un bloque de construcción de UML, permitiendo añadir nueva información en la especificación de ese elemento.

- Restricciones: Extienden la semántica de un bloque de construcción UML, permitiendo añadir nuevas reglas o modificar las existentes. El lenguaje OCL (ObjectConstraint Language) permite especificar formalmente las restricciones.

1.8 Herramientas CASE.

Las **Herramientas CASE** (***C****omputer* ***A****ided* ***S****oftware* ***E****ngineering*, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas nos pueden ayudar 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. (GÓMEZ 2003)

Las herramientas CASE alcanzaron su techo a principios de los años 90. En la época en la que IBM había conseguido una alianza con la empresa de software AD/Cycle para trabajar con sus mainframes, estos dos gigantes trabajaban con herramientas CASE que abarcaban todo el ciclo de vida del software. Pero poco a poco los mainframes han ido siendo menos utilizados y actualmente el mercado de las Big CASE

ha muerto completamente abriendo el mercado de diversas herramientas más específicas para cada fase del ciclo de vida del software.

1.8.1 Rational Rose Enterprise Edition 2003.

Rational Rose es una poderosa herramienta de modelado visual, excelente para desarrollar el análisis y diseño orientado a objetos de un sistema. Su versión Enterprise Edition 2003 fue la seleccionada para la elaboración y desarrollo de los artefactos que componen el modelo de diseño (CAMEJO 2007). Para su selección se tuvieron en cuenta una serie de criterios definidos en (FUSTER 2006):

a) Permite el modelado visual de artefactos UML. Presenta un gran número de modelos que son necesarios para la realización del diseño y brinda facilidades para desarrollar el dibujo en dichos modelos.
b) Establece jerarquías de paquetes de artefactos.
c) Soporta diferentes tipos de diagramas: vista estática, vista dinámica, vista de procesos de negocio, revisión UML 2.0.
d) Permite realizar modelado de procesos de negocio, modelado de datos, y modelado de aplicaciones Web.
e) Brinda procesos de ingeniería desde modelos de objetos a modelos de datos y esquemas de base de datos, además de ingeniería inversa desde esquemas de base de datos a modelos de datos y modelos de objetos.
f) Presenta distintos niveles de generación de código, especificación de modelos independientes de plataforma, y conectores para generar código de múltiples plataformas.
g) Presenta integración con soporte de patrones de funcionalidad, de análisis, de diseño, e importación de diagramas para confeccionar presentaciones.

1.9 Conclusiones

Después de haber realizado un análisis sobre los sistemas de gestión estadística similares al que se pretende diseñar en este trabajo, observamos que carecen de funcionalidades básicas que hoy día han

evolucionado significativamente. Muchos de ellos, debido a sus insuficiencias, están siendo migrados a otros lenguajes de programación. La información pública referente al proceso de automatización del cálculo de IPC es relativamente escasa, por lo que el análisis de sistemas de este tipo es difícil de realizar y a su vez el desarrollo de un nuevo sistema.

Con las características analizadas sobre el diseño, llegamos a la conclusión de que este flujo de trabajo es uno de los más importantes de todo el proceso de desarrollo de software. Es donde se modela, comprende y detalla el sistema a construir. Como técnicas, los patrones colaboran a que el diseño sea más robusto y permiten la solución de determinados problemas que enfrentaremos durante la elaboración del diseño.

Sobre las métricas para medir la calidad del diseño, podemos concluir diciendo que en la actualidad es un problema cotidiano el cómo usarlas, debido a la informalidad que todavía existe en algunas de las propiedades en las que se basan las métricas, entre ellas, la complejidad, la cohesión, el acoplamiento, el tamaño, etc.

Finalmente, para la realización del diseño del sistema, utilizaremos la herramienta Rational Rose por sus potencialidades y características que permiten el modelado visual de artefactos UML.

Capítulo 2 “El modelo de diseño propuesto”.

2.1 Introducción.

En este capítulo se presentan y describen todos los artefactos generados durante esta etapa de desarrollo que componen el Modelo de Diseño, los mismos son: subsistemas de diseño, clases del diseño y la realización de los casos de uso-diseño. De forma previa se realiza un análisis de la arquitectura propuesta, y una breve descripción de los casos de uso.

2.2 Análisis de la arquitectura propuesta para el desarrollo del sistema.

La realización de todo diseño va dirigida según la arquitectura propuesta para el desarrollo del sistema. Por tanto, en esta sección se realizará un análisis de la línea base de la arquitectura propuesta por el arquitecto del sistema (CAMEJO 2007), donde se hará referencia a elementos y cuestiones importantes relacionadas con la realización del diseño del software.

2.2.1 Concepciones generales y organigrama de la arquitectura.

La arquitectura del sistema IPC-ONE consta de dos características principales. La primera; es un modelo de desarrollo orientado a casos de usos. Esto permite que toda la organización y desarrollo del producto se ajuste a las necesidades de los clientes, y establece cierta claridad en la comprensión de cada caso de uso (CU) por parte del equipo de desarrollo y del cliente. La segunda característica se refiere a que todo el ciclo de elaboración del sistema se ajustará a un modelo iterativo e incremental, lo cual permite que se vayan obteniendo resultados parciales y dar un mejor seguimiento al proyecto y mantener un chequeo constante del avance del mismo.

Teniendo en cuenta los requerimientos del sistema IPC-ONE y considerando la poca complejidad funcional del mismo, se definió un estilo arquitectónico organizado desde un enfoque vertical respondiendo a un modelo multicapas.

2.2.2 Vista lógica de la arquitectura en capas.

La vista lógica describe la arquitectura en capas desde un alto nivel de abstracción donde se representan los paquetes contenidos en cada capa.

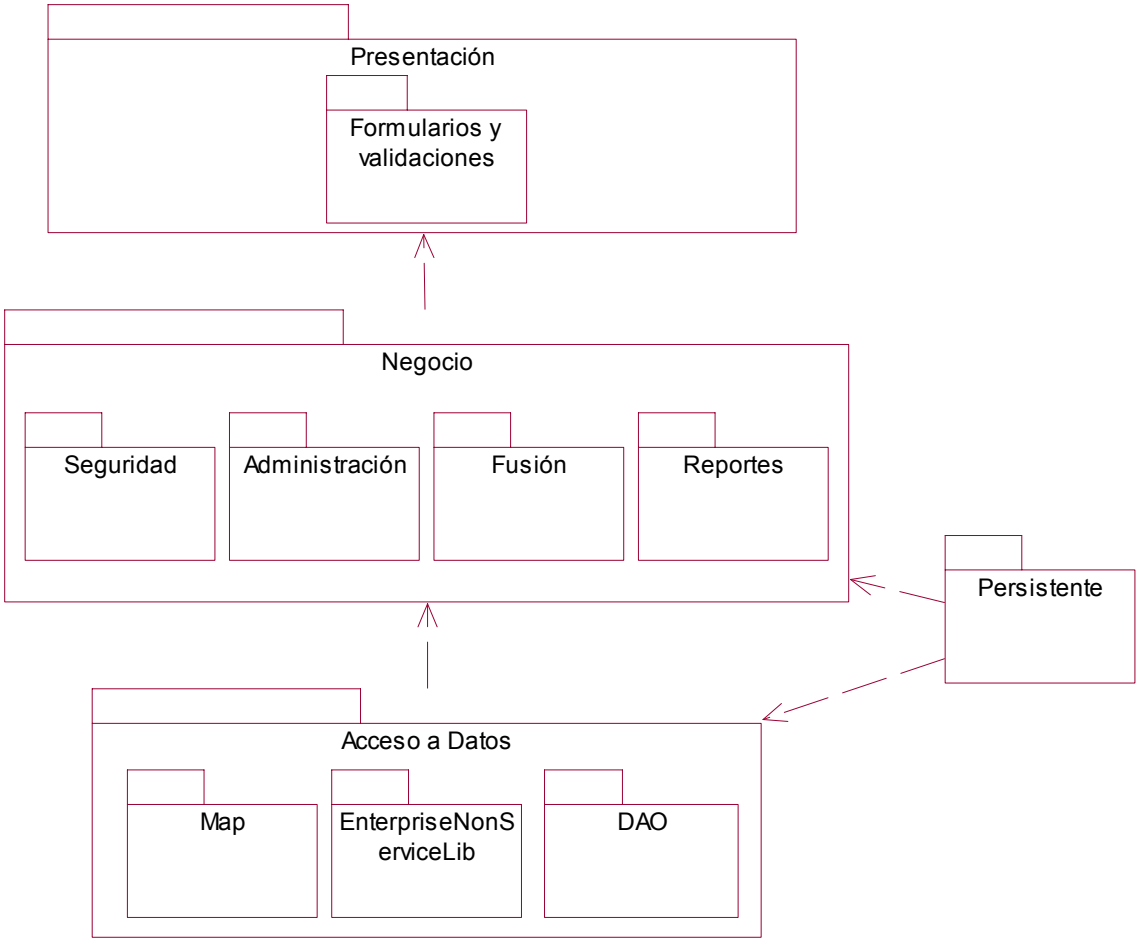

Figura 2. Arquitectura en capas.

Capa Presentación: Contiene todos los formularios para la captura y salida de datos de los usuarios, así como componentes de validación de datos. Estas clases permiten el intercambio de los usuarios con el sistema, los mismos pueden entrar y obtener información acerca de los productos, índices de precios, clasificadores, etc.

Capa Negocio: Contiene las clases que encapsulan la lógica del negocio; clases controladoras y algunas entidades utilizadas en cada uno de los subsistemas para el control del flujo de información y el cálculo de operaciones.

Capa de Acceso a Datos: Encapsula tres paquetes principales. El paquete Map contiene las clases generadas por el Tier Developer, las cuales se encargan del mapeo de cada una de las tablas de la base de datos. El paquete EnterpriseNonServiceLib representa el framework del Tier Developer y brinda servicios para el manejo de la lógica de acceso a datos. Por último, el paquete DAO, contiene clases correspondientes a cada entidad persistente, las cuales se apoyan en las clases del paquete Map para obtener los datos de la BD y crear los objetos que van a pasar a la capa del negocio.

Paquete **Persistente:** Este paquete pretende que las capas de negocio y de acceso a datos conozcan a los objetos de persistencia. Aquí se agrupan todas las clases persistentes. De esta manera las clases DOA pueden crear objetos persistentes una vez que hayan recuperado alguna información de la BD.

La arquitectura multicapas propuesta es factible para el sistema a desarrollar. Cada componente de una determinada capa solo conoce los componentes de la capa inmediata inferior, de este modo se garantiza un mejor manejo de los datos y un flujo de información organizado y mejor controlable. Cada capa presenta una estructura lógica de funcionamiento que puede ser modificable sin influir en el funcionamiento de las demás capas. Esto brinda cierta flexibilidad a la arquitectura y contribuye a una mejor configuración de los cambios que se puedan realizar.

2.3 Casos de uso del sistema.

Los casos de uso del sistema fueron definidos por el analista (RÍOS 2007) durante la etapa de análisis del sistema con el fin de proporcionar soluciones concretas para el cumplimiento de los requisitos funcionales y a las necesidades del cliente. Para lograr esto, fueron necesarios diez casos de uso, los cuales se muestran en la siguiente figura y en secciones posteriores se describe la realización de cada uno de ellos.

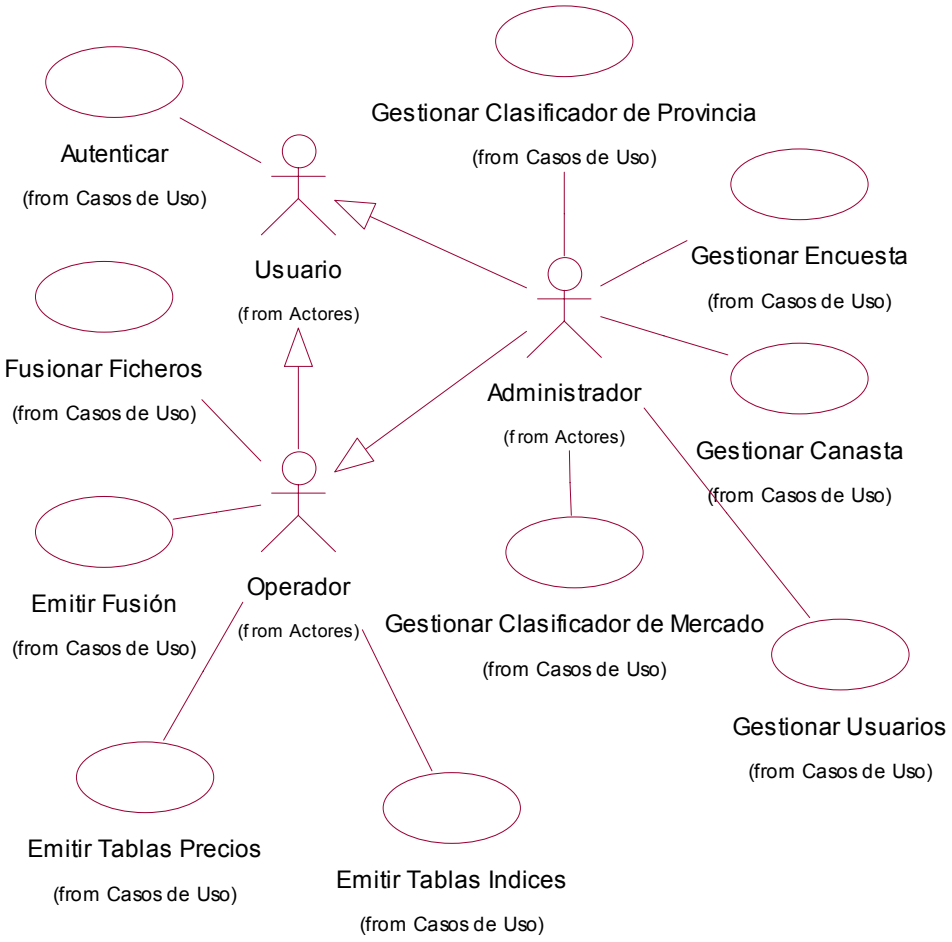

Figura 3. Diagrama de casos de uso del sistema.

2.4 Subsistemas de diseño.

El sistema ONE-IPC no requiere de la realización de grandes operaciones, pero sí necesita una buena estructura para dar cumplimiento de manera eficiente a los requisitos funcionales. Para ello se ha decidido dividir el sistema en subsistemas de diseño, con el objetivo de organizar mejor los componentes del modelo de diseño.

Los subsistemas de diseño son una forma de organizar los artefactos del modelo de diseño en piezas más manejables. Un subsistema puede constar de clases del diseño, realizaciones de casos de uso-diseño, interfaces y otros subsistemas (recursivamente). Por otro lado, un subsistema puede proporcionar interfaces que representan la funcionalidad que exportan en términos de operaciones. Un subsistema debe ser cohesivo; es decir, sus contenidos deben encontrarse fuertemente asociados. Además, los subsistemas deben ser débilmente acoplados; esto es, las dependencias entre unos y otros, o entre sus interfaces, deben ser mínimas. (JACOBSON 2004)

Para dividir el diseño en subsistemas, primeramente se tuvo en cuenta el criterio de agrupar funcionalidades y asignar responsabilidades, de forma tal que la dependencia entre cada subsistema fuera mínima y que cada uno de ellos cumpla con sus funciones. El objetivo principal de la división en subsistemas de diseño realizada es poder manipular de forma más eficiente los cambios que se puedan realizar en un futuro, esto es, poder realizar cambios a un subsistema de diseño sin que se vean afectados los demás. Cada subsistema contiene la estructura de la arquitectura en capas definida, y a su vez, engloba un conjunto de clases de diseño.

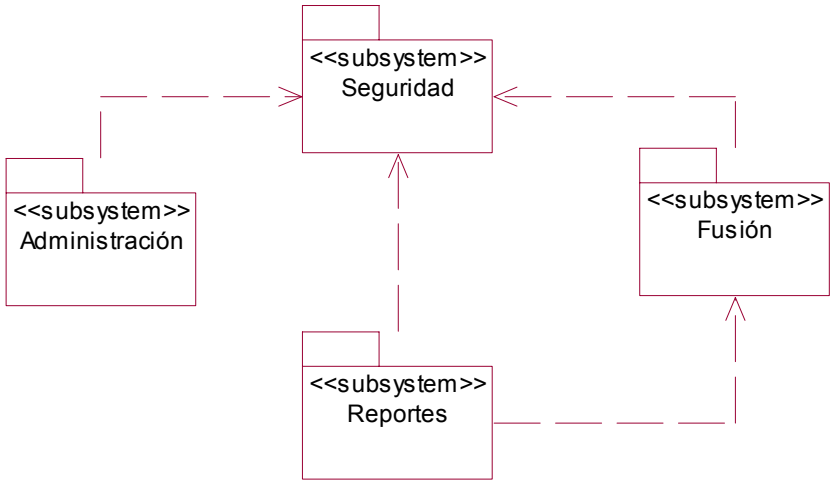

Figura 4. Relaciones de los subsistemas de diseño.

2.4.1 Subsistema de diseño: Seguridad.

En este subsistema se encierran los casos de uso autenticar y gestionar usuarios respectivamente. De acuerdo a las funcionalidades de estos casos de uso, es posible decir entonces que el subsistema de diseño Seguridad tiene la responsabilidad de brindar acceso al sistema a los usuarios y permitir la gestión de los usuarios del sistema, (insertar nuevo usuario, eliminar un usuario y modificar un usuario). Ningún usuario puede realizar operaciones en el sistema antes de autenticarse, de aquí parte la dependencia que tienen los demás subsistemas de diseño con el que se describe. Las clases de diseño que interactúan para resolver las funcionalidades de este subsistema se muestran a continuación.

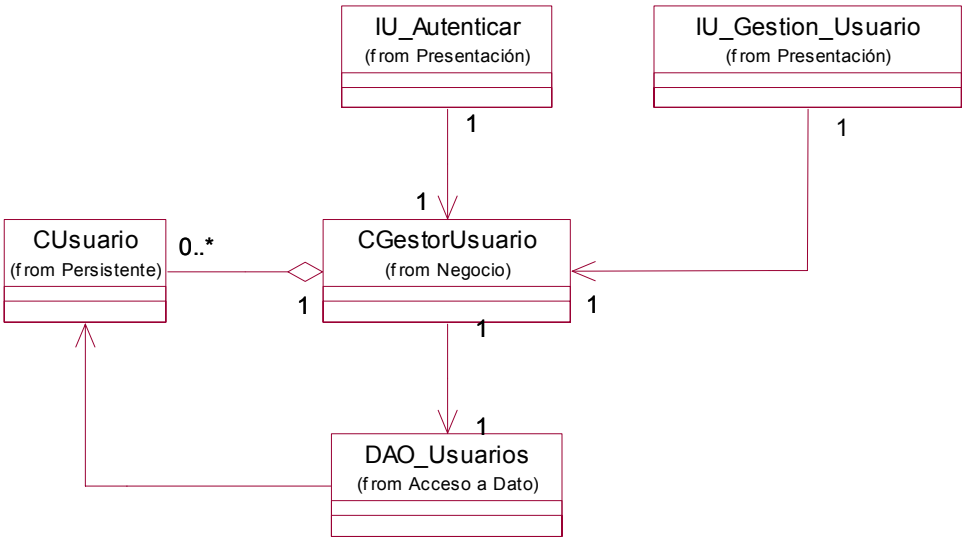

Figura 5. Clases de diseño del SD Seguridad.

2.4.2 Subsistema de diseño: Fusión.

Este subsistema de diseño fue definido con la intensión de agrupar las funcionalidades de captura y almacenamiento de la información contenida en los ficheros enviados desde las distintas provincias. Todo este proceso se realiza en el caso de uso Fusionar Ficheros. Este caso de uso es considerado arquitectónicamente significativo, ya que esta es una de las tareas principales del sistema que tiene gran prioridad. Si no se recogen los datos relacionados a una fecha determinada, pues no se podrán procesar nunca dichos datos para calcular los índices de precios porque no existen en la base de datos.

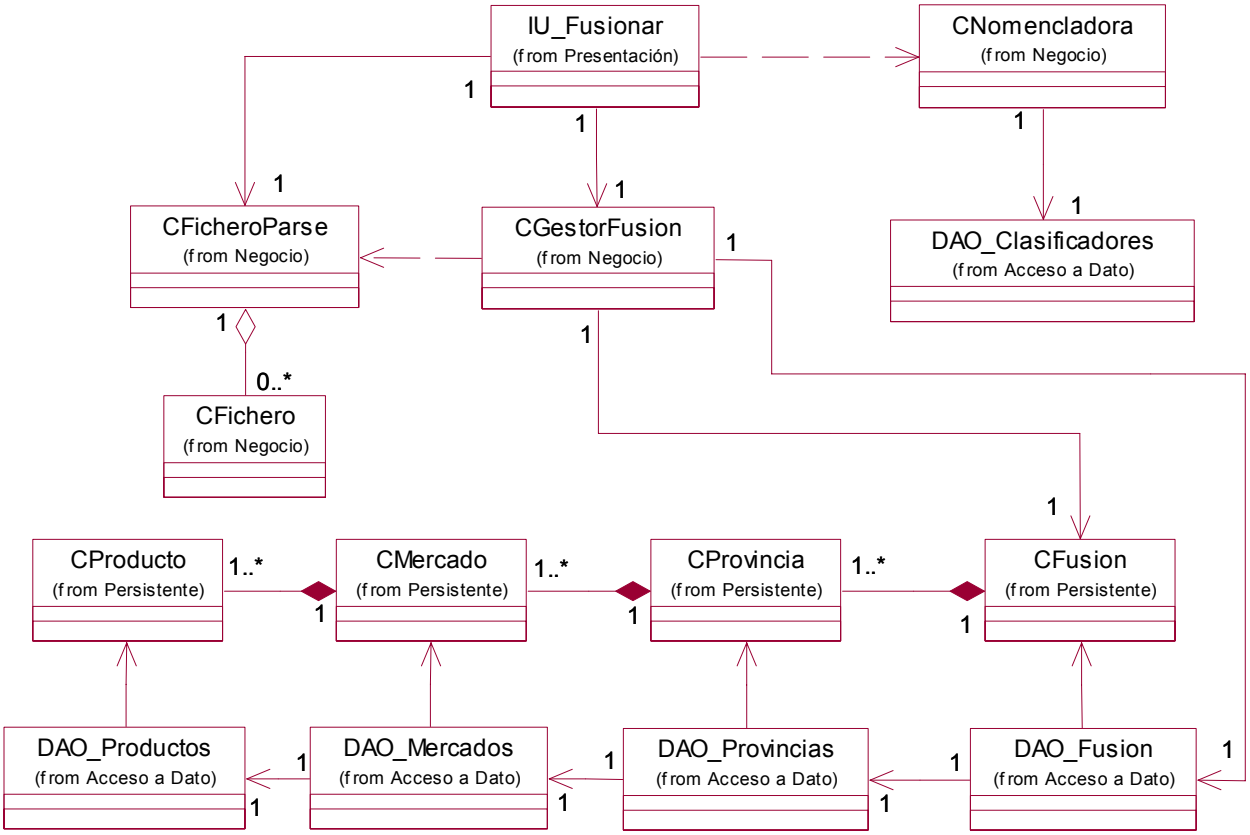

Figura 6. Clases del diseño SD Fusión.

El proceso de lectura de los ficheros:

Este proceso es de gran importancia dentro del sistema y exige una buena calidad en su realización, un mínimo error en el procesamiento de la información podría causar resultados posteriores no deseados. La clave para la captura de datos se encuentra en la estructura del nombre del fichero. Antes de hacer una fusión de datos, el usuario especifica el año y el mes que se desea fusionar. El nombre del fichero tiene el siguiente formato: Tipo de mercado + código de provincia + mes + año. Una vez que el usuario selecciona la fecha a fusionar, debe indicar el directorio donde se encuentran los ficheros. Posteriormente el sistema carga en una lista, todos los ficheros que son válidos por el nombre, (entiéndase válido por el nombre a todos los ficheros cuyos nombres cumplan con el formato descrito anteriormente), y después va

eliminando de esa lista todos los ficheros que no cumplen con la fecha seleccionada, (los datos contenidos en la lista son las URL de cada fichero, no el fichero en sí). La información que contiene cada fichero está estructurada de la siguiente manera:

Ejemplo de un fichero.

Nombre Fichero: ES030507.txt

Tabla 1. Ejemplo del contenido de un fichero.

Código del producto	Precio del producto
11568763	8.50
45111755	23.00
33796152	2.20

Las siglas ES representan al mercado formal, el fichero pertenece a Ciudad de la Habana (03), al mes de mayo (05), y al año 2007 (07). Información que contiene: tres códigos de productos y sus respectivos precios recogidos en ese mes.

La clase controladora de este caso de uso contiene una operación que prepara la lista de ficheros que se van a fusionar, y otra operación se encarga después de ir recibiendo cada uno de esos ficheros y realizar la lectura de los mismos, obteniendo como resultado de cada lectura una lista de productos, la cual es insertada en su mercado correspondiente y con su respectiva fecha.

2.4.3 Subsistema de diseño: Reportes.

Este subsistema de diseño engloba los casos de uso Emitir Fusión, Emitir Tablas de Precio y Emitir Tablas de Índices. Además, se calculan los índices de precios antes de ser emitidos. La característica fundamental que se tuvo en cuenta a la hora de diseñar este subsistema fue poder cambiar la forma en que se emite cada reporte sin afectar el funcionamiento de los otros subsistemas. La información a partir de la cual de emiten los reportes está almacenada en base de datos. Esto quiere decir que el

funcionamiento básico de este subsistema es pedir la información solicitada para mostrarse a la base de datos y mostrarla en pantalla utilizando diferentes mecanismos que pueden ser configurables sin alterar el funcionamiento de otros componentes.

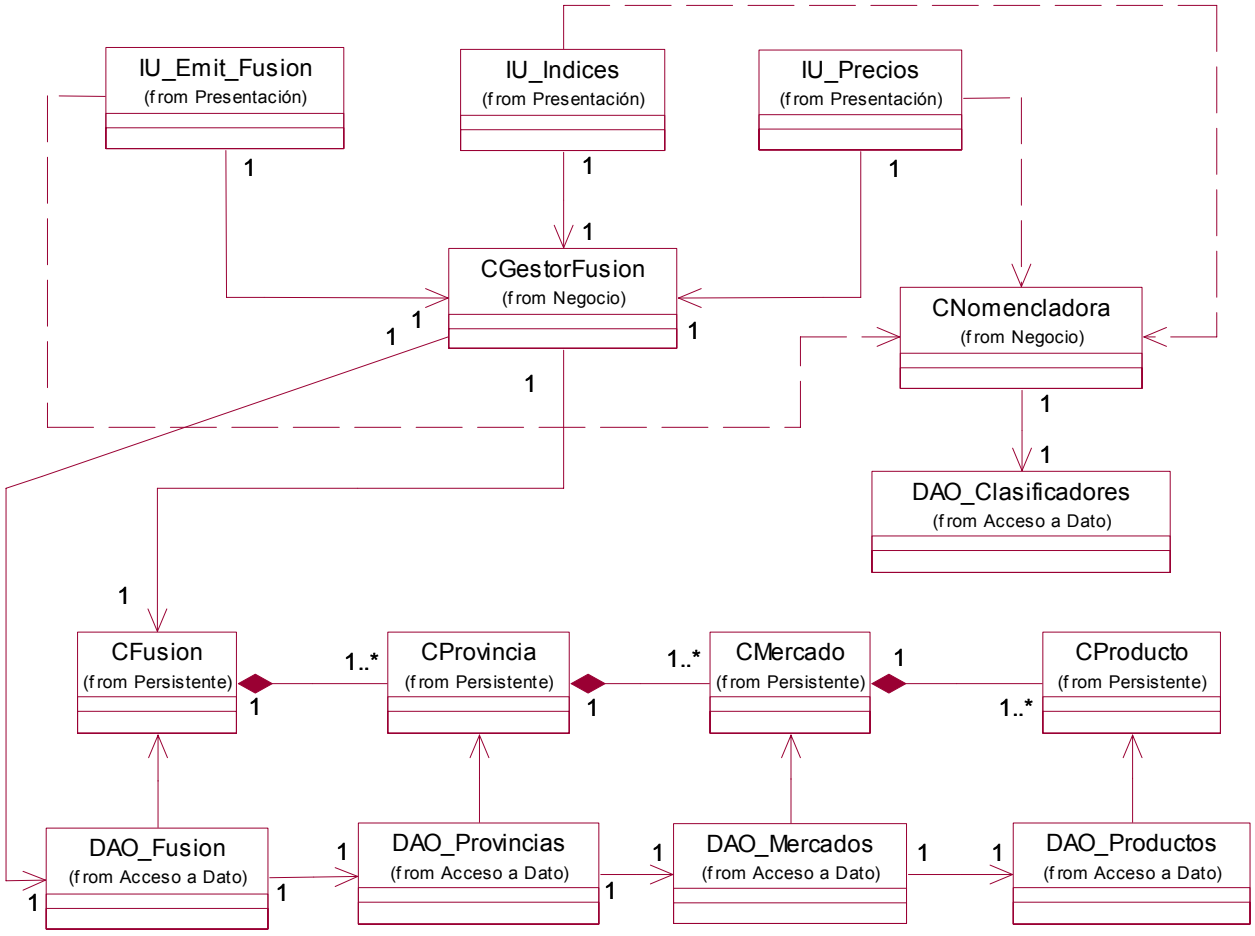

Figura 7. Clases del diseño SD Reportes.

2.4.4 Subsistema de diseño: Administración.

Este subsistema de diseño brinda servicios de gestión con el objetivo de establecer cambios en la información con la que se trabaja en los demás subsistemas. Esto facilita no tener que realizar los cambios en los demás subsistemas y en el código de la aplicación. Para tratar estos cambios se tuvieron en cuenta un conjunto de factores importantes que pueden presentarse en un futuro inmediato.

Este subsistema sirve de soporte de información a los restantes subsistemas, o sea, es el encargado de gestionar toda la información del proyecto. Ejemplos: a) Manipula la información relacionada con los productos como son: nombres, unidades de medida, precios máximos y mínimos, códigos. b) Manipula la información de los mercados: tipos de mercados, códigos. c) Manipula la información de las provincias: códigos, nombres. d) Manipula la información de la encuesta: precios de los productos en los diferentes mercados.

Las operaciones que se realizan sobre estos clasificadores son las mismas para los cuatro casos de uso: eliminar, insertar y modificar, entre otras. Cada clase controladora contiene una lista del tipo de objetos que ellas manipulan, con el objetivo de darle solución al problema de que en un momento determinado, la conexión con BD pueda caerse. Se decidió trabajar con los objetos clasificadores en memoria virtual, o sea, las clases controladoras antes de realizar alguna operación, verifican la conexión con BD y en caso que no haya, pues almacenan el objeto a manipular en su lista y lo mantiene “vivo” hasta que la conexión se reestablezca para poderlo insertar. Esta estrategia podría complicarse en caso de trabajar con un gran número de datos, pero en nuestro caso, estamos hablando de objetos que solo tienen 3 atributos, ocupan poco espacio y se impide el estar abriendo y cerrando conexiones cada vez que se vaya a realizar una simple operación de obtención de datos. Es preferible obtener todo con lo que se va a trabajar al instante.

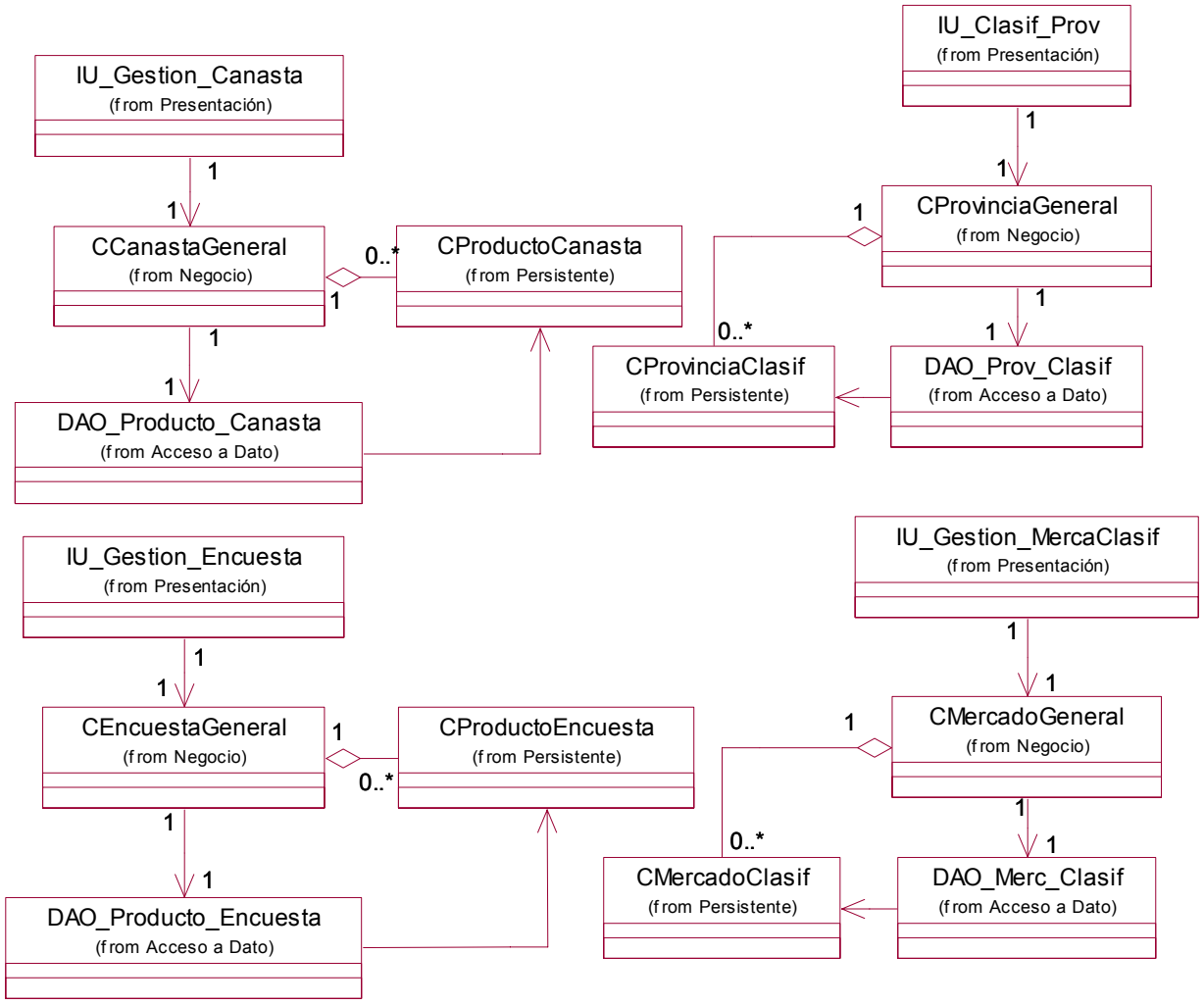

Figura 8. Otra parte de las clases del diseño SD Administración.

La principal idea considerada para la gestión de los clasificadores se basa en que los códigos de los productos a nivel nacional pueden cambiar, se pueden agrupar varias provincias en trabajo conjunto a la hora de emitir la información mensual a la ONE, pueden eliminarse mercados, y pueden cambiar en cierto sentido las unidades de medida de algunos productos.

Este subsistema brinda las facilidades para poder realizar estos cambios, y además provee de los cambios realizados a los demás subsistemas.

2.5 Clases del diseño.

Una clase de diseño es una abstracción sin costura de una clase o construcción similar en la implementación del sistema. (JACOBSON 2004)

Esta abstracción es sin costuras en el sentido de que el lenguaje utilizado para especificar una clase de diseño debe ser el mismo lenguaje de programación en el que se va a desarrollar. Consecuentemente, los atributos, operaciones, parámetros y tipos de datos deben ser especificados utilizando la sintaxis del lenguaje de programación. En esta sección se representa la descripción de las clases más importantes del diseño, en el **¡Error! No se encuentra el origen de la referencia.** se encuentran las especificaciones de las restantes clases.

2.5.1 Especificaciones de las clases de diseño.

Tabla 2. Especificación de la clase CProducto.

Nombre: public class CProducto	
Tipo de clase: Entidad.	
Atributo	Tipo
private int idp	int
private float precio	float
Responsabilidades	
Nombre:	public CProducto()
Descripción	Constructor sin parámetros de la clase.
Nombre:	public CProducto(int cod, float p)
Descripción	Constructor pasándole parámetros.
Nombre:	public int IDP()
Descripción:	Obtiene y asigna el atributo idp.

Nombre:	public float Precio()
Descripción:	Obtiene y asigna el atributo precio.

Tabla 3. Especificación de la clase CFichero.

Nombre: public class CFichero	
Tipo de clase: Entidad.	
Atributo	Tipo
private string nombreFichero	string
private string dirreccionURL	string
private string extension	string
Responsabilidades	
Nombre:	public CFichero()
Descripción:	Constructor sin parámetros de la clase.
Nombre:	public CFichero(string dir, string ext, string nomb)
Descripción:	Constructor pasándole parámetros.
Nombre:	public bool válido_por_nombre()
Descripción:	Para saber si el fichero es válido por su nombre.
Nombre:	public bool Válido_por_fecha(int mes, int anno, string ext)
Descripción	Para saber si el fichero es válido por su fecha.
Nombre:	public string NombreFichero()
Descripción	Obtiene y asigna el atributo nombreFichero.
Nombre:	public string DirrecionURL()
Descripción	Obtiene y asigna el atributo direccionURL
Nombre:	public string Extension()
Descripción	Obtiene y asigna el atributo extensión.
Nombre:	public string TipoMercado()
Descripción	Devuelve el tipo de mercado al que pertenece el fichero.
Nombre:	public int Provincia()
Descripción	Devuelve el código de la provincia a la que pertenece el fichero.
Nombre:	public int Mes()
Descripción	Devuelve el mes al que pertenece el fichero.
Nombre:	public int Anno()
Descripción	Devuelve el año al que pertenece el fichero.

Tabla 4. Especificación de la clase CFusion.

Nombre: public class CFusion	
Tipo de clase: Entidad.	
Atributo	Tipo
private int anno	int
private int mes	int
private List<CProvincia> provincias	List<CProvincia>
Responsabilidades	
Nombre:	public CFusion()
Descripción:	Constructor sin parámetros de la clase.
Nombre:	public CFusion(int a, int m, List<CProvincia> pprovincias)
Descripción:	Constructor pasándole parámetros.
Nombre:	public void InsertarProvincia(CProvincia temp)
Descripción:	Inserta una provincia pasándole in item de tipo CProvincia
Nombre:	public bool ExisteProvincia(int cod)
Descripción:	Saber si determinada provincia existe pasándole el código.
Nombre:	public int Mes()
Descripción:	Obtiene y asigna el mes.
Nombre:	public int Anno ()
Descripción	Obtiene y asigna el anno.
Nombre:	public List<CProvincia> Provincias
Descripción	Devuelve la lista de provincias.
Nombre:	public int BuscarProvincia(int cod)
Descripción	Busca la posición de una provincia de de la lista de provincias.
Nombre:	public string Extension()
Descripción	Obtiene y asigna la extensión.
Nombre:	public void Fusionar(List<CFichero> ficheros)
Descripción	Prepara la lista de ficheros a fusionar.
Nombre:	private List<CProducto> ParsearFichero(string URL)
Descripción	Devuelve la lista de productos del fichero que se parsea.
Nombre:	public bool FusionValida(int mes, int anno)

Descripción	Saber si una fusión es válida por la fecha.
Nombre:	public string PrecioProducto(int codP, int codM, int codProd)
Descripción	Devuelve el precio de un producto como un string.

Tabla 5. Especificación de la clase CProvincia.

Nombre: public class CProvincia	
Tipo de clase: Entidad.	
Atributo	Tipo
private int id	int
private List<CMercado> mercados	List<CMercado>
Responsabilidades	
Nombre:	public CProvincia()
Descripción:	Constructor sin parámetros de la clase.
Nombre:	public CProvincia(int p, string nom, List<CMercado> mercad)
Descripción:	Constructor pasándole parámetros.
Nombre:	public int ID ()
Descripción	Obtiene y asigna el atributo id.
Nombre:	public void InsertarMercado(CMercado temp)
Descripción:	Inserta un mercado en la lista de mercados.
Nombre:	public int BuscarMercado(string tip)
Descripción	Devuelve la posición de un mercado en la lista de mercados.
Nombre:	public bool ExisteMercado(string tip)
Descripción	Saber si un mercado existe en una provincia.
Nombre:	public List<CMercado> ObtenerMercados()
Descripción	Obtiene la lista de mercados.
Nombre:	public List<CProducto> ObtenerProductosDadoMercado(string p)
Descripción	Obtiene los productos de un mercado dado el tipo de mercado.

Tabla 6. Especificación de la clase CMercado.

Nombre: public class CMercado	
Tipo de clase: Entidad.	
Atributo	Tipo
private int id	int

private List<CProducto> prod		List<CProducto>
Responsabilidades		
Nombre:	public CMercado()	
Descripción	Constructor sin parámetros de la clase.	
Nombre:	public CMercado(int a, List<CProducto> aproductos)	
Descripción	Constructor pasándole parámetros.	
Nombre:	public int ID()	
Descripción	Obtiene y asigna el atributo id.	
Nombre:	public void InsertarProductoEnMercado(CProducto temp)	
Descripción:	Inserta el producto que se le pasa en la lista de productos.	
Nombre:	public bool ExisteProducto(int cod)	
Descripción:	Saber si un producto existe pasándole el código del producto.	
Nombre:	public int BuscarProducto(int cod)	
Descripción	Devuelve la posición de un producto en la lista de productos.	
Nombre:	public float ObtenerPrecio(int cod)	
Descripción	Devuelve el precio de un producto.	
Nombre:	public List<CProducto> ObtenerProductos()	
Descripción	Devuelve la lista de productos.	

Tabla 7. Especificación de la clase CProductoCanasta.

Nombre: public class CProductoCanasta		
Tipo de clase: Entidad.		
Atributo		Tipo
private int idp		int
private string nombre		string
private string unidad		string
private float preciomin		float
private float preciomax		float
private float preciomindiv		float
private float preciomaxdiv		float
Responsabilidades		
Nombre:	public CProductoCanasta()	
Descripción	Constructor sin parámetros de la clase.	

Nombre:	public CProductoCanasta(int acod, string nom, double min, double max, double mindiv, double maxdiv, string uni)
Descripción	Constructor pasándole parámetros.
Nombre:	public int IDP()
Descripción:	Obtiene y asigna el atributo idp.
Nombre:	public string Nombre()
Descripción:	Obtiene y asigna el atributo nombre.
Nombre:	public double PreMin()
Descripción:	Obtiene y asigna el atributo preciomin.
Nombre:	public double PreMax()
Descripción	Obtiene y asigna el atributo preciomax.
Nombre:	public double PreMinDiv()
Descripción	Obtiene y asigna el atributo preciomindiv.
Nombre:	public double PreMaxDiv()
Descripción	Obtiene y asigna el atributo preciomaxdiv.
Nombre:	public string Unidad()
Descripción	Obtiene y asigna el atributo unidad.

Tabla 8. Especificación de la clase CCanastaGeneral.

Nombre: public class CCanastaGeneral	
Tipo de clase: Controladora	
Atributo	Tipo
private List<CProductoCanasta> canasta	List<CProductoCanasta>
Responsabilidades	
Nombre:	public CCanastaGeneral()
Descripción:	Constructor sin parámetros de la clase.
Nombre:	public CCanastaGeneral(List<CProductoCanasta> cana)
Descripción:	Constructor pasándole parámetros.
Nombre:	public bool InsertarProductoCanasta(CProductoCanasta temp)
Descripción:	Inserta un producto en la canasta.
Nombre:	public bool ExisteProductoCanasta(int cod)
Descripción	Saber si un producto existe en la canasta pasándole el código.
Nombre:	public int BuscarProductoCanasta(int cod)

Descripción	Devuelve la posición de un producto en la canasta.
Nombre:	public bool EliminarProductoCanasta(int cod)
Descripción	Eliminar un producto de la canasta.
Nombre:	public bool ModificarProductoCanasta (int cod, CProductoCanasta)
Descripción	Modificar un producto de la canasta.
Nombre:	public CProductoCanasta ObtenerProductoCanasta(int cod)
Descripción	Devuelve un producto de la canasta.

Tabla 9. Especificación de la clase CUsuario.

Nombre: CUsuario	
Tipo de clase: Entidad.	
Atributo	Tipo
private string user	string
private string pass	string
private string rol	string
Responsabilidades	
Nombre:	public CUsuario()
Descripción	Constructor sin parámetros
Nombre:	public CUsuario(string auser, string apass, string arol)
Descripción	Constructor de la clase pasándole parámetros.
Nombre:	public string User()
Descripción:	Asigna y devuelve el valor del atributo user.
Nombre:	public string Pass()
Descripción:	Asigna y devuelve el valor del atributo pass.
Nombre:	public string Rol()
Descripción	Asigna y devuelve el valor del atributo rol.

Tabla 10. Especificación de la clase CGestorUsuario.

Nombre: CGestorUsuario	
Tipo de clase: Controladora	
Atributo	Tipo
private List<CUsuario> usuarios	List<CUsuario>
Responsabilidades	

Nombre:	public CGestorUsuario()
Descripción:	Constructor sin parámetros de la clase.
Nombre:	public CGestorUsuario(List<CUsuario> users)
Descripción:	Constructor pasándole parámetros.
Nombre:	public bool comparar_pass(CUsuario, string pass2)
Descripción:	Compara la contraseña de dos usuarios.
Nombre:	public bool InsertarUsuario(CUsuario temp)
Descripción	Inserta un nuevo usuario.
Nombre:	public bool ExisteUsuario(string aUser)
Descripción	Saber si existe un usuario.
Nombre:	public int BuscarUsuario(string aUser)
Descripción	Devuelve la posición de un usuario.
Nombre:	public bool EliminarUsuario(string aUser)
Descripción	Elimina un usuario.
Nombre:	public CUsuario ObtenerUsuario(string aUser)
Descripción	Devuelve un usuario.
Nombre:	public bool ModificarUsuario(string aUser, CUsuario)
Descripción	Modifica un usuario.

Tabla 11. Especificación de la clase CNomencladora.

Nombre: public class CNomencladora	
Tipo de clase: Controladora.	
Atributo	Tipo
private static CNomencladora instance	CNomencladora
Responsabilidades	
Nombre:	private CNomencladora()
Descripción:	Constructor sin parámetros.
Nombre:	public string ConvertirProvincia(int cod)
Descripción:	Devuelve el nombre de una provincia dado su código.
Nombre:	public string ConvertirMercado(int cod)
Descripción:	Devuelve el tipo de mercado dado el código del mismo.
Nombre:	public string ConvertirProducto(int cod)
Descripción	Devuelve el nombre del producto dado su código.

Nombre:	public static CNomencladora getInstance
Descripción	Devuelve la instancia única de esta clase.

Tabla 12. Especificación de la clase DAO_Fusion.

Nombre: public class DAO_Fusion	
Tipo de clase: Controladora	
Atributo	Tipo
private DAO_Provincias Provincias	DAO_Provincias
Responsabilidades	
Nombre:	public DAO_Fusion()
Descripción:	Constructor de la clase.
Nombre:	public void Salvar(CFusion pFusion)
Descripción:	Guarda un objeto CFusion en la B.Datos
Nombre:	public CFusion Cargar(int anno, int mes)
Descripción:	Devuelve un objeto CFusion de la B.Datos
Nombre:	public void Eliminar(int anno, int mes)
Descripción:	Elimina un objeto CFusion en la B.Datos

Tabla 13. Especificación de la clase DAO_Provincias.

Nombre: public class DAO_Provincias	
Tipo de clase: Controladora	
Atributo	Tipo
private DAO_Mercados miDAO_Mercados	DAO_Mercados
Responsabilidades	
Nombre:	public DAO_Provincias()
Descripción:	Constructor de la clase.
Nombre:	public void Salvar(int anno, int mes, CProvincia Provincia)
Descripción:	Guarda en la B.Datos un objeto CProvincia de una fusion
Nombre:	public List<CProvincia> Cargar(int anno, int mes)
Descripción:	Devuelve la lista de objetos CProvincia de una fusion
Nombre:	public void Salvar(CProvincia provincia)
Descripción:	Guarda un objeto CProvincia en la B.Datos

Tabla 14. Especificación de la clase DAO_Mercados.

Nombre: public class DAO_Mercados	
Tipo de clase: Controladora	
Atributo	Tipo
private DAO_Productos miDAO_Productos	DAO_Productos
Responsabilidades	
Nombre:	public DAO_Mercados()
Descripción:	Constructor de la clase.
Nombre:	public void Salvar(int anno, int mes, int cod_provincia, CMercado Mercado)
Descripción:	Guarda en la B.Datos un objeto CMercado de una fusion
Nombre:	public void Salvar(CMercado Mercado)
Descripción:	Guarda un objeto CMercado en la B.Datos
Nombre:	public void Salvar(int cod_mercado, List<int> ListaCodProv)
Descripción:	Guarda en la B.Dastos las provincias con un mercado determinado.
Nombre:	public CMercado Cargar(int anno, int mes,int id_mercado, int cod_provincia)
Descripción:	Devuelve un objeto CMercado de una fusion
Nombre:	public List<CMercado> Cargar(int anno, int mes, int cod_provincia)
Descripción:	Devuelve una lista de objetos CMercados de una fusion
Nombre:	public void Eliminar(int idmercado)
Descripción:	Elimina un objeto CMercado en la B.Datos
Nombre:	public void Eliminar(int idmercado, List<int> ListaCodProv)
Descripción:	Elimina en la B.Datos un mercado en las provincias especificadas
Nombre:	public void Actualizar(CMercado mercado)
Descripción:	Actualiza un objeto CMercado en la B.Datos

Tabla 15. Especificación de la clase DAO_Productos.

Nombre: public class DAO_Productos	
Tipo de clase: Controladora	
Atributo	Tipo
Responsabilidades	

Nombre:	public DAO_Productos()
Descripción:	Constructor de la clase.
Nombre:	public void Salvar(int anno, int mes, int idmercado, int cod_provincia, CProducto Producto)
Descripción:	Guarda en la B.Datos un objeto CProducto de una fusion
Nombre:	public CProducto Cargar(int anno, int mes, int idmercado, int cod_provincia, int cod_producto)
Descripción:	Devuelve un objeto CProducto de una fusion
Nombre:	public void Salvar(int anno, int mes, int idmercado, int cod_provincia, List<CProducto> Productos)
Descripción:	Guarda en la B.Datos una lista de objetos CProductos de una fusion
Nombre:	public void Salvar(int idprov, int idmerc, List<int> ListaIdProd)
Descripción:	Guarda en la B.Datos los productos existentes en un mercado de una provincia
Nombre:	public List<CProductoCanasta> Cargar(int idprov, int idmerc)
Descripción:	Devuelve todos los objetos CProductoCanasta de un mercado en una provincia
Nombre:	public void Eliminar(int idprov, int idmerc, int idprod)
Descripción:	Elimina un producto del mercado de una provincia
Nombre:	public List<CProducto> Cargar(int anno, int mes, int idmercado, int cod_provincia)
Descripción:	Devuelve una lista de objetos CProductos de una fusion
Nombre:	public void Eliminar(int cod_producto)
Descripción:	Elimina un objeto CProducto en la B.Datos
Nombre:	public CCanastaGeneral CargarCanastaGeneral()
Descripción:	Devuelve el objeto CCanastaGeneral de la B.Datos
Nombre:	public CEncuestaGeneral CargarEncuestaGeneral()
Descripción:	Devuelve el objeto CEncuestaGeneral de la B.Datos
Nombre:	public void Salvar(CCanastaGeneral CanastaGeneral)
Descripción:	Guarda un objeto CCanastaGeneral en la B.Datos
Nombre:	public void Salvar(CEncuestaGeneral EncuestaGeneral)
Descripción:	Guarda un objeto CEncuestaGeneral en la B.Datos

Nombre:	public void Actualizar(CProductoCanasta ProductoCanasta)
Descripción:	Actualiza un objeto CProductoCanasta en la B.Datos

Tabla 16. Especificación de la clase DAO_Usuarios.

Nombre: public class DAO_Usuarios	
Tipo de clase: Controladora	
Atributo	Tipo
Responsabilidades	
Nombre:	public DAO_Usuarios()
Descripción:	Constructor de la clase.
Nombre:	public CUsuario Cargar(string login)
Descripción:	Devuelve un objeto CUsuario de la B.Datos
Nombre:	public void Salvar(CUsuario usuario)
Descripción:	Guarda un objeto CUsuario en la B.Datos
Nombre:	public void Eliminar(string login)
Descripción:	Elimina un objeto CUsuario en la B.Datos
Nombre:	public void Actualizar(CUsuario usuario)
Descripción:	Actualiza un objeto CUsuario en la B.Datos.

2.6 Realización de los casos de uso-diseño.

Una realización de caso de uso-diseño (RCU-D) es una colaboración en el modelo de diseño que describe cómo se realiza un caso de uso específico, y cómo se ejecuta, en términos de clases de diseño y objetos. Además cada RCU-D tiene una descripción de flujo de eventos textual, diagramas de clases que muestran sus clases de diseño participantes, y diagramas de interacción que muestran el flujo o escenario concreto de un caso de uso en términos de interacción entre objetos del diseño. (JACOBSON 2004)

En los diagramas de clases que se representan en esta sección no se incluyen los atributos y operaciones de cada clase, para ello se realizó en la sección anterior una especificación de las clases de diseño donde

se expusieron sus atributos y métodos. En el ***¡Error! No se encuentra el origen de la referencia.*** se encuentran las interfaces de usuarios del sistema correspondientes a cada uno de los casos de uso.

2.6.1 Realización del caso de uso: Autenticar.

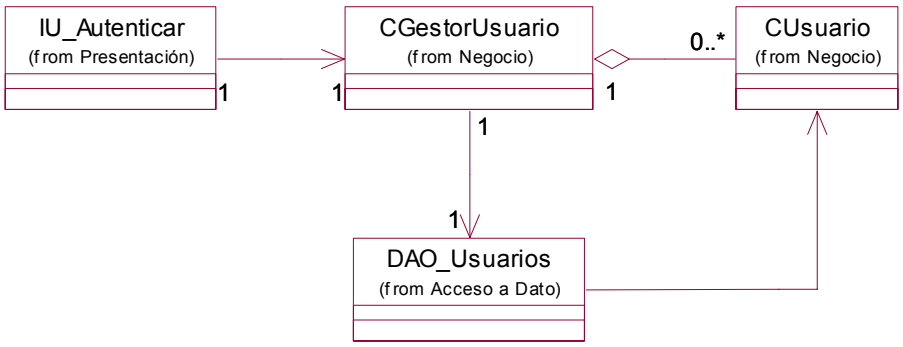

Figura 9. Diagrama de clases CU Autenticar.

Descripción textual: Este caso de uso lo inicia el usuario que pretende ser autenticado. Para ello introduce el nombre de usuario y la contraseña. Una vez realizado este paso, el sistema establece una conexión a la base de datos para verificar si el usuario existe, y en caso positivo se devuelve el rol del usuario. Por último, mediante el rol obtenido, el sistema es capaz de establecer los permisos para ese usuario activando o desactivando opciones del menú.

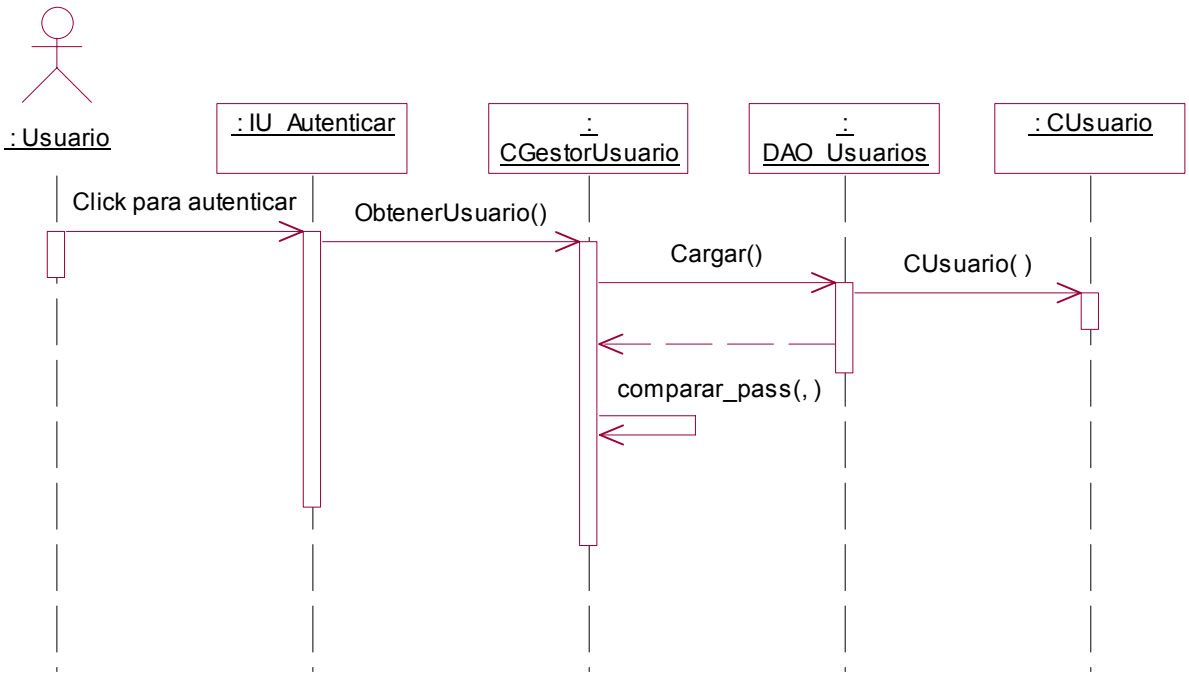

Figura 10. Diagrama de secuencia CU Autenticar.

2.6.2 Realización del caso de uso: Gestionar Usuario.

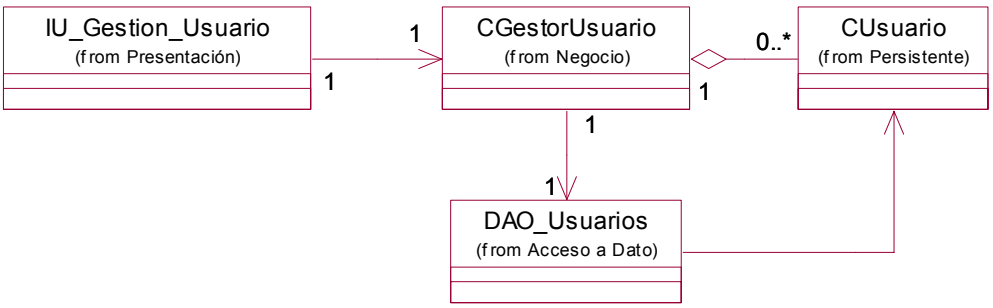

Figura 11. Diagrama de clases CU Gestionar Usuario.

Descripción textual: Este caso de uso está compuesto por tres escenarios fundamentales.

Escenario **Insertar Usuario**. El administrador introduce los datos del usuario nuevo que quiere crear. El sistema verifica que no exista un usuario con el mismo nombre. Si no existe ninguno, se inserta el nuevo usuario en la BD.

Escenario **Eliminar Usuario**. El sistema muestra una lista de los usuarios existentes. El administrador selecciona el que quiere eliminar y ejecuta la acción eliminar. El sistema hace una consulta a BD y elimina el usuario seleccionado.

Escenario **Modificar Usuario**. El sistema muestra una lista de usuarios. El administrador selecciona uno de ellos, introduce los datos que quiere modificar y ejecuta la acción modificar. El sistema hace una consulta a BD y actualiza el usuario seleccionado con los nuevos campos.

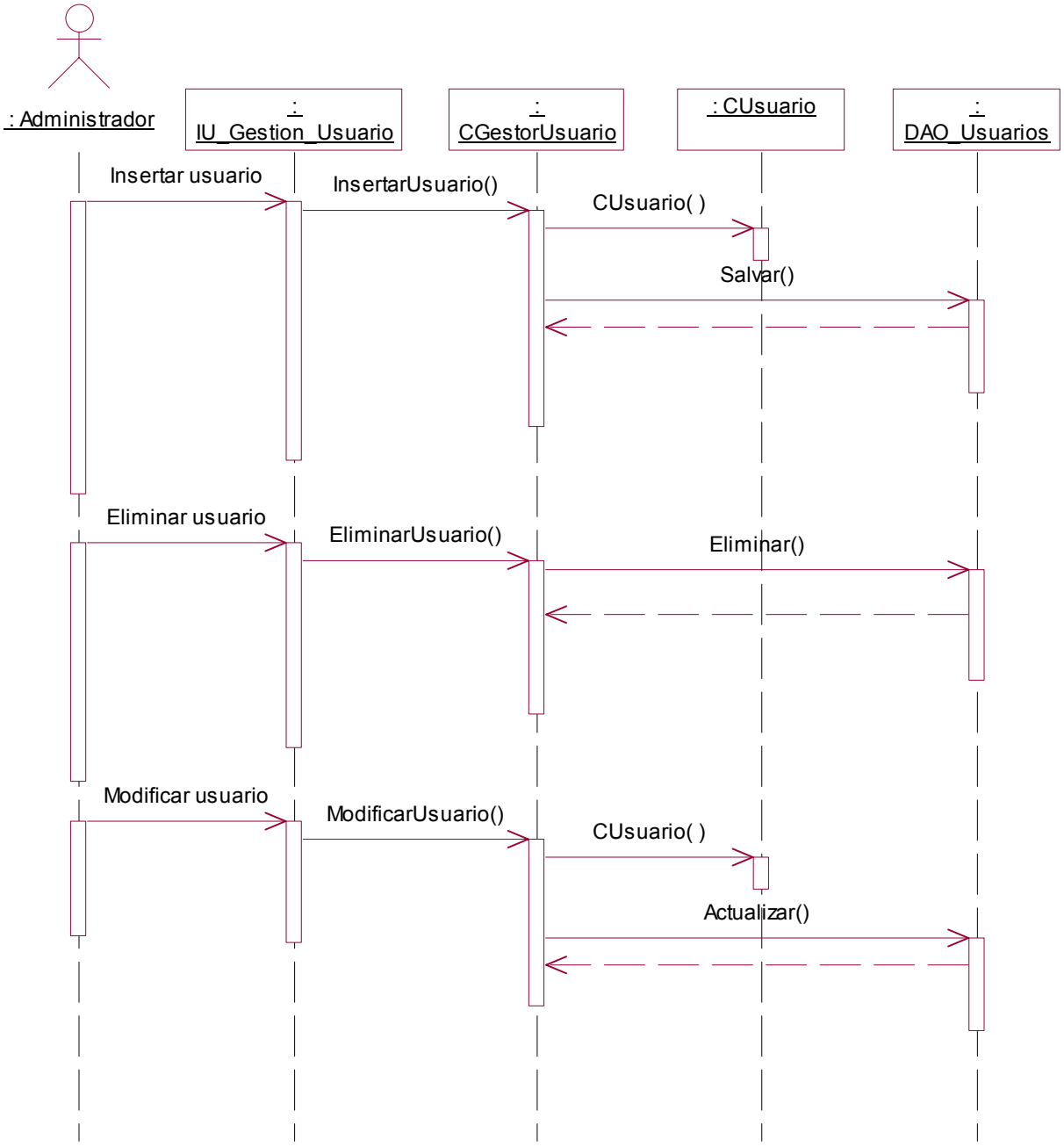

Figura 12. Diagrama de secuencia CU Gestionar Usuario.

2.6.3 Realización del caso de uso: Fusionar Ficheros.

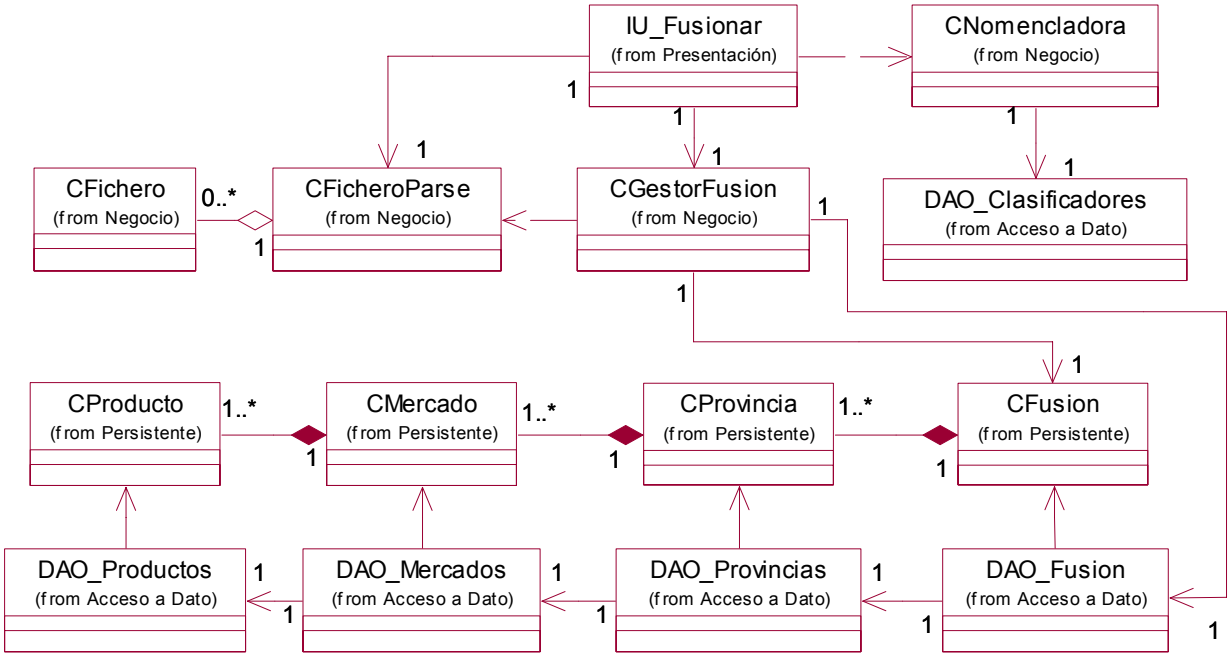

Figura 13. Diagrama de clases CU Fusionar Ficheros.

Descripción textual: El operador introduce la fecha que se quiere fusionar y selecciona el directorio a buscar. El sistema busca los ficheros que sean válidos por el nombre y válidos para esa fecha. Los ficheros son leídos y se obtiene como resultado de cada lectura un conjunto (lista) de productos. Los productos obtenidos son almacenados en la BD según el tipo de mercado y a la provincia a que pertenecen.

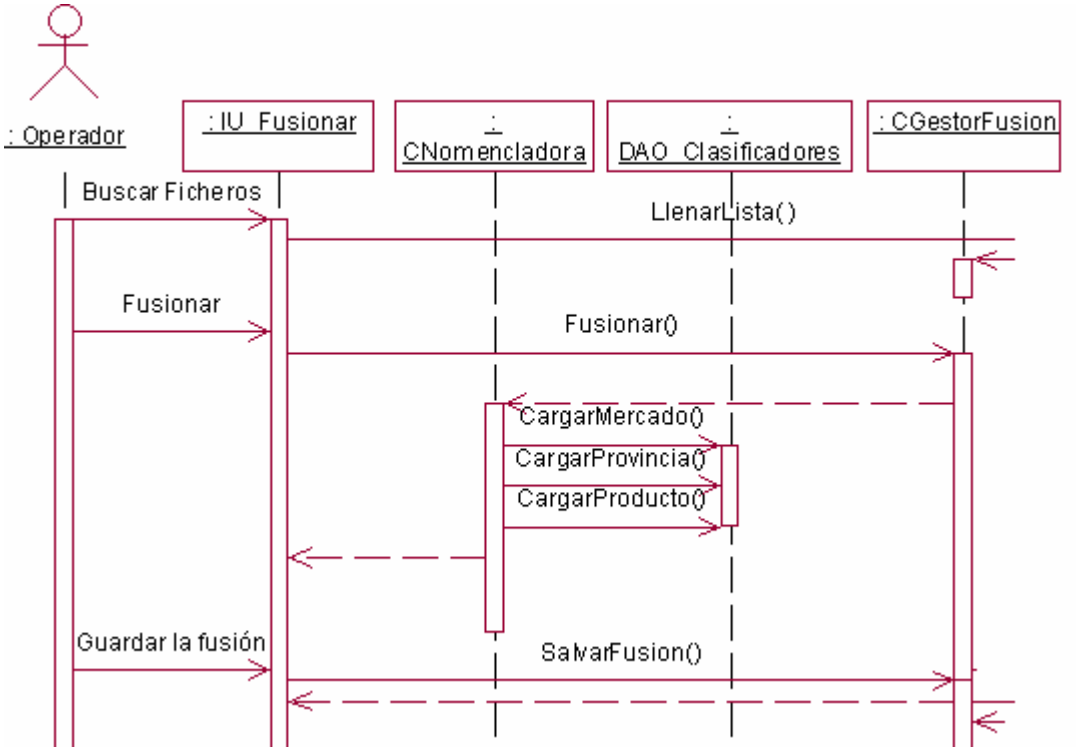

Figura 14. Diagrama de secuencia. CU Fusionar Ficheros. Parte 1.

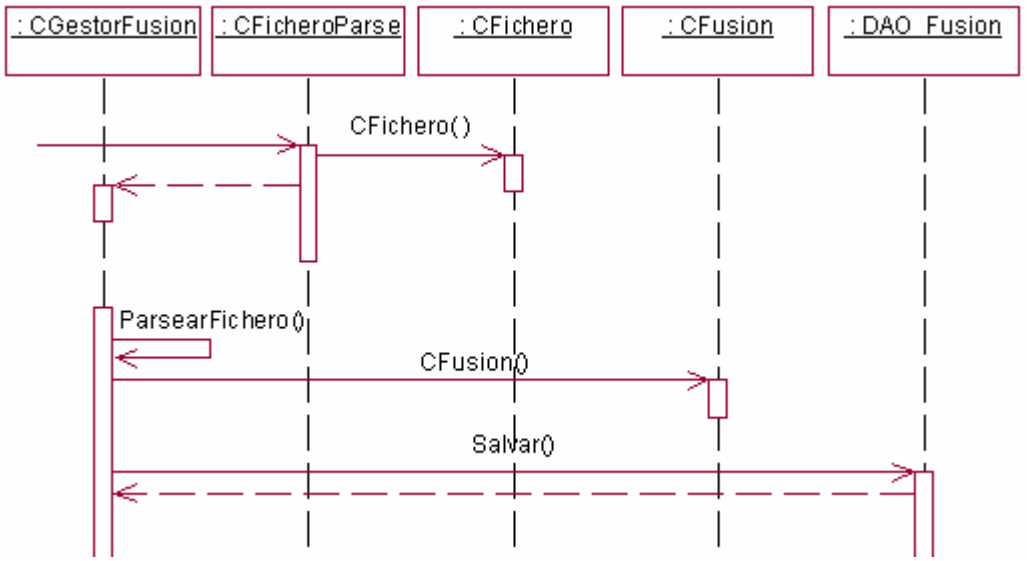

Figura 15. Diagrama de secuencia. CU Fusionar Ficheros. Parte 2.

2.6.4 Realización del caso de uso Emitir Fusión.

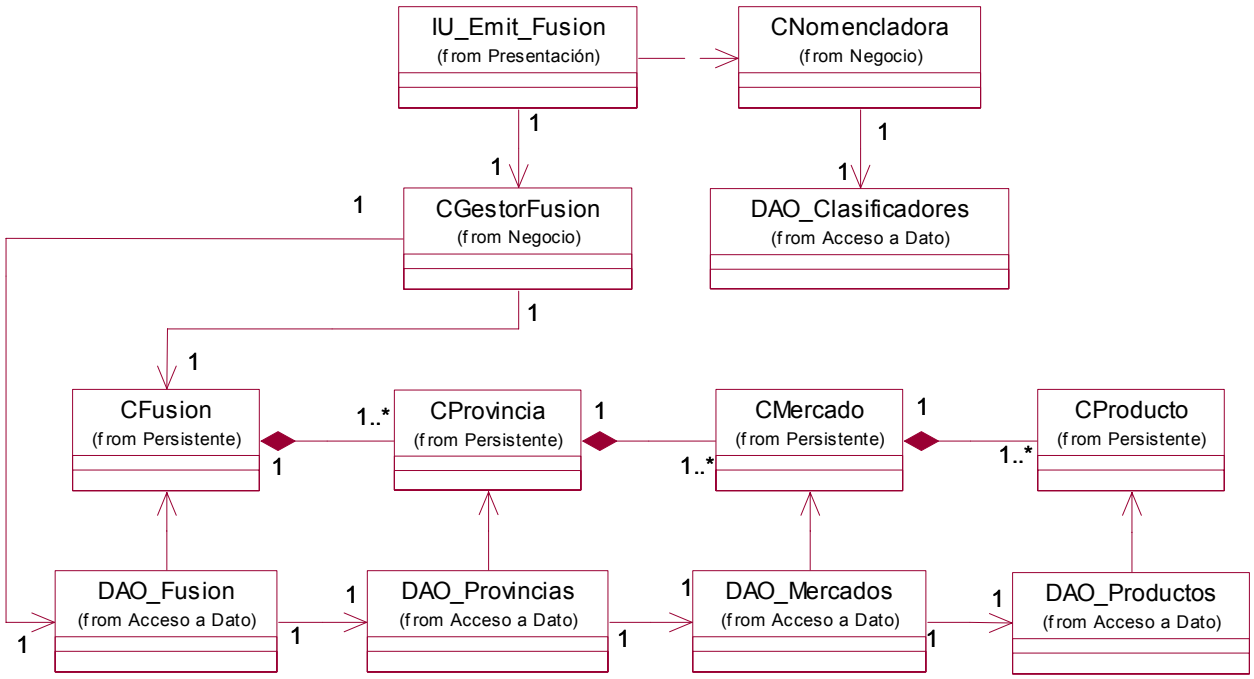

Figura 16. Diagrama de clases CU Emitir Fusión.

Descripción textual: El operador selecciona la fecha para emitir la fusión y ejecuta la acción mostrar. El sistema consulta la BD para verificar si existen datos relacionados a la fusión de la fecha seleccionada. La BD devuelve en respuesta a la consulta la fusión correspondiente a la fecha seleccionada. El sistema se encarga de visualizar los datos de la fusión a través de un componente visual.

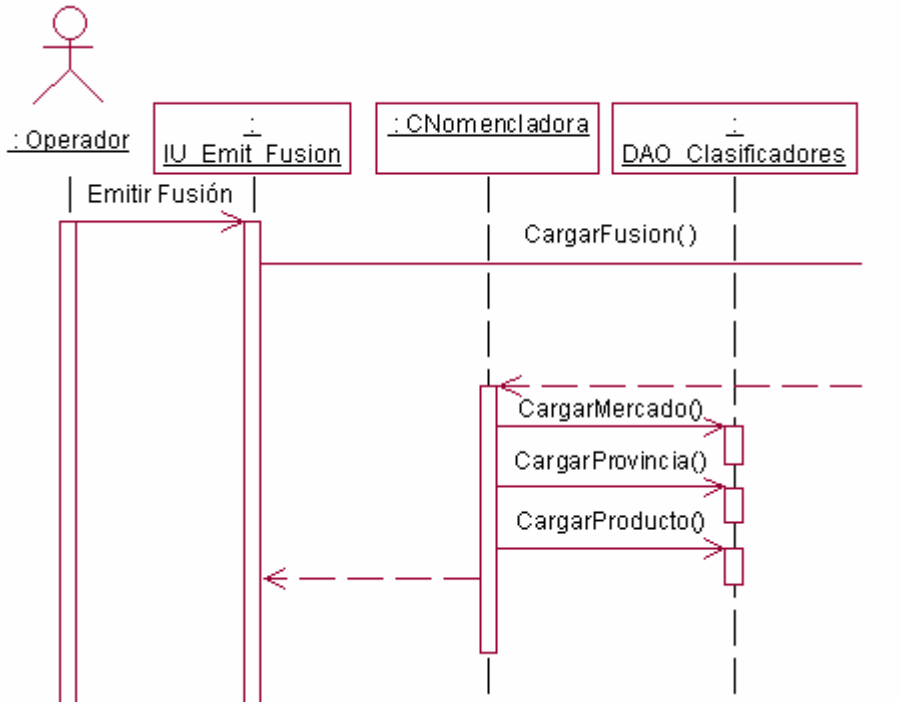

Figura 17. Diagrama de secuencia CU Emitir Fusión. Parte 1.

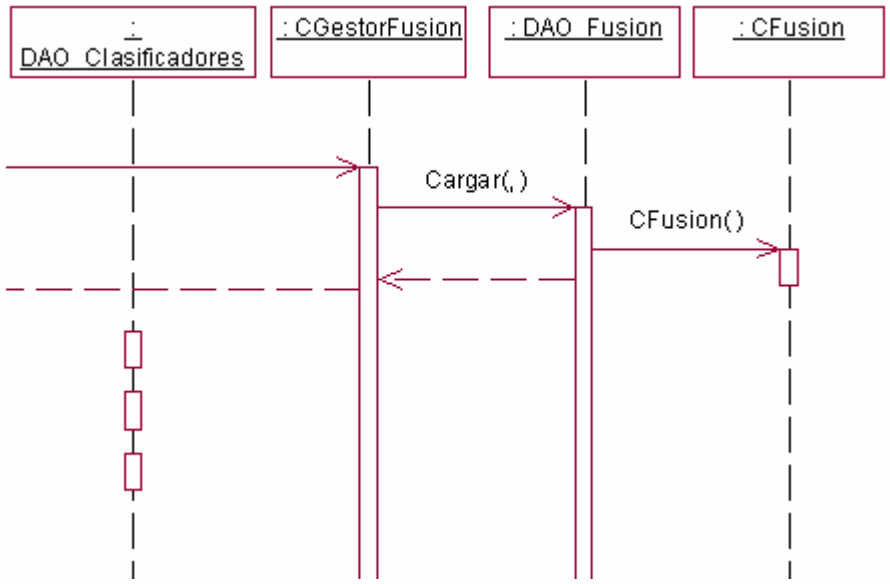

Figura 18. Diagrama de secuencia CU Emitir Fusión. Parte 2.

2.6.5 Realización del caso de uso Gestionar Canasta.

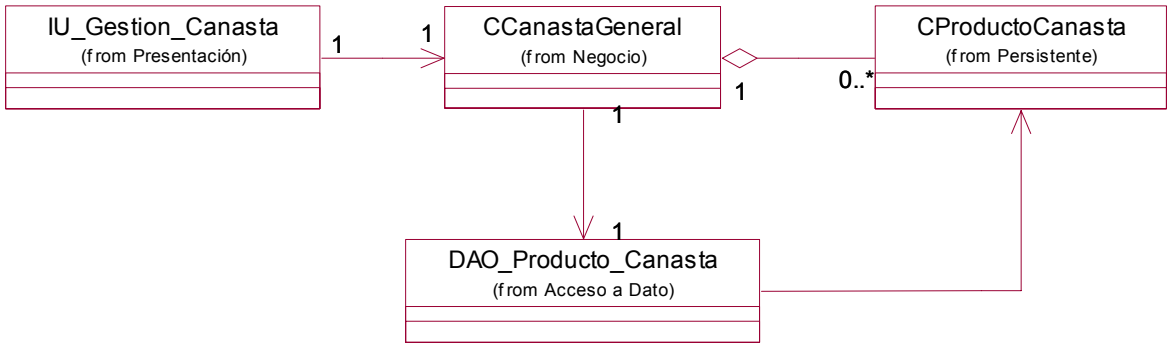

Figura 19. Diagrama de clase CU Gestionar Canasta.

Descripción textual: Este caso de uso se compone de tres escenarios principales.

Escenario **Insertar Clasificador de Producto**: El administrador introduce los datos de un nuevo clasificador de producto y ejecuta la acción insertar. El sistema consulta la base de datos para verificar si ya existe ese clasificador. En caso que no exista, el nuevo clasificador es almacenado en la BD.

Escenario **Eliminar Clasificador de Producto**: El sistema muestra un listado de todos los clasificadores de productos que existen en la BD. El administrador selecciona el que quiera eliminar y ejecuta la acción eliminar. El sistema elimina de la BD el clasificador seleccionado mediante una consulta.

Escenario **Modificar Clasificador de Producto**: El sistema muestra un listado de todos los clasificadores de productos que existen en la BD. El administrador selecciona el que quiera modificar, introduce los nuevos datos y ejecuta la acción modificar. El sistema actualiza en la BD el clasificador seleccionado mediante una consulta.

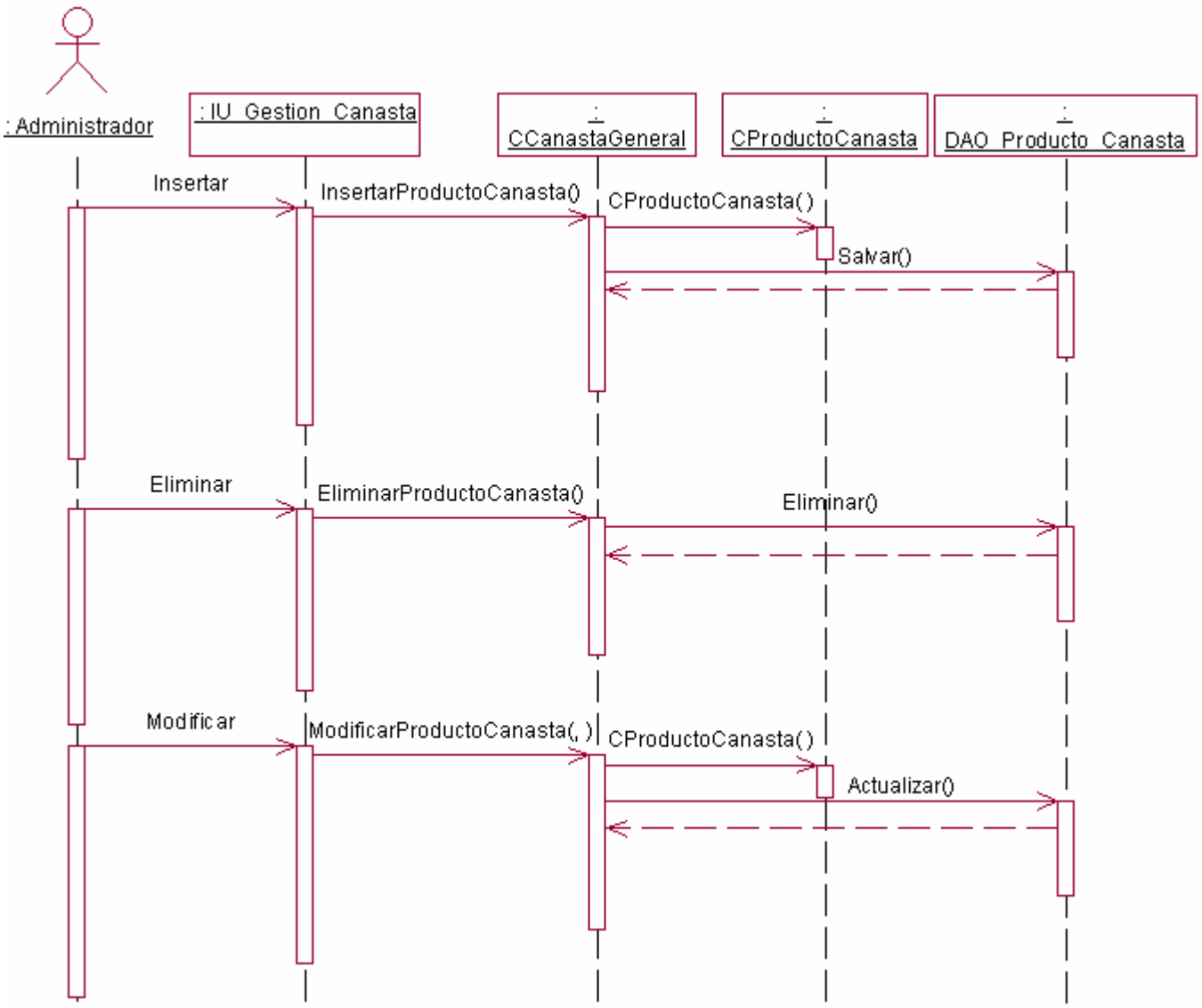

Figura 20. Diagrama de secuencia CU Gestionar Canasta.

2.6.6 Realización del caso de uso Gestionar Encuesta.

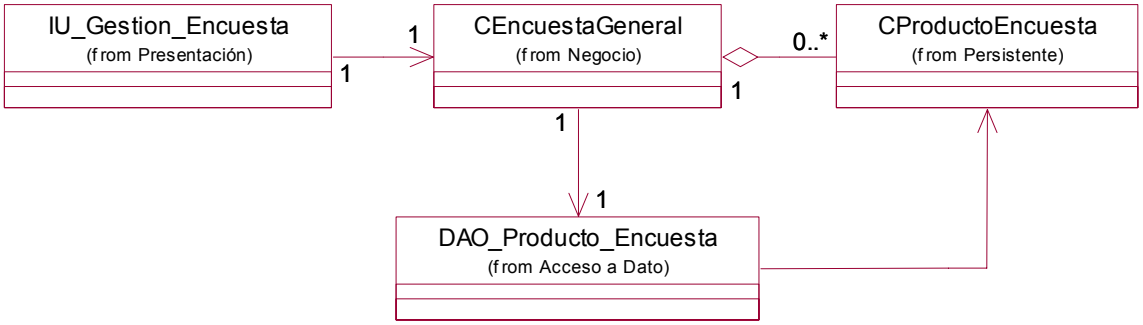

Figura 21. Diagrama de clases CU Gestionar Encuesta.

Descripción textual: Este caso de uso se compone de tres escenarios principales.

Escenario **Insertar Producto Encuesta**: El administrador introduce los datos de un nuevo producto encuestado y ejecuta la acción insertar. El sistema consulta la base de datos para verificar si ya existe ese producto encuestado. En caso que no exista, el nuevo producto encuestado es almacenado en la BD.

Escenario **Eliminar Producto Encuesta**: El sistema muestra un listado de todos los productos encuestados que existen en la BD. El administrador selecciona el que quiera eliminar y ejecuta la acción eliminar. El sistema elimina de la BD el producto encuestado seleccionado mediante una consulta.

Escenario **Modificar Producto Encuesta**: El sistema muestra un listado de todos los productos encuestados que existen en la BD. El administrador selecciona el que quiera modificar, introduce los nuevos datos de la nueva encuesta y ejecuta la acción modificar. El sistema actualiza en la BD el producto encuestado seleccionado mediante una consulta.

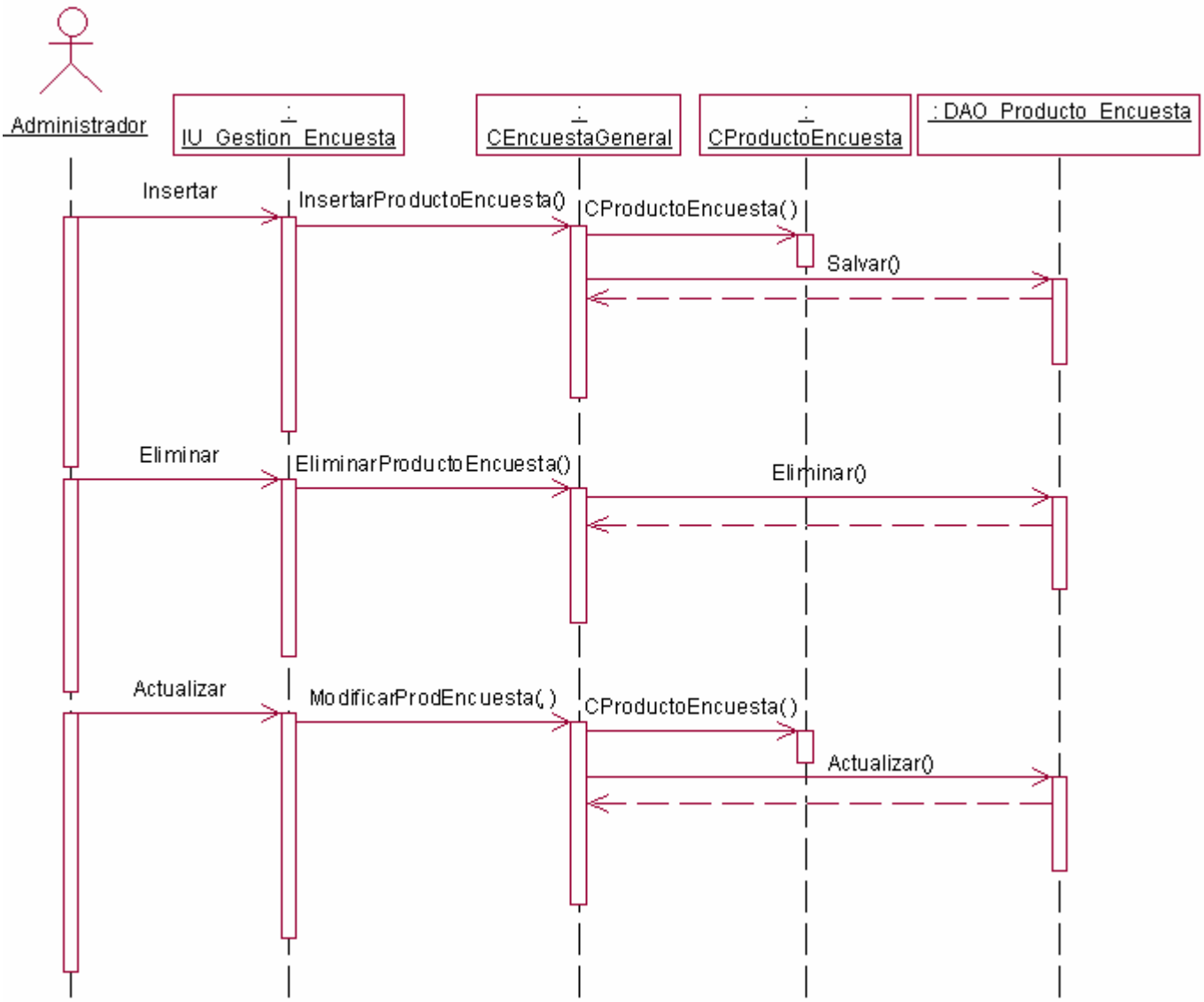

Figura 22. Diagrama de secuencia CU Gestionar Encuesta.

2.6.7 Realización del caso de uso Gestionar Clasificador de Mercado.

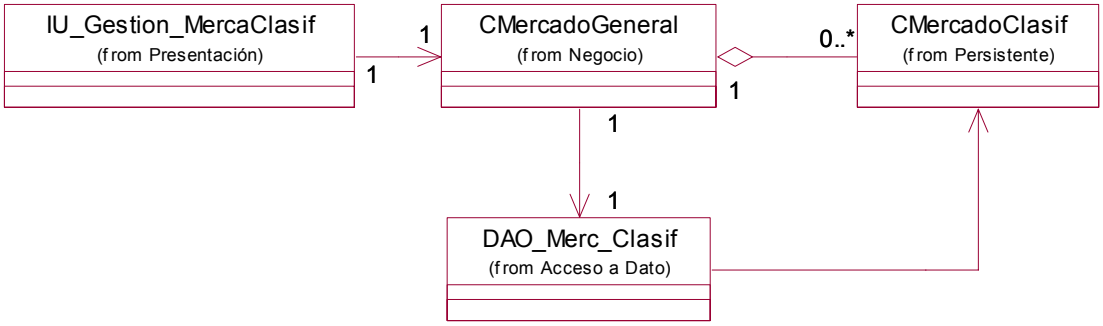

Figura 23. Diagrama de clases CU Gestionar Clasificador de Mercado.

Descripción textual: Este caso de uso se compone de tres escenarios principales.

Escenario **Insertar Clasificador de Mercado**: El administrador introduce los datos de un nuevo clasificador de mercado y ejecuta la acción insertar. El sistema consulta la base de datos para verificar si ya existe ese clasificador. En caso que no exista, el nuevo clasificador es almacenado en la BD.

Escenario **Eliminar Clasificador de Mercado**: El sistema muestra un listado de todos los clasificadores de mercado que existen en la BD. El administrador selecciona el que quiera eliminar y ejecuta la acción eliminar. El sistema elimina de la BD el clasificador seleccionado mediante una consulta.

Escenario **Modificar Clasificador de Mercado**: El sistema muestra un listado de todos los clasificadores de mercado que existen en la BD. El administrador selecciona el que quiera modificar, introduce los nuevos datos que quiere para el clasificador y ejecuta la acción modificar. El sistema actualiza en la BD el clasificador seleccionado mediante una consulta.

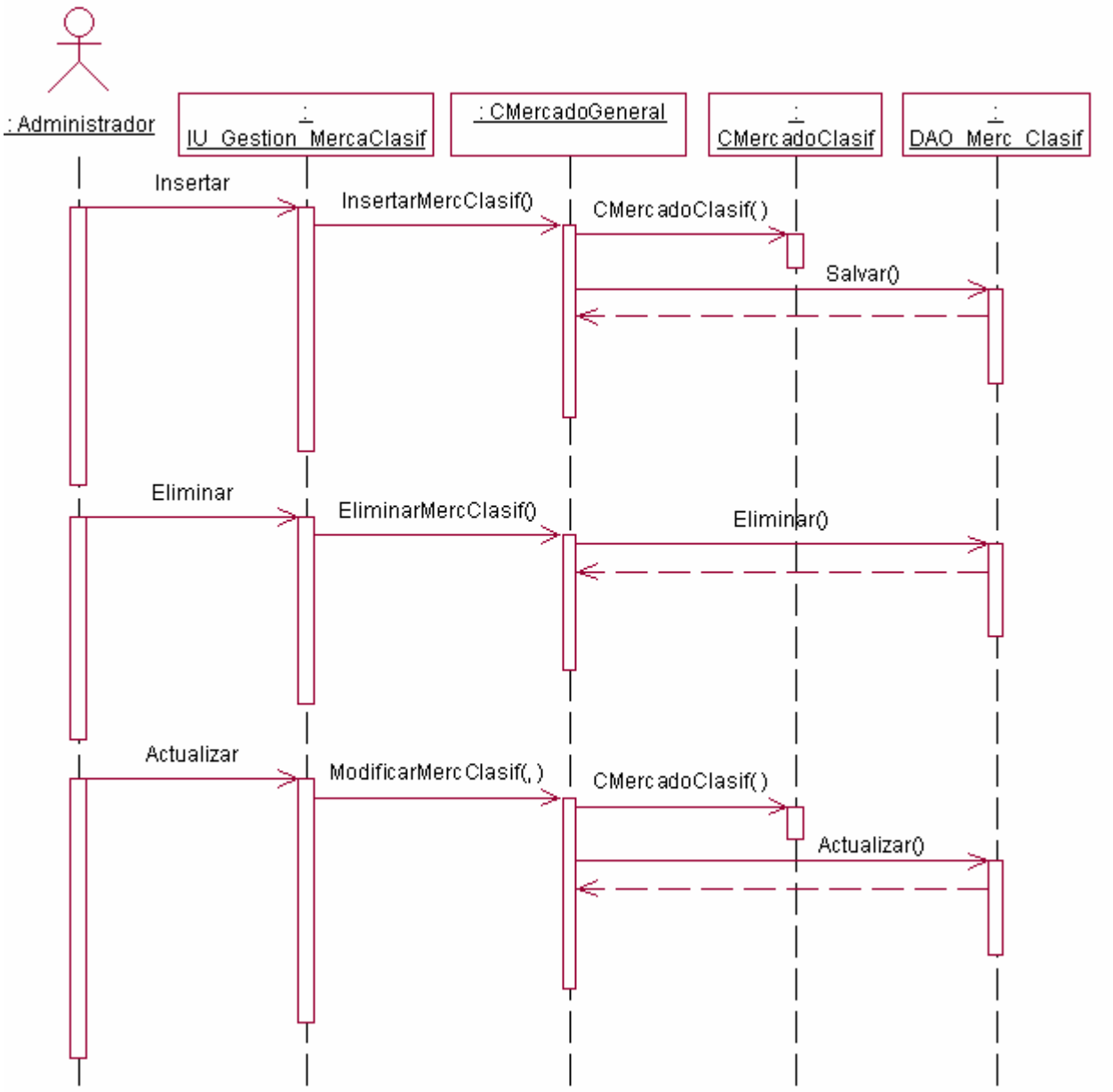

Figura 24. Diagrama de secuencia CU Gestionar Clasificador de Mercado.

2.6.8 Realización del caso de uso Gestionar Clasificador de Provincia.

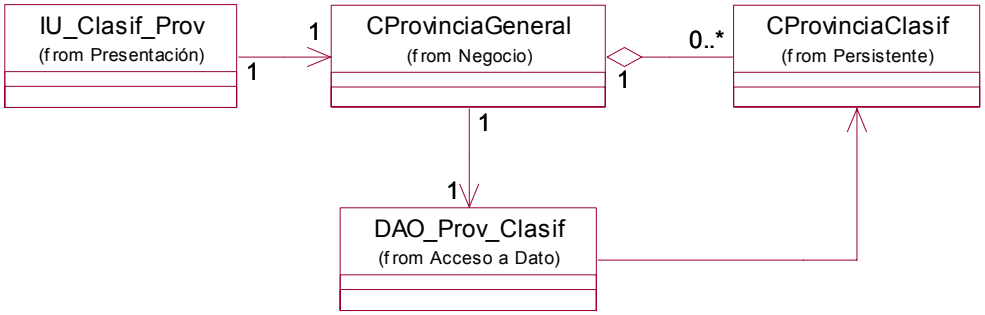

Figura 25. Diagrama de clases CU Gestionar Clasificador de Provincia.

Descripción textual: Este caso de uso se compone de tres escenarios principales.

Escenario **Insertar Clasificador de Provincia**: El administrador introduce los datos de un nuevo clasificador de provincia y ejecuta la acción insertar. El sistema consulta la base de datos para verificar si ya existe ese clasificador. En caso que no exista, el nuevo clasificador es almacenado en la BD.

Escenario **Eliminar Clasificador de Provincia**: El sistema muestra un listado de todos los clasificadores de provincia que existen en la BD. El administrador selecciona el que quiera eliminar y ejecuta la acción eliminar. El sistema elimina de la BD el clasificador seleccionado mediante una consulta.

Escenario **Modificar Clasificador de Provincia**: El sistema muestra un listado de todos los clasificadores de provincia que existen en la BD. El administrador selecciona el que quiera modificar, introduce los nuevos datos que quiere para el clasificador y ejecuta la acción modificar. El sistema actualiza en la BD el clasificador seleccionado mediante una consulta.

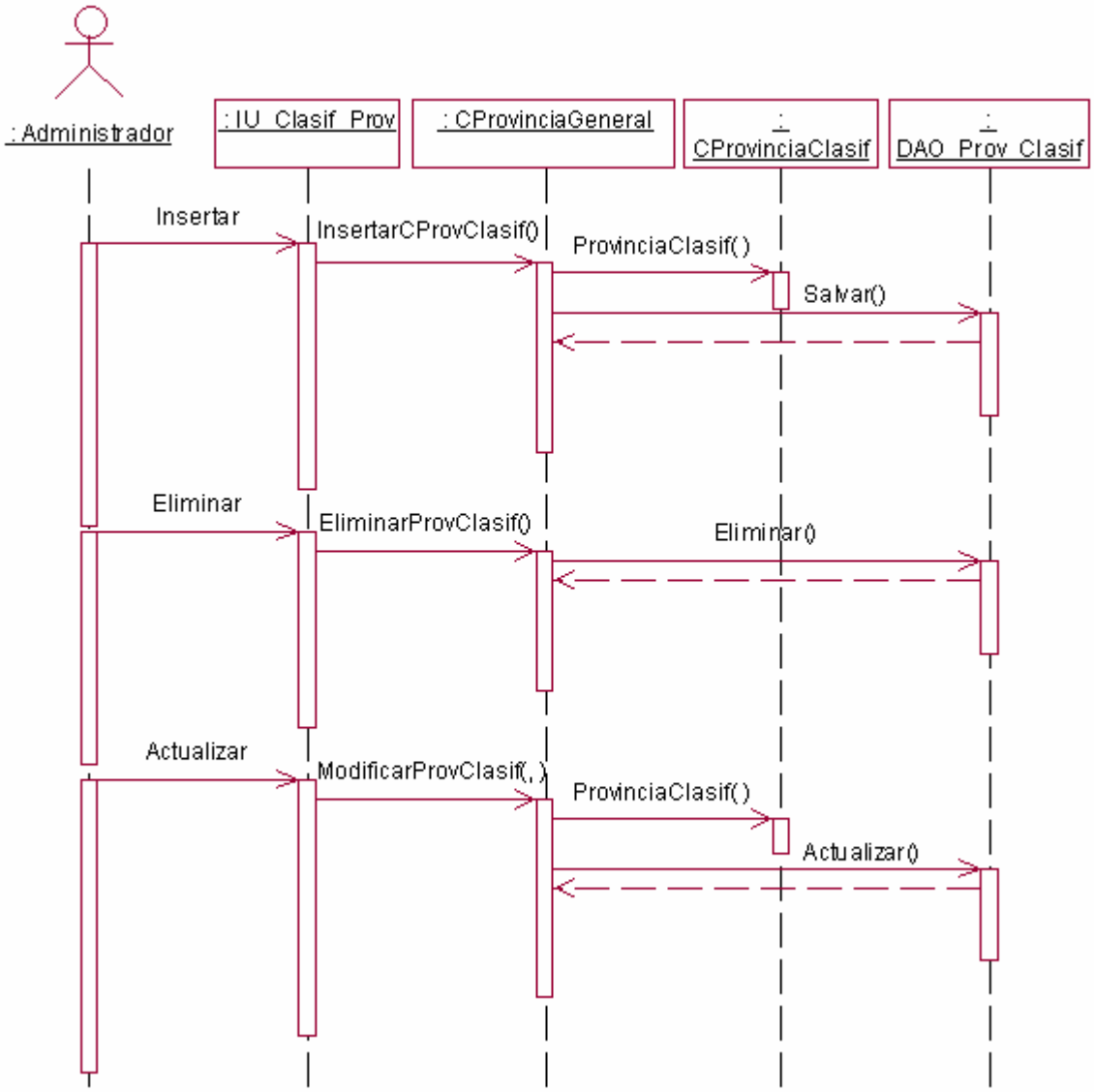

Figura 26. Diagrama de secuencia CU Gestionar Clasificador de Provincia.

2.6.9 Realización del caso de uso Emitir Tablas de Índices.

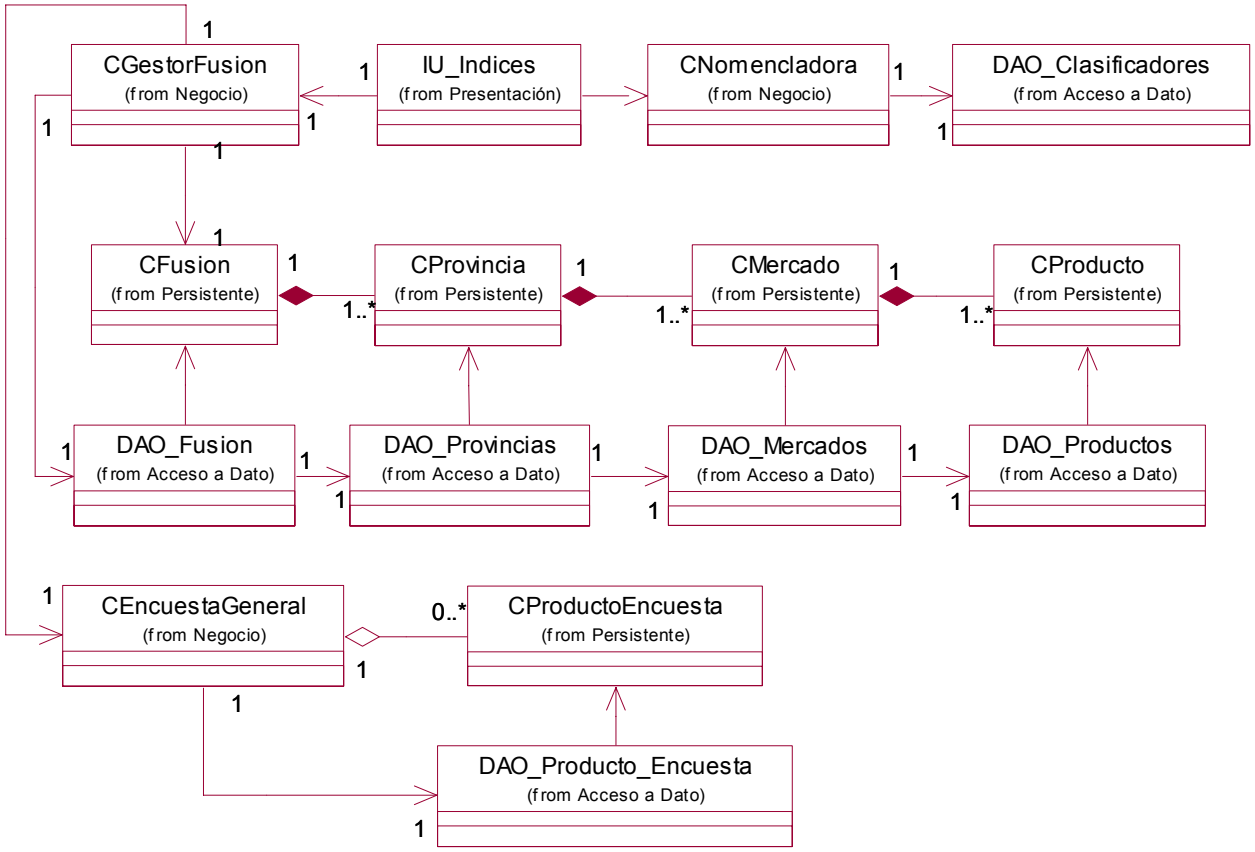

Figura 27. Diagrama de clases CU Emitir Tablas de Índices.

Descripción textual: El operador solicita emitir las tablas de índices indicando la fecha y el tipo de mercado a emitir. El sistema consulta a la BD y obtiene la información relacionada a ese mercado y a esa fecha. El sistema consulta la base de datos para ir obteniendo los precios del período base a medida que va calculando los índices para cada producto. El sistema muestra a través de un componente visual los índices calculados a cada producto en la fecha y para el tipo de mercado seleccionados.

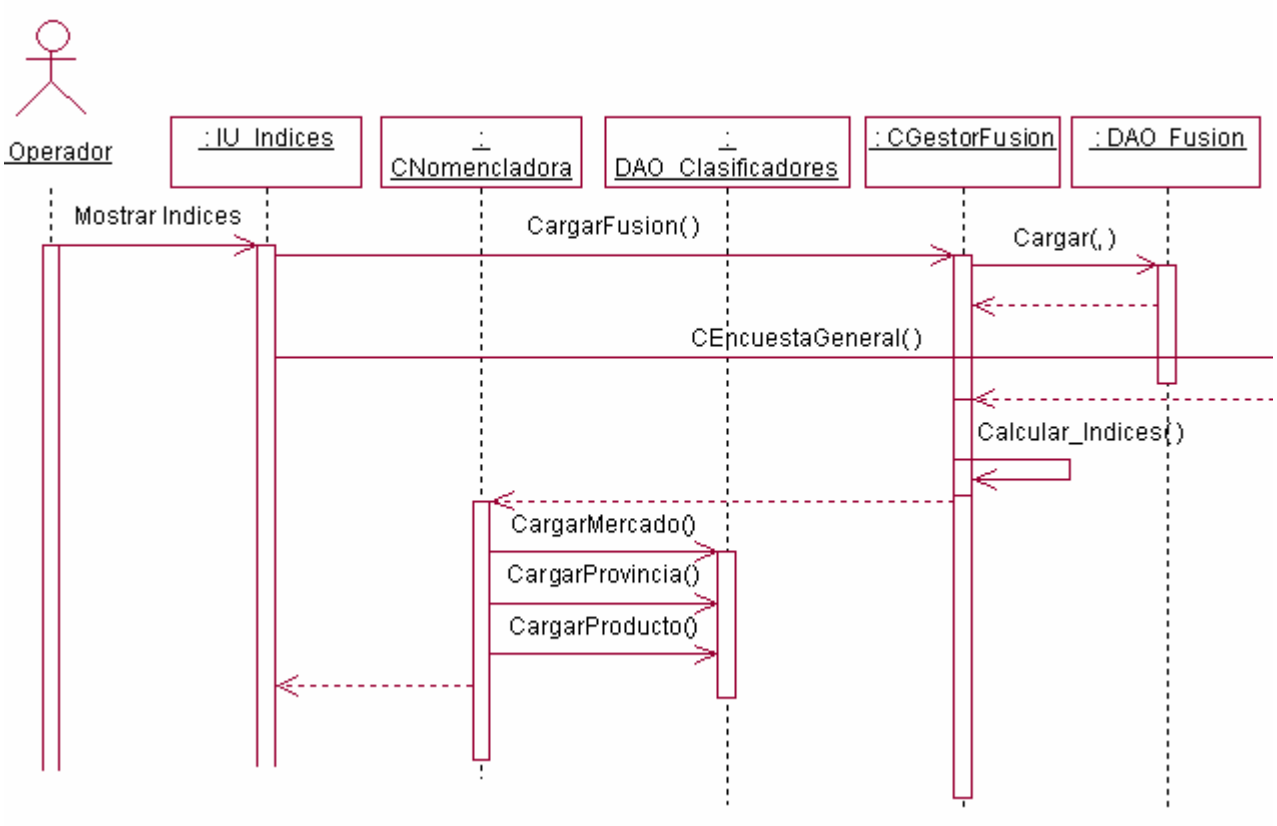

Figura 28. Diagrama de Secuencia CU Emitir Tablas de Índices. Parte 1.

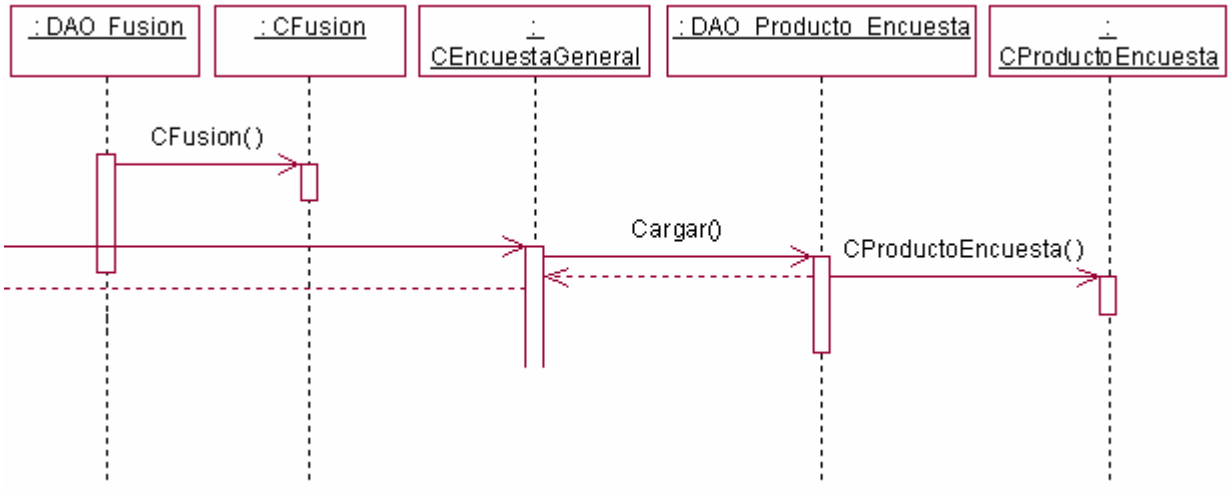

Figura 29. Diagrama de Secuencia CU Emitir Tablas de Índices. Parte 2.

2.6.10 Realización del caso de uso Emitir Tablas de Precios.

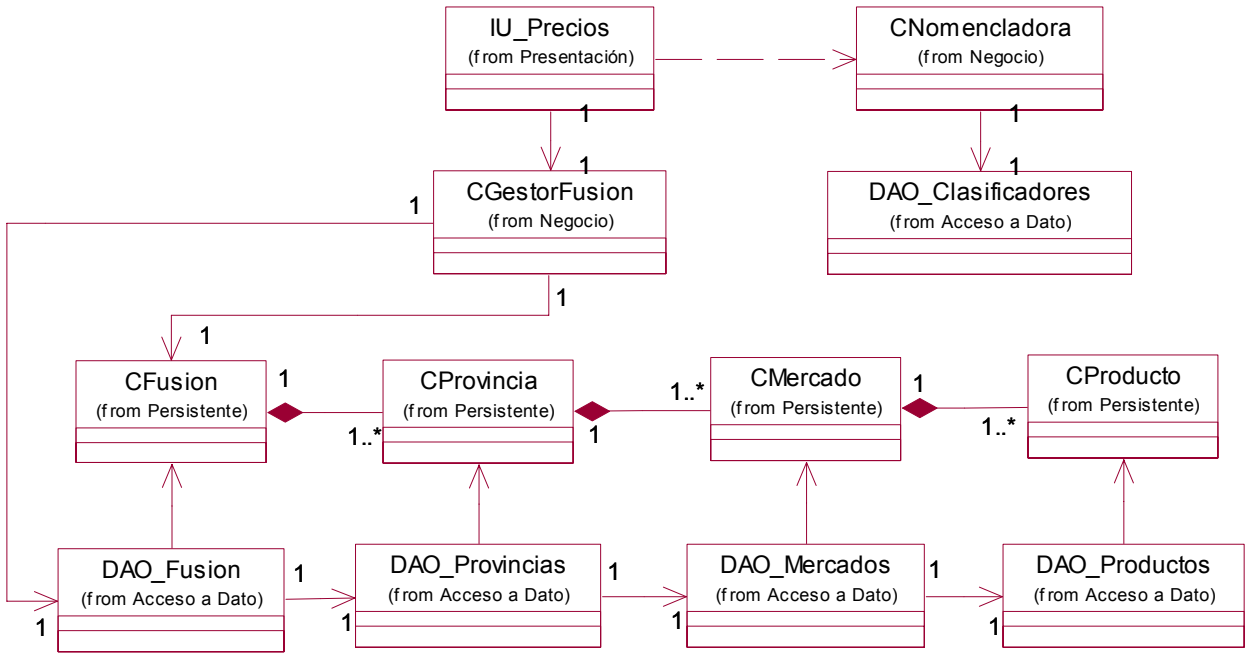

Figura 30. Diagrama de clases CU Emitir Tablas de Precios.

Descripción textual: El operador solicita emitir las tablas de precios indicando la fecha y el tipo de mercado a emitir. El sistema consulta a la BD y obtiene la información relacionada a ese mercado y a esa fecha. El sistema consulta la base de datos para ir obteniendo los datos del clasificador de producto a medida que se vaya procesando cada producto del mercado seleccionado. El sistema muestra a través de un componente visual los datos correspondientes a cada producto en la fecha y para el tipo de mercado seleccionados.

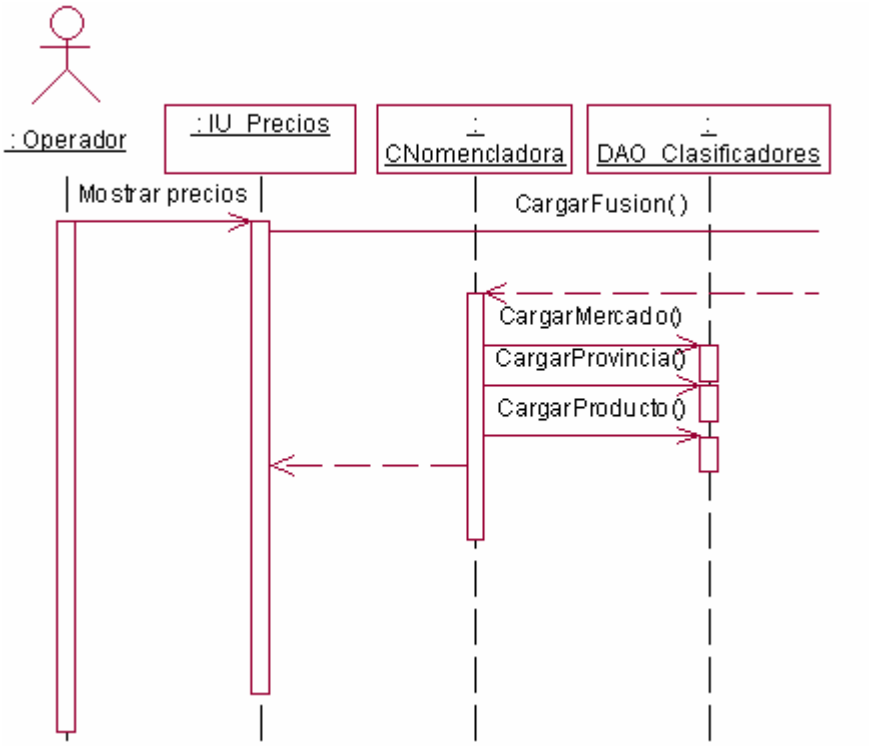

Figura 31. Diagrama de Secuencia CU Emitir Tablas de Precios. Parte 1.

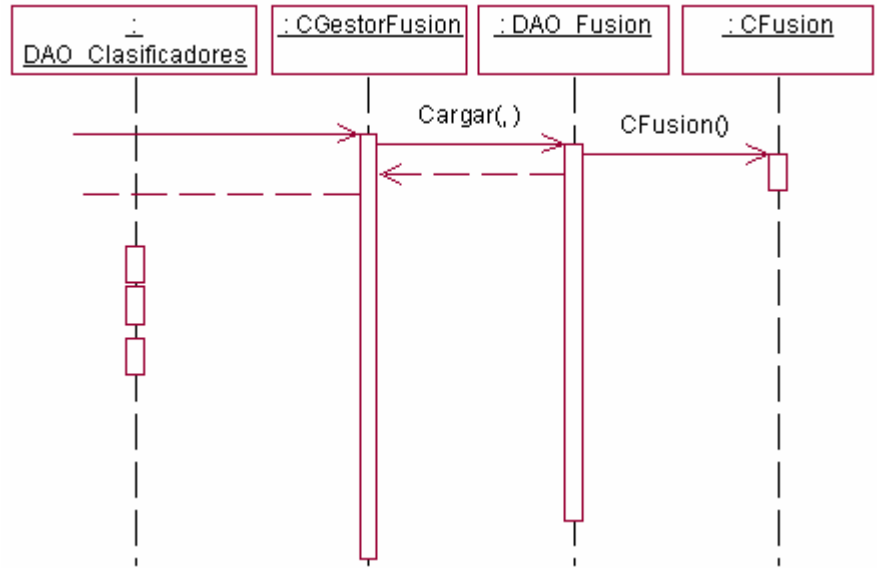

Figura 32. Diagrama de Secuencia CU Emitir Tablas de Precios. Parte 2.

2.7 Patrones de diseño utilizados.

Para el desarrollo del diseño, hemos aplicado un conjunto de patrones que nos permiten en cierta medida logar un diseño más flexible, elegante y reutilizable. En esta sección se exponen y se explican los diferentes patrones que han sido utilizados en el diseño.

2.7.1 Patrones GRASP.

Los patrones GRASP *(General Responsibility Assignment Software Patterns)*, no compiten con los patrones de diseño; son una guía para ayudar a encontrar los patrones de diseño (que son más concretos). Pero de cierta forma, ha sido aplicado el catálogo de patrones GRASP que se referencia en (LARMAN 2004).

Bajo acoplamiento:

Plantea que las dependencias entre las clases deben ser en el menor grado posible. En nuestro diseño propuesto existen pocas dependencias entre las clases, por lo que se ha logrado que el acoplamiento sea bajo y las posibilidades de poder reutilizar el diseño aumenten. Algo que favorece a nuestro diseño en ese sentido es la no existencia de un árbol profundo de herencia. Por lo general, la gran profundidad de un árbol de herencia conlleva a subir considerablemente los niveles de acoplamiento en los sistemas. En los diagramas de clases generados en la sección anterior, se observa más detalladamente el nivel de acoplamiento de las clases del diseño.

Experto:

Plantea que la responsabilidad de realizar una tarea es de la clase que contiene los datos involucrados. En nuestro diseño existen varias clases que poseen este patrón. La clase CFichero es la encargada de realizar las operaciones sobre los ficheros, ya que es esta la que contiene los datos necesarios para realizar dichas operaciones. Las clases CCanastaGeneral, CEncuestaGeneral, entre otras, son especialistas en las operaciones que se realizan a partir de los atributos contenidos en ellas. Con la

aplicación de este patrón se ha logrado una mejor especialización de las clases, esto es, no asignar responsabilidades a una clase, habiendo otra que pueda manejarlas de forma más eficiente y mejor. Además, se ha conservado el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide.

Alta cohesión:

Su principio se basa en que cada clase debe tener responsabilidades moderadas en un área funcional y colaborar con otras para llevar a cabo las tareas. Un ejemplo de cómo se cumple este patrón en nuestro diseño es: para la interacción con la base de datos existen varios conjuntos de clases, las clases DAO realizan una parte de las funciones, colaboran con las clases Map para recuperar y guardar objetos, y de esta forma se comparte el esfuerzo de estas tareas. Otra regla que hace cumplir este patrón en nuestro diseño es el número relativamente pequeño de operaciones en las clases, con una importante funcionalidad relacionada y poco trabajo por hacer.

Controlador:

Se basa en asignarle a clases la responsabilidad de controlar el flujo de eventos del sistema. En nuestro diseño hemos asignado responsabilidades como esta a clases que controlan el flujo en procesos similarmente funcionales. Por ejemplo, se ha creado una clase controladora para manejar los eventos relacionados con los usuarios, otras para manejar el flujo de eventos de cada uno de los clasificadores, entre otras. Este patrón ha facilitado una mayor centralización de las actividades, ya que las clases controladoras no realizan todas las actividades, sino que las delegan en otras clases con las que mantiene un modelo de alta cohesión.

2.7.2 Patrón Data Access Object.

Este patrón, más conocido como DAO, (LAGO 2007), se ha utilizado con el objetivo de abstraer y encapsular todos los accesos a la fuente de datos, logrando así desacoplar la lógica de negocios de la lógica de acceso a datos. El DAO, en este caso, maneja la conexión con la fuente de clases Map, para

obtener y almacenar datos. Este patrón presenta la característica de ocultar completamente los detalles de implementación de la fuente de datos a sus clientes. Otra característica importante es que la interfaz que él representa no cambia cuando cambia la implementación de la fuente de datos.

Hemos logrado mediante su uso, resolver el problema de disminuir considerablemente la dependencia y el acoplamiento entre la capa del negocio y la de acceso a datos, de igual manera, hemos reducido la complejidad de la implementación de la lógica del negocio.

Es importante destacar que generalmente el uso de este patrón, aumenta la complejidad del diseño, debido principalmente al conjunto de clases que colaboran en el DAO, ya que de forma general, se requiere de una clase DAO para modelar una tabla del modelo relacional. Pero ese no es el caso para nuestro diseño; para la lógica de acceso a datos hemos requerido solamente de diez clases DAO, las cuales no son grandes y su nivel de complejidad estimado es pequeño, además influye positivamente su uso desde el punto de vista que, si las clases del paquete persistente hubieran tomado esta responsabilidad, su hubieran sobrecargado demasiado y hubieran colaborado a incrementar la complejidad del sistema, el tamaño de clases, y lo más seguro, la carencia de cohesión en sus métodos.

A continuación ilustramos uno de los ejemplos donde utilizamos el patrón DAO.

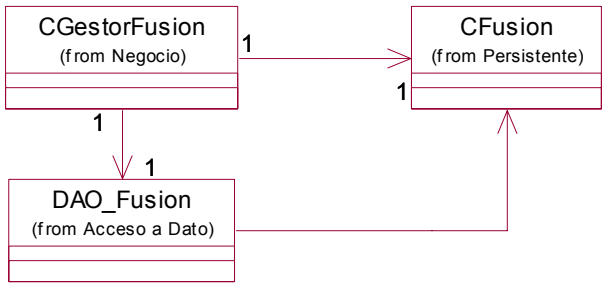

Figura 33. Ejemplo de aplicación del patrón DAO.

2.7.3 Patrón Singleton.

El patrón Singleton pertenece al conjunto de patrones GOF (Gang of Four), dentro de ellos al grupo de patrones de creación. Garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a esta instancia.

Este patrón define una operación getInstance que permite que los clientes accedan a su única instancia. getInstance es una operación de clase static en C#. También puede ser responsable de crear su única instancia. Generalmente se usa cuando deba haber una instancia de una clase y esta deba ser accesible por los clientes desde un punto de acceso conocido. (WELICKI 2007)

El funcionamiento de este patrón es muy sencillo y podría reducirse a los siguientes conceptos: (WELICKI 2007)

1. Ocultar el constructor de la clase Singleton, para que los clientes no puedan crear instancias.
2. Declarar en la clase Singleton una variable miembro privada que contenga la referencia a la instancia única que queremos gestionar.
3. Proveer en la clase Singleton una función o propiedad que brinde acceso a la única instancia gestionada por el Singleton. Los clientes acceden a la instancia a través de esta función o propiedad.

La clase CNomencladora resuelve el problema de obtener los nombres de los productos, de los mercados y de las provincias a partir de un código, y es utilizada en varios casos de uso. Para resolver el problema de garantizar una sola instanciación de la misma se ha aplicado el patrón Singleton a esta clase.

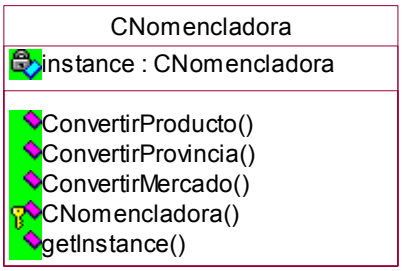

Figura 34. Aplicación del patrón Singleton.

2.8 Conclusiones.

Durante esta etapa de diseño logramos dar una solución factible para cada funcionalidad requerida por el sistema y proporcionamos una visión clara y precisa de los elementos constitutivos del diseño propuesto mediante la descripción detallada de todos los escenarios de flujo de información y de los diagramas realizados.

La arquitectura propuesta fue cumplida en todo momento y nos proporcionó una base adecuada para dar cumplimiento a todas las especificaciones de los casos de uso. Se logró un diseño flexible y reusable mediante el uso de patrones de diseño, capaz de adaptarse a nuevas funcionalidades que se pretendan agregar al sistema.

El uso de la herramienta CASE Rational Rose facilitó de forma eficiente realizar el modelado visual de los diagramas de clases y de interacción.

Capítulo 3 “Evaluación del modelo de diseño propuesto”.

3.1 Introducción

El flujo de trabajo de diseño se propone crear una entrada apropiada para las tareas de implementación subsiguientes. El flujo de trabajo de implementación describe cómo los elementos del modelo de diseño se implementan en términos de componentes y cómo se organizan los mismos. Pero primero debemos estar seguros de que el modelo de diseño cumpla con la eficiencia requerida, que sea fiable, mantenible, flexible y reusable.

En este capítulo se pretende medir el diseño propuesto en el capítulo anterior para demostrar su eficiencia. Para ello se seleccionan una serie de métricas propuestas por (PRESSMAN 2005), las cuales se aplican al modelo de diseño propuesto. Posterior a este paso, se hace un análisis y valoración críticos de los resultados obtenidos, a partir de los cuales se generan conclusiones finales y se hacen algunas recomendaciones para el enriquecimiento de este trabajo.

3.2 Tamaño de clase (TC).

Esta es una de las métricas propuestas por Lorenz y Kidd, referenciada en (PRESSMAN 2005). El tamaño de una clase (TC), puede medirse determinando el total de métodos (TM), tanto heredados como privados de la instancia, que se encapsulan dentro de la clase y que se consideran complejos.

Se indica que valores grandes para TC implican que la clase debe tener bastante responsabilidad. Esto reducirá la reutilización de la clase y complicará la implementación y las pruebas. En general, métodos y atributos heredados o públicos deben ser ponderados con mayor importancia, cuando se determina el TC. Métodos y atributos privados, permiten la especialización y son más propios del diseño.

También se pueden calcular los promedios para el número de atributos y operaciones de clases. Cuanto menor sea el valor del promedio para el tamaño, será más posible que las clases dentro del sistema puedan reutilizarse.

Lorenz y Kidd establecen un umbral de 40 ó 20 en dependencia si la clase es interfaz de usuario o no. Para la aplicación de esta métrica no tendremos en cuenta las clases de interfaz de usuario, por lo que tomaremos la escala: para valores de TC≤20 la clase es pequeña, para valores de 20<TC≤30 la clase es mediana, y para valores de TC>30 entonces la clase es grande.

Teniendo en cuenta estos criterios, apliquemos esta métrica a algunas de las clases de diseño propuestas en el capítulo anterior, tomando solo la cantidad de métodos como referencia para calcular el TC.

Tabla 17. Tamaños de clase para algunas clases del sistema.

Clase	Total de métodos (TM)	Tamaño de clase (TC)	Clasificación
CProducto	4	4	Pequeña
CMercado	8	8	Pequeña
CProvincia	8	8	Pequeña
CFusion	13	13	Pequeña
CProductoEncuesta	7	7	Pequeña
CEncuestaGeneral	8	8	Pequeña
CProductoCanasta	9	9	Pequeña
CCanastaGeneral	8	8	Pequeña
CFichero	11	11	Pequeña
CFicheroParse	9	9	Pequeña
CUsuario	5	5	Pequeña
CGestorUsuario	9	9	Pequeña
CMercadoClasif	4	4	Pequeña
CMercadoGeneral	8	8	Pequeña
CProvinciaClasif	4	4	Pequeña
CProvinciaGeneral	8	8	Pequeña
CNomencladora	5	5	Pequeña
DAO_Fusion	4	4	Pequeña
DAO_Provincias	4	4	Pequeña
DAO_Mercados	9	9	Pequeña

DAO_Productos	14	14	Pequeña
DAO_Usuarios	5	5	Pequeña
DAO_Producto_Canasta	5	5	Pequeña
DAO_Producto_Encuesta	5	5	Pequeña
DAO_Prov_Clasif	5	5	Pequeña
DAO_Merc_Clasif	5	5	Pequeña
DAO_Clasificadores	4	4	Pequeña

3.2.1 Análisis de los resultados:

Para tener una mejor visión de los resultados, hallaremos un promedio P(m) del total de métodos por clase en el diseño propuesto, o sea, un promedio del tamaño de clase para nuestro diseño sumando cada uno de los totales de métodos y dividiéndose después entre el total de clases N.

$$P(m) = \frac{\sum (TM)}{N}$$

Resolviendo:

$$P(m) = \frac{188}{27} = 6.9 \approx 7$$

El resultado obtenido representa que las clases del diseño propuesto tienen un tamaño promedio de 7 métodos. Este resultado, representa en la escala un valor relativamente bajo, y además, el 100% de las clases utilizadas en el diseño son de tamaño pequeño. Los valores obtenidos garantizan una mejor reutilización de las clases dentro del sistema, por lo que la implementación y las pruebas no tendrán un alto nivel de complicación.

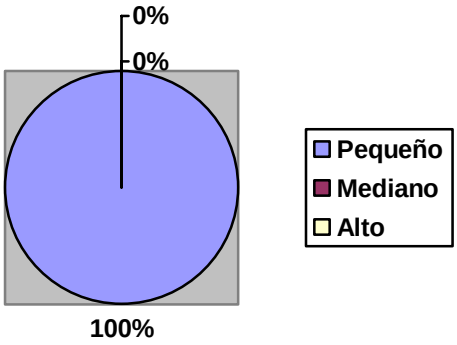

Figura 35. Representación gráfica de los resultados de TC.

3.3 Carencia de cohesión en los métodos (CCM).

Esta métrica pertenece al conjunto de métricas propuestas por Chidamber y Kemerer, (métricas CK), referenciadas en (PRESSMAN 2005). CCM se define como la cantidad de métodos de una clase que acceden a uno o más de los mismos atributos. Si no existen métodos que accedan a los mismos atributos, entonces CCM=0. Lo que se pretende con esta métrica es mantener el valor de CMM lo más bajo posible, valores altos de CMM incrementan la complejidad del diseño de clases, por lo que se requiere acoplar mejor los métodos a otros por medio de los atributos. En general, los valores altos de CCM implican que la clase debe diseñarse mejor, descomponiendo en dos o más clases distintas. Aunque existan casos en que valores altos de CCM sean justificables, es deseable mantener la cohesión alta, o sea, mantener CCM bajo.

Apliquemos entonces esta métrica a las clases más importantes del diseño. Es importante aclarar que Chidamber y Kemerer no definieron valores umbrales para esta métrica, pero tomaremos a consideración que por ejemplo, una clase con 10 métodos que accedan a atributos comunes ya está sobrepasando los límites. Consideraremos entonces la escala: para CMM≤8, valores bajos; para 8<CCM≤12, valores medios y valores altos para CCM>12.

La clase CProducto tiene dos atributos y cuatro operaciones. De las operaciones, dos acceden a los mismos atributos. Entonces podemos decir que CCM=2.

La clase CMercado tiene dos atributos y ocho métodos. De ellos, cinco acceden a atributos comunes, por tanto CCM=5.

La clase CProvincia tiene dos atributos y ocho métodos. De ellos, cinco acceden a atributos comunes, por tanto CCM=5.

La clase CFusion tiene tres atributos y trece métodos. De ellos, cinco acceden a atributos comunes, por tanto CCM=5.

La clase CProductoEncuesta tiene cinco atributos y siete métodos. Dos de los métodos acceden al mismo atributo, por tanto CCM=2.

La clase CEncuestaGeneral tiene un atributo y ocho métodos de los cuales siete acceden a ese atributo. Por tanto, CCM=7.

La clase CProductoCanasta tiene siete atributos y nueve métodos. De ellos, existen dos que acceden a atributos comunes, por tanto CCM=2.

La clase CCanastaGeneral tiene un atributo, y de los ocho métodos que tiene, siete acceden al mismo atributo. Por tanto CCM=7.

La clase CUsuario tiene tres atributos y cinco métodos. De ellos existen dos que acceden a atributos comunes, por tanto CCM=2.

La clase CGestorUsuario tiene un atributo y nueve meétodos. De ellos, ocho acceden a atributos comunes. Por tanto CCM=8.

La clase CFichero tiene tres atributos y once métodos. De ellos, existen cinco que acceden a atributos comunes. Por tanto CCM=5.

La clase CFicheroParse tiene cinco atributos y nueve métodos. De ellos, cinco acceden a atributos comunes. Por tanto CCM=5.

La clase CMercadoClasif tiene dos atributos y cuatro métodos. De ellos, dos acceden a atributos comunes. Por tanto CCM=2.

La clase CMercadoGeneral tiene un atributo y ocho métodos. De ellos, siete acceden a atributos comunes. Por tanto CCM=7.

La clase CProvinciaClasif tiene dos atributos y cuatro métodos. De ellos, dos acceden a atributos comunes. Por tanto CCM=2.

La clase CProvinciaGeneral tiene un atributo y ocho métodos. De ellos, siete acceden a atributos comunes. Por tanto CCM=7.

La clase CNomencladora tiene un atributo y cinco métodos. De ellos, dos acceden a ese atributo. Por tanto, CCM =2.

La clase DAO_Clasificadores no tiene atributos, por tanto CCM=0.

La clase DAO_Merc_Clasif no tiene atributos, por tanto CCM=0.

La clase DAO_Prov_Clasif no tiene atributos, por tanto CCM=0.

La clase DAO_Producto_Encuesta no tiene atributos, por tanto CCM=0.

La clase DAO_Producto_Canasta no tiene atributos, por tanto CCM=0.

La clase DAO_Usuarios no tiene atributos, por tanto CCM=0.

La clase DAO_Productos no tiene atributos, por tanto CCM=0.

La clase DAO_Mercados tiene un atributo y nueve métodos. De ellos, siete acceden a ese atributo. Por tanto CCM=7.

La clase DAO_Provincias tiene un atributo y cuatro métodos. De ellos, tres acceden a ese atributo. Por tanto CCM=3.

La clase DAO_Fusion tiene un atributo y cuatro métodos. De ellos, dos acceden a ese atributo. Por tanto CCM=2.

3.3.1 Análisis de los resultados:

Los resultados de la aplicación de la métrica CCM, están recogidos en la siguiente tabla:

Tabla 18. Resultados de la métrica CCM.

Clase	Valor de CCM	Clasificación del resultado.
CProducto	2	Bajo
CMercado	5	Bajo
CProvincia	5	Bajo
CFusion	5	Bajo
CProductoEncuesta	2	Bajo
CEncuestaGeneral	7	Bajo
CProductoCanasta	2	Bajo
CCanastaGeneral	7	Bajo
CUsuario	2	Bajo
CGestorUsuario	8	Bajo

CFichero	5	Bajo
CFicheroParse	5	Bajo
CMercadoClasif	2	Bajo
CMercadoGeneral	7	Bajo
CProvinciaClasif	2	Bajo
CProvinciaGeneral	7	Bajo
CNomencladora	2	Bajo
DAO_Fusion	2	Bajo
DAO_Provincias	3	Bajo
DAO_Mercados	7	Bajo
DAO_Productos	0	Bajo
DAO_Usuarios	0	Bajo
DAO_Producto_Canasta	0	Bajo
DAO_Producto_Encuesta	0	Bajo
DAO_Prov_Clasif	0	Bajo
DAO_Merc_Clasif	0	Bajo
DAO_Clasificadores	0	Bajo

De los resultados mostrados en la tabla anterior, podemos llegar a la conclusión de que el 100% de las clases tomadas como muestra cumplen con una carencia de cohesión en los métodos baja, lo que significa que la complejidad del diseño no se ve afectada prácticamente. Además, ninguna de las clases arrojó resultados de CCM alto. En general, los valores obtenidos de CCM significan una alta cohesión para el diseño. En la siguiente figura se muestran los resultados graficados.

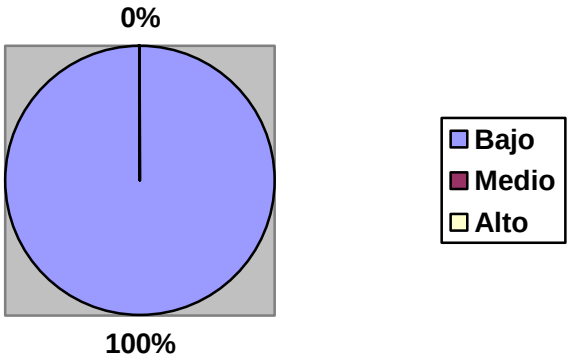

Figura 36. Representación gráfica de los valores de CCM obtenidos.

3.4 Acoplamiento entre clases objetos (ACO).

Esta es otra de las métricas CK, referenciada en (PRESSMAN 2005) y se define como el número de clases a las cuales una clase está ligada. Se da dependencia entre dos clases cuando una de ellas usa métodos o variables de la otra clase. Las clases relacionadas por herencia no se tienen en cuenta. Los autores proponen que esta métrica sea un indicador del esfuerzo necesario para el mantenimiento y las pruebas. También indican que cuanto más acoplamiento se da en una clase, más difícil será reutilizarla.

Para esta métrica tampoco fueron definidos valores umbrales, por lo que, teniendo en cuenta la experiencia adquirida de otros proyectos productivos, consideraremos que valores de ACO superiores a 6 no son bajos. Definiremos la siguiente escala: bajo para ACO≤6, medio para 6<ACO≤10 y altos para ACO>10. Aplicaremos esta métrica a las principales clases que opinamos deben estar más relacionadas con otras.

Tabla 19. Valores de ACO de algunas clases del diseño.

Clase	Valor de ACO	Clasificación
CGestorUsuario	2	Bajo
CGestorFusion	6	Bajo

CFusion	3	Bajo
DAO_Clasificadores	3	Bajo
DAO_Fusion	6	Bajo
CProvincia	2	Bajo
DAO_Provincias	5	Bajo
DAO_Mercado	3	Bajo

3.4.1 Análisis de los resultados.

Los valores obtenidos se encuentran en el rango de los valores bajos. Esto quiere decir que se ha logrado un mínimo acoplamiento entre las clases del diseño, en otras palabras, el 100% de las clases del diseño presentan un bajo acoplamiento. Estos resultados obtenidos hacen que sea más fácil el mantenimiento y la comprensión del diseño, por lo que no serán necesarios un gran esfuerzo y unas pruebas rigurosas. Al reducir el acoplamiento, hemos reducido la complejidad.

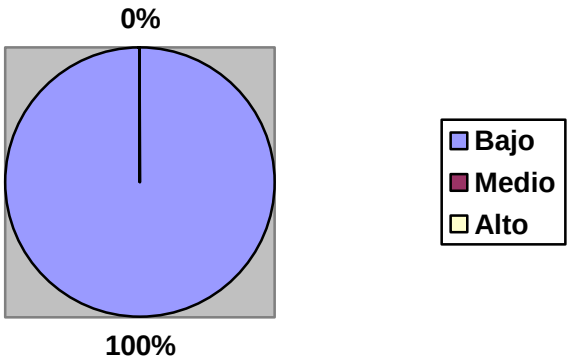

Figura 37. Representación gráfica de los valores de ACO obtenidos.

3.5 Número de clases clave (NCC).

Esta es otra de las métricas propuestas por Lorenz y Kidd, referenciada en (PRESSMAN 2005). Una clase clave se centra directamente en el dominio del negocio para el problema, y tendrá una menor probabilidad de ser implementada por medio de la reutilización. Por esta razón, valores altos para NCC indican gran trabajo de desarrollo substancial. Lorenz y Kidd sugieren que el NCC esté entre el 20 y el 40 por ciento de todas las clases.

Aplicando esta métrica:

Número de clases claves (NCC): 10

Total de clases del sistema: 37

Calculando por ciento de NCC:

$$(NCC) = \frac{1000}{37} = 27.02 \approx 27\%$$

3.5.1 Análisis de los resultados.

Hemos logrado que la cantidad de clases clave se reduzca al 27% del total de clases. Como se explicaba anteriormente, las clases claves tienen menos probabilidades de ser reutilizadas, por tanto, hemos logrado aumentar el nivel de reutilización del diseño disminuyendo la cantidad de clases claves.

3.6 Conclusiones.

Teniendo en cuenta los resultados obtenidos durante la aplicación de las diferentes métricas seleccionadas, podemos decir que los resultados obtenidos se apropian a las necesidades que requiere un buen diseño OO.

Los valores de tamaño obtenidos fueron bajos en un 100%, y con ellos logramos aumentar la reutilización de las clases, disminuir las responsabilidades de las mismas y, en un tercer lugar, complicar en un menor grado las pruebas y la implementación.

Los valores de cohesión obtenidos fueron bajos en un 100%. Este resultado hace que la complejidad del diseño propuesto no sea alta y que las clases se encuentran bien diseñadas, sin necesidad de tener que acoplar métodos unos dentro de otros.

Conclusiones.

Dando por cumplidas las tareas de investigación planificadas para la realización de este trabajo de diploma, se ha logrado realizar un diseño flexible y reusable para el Sistema Integral de Cálculo de Índices de Precios al Consumidor que da cumplimiento a los requisitos del cliente y que brindó información detallada, para facilitar los trabajos de implementación.

El uso de los patrones de diseño contribuyó a elevar la cohesión del diseño, así como disminuir el acoplamiento del mismo. Los diagramas de clases y de secuencia proporcionaron la lógica de realización de cada caso de uso, y de esta misma manera, brindaron información detallada y específica a los demás desarrolladores que fue usada para la implementación del sistema.

La aplicación de las métricas para el diseño demostró, a pesar de que los resultados obtenidos fueron buenos, que todavía existe cierto grado de incertidumbre en la formalización de algunas de las propiedades en las que se basan las métricas. Se ha demostrado la necesidad de un conjunto formal de métricas que cuantifiquen de manera formal la calidad del diseño de software.

Recomendaciones.

Toda la información expuesta en este trabajo de diploma pretende ser la idea principal sobre la cual seguir trabajando sobre el diseño propuesto.

Con el objetivo de hacer el diseño más potente, se recomienda:

- Rediseñar el proceso de autenticación de usuarios y los procesos de gestión teniendo en cuenta para un futuro que la aplicación sea multiusuario y se ejecute desde diferentes nodos.

- Diseñar un caso de uso para gestionar la tasa de cambio. El sistema trabaja con valores monetarios en divisa, sería recomendable en momentos determinados mostrarlos en su equivalente con los demás mercados para realizar análisis comparativos de precios, entre otras opciones.

- Diseñar un mecanismo de identificación que permita advertir a los usuarios sobre precios alterados y precios fuera de rangos para un producto determinado.

- Diseñar un mecanismo que permita especificar ajustes de calidad de los productos. Esto se debe a que el tiempo de vida del período base es largo por lo general, y cuando hayan transcurrido varios años la calidad de un producto determinado no debe ser la misma que cuando se tomó como muestra. De este modo los valores de índices de precios podrían ponderarse de una forma más eficiente.

Bibliografía.

ALFARO, F. M. *Procesamiento y Difusión del IPC* 2001. [Disponible en: http://www.inei.gob.pe/biblioineipub/bancopub/Est/Lib0344/procesa.HTM

CAMEJO, D. *Propuesta de la arquitectura para el Sistema Integral de Cálculo de Indices de Precios al Consumidor.*, 2007. p.

CEBALLOS, F. J. El lenguaje de programación C#. 2002. Madrid, Editorial Addison-Wesley, 2002. p. 84-789-7500-4

DODERO, J. M. Patrones de creación, 2002. [Disponible en: www.dei.inf.uc3m.esdocenciap_s_ciclopawebapuntesfactory.pdf

FERNANDEZ, J. F. *Sistema para el Cálculo de IPC*, 1999.

FUSTER, G. G. *Evaluación comparativa de herramientas CASE*, 2006. [Disponible en: http://dialnet.unirioja.es/

GÓMEZ, R. P. L. *Herramientas CASE*, 2003. [Disponible en: http://www.ilustrados.com/publicaciones/EpykZkulZZsYzwABxh.php

JACOBSON, I. *El lenguaje unificado de modelado. Volumen 1, 2 y 3*. 2000. Madrid, Editorial Addison-Wesley, 2000. p. 978-84-7829-044-4

---. *El proceso unificado de desarrollo de software. Volumen 1 y 2*. 2004. La Habana, Editorial Félix Varela, 2004. p. 0-201-57169-2

LAGO, R. *Patrón Data Access Object*, 2007. [Disponible en: http://www.proactiva-calidad.com/java/patrones/DAO.html

LARMAN, C. *UML y Patrones. Introducción al análisis y diseño orientado a objetos. Volumen 1 y 2*. 2004. La Habana, Editorial Félix Varela, 2004. p. 84-205-3438-2

ONE. *Indice de Precios al Consumidor*, 2000.

PRESSMAN, R. *Ingeniería del software. Un enfoque práctico. Parte 1 y 2*. 2005. La Habana, Editorial Félix Varela, 2005. p. 84-481-1186-9

RÍOS, E. Sistema de Cálculo de Indices de Precios al Consumidor, 2007.

WELICKI, L. *El Patrón Singleton*, 2007. [Disponible en: http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_4081.asp