UNIVERSIDAD DE LAS CIENCIAS INFORMATICAS

"PROPUESTA DE LA ARQUITECTURA PARA EL SISTEMA INTEGRAL DEL CALCULO DE INDICES AL CONSUMIDOR "

AUTOR

Disnier Alberto Camejo Dominguez

TUTORES

Lic. Merlyn Avilés Espinosa

Ciudad de la Habana

Enero 2007

Declaración de Autoria

Por este medio declaro que soy los único autore de este trabajo y autorizo a la Universidad de Ciencias Informáticas (UCI) para que hagan el uso que estimen pertinente con el mismo.

Para que así conste firmamo la presente a los ¿ días del mes de ¿ de 2007.

_____ _____

Firma del Autor Firma del Tutor

Agradecimientos

A mis padres por la educación que me han brindado, a la revolución por haberme dado la oportunidad de superarme cada día y en especial a nuestro Comandante Fidel Castro .

Dedicatoria

Al ser más amado de mi vida, mi madre Maria Dominguez Bonell, luz y faro guiador de cada uno de los actos de mi vida. A mis hermanos Daikel Camejo y Daliana Camejo, mis compañeros inseparables desde la infancia. Los quiero

A mi padre Alberto Camejo, a mis Tios, primos, abuelos, a mi tia Maricela, mi Abuela mami Gladys y a mi abuelo Papi Mellizo, a mi novia y a mi inseparable amigo Dabiel, a mis compañeros de estudio Yusniel, Jose Raul, Jorjito , Yoandry, Yordanis, Tobias, Ramonsito y muchos mas que han estado en las buenas y en las malas, me es muy dificil mencionarlos a todos.

Resumen

El presente trabajo de diploma esta dirigido al proceso de cálculo del Índice de Precios al Consumidor en la Oficina Nacional de Estadística. Para ello se ha propuesto llevar a cabo una Arquitectura del Software para la automatización de los procesos de dicha actividad.

La entidad anteriormente mencionada, con su sede en Ciudad de la Habana esta encargada de archivar y procesar gran cantidad de información de muchas esferas de nuestra sociedad. Dentro de las disímiles funciones que cumple se encuentra el cálculo del ya citado índice, el cual tiene una gran importancia para analizar el comportamiento de varios indicadores económicos y sociales de nuestro país.

La infraestructura creada para soportar las acciones comprendidas en el ejercicio de esta actividad genera documentos oficiales de apoyo, informes y reportes de suma importancia, por lo que se hace imprescindible llevar a cabo un adecuado tratamiento de la información, pues de ello depende el correcto funcionamiento de dicha infraestructura.

Con el objetivo de proporcionarle a la ONE dicho mecanismo se propone la realización de una Arquitectura para la posterior implementación del sistema de cálculo del índice, que posibilite el adecuado funcionamiento del mismo.

Al final de todo este proceso se deben obtener como resultados relevantes:

Proveer a la Oficina Nacional de Estadística de una herramienta que sea capaz de resolver todo el procesamiento de información que se requiere para el cálculo del índice de precios.

El presente documento deja plasmado los resultados de estudios realizados a sistemas que tratan con el citado índice de precios, además, se incluyen algunos conceptos relacionados con esta materia , así como el fruto de las investigaciones realizadas durante todo el proceso. Finalmente se muestran los resultados de la Arquitectura propuesta del sistema, y se proponen algunas recomendaciones para el desarrollo futuro del mismo.

Abstract

The present work is intended to deal with the calculus for the consumer index prices at the National Statistic Office. For this is proposed to carry out software Architecture to develop the automatization of the process involved.

The aforementioned entity, located in Havana City is in charge of filing and processing great quantity of information of several spheres of our society. Among the many functions that it achieve is the already stated index, which have immense relevance to analyze the behavior of many economic and social indicators of our country.

The created infrastructure to maintain the related actions in this activity generates supporting official documents and reports of enormous value, due to this is necessary to perform a satisfactory treatment of the information, in which lies the correct execution of it.

At the end of this process relevant results are to be attained.

To provide the national Statistic Office with a tool capable of solving the whole processing of the information required for the index calculus.

The current document explains the results of studies done to systems that deal with the index of prices, besides are included concepts related to this branch, as well as the consequence of the investigations during the process. Finally are shown the resultant architecture proposed for the system and some recommendations for the future development of it.

Tabla de contenidos

Indice de Figuras

Indice de Tablas

Introducción

¿Que es el índice de precios?

El Índice de Precios de Consumo (IPC) requiere para su elaboración la selección de una muestra de bienes y servicios representativa de los distintos comportamientos de consumo de la población, así como la estructura de ponderaciones que define la importancia de cada uno de estos productos. Dicho IPC obtiene esa información de la encuesta de presupuestos a hogares cubanos y se halla por la variación de precios de productos en 2 períodos, el primero (base) tomado de la encuesta anteriormente mencionada y el segundo en el momento que se desee conocer la variación. Para este último debe realizarse un sondeo de precios de los productos que han sido seleccionados previamente.

Sistemas automatizados existentes.

En todos o casi todos los países del mundo existen sistemas automatizados que realizan este proceso de cálculo del IPC. Por ejemplo en España, existe un sistema que realiza este proceso de cálculo. Dicho sistema toma los productos y precios de la encuesta de presupuestos familiares, realizada en el año 1997. Esta encuesta española tomó como muestra productos que pertenecen única y exclusivamente a un solo mercado, el mercado capitalista español. Así ocurre en todos lo países del mundo, donde los productos con sus respectivos precios se toman exclusivamente de un solo tipo de mercado, que es el que consume la población. En nuestro país no ocurre de esta manera. Por las dificultades económicas que se han originado como resultado del férreo bloqueo económico impuesto, el estado cubano se ha visto en la necesidad de establecer 2 tipos de moneda y a la vez crear 2 tipos de mercado uno para cada tipo de moneda. Es por ello que para obtener un correcto IPC es necesario analizar distintos precios, o sea, en el mercado en moneda nacional y en el mercado en divisas. Este mercado en moneda nacional se divide en 3 tipos: mercado formal, que revela los productos que se consumen en la tienda de víveres, el mercado informal o bolsa negra y el mercado agropecuario. Por todo lo anteriormente expuesto podemos llegar a la conclusión de que no existe en el mundo un sistema automatizado exactamente como el que se requiere implementar en la Oficina Nacional de Estadística (ONE) en lo referente al IPC. Las características expuestas previamente lo hacen un sistema realmente peculiar.

Situación Problemática.

En la Oficina Nacional de Estadística existe un sistema para calcular el índice de los precios de los productos de consumo en la sociedad cubana pero el mismo se implementa sobre una arquitectura mal concebida, sin hacer uso de bases de datos, sino más bien de ficheros que no eran suficientes para cubrir las expectativas del negocio. Por otra parte los funcionarios de esta oficina consideran el sistema obsoleto porque han surgido nuevas funcionalidades en el negocio que el sistema no puede abarcar, el software no era escalable porque no consideraba aspectos que se podían presentar en el futuro.

Propuesta de solución.

Como hemos podido apreciar, los sistemas desarrollados por los distintos países en lo referente al cálculo del IPC suelen resultar muy eficientes, pero están vinculados exclusivo y particularmente a su país, es decir, son totalmente incompatibles con el nuestro. La ONE se ve imposibilitada de la compra de cualquier sistema de este tipo proveniente de otra nación. Dada estas condiciones donde también se ve inmerso el bloqueo económico por Estados Unidos a nuestro país, se suman los problemas presentes en el Sistema que era utilizado en dicha entidad.

Es por ello que se hace necesario desarrollar un sistema compatible, funcional y acorde con las necesidades y especificaciones anteriormente mencionadas, en mutuo acuerdo con la Oficina Nacional de Estadística.

Problema Científico de la Investigación

Se define como Problema Científico de la Investigación la no existencia de una arquitectura de software bien definida que sea capaz de centrar el desarrollo del sistema IPC de la ONE y que permita la implementación del sistema acorde a las necesidades de dicha entidad.

Objetivo

Se plantea como Objetivo General la propuesta de dicha arquitectura de forma tal que la misma cree una vista abstracta de los principales componentes del sistema y la interacción entre ellos, de este modo se podrá guiar todo el proceso de desarrollo a partir del esbozo del sistema que ofrecer la arquitectura.

Objeto de Estudio

El Objeto de estudio estará centrado en el análisis del desarrollo de arquitecturas para sistemas empresariales y de gestión.

Campo de Acción

El Campo de Acción estará enmarcado en la propuesta de la arquitectura específica para el sistema IPC de la ONE.

Hipótesis

Se plantea entonces como Hipótesis que si se crea una arquitectura para el sistema IPC de la ONE se logrará crear un esbozo sobre el cual se centrará el desarrollo de la aplicación.

Objetivo General

Desarrollar una prupuesta de arquitectura capaz de centrar los esfuerzos de los desarrolladores crear un ezbozo del sistema a desarrollar y facilitar la toma de decisiones significativas.

Objetivos Específicos

- Para dar cumplimiento al objetivo general se delinean una serie de objetivos específicos
- Definir Estilos Arquitectónicos a utilizar.
- Definir Patrones Arquitectónicos a utilizar.
- Definir las herramientas a utilizar para el desarrollo.
- Definir la configuración de los puestos de trabajo según los roles.
- Definir Linea Base de la Arquitectura.
- Definir las vistas arquitectónicas que permiten observar la maduración de la arquitectura.

Tareas a realizar para dar cumplimiento a los objetivos

- Revisar la bibliografía existentes para establecer una comparación entre los distintos enfoques de la arquitectura y ver como han evolucionado.
- Revisar bibliografía científica teórica usadas en el desarrollo de arquitecturas para sistemas empresariales desarrollados anteriormente para estimar el grado de novedad de los posibles resultados.
- Estudiar las soluciones arquitectónicas que propone la metodología de desarrollo RUP.
- Fundamentación de los patrones a utilizar, así como su aplicación.

Métodos

Para dar cumplimiento a estas tareas se plantea el seguimiento y estudio de dos tipos de métodos, los teóricos y los empíricos, los cuales aparecen a continuación.

Métodos teóricos

- Análisis y Síntesis: Permite procesar información, proponer la arquitectura teniendo en cuenta tendencias y modelos de arquitectura, arribar a las conclusiones de la investigación y precisar las características del los modelos a utilizar en el desarrollo del sistema.
- Inducción y deducción : Proponer modelos y estilos de arquitectura a partir del estudio de los modelos utilizados para el desarrollo arquitectonico de software.
- Histórico Lógico: Permite determinar las tendencias actuales tanto de enfoques como de modelos de arquitecturas a utilizar.
- Método Sistémico: Determinar y definir los componentes de la arquitectura y las relaciones existentes entre estos.

Métodos empíricos

- Análisis de documento: para la fundamentación teórica del problema.

CAPITULO 1 Fundamentación Teórica

Introducción

En este capítulo se realizará un estudio del arte de la Arquitectura de Software, donde se incluirán puntos medulares como el análisis de las metodologías de desarrollo de mayor uso para así seleccionar la más conveniente a utilizar, por otra parte se analizarán los estilos arquitectónicos predominantes para seleccionar el estilo que guiará el desarrollo de la arquitectura, análisis de los patrones de arquitectura mas difundidos y analisis de los Lenguajes de Descripción Arquitectónica(ADL).

¿Que es la Arquitectura de Software?

La arquitectura de Software es una disciplina reciente [Szyperski, 2000]. A medida que crece la complejidad de las aplicaciones, y que se extiende el uso de sistemas distribuidos y sistemas basados en componentes, los aspectos arquitectónicos del desarrollo de software están recibiendo un interés cada vez mayor, tanto desde la comunidad científica como desde la propia industria del software. En la actualidad pocas definiciones de la Arquitectura de Software son respaldadas unánimemente por la totalidad de los arquitectos. El número de definiciones existentes alcanza un orden de tres dígitos,

amenazando llegar a cuatro. De hecho, existen grandes compilaciones de definiciones alternativas o contrapuestas.

En general, las definiciones se entremezclan en:

- Mary Shaw y David Garlan, sugieren que la arquitectura del software sea un nivel del diseño referido a las ediciones "…más allá de las estructuras de los algoritmos y de datos del cómputo; diseñar y especificar la estructura del sistema total emerge como nueva clase de problema. Las ediciones estructurales incluyen la organización gruesa y la estructura global del control; los protocolos para la comunicación, la sincronización, y el acceso a los datos; asignación de la funcionalidad para diseñar elementos; distribución física; composición de los elementos del diseño; escalamiento y funcionamiento; y selección entre alternativas del diseño".

Esta definición no es capaz de definir el momento en el ciclo de desarrollo del software donde se debe llevar a cabo la arquitectura, según esta, el desarrollo de la misma comienza en un momento entre la definición de los requisitos y la modelación del sistema o entre la modelación del sistema, el análisis y el diseño.

- Rational Unified Process, 1999, propone que: Una arquitectura es el sistema de decisiones significativas sobre la organización de un sistema de software, la selección de los elementos estructurales y de sus interfaces por los cuales el sistema es compuesto, junto con su comportamiento según lo especificado en las colaboraciones entre esos elementos, la composición de estos elementos estructurales y del comportamiento en subsistemas progresivamente más grandes, y el estilo arquitectónico que dirigen esta organización, estos elementos y sus interfaces, sus colaboraciones, y su composición [Jacobson, et al, 1999].

Esta define la arquitectura como una etapa más de la Ingeniería de Software enfocada a la orientación a objetos y a UML, como lo plantea la metodología de desarrollo RUP, representa la arquitectura como algo más que la simple composición de herramientas o tecnologías a usar en el desarrollo del sistema.

Desde nuestro punto de vista la arquitectura es una representación del sistema que incluye los componentes principales del mismo, el comportamiento que existe entre ellos y las formas en que

interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.

Existen otras corrientes de arquitecturas como:

- Basada en patrones (SEI), que se basa principalmente en la redefinición de los estilos como patrones POSA, un ejemplo significativo de esta corriente es Microsoft Pattern & Practices. Esta corriente de arquitectura de Microsoft utiliza los patrones POSA como estilos de arquitectura, ejemplificados en sus frameworks de desarrollo (MSF).

- Arquitecturas procedural y metodológicas, sus principales exponentes son: Kazman y Bass (SEI), que proponen variantes de arquitecturas basadas en escenarios.

En el año 2000 la IEEE 1471 - 2000 definió lo que es Arquitectura de Software como: la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

Cuando se habla de componentes es importante establecer una diferencia entre componentes de software y los componentes de la arquitectura, por lo que los arquitectos en las definiciones modernas prefieren llamarle “elementos”.

Los componentes de la arquitectura o “elementos” pueden ser dinámicos o estáticos, estos son elementos ya sea de procesamiento, de datos o de conexión [Reynoso, 2004] mientras que los componentes de software se refiere a subsistemas, clases, interfaces, etc.

Esta definición dada por la IEEE, define la arquitectura no como un flujo de trabajo dentro de una metodología, deja claro que la Arquitectura de Software no se inscribe en ninguna metodología de desarrollo, sino que es en sí, una disciplina; se desarrolla durante todo el ciclo de desarrollo de software y constituye la organización del sistema al nivel más alto de abstracción.

Podemos plantear que la Arquitectura de Software no es una normativa madura, que además tiene un vínculo estrecho con el diseño de software pero no son lo mismo y no se encuentra ligada a ninguna metodología en específico.

Se debe resaltar que la misma si tiene una relación directa con los requisitos no funcionales puesto que son estos los que permiten crear escenarios que facilitan representarla y describirla. Los requisitos funcionales quedan expresados mediante el modelado y el diseño de la aplicación.

Por otra parte se puede plantear que la misma es una vista estructural de alto nivel donde se definen los estilos arquitectónicos o la combinación de los mismos para dar una solución factible.

1.1 Clases de Estilos Arquitectónicos

No es necesario crear una nueva arquitectura software para cada sistema. Lo habitual es adoptar una arquitectura conocida en función de sus ventajas e inconvenientes para cada caso en concreto.
Así los estilos más universales se encuentran encapsulados en las clases de estilos [Shaw y Garlan, 1996, Bass et al., 1998].
.A continuacion de dara una breve expliación de algunas de estas clases de estílos y se enumeraran los estílos arquitectónicos que encapsulan.

1.1.1 Estilos de Flujo de Datos

Esta clase de estilos enfatiza en la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. Ejemplares de la misma serían las arquitecturas de tubería-filtros y las de proceso secuencial en lote.

- *Tubería y filtros*

1.1.2 Estilos Centrados en Datos

Esta clase de estilos enfatiza en la integrabilidad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Sub-estilos característicos

de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra.

- *Arquitecturas de Pizarra o Repositorio*

1.1.3 Estilos de Llamada y Retorno

Esta clase de estilos enfatiza en la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Miembros de la clase son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.

- *Model-View-Controller (MVC)*
- *Arquitecturas en Capas*
- *Arquitectura Orientada a Objetos*
- *Arquitecturas Basadas en Componentes*

1.1.4 Estilos de Código Móvil

Esta clase de estílos enfatiza en la portabilidad. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando. Fuera de las máquinas virtuales y los intérpretes, los otros miembros del conjunto han sido rara vez estudiados desde el punto de vista estilístico. Los sistemas basados en reglas, que a veces se agrupan como miembros de la clase de estilos basados en datos

- *Arquitectura de Máquinas Virtuales*

1.1.5 Estilos heterogéneos

Es la clase de estílos más fuertemente referida en los últimos tiempos, se incluyen en este grupo formas compuestas o indóciles a la clasificación en las categorías habituales. Podrían agregarse formas que aparecen esporádicamente en los estudios de nuevos estílos, como los sistemas de control de procesos

industriales, sistemas de transición de estados, arquitecturas específicas de dominios o estilos derivados de otros estilos.

- *Sistemas de control de procesos*
- *Arquitecturas Basadas en Atributos*

1.1.6 Estilos Peer-to-Peer

Esta familia, también llamada de componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente. Los mensajes pueden ser enviados a componentes nominados o propalados mediante broadcast.

- *Arquitecturas Basadas en Eventos*
- *Arquitecturas Orientadas a Servicios (SOA)*
- *Arquitecturas Basadas en Recursos (REST)*

1.2 Lenguajes de Descripción Arquitectónicos (ADLs)

Una ves que el arquitecto se decide a delinear su estrategia para proponer la arquitectura de un sistema, debe tener en cuenta que utilizará para modelar la misma de forma visual y que pueda tener un alto nivel de abstracción. Los ADLs se utilizan, además, para satisfacer requerimientos descriptivos de alto nivel de abstracción que las herramientas basadas en objeto en general y UML en particular no cumplen satisfactoriamente. Aunque algunos arquitectos alegan que el período de gloria de la OOP podría estar acercándose a su fin, el hecho concreto es que el modelado orientado a objetos de sistemas basados en componentes posee cierto número de rasgos muy convenientes a la hora de diseñar o al menos describir un sistema. En primer lugar, las notaciones de objeto proporcionan un mapeo directo entre un modelo y una implementación e implementan además métodos bien definidos para desarrollar sistemas a partir de un conjunto de requerimientos.

Contando con un ADL, un arquitecto puede razonar sobre las propiedades del sistema con precisión, pero a un nivel de abstracción convenientemente genérico. Algunas de esas propiedades podrían ser, por ejemplo, protocolos de interacción, anchos de banda y latencia, localización del almacenamiento, conformidad con estándares arquitectónicos y previsiones de una posible evolución del sistema. [Shaw y Garlan, 1996]

Entre los ADL mas difundidos podemos encontrar:

1.2.1 Acme – Armani

Acme se define como una herramienta capaz de soportar el mapeo de especificaciones arquitectónicas entre diferentes ADLs, o en otras palabras, como un lenguaje de intercambio de arquitectura. No es entonces considerado por muchos un ADL en sentido estricto. De hecho, posee numerosas prestaciones que también son propias de los ADLs. En su sitio oficial se reconoce que como ADL no es necesariamente apto para cualquier clase de sistemas, al mismo tiempo que se destaca su capacidad de describir con facilidad sistemas “relativamente simples”.

1.2.2 Aesop

El nombre oficial es Aesop Software Architecture Design Environment Generator. Se ha desarrollado como parte del proyecto ABLE de la Universidad Carnegie Mellon, cuyo objetivo es la exploración de las bases formales de la arquitectura de software, el desarrollo del concepto de estilo arquitectónico y la producción de herramientas útiles a la arquitectura, de las cuales Aesop es precisamente la más relevante. Aesop genera código C++ y aunque opera primariamente desde una interfaz visual, el código de es marcadamente más procedural que el de Acme.

Disponibilidad de plataforma – Aesop no está disponible en plataforma Windows, aunque naturalmente puede utilizarse para modelar sistemas implementados en cualquier plataforma.

1.2.3 ArTek

ArTek fue desarrollado por Teknowledge. Se lo conoce también como ARDEC/Teknowledge Architecture Description Language. Para algunos arquitectos no es considerado un genuino ADL, por cuanto la configuración es modelada implícitamente mediante información de interconexión que se distribuye entre la definición de los componentes individuales y los conectores. Aunque pueda no ser un ADL en sentido estricto, se le reconoce la capacidad de modelar ciertos aspectos de una arquitectura.

Disponibilidad de plataforma – Hoy en día ArTek no se encuentra disponible en ningún sitio y para ninguna plataforma

1.2.4 Darwin

Darwin es un lenguaje de descripción arquitectónica desarrollado por Jeff Magee y Jeff Kramer. El mismo describe un tipo de componente mediante una interfaz consistente en una colección de servicios que son ya sea provistos (declarados por ese componente) o requeridos (o sea, que se espera ocurran en el entorno). Las configuraciones se desarrollan instanciando las declaraciones de componentes y estableciendo vínculos entre ambas clases de servicios.
En una implementación generada en Darwin, se supone que cada componente primitivo está implementado en algún lenguaje de programación, y que para cada tipo de servicio se necesita un ligamento (glue) que depende de cada plataforma. El algoritmo de elaboración actúa, esencialmente, como un servidor de nombre que proporciona la ubicación de los servicios provistos a cualquier componente que se ejecute.

Disponibilidad de plataforma – Aunque el ADL fue originalmente planeado para ambientes tan poco vinculados al modelado corporativo como hoy en día lo es Macintosh, en Windows se puede modelar en lenguaje Darwin.

1.2.5 Jacal

Es un lenguaje de descripción de arquitecturas de software de propósito general creado en la Universidad de Buenos Aires, por un grupo de investigación del Departamento de Computación de la Facultad de Ciencias Exactas y Naturales.

El objetivo principal de Jacal es lo que actualmente se denomina “animación” de arquitecturas. Esto es, poder visualizar una simulación de cómo se comportaría en la práctica un sistema basado en la arquitectura que se ha representado.

Más allá de este objetivo principal, el diseño de Jacal contempla otras características deseables en un ADL, como por ejemplo contar con una representación gráfica que permita a simple vista transmitir la arquitectura del sistema, sin necesidad de recurrir a información adicional. Para este fin, se cuenta con un conjunto predefinido (extensible) de conectores, cada uno con una representación distinta.

1.2.6 UML

UML forma parte del repertorio conocido como lenguajes semi-formales de modelado. En términos de número, muchos de los conocedores de UML no admite comparación con la todavía modesta población de especialistas en ADLs. Existen dos fracciones claramente definidas; la primera, vinculada con el mundo de Rational y UML, impulsa el uso casi irrestricto de UML como si fuera un ADL normal; la segunda ha señalado reiteradas veces las limitaciones de UML no sólo como ADL sino como lenguaje universal de modelado.

1.3 Patrones de Arquitectura

Los patrones de arquitectura expresan el esquema fundamental de organización para sistemas de software. Proveen un conjunto de subsistemas predefinidos; especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos. Los patrones de arquitectura representan el nivel más alto en el sistema.

1.3.1 Patrón Modelo Vista Controlador

Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve

frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el controlador es el Sistema de Gestión de Base de Datos y el modelo es el modelo de datos.

Modelo

Esta es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos; por ejemplo, no permitiendo comprar un número de unidades negativo, calculando si hoy es el cumpleaños del usuario o los totales, impuestos o portes en un carrito de la compra.

Vista:

Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.

Controlador:

Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista. Muchos sistemas informáticos utilizan un Sistema de Gestión de Base de Datos para gestionar los datos. En MVC corresponde al controlador.

Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el siguiente:

- El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace)
- El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.

- El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario).

Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

- El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, el patrón de observador puede ser utilizado para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.
- La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

1.3.2 Patrón Capas

Con anterioridad se había explicado como uno de los estilos arquitectónicos, ahora se hará referencia a cada uno de los ejemplos específicos que pueden aparecer de este patrón partiendo que la funcionalidad principal del mismo es crear una modularidad del sistema asignando a cada capa funcionalidades específicas y bien definidas.

La capa de la Presentación

Esta capa reúne todos los aspectos del software que tiene que ver con las interfaces y la interacción con los diferentes tipos de usuarios. Estos típicamente incluyen el manejo y aspecto de las ventanas, el formato de los reportes, menús, gráficos y elementos multimedia en general.

La capa del Dominio de la Aplicación

Esta capa reúne todos los aspectos del software que tienen que automatizan o apoyan los procesos de negocio que llevan a cabo los usuarios. Estos aspectos típicamente incluyen las tareas que forman parte

de los procesos, las reglas y restricciones que aplican. Esta capa también recibe el nombre de la capa de la Lógica de la Aplicación.

La capa del Acceso a Datos

Esta capa reúne todos los aspectos del software que tienen que ver con el manejo de los datos persistentes, por lo que también se le denomina la capa de las Bases de Datos.

Existen especificaciones de este patrón donde se manipula como será la comunicación entre cada una de las capas definidas.

Arquitectura top-down de capas:

Los elementos de una capa i+1 pueden enviar solicitudes de servicio a elementos de la capa inferior i. Típicamente se produce una cascada de solicitudes, es decir para satisfacer una solicitud a una capa i+2, ésta requiere enviar varias solicitudes a la capa i+1; cada una de estas solicitudes a la capa i+1 genera a su vez un conjunto de solicitudes a la capa i y así sucesivamente. Una arquitectura top-down es laxa (o no estricta) si los elementos de una capa i+1 pueden enviar solicitudes de servicio directamente a un elemento de cualquiera de las i capas inferiores.

Arquitectura bottom-up de capas:

Cada elemento de una capa i puede notificar a elementos de la capa superior i+1 de que ha ocurrido algún evento de interés (ej. manejadores de dispositivos). La capa i+1 puede juntar varios eventos antes de notificar a su vez al elemento de la capa i+2. Una arquitectura bottom-up también puede ser no estricta si el elemento de la capa i puede notificar a cualquier elemento de cualquier capa superior a la capa i.

Arquitectura bidireccional de capas

En su forma más común involucra dos pilas de N capas que se comunican entre si. El ejemplo más conocido es el de los protocolos en Redes de Computadores.

1.3.3 Patrón Cliente- Servidor

En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa.

Una disposición muy común son los sistemas multicapa en los que el servidor se descompone en diferentes programas que pueden ser ejecutados por diferentes computadoras aumentando así el grado de distribución del sistema.

La arquitectura cliente-servidor sustituye a la arquitectura monolítica en la que no hay distribución, tanto a nivel físico como a nivel lógico.

Ventajas de la arquitectura cliente-servidor

Centralización del control:

Los accesos, recursos y la integridad de los datos son controlados por el servidor de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema.

Escalabilidad:

Se puede aumentar la capacidad de clientes y servidores por separado.

El servidor de cliente es la arquitectura de red que separa al cliente (a menudo un uso que utiliza un interfaz utilizador gráfico) de un servidor. Cada caso del software del cliente puede enviar peticiones a un servidor. Los tipos específicos de servidores incluyen los servidores de la web, los servidores del uso, los servidores de archivo, los servidores terminales, y los servidores del correo. Mientras que sus propósitos varían algo, la arquitectura básica sigue siendo igual.

1.4 ¿Qué metodología usar?

Es una pregunta que cobra mucha importancia, pues el arquitecto debe tener un plano en que apoyarse. Todo desarrollo de software es riesgoso y difícil de controlar, pero si no se lleva una metodología de por medio, lo que se obtiene es clientes insatisfechos con el resultado y desarrolladores aún más insatisfechos. A continuación de expondrán las principales características de 3 metodologías de desarrollo, Rational Unifield Process, Extreme Programming y Microsoft Solution FrameWork.

1.4.1 Rational Unified Process (RUP)

RUP es un proceso que define claramente quien, cómo, cuándo y qué debe hacerse; y, como su enfoque esta basado en modelos y utiliza un lenguaje bien definido para tal fin, UML. Éste aporta herramientas como los casos de uso, que definen los requerimientos. Permite la ejecución iterativa del proyecto y del control de riesgos.

Descripción general

- El RUP es un proceso de ingeniería de software.
- Utiliza el paradigma de orientación a objetos para su descripción.
- Es un marco de proceso configurable para satisfacer necesidades específicas.
- Implementa las mejores prácticas de desarrollo de software.

Casos de Uso

- Secuencia de pasos que un sistema realiza para proveer un resultado de valor para un actor particular.

- Se enfocan en la funcionalidad del sistema y necesitan de requerimientos adicionales para proveer una especificación completa de los requerimientos de software.
- Se concentra en el qué y no en el cómo.
- Los casos de uso se utilizan principalmente para capturar los requerimientos de comportamiento de un sistema.
- Posicionan los requerimientos de software en contexto: muestran cómo el sistema provee valor a sus patrocinadores al mismo tiempo que los hace más fáciles de entender.

Proceso iterativo e incremental

Es el proceso de construir sistemas haciendo aproximaciones que se acercan progresivamente a la solución ideal. De esta forma se obliga a identificar los riesgos del proyecto en etapas tempranas y es además un enfoque de descubrimiento, invención e implementación continuos donde cada iteración termina en una forma predecible y repetible.

La siguiente imagen muestra los flujos de trabajo por las cuales pasa el ciclo de vida del software y el carácter iterativo del mismo.

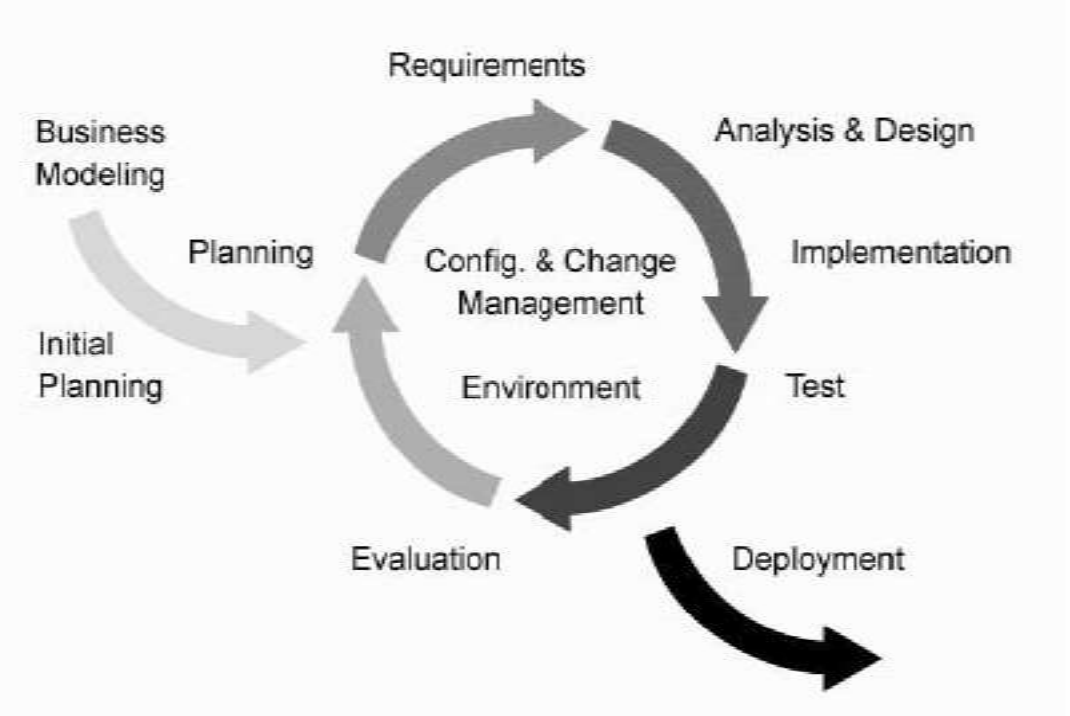

Figure 1 Flujos de Trabajo y Proceso Incremental

Características

1. *Guiado por los Casos de Uso*: Los casos de uso capturan requerimientos funcionales y representan piezas de funcionalidad que brindan un resultado de valor al usuario.
2. *Centrado en la Arquitectura*: Comprende los aspectos estáticos y dinámicos más importantes del sistema.
3. *Iterativo***:** El trabajo se divide en piezas pequeñas o mini proyectos; cada uno proveyendo un subproducto incremental.

Guiado por los Riesgos

A través de un proyecto guiado por RUP, los requerimientos funcionales son expresados en la forma de Casos de Uso, que guían la realización de una arquitectura ejecutable de la aplicación. Además el proceso focaliza el esfuerzo del equipo en construir los elementos críticos estructuralmente y del

comportamiento (llamados Elementos Arquitecturales) antes de construir elementos menos importantes. Finalmente RUP particiona el ciclo de vida en iteraciones que producen versiones increméntales de los ejecutables de la aplicación.

RUP implementa las siguientes mejores prácticas asociadas al proceso de Ingeniería de Software:

- Desarrollo Iterativo
- Manejo de los Requerimientos
- Uso de una Arquitectura basada en componentes
- Modelización Visual
- Verificación Continua de la Calidad
- Manejo de los Cambios

Disciplina de Desarrollo

- Ingeniería de Negocios: Entendiendo las necesidades del negocio.
- Requerimientos: Trasladando las necesidades del negocio a un sistema automatizado.
- Análisis y Diseño: Trasladando los requerimientos dentro de la arquitectura de software.
- Implementación: Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.
- Pruebas: Asegurándose que el comportamiento requerido es el correcto y que todo los solicitado esta presente.

Disciplina de Soporte

Configuración y administración del cambio: Guardando todas las versiones del proyecto.

- Administrando el proyecto: Administrando horarios y recursos.
- Ambiente: Administrando el ambiente de desarrollo.
- Distribución: Hacer todo lo necesario para la salida del proyecto

En la imagen se recrean los flujos de trabajo de la metodología RUP así como el esfuerzo realizado en cada flujo en cada iteración.

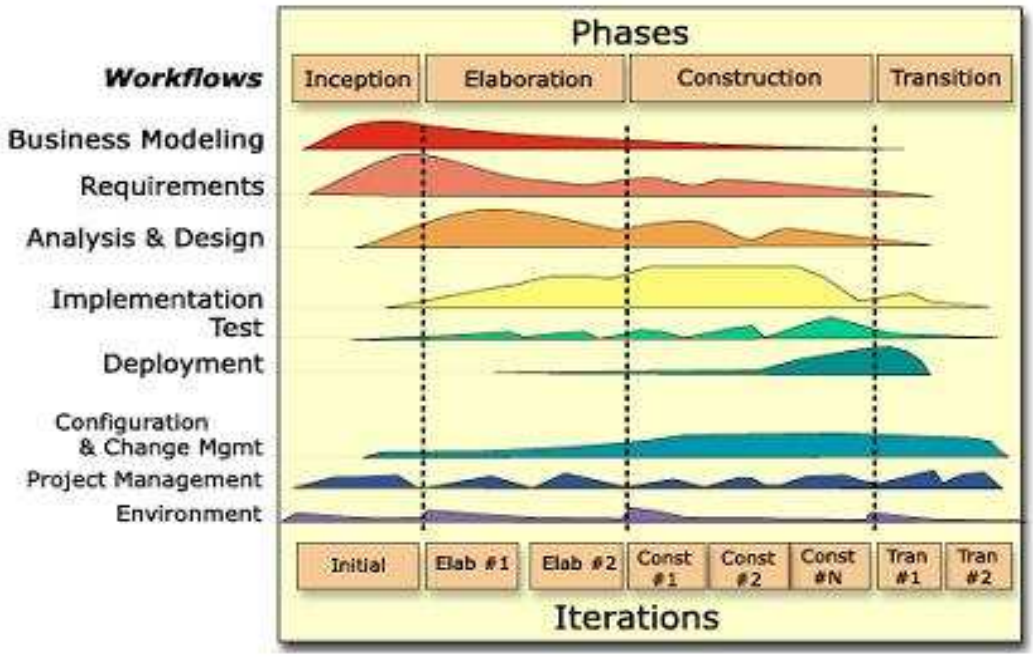

Figure 2 Flujos de Trabajo, Fases y esfuerzo por Iteración.

Es recomendable que a cada una de estas iteraciones se les clasifique y ordene según su prioridad, y que cada una se convierte luego en un entregable al cliente. Esto trae como beneficio la retroalimentación que se tendría en cada entregable o en cada iteración.

Table 1 Fases de la Metodología RUP

Fase	Objetivos	Puntos de Control
Inicio	• Definir el alcance del proyecto • Entender que se va a construir	Objetivo del proyecto
Elaboración	• Construir una versión ejecutable de la arquitectura de la aplicación	Arquitectura de la Aplicación

Fase	Objetivos	Puntos de Control
	• Entender cómo se va a construir	
Construcción	• Completar el esqueleto de la Aplicación con la funcionalidad • Construir una versión Beta	Versión Operativa Inicial de la Aplicación
Transición	• Disponibilizar la aplicación para los usuarios finales • Construir la versión Final	Liberación de la versión de la Aplicación

Elementos del RUP

Actividades: Son los procesos que se llegan a determinar en cada iteración.

Trabajadores: Personas o entidades involucrados en cada proceso.

Artefactos: Un artefacto puede ser un documento, un modelo, o un elemento de modelo.

La imagen muestra cada uno de los elementos antes mencionados en forma de ejemplos.

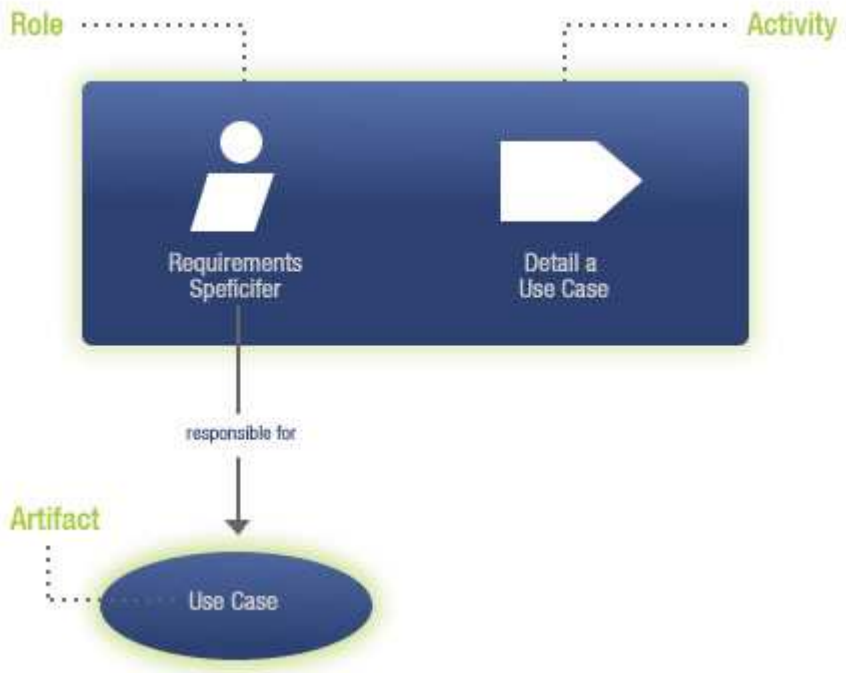

Figure 3 Artefactos, Actividades, roles

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.

1.4.2 Extreme Programing (XP).

Es una de las metodologías de desarrollo de software más exitosas en la actualidad utilizadas para proyectos de corto plazo y corto equipo. La misma consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

La figura muestra cada una de las fases del desarrollo siguiendo esta metodologia, Planificacion, Diseño, Codificacion y Pruebas.

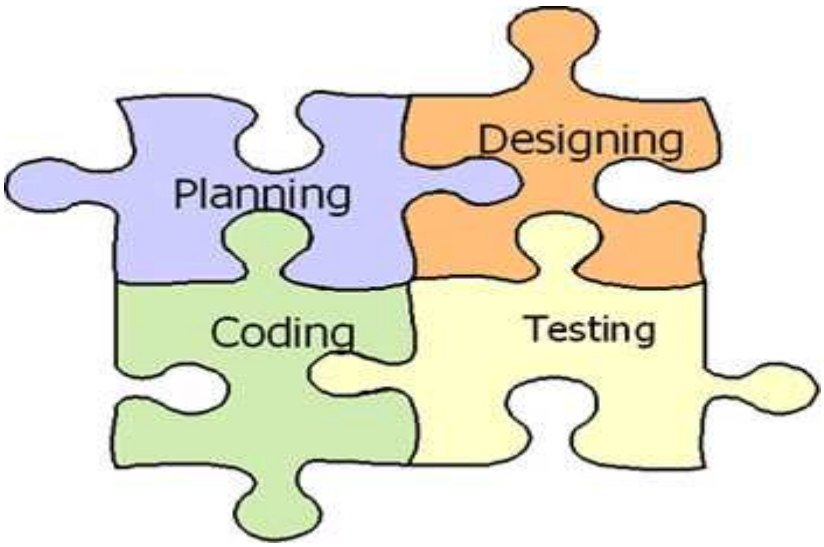

Figure 4 Flujos de Trabajo de la Metodología XP

La metodología se basa en:

- *Pruebas Unitarias*: Se basa en las pruebas realizadas a los principales procesos, de tal manera que adelantándonos en algo hacia el futuro, podamos hacer pruebas de las fallas que pudieran ocurrir. Es como si nos adelantáramos a obtener los posibles errores.

- *Re fabricación*: Se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

- *Programación en pares*: Una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

¿Qué propone XP?

- Empieza en pequeño y añade funcionalidad con retroalimentación continua
- El manejo del cambio se convierte en parte sustantiva del proceso
- El costo del cambio no depende de la fase o etapa
- No introduce funcionalidades antes que sean necesarias
- El cliente o el usuario se convierte en miembro del equipo

Derechos del Cliente

- Decidir que se implementa
- Saber el estado real y el progreso del proyecto
- Añadir, cambiar o quitar requerimientos en cualquier momento
- Obtener lo máximo de cada semana de trabajo
- Obtener un sistema funcionando cada 3 o 4 meses

Derechos del Desarrollador

- Decidir como se implementan los procesos
- Crear el sistema con la mejor calidad posible
- Pedir al cliente en cualquier momento aclaraciones de los requerimientos
- Estimar el esfuerzo para implementar el sistema

Cambiar los requerimientos en base a nuevos descubrimientos

Lo fundamental en este tipo de metodología es:

- La comunicación, entre los usuarios y los desarrolladores
- La simplicidad, al desarrollar y codificar los módulos del sistema
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales

1.4.3 Microsoft Solution Framework (MSF)

Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

La figura muestra el flujo de las actividades prioritarias según la metodología MSF, Planificación, Construcción y Administración.

Figure 5 Flujos de Trabajo de la Metodología MSF

Caracteristicas:

Adaptable: Es usado en cualquier parte, su uso es limitado a un lugar específico.

Escalable: Organizar equipos pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.

Flexible: Es utilizada en el ambiente de desarrollo de cualquier cliente.

Tecnología Agnóstica: Puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el modelo de Aplicación.

Modelo de Arquitectura del Proyecto:

Diseñado para acortar la planificación del ciclo de vida. Este modelo define las pautas para construir proyectos empresariales a través del lanzamiento de versiones.

Modelo de Equipo:

Este modelo ha sido diseñado para mejorar el rendimiento del equipo de desarrollo. Proporciona una estructura flexible para organizar los equipos de un proyecto. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles.

Modelo de Proceso:

Diseñado para mejorar el control del proyecto, minimizando el riesgo, y aumentar la calidad acortando el tiempo de entrega. Proporciona una estructura de pautas a seguir en el ciclo de vida del proyecto, describiendo las fases, las actividades, la liberación de versiones y explicando su relación con el Modelo de equipo.

Modelo de Gestión del Riesgo:

Diseñado para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.

Modelo de Diseño del Proceso:

Diseñado para distinguir entre los objetivos empresariales y las necesidades del usuario. Proporciona un modelo centrado en el usuario para obtener un diseño eficiente y flexible a través de un enfoque iterativo.

Las fases de diseño conceptual, lógico y físico proveen tres perspectivas diferentes para los tres tipos de roles: los usuarios, el equipo y los desarrolladores.

Modelo de Aplicación:

Diseñado para mejorar el desarrollo, el mantenimiento y el soporte, proporciona un modelo de tres niveles para diseñar y desarrollar aplicaciones software. Los servicios utilizados en este modelo son escalables, y pueden ser usados en un solo ordenador o incluso en varios servidores.

Capítulo 2 Descripción de la Arquitectura

Introducción

La descripción de la arquitectura consta de dos documentos bien definidos, llamados también por el RUP artefactos, sobre los cuales el arquitecto deberá enfocar el trabajo de descripción de la arquitectura propuesta, estos documentos son:

- Línea Base de la Arquitectura
- Descripción de la Arquitectura

La línea base de la arquitectura se convierte gradualmente en el documento de descripción de la arquitectura donde existirá un nivel mas detallado en cuanto a la propuesta, pues contendrá las 4+1 vistas que serán de gran importancia a lo largo del ciclo de vida de desarrollo del software.

2.1 Línea Base de la Arquitectura

Introducción

La Línea de Base de la Arquitectura contiene los elementos más importantes para lograr la máxima abstracción en el diseño arquitectónico de la aplicación. Está basada principalmente en la metodología de desarrollo Proceso Unificado de Desarrollo (Rational Unified Process), aunque no es un artefacto propuesto por la metodología. En la misma se exponen los estilos arquitectónicos para la aplicación, se expondrán los principales componentes o elementos de la arquitectura, los conectores y sus configuraciones. Se describen los principales patrones de arquitectura utilizados, las restricciones de hardware o software de la arquitectura y se describe las tecnologías y herramientas que se utilizarán en el desarrollo del sistema.

Propósito

El propósito de la Línea Base de la Arquitectura es proporcionar la información necesaria para estructurar el sistema desde el más alto nivel de abstracción. En ella se describe la estructura del sistema en cuanto a los elementos, los conectores, las configuraciones y sus restricciones.

Los usuarios de la Línea Base de la Arquitectura son:

- El equipo de arquitectos del proyecto, que le sirve de guía para la toma de decisiones arquitectónicas y son los encargados del mantenimiento y refinamiento de esta línea base.
- Los miembros del equipo de desarrollo encargados de desarrollar la aplicación, la utilizan como la guía para la implementación del sistema.
- Los clientes tienen en ella una garantía de la calidad y el conocimiento sobre en que tecnología está desarrollada su solución.

Alcance

La Línea Base de la Arquitectura describe la estructura de la aplicación en su más alto nivel de abstracción. Describe detalladamente el organigrama de la arquitectura encarnada en los estilos arquitectónicos que se utilizarán; los principales frameworks de desarrollo y como se adaptarán a la solución; se propone la utilización de un conjunto de patrones que resuelven problemas que se podrían presentar a lo largo del desarrollo del sistema. Se proponen las principales tecnologías y herramientas que soportan los estilos y patrones especificados y cumplen con las restricciones del sistema.

2.1.2 Estilos Utilizados

Arquitectura en Capas

La arquitectura en capas permite agrupar funcionalidades bien definidas facilitando los cambios en el proyecto sin necesidad de que los cambios en una capa repercutan en el funcionamiento de otras capas.

Ventajas

- Modularidad del sistema.

Desventajas

- Se hace difícil definir que componentes ubicar en cada una de las capas.

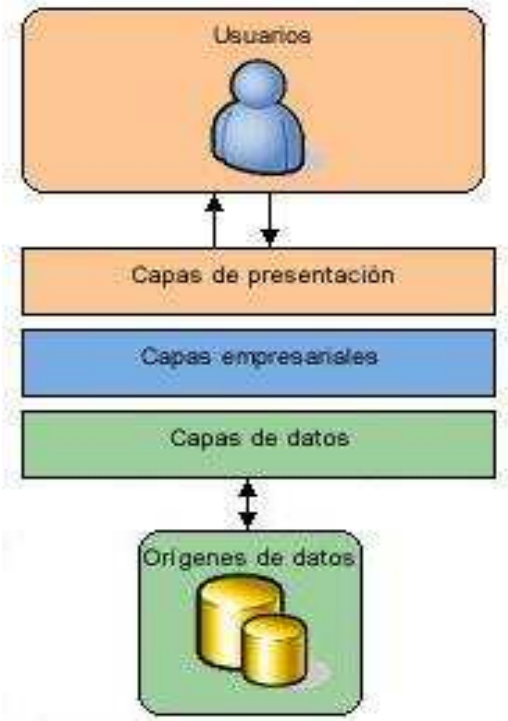

Figure 6 Estilo en Capas

Arquitectura Orientada a Objetos

Este estilo define como sus componentes principales los objetos, todos estos se basan en caracterizticas puramente orientada a objetos haciendo uso de herencia, encapsulamiento y polimorfismo, son ademas la unidades basicas del modelado, diseño e implementación y los objetos y sus interacciones son consideradas como punto de partida y el centro del diseño arquitectonico de la aplicación. En cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases. En tanto componentes, los objetos interactúan a través de invocaciones de funciones y procedimientos.

Arquitectura Basada en Componentes

Los componentes no son separables de la arquitectura. Así mismo, todos los modelos y plataformas de componentes imponen una serie de compromisos sobre dichos componentes, indicando la forma en que se describe su interfaz o que mecanismos básicos tienen que proporcionar [Brown et al, 1998]. La interfaz de un componente no solo muestra su funcionalidad, de forma abstracta, sino que implica además toda una serie de restricciones arquitectónicas ligadas a la plataforma o modelo del que forma parte [Szyperski, 2000]. De aquí que la Arquitectura al estar basada en componentes proporcione la herramienta fundamental para lograr hacer una Ingeniería Basada en Componentes, que cumpla con las expectativas de reusabilidad y reutilización de las partes de software.

Uno de los campos en los que la tecnología de componentes se ha mostrado más activa es en el desarrollo de marcos de trabajo (Frameworks) [Fayad et al., 1999]. Estos se definen como un diseño reutilizable de todo o parte de un sistema, representado por un conjunto de componentes abstractos, y la forma en la que dichos componentes interactúan.

2.1.3 Selección del Lenguaje y el Entorno de Desarrollo

En esta seccion se decribe el lenguaje a utilizar para la implemetación de la aplicación así como el Entorno de Desarrollo a utilizar.

Para la implementación del sistema se escogió la Plataforma .NET.

La Plataforma .NET

Es un conjunto de tecnologías dispersas, que en muchos casos ya existían, que Microsoft ha integrado en una plataforma común con el objetivo de facilitar el desarrollo de este nuevo tipo de servicios de tercera generación.

Pilares de esta nueva plataforma:

• *Integración*: Proporcionar Mecanismos para que una empresa pueda ofrecer servicios a otras empresas o clientes de una forma sencilla y rápida. En general, este tipo de servicios se suelen denominar B2B: Business to Business y B2C: Business to Client.

• *Nuevos dispositivos*: La forma más común de acceso a Internet hasta ahora ha sido el ordenador personal con sus limitaciones de movilidad. Pero recientemente han ido apareciendo una serie de dispositivos que permiten el acceso a servicios Internet de forma rápida y directa, como por ejemplo agendas electrónicas, teléfonos móviles, WebTV, videoconsolas, etc. Esto supone un cambio radical en la forma de acceder a este tipo de servicios.

Con estos objetivos, Microsoft .NET es una plataforma para construir, ejecutar y experimentar la tercera generación de aplicaciones distribuidas, que consiste en los siguientes elementos:

Un modelo de programación basado en XML.

Un conjunto de servicios Web XML, como Microsoft .NET My Services para facilitar a los desarrolladores integrar estos servicios.

Un conjunto de servidores que permiten ejecutar estos servicios (como .NET Enterprise Servers).

Para facilitar esta integración y el desarrollo de este Software en el cliente para poder utilizar estos servicios (como Windows XP, agendas electrónicas, etc.).

Herramientas para el Desarrollo como Visual Studio.Net. En la siguiente figura se muestran los siguientes elementos que componen la plataforma .NET.

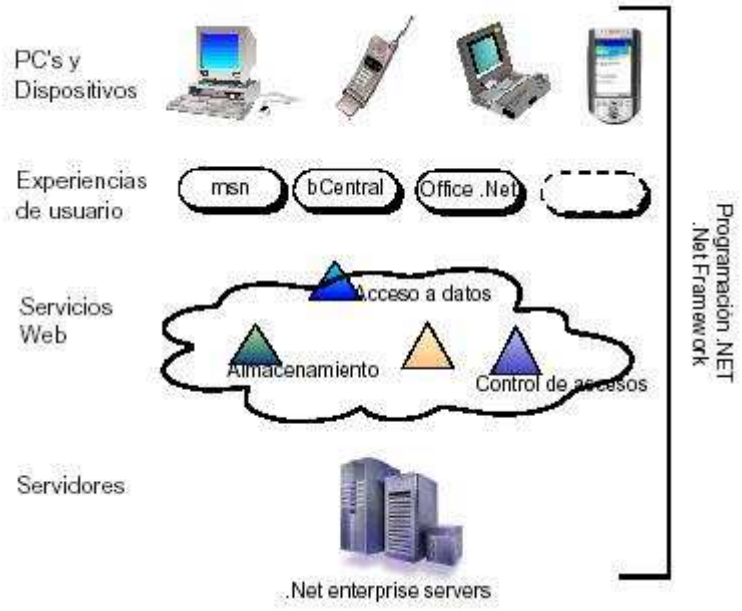

Figure 7 Elementos que conforman la plataforma.Net.

Una parte importante de esta plataforma es el software de los dispositivos clientes y servidores, que ha sido el mercado habitual de Microsoft. Para los dispositivos clientes, Microsoft planea integrar .NET en cualquier dispositivo imaginable, como PCs con Windows, agendas electrónicas con Pocket PC, teléfonos móviles, su consola de videojuegos X-Box, en WebTV, etc. Esto supone para las empresas aumentar el número de potenciales clientes que puedan utilizar su servicios (ya no están limitados al PC). Para poder ejecutar estos servicios, Microsoft introduce una serie de softwares englobados dentro de los .NET Enterprise Servers, como es el Application Center, Commerce Server, etc.

Estos servicios se ofrecerán al cliente a través de distintos canales, lo que Microsoft ha denominado Experiencias de Usuarios. Así, Microsoft ha pensado que MSN sea el canal para clientes domésticos y bCentral es el canal de comercio electrónico para empresas.

Arquitectura de .NET

Una definición general de la arquitectura .NET podría ser la siguiente [4]: "Una plataforma independiente del lenguaje para el desarrollo de servicios Web". (Enrique Hernández Orallo)[Introducción a Microsoft. NET].

La arquitectura .NET (.NET Framework) es el modelo de programación de la plataforma .NET para construir y ejecutar los servicios .NET. El objetivo de esta arquitectura es la de reducir la complejidad en el desarrollo de este tipo de aplicaciones, permitiendo a los desarrolladores centrarse en escribir la lógica específica del servicio a desarrollar. Esta arquitectura está compuesta por librerías tal como muestra la figura:

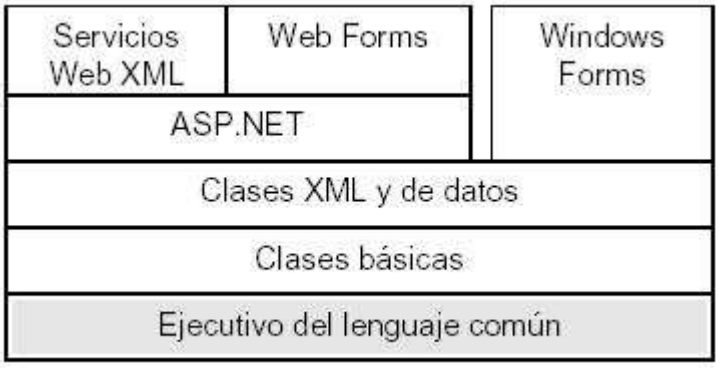

Figure 8 Composición de la arquitectura .Net

El ejecutivo del lenguaje común (CLR: Common Languaje Runtime) es un soporte que permite ejecutar los servicios .NET en cualquier máquina que lo disponga. Está basado en la idea de Java, que también tiene un módulo de ejecución independiente del sistema operativo donde se vaya a ejecutar. La gran diferencia con Java es que este ejecutivo es multilenguaje, esto es, no está limitado a un único lenguaje como Java. Esto permite al desarrollador utilizar una amplia variedad de lenguajes como C++, Visual Basic y C#.

Las librerías básicas proporcionan una serie de funcionalidades que son necesarias a la hora de desarrollar los servicios Web. Las clases básicas gestionan las operaciones más básicas como las

comunicaciones, entrada/salida, seguridad, etc. Las clases XML y de datos gestionan el acceso a base de datos y la gestión de datos en XML. El objetivo de las librerías Servicios Web XML es la de dar soporte para el desarrollo de aplicaciones distribuidas que ofrezcan servicios XML a otras entidades. Las Web forms permiten desarrollar la parte gráfica de una aplicación para la Web, mientras las Windows Forms están orientadas a implementar la parte gráfica de las aplicaciones clásicas para Windows.

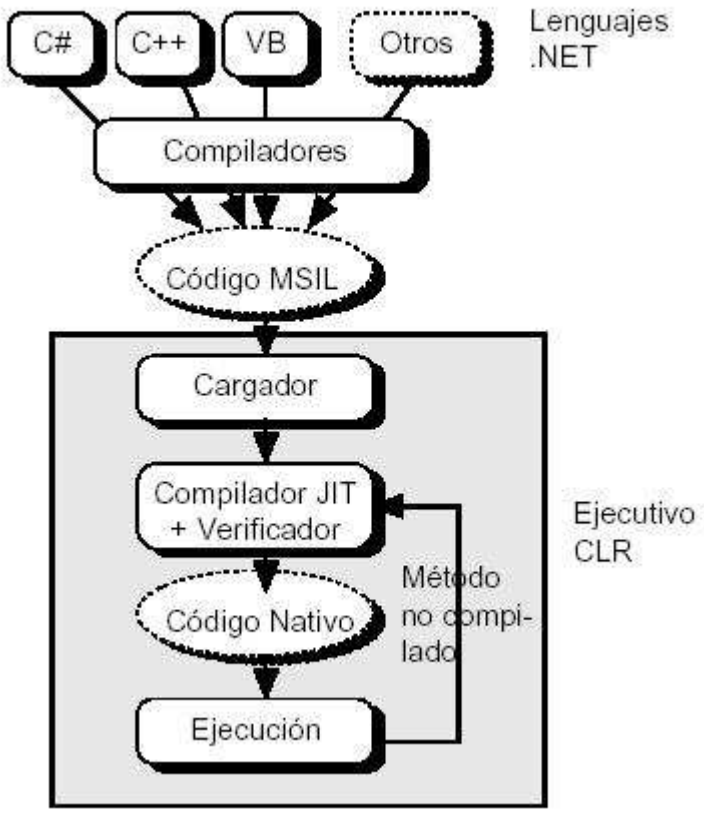

Figure 9 Lenguajes de .Net

Los compiladores producen código MSIL (Microsoft Intermediate Language), que es un lenguaje intermedio que se puede ejecutar en la máquina virtual. Este código no es interpretado por el ejecutivo, sino que es compilado de nuevo en tiempo de ejecución (JIT: Just in Time) al código nativo de la máquina. Este código compilado no se ejecuta independientemente sino dentro de este ejecutivo. Esto se

denomina código manejado, lo cual permite que el ejecutivo controle ciertos aspectos de la aplicación que ejecuta como son seguridad, gestión de memoria, compartición de datos, etc. Aparte de Microsoft, existe actualmente un proyecto de Software Abierto para implementar toda esta arquitectura en Linux que se denomina MONO [5]. El objetivo es portar el ejecutivo CLR a Linux e implementar un compilador C#. Esto es muy interesante, ya que rompería uno de los objetivos de Microsoft, que es que la plataforma .NET sólo se ejecute en sus sistemas operativos Windows.

Herramienta de Desarrollo Visual Studio.NET.

El objetivo principal de este entorno de desarrollo es la de simplificar el desarrollo de aplicaciones Windows y servicios Web permitiendo la elección del lenguaje de programación más adecuado (Visual Basic, C++ o C#).

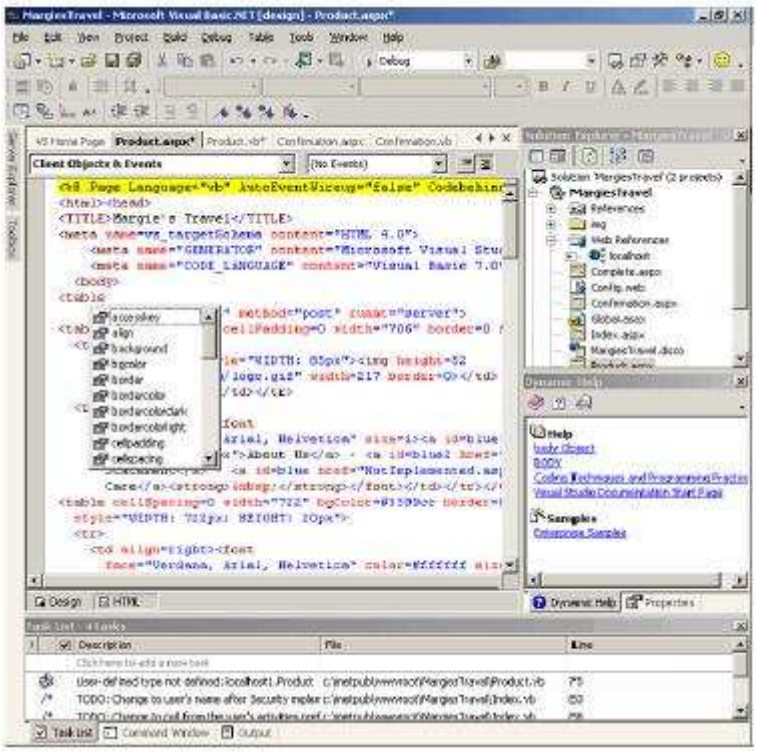

Figure 10 Ambiente.Net.

Aparte de poder elegir el lenguaje de programación hay que decidir qué tipo de aplicación se va a desarrollar, y en este caso se distingue ya entre las aplicaciones Windows tradicionales y los servicios Web, todo en el mismo entorno.

Visual Studio .NET sigue teniendo muchas de las características de versiones anteriores, como el entorno de edición, compilación y depuración integrado, gestión de proyectos complejos, diseño con notación UML. Pero con lo que respecta a la plataforma .NET se enumeran las principales características o mejoras al desarrollo que proporciona este entorno de desarrollo:

• *Ejecutivo común*: Como se ha comentado antes todos los lenguajes en la arquitectura .NET utilizan un módulo de ejecución común con librerías comunes. Con esto se termina con los distintos módulos de ejecución para cada lenguaje (como vbrun.dll para Visual Basic o msvc42.dll para Visual C++).

• *Clases unificadas*: Hasta ahora, cada lenguaje tenía su conjunto de clases o librerías para poder desarrollar programas Windows (las MFC en C++, VB Framework en Visual Basic). Esto implicaba que para cambiar de lenguaje, era necesario, aparte de conocer la sintaxis del lenguaje, conocer las librerías a utilizar. En la nueva plataforma .NET estas librerías o clases son comunes para todos los lenguajes, con lo que los desarrolladores no tienen que aprender una nueva librería cuando cambian de lenguaje.

• *Integración multilenguaje*: Además de los puntos anteriores, se incluye la posibilidad de llamada a métodos de otros objetos desarrollados en otros lenguajes e incluso su herencia. Esto permite desarrollar objetos en el lenguaje más apropiado para el problema a solucionar.

• *ASP.NET*: Esta librería proporciona un nuevo modelo para la creación de aplicaciones Web. Esto permite crear gráficamente páginas Web utilizando una serie de controles (desde el tipo campo de edición, hasta calendarios). Estos servicios se compilarán en el servidor y al cliente se le genera en tiempo de ejecución la página HTML apropiada para el navegador que se utilice.

• *ADO.NET*: Esta librería proporciona un acceso común a los datos, ya sea en bases de datos o XML.

• *Plataforma abierta*: A este entorno de desarrollo se le pueden añadir herramientas o nuevos lenguajes de programación, de tal forma que estén perfectamente integrados en Visual Studio.

De esta forma se van a poder utilizar distintos lenguajes de programación como Eiffel, Perl, Java e incluso lenguajes tan venerables como Cobol y Fortran. Además de todo este soporte, Microsoft ha desarrollado un lenguaje nuevo de programación basado en C++ denominado C# (C sharp).

Selección del Lenguaje C#

Se eligió este lenguaje ya que posee en primer lugar las estructuras de programación necesarias para poder implementar los requerimientos planteados en la fase de análisis y diseño de este sistema como son: un lenguaje que fuera totalmente orientado a objetos, con interfaces de implementación, que diera soporte al manejo de eventos, delegados de la manera más orientada a objeto posible.

Caracteristicas de C#

Sencillez: C# elimina muchos elementos que otros lenguajes incluyen y que son innecesarios en .NET. Por ejemplo: el código escrito en C# es auto contenido (no necesita de ficheros adicionales al propio fuente tales como ficheros de cabecera) y el tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o máquina para quienes se compile (no como en C++), lo que facilita la portabilidad del código.

Modernidad: C# incorpora en el propio lenguaje elementos que a lo largo de los años ha ido demostrándose son muy útiles para el desarrollo de aplicaciones: tipo básico decimal (128 bits), instrucción foreach, tipo básico string, etc.

Orientación a Objetos: C# no admite ni funciones ni variables globales, sino que todo el código y datos han de definirse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código. Soporta todas las características propias del paradigma de programación Orientado a objetos: polimorfismo, la encapsulación y la herencia.

Orientación a componentes: La sintaxis de C# permite definir cómodamente propiedades (similares a campos de acceso controlado), eventos (asociación controlada de funciones de respuesta a notificaciones) o atributos (información sobre un tipo o sus miembros).

Gestión automática de memoria. Todo lenguaje de .NET tiene a su disposición el recolector de basura del CLR.

Seguridad de tipos. C# incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente.

No se pueden usar variables no inicializadas: Se comprueba que los accesos a los elementos de una tabla se realicen con índices que se encuentren dentro del rango de la misma.

Delegados : Son similares a los punteros a funciones de C++ pero siguen un enfoque orientado a objetos: pueden almacenar referencias a varios métodos simultáneamente, y se comprueba que los métodos a los que apunten tengan parámetros y valores de retorno del tipo indicado al definirlos.

Instrucciones seguras. En C# se han impuesto una serie de restricciones en el uso de las instrucciones de control más comunes (toda condición está controlada por una expresión condicional, todo caso de un switch ha de terminar en un break o goto, etc.).

Extensibilidad de operadores: C# permite redefinir el significado de la mayoría de los operadores, incluidos los de conversión, tanto para conversiones implícitas como explícitas cuando se apliquen a diferentes tipos de objetos.

Extensibilidad de modificadores: C# ofrece, a través del concepto de atributos, la posibilidad de añadir, a los meta datos del módulo resultante de la compilación de cualquier fuente, información adicional a la generada por el compilador que luego podrá ser consultada en tiempo ejecución a través de la biblioteca de Reflexión de .NET. Esto, que más bien es una característica propia de la Plataforma .NET y no de C#, puede usarse como un mecanismo para definir nuevos modificadores.

Versionable: C# incluye una política de versionado que permite crear nuevas versiones de tipos sin temor a que la introducción de nuevos miembros provoque errores difíciles de detectar en tipos hijos previamente desarrollados y ya extendidos con miembros de igual nombre a los recién introducidos.

2.1.4 Metodología seleccionada (RUP)

Según el Proceso Unificado de Desarrollo de Software la arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros por plataformas de software, sistemas operativos, manejadores de bases de datos, protocolos, consideraciones de desarrollo como sistemas heredados y requerimientos no funcionales. Se representa mediante varias vistas que se centran en aspectos concretos. [Ayuda Rational, 2003]

Todas las vistas juntas forman el llamado modelo 4+1 de la arquitectura, recibe este nombre porque lo forman las vistas lógica, de implementación, proceso y despliegue, más la de casos de uso que es la que da cohesión a todas.

La metodología de desarrollo RUP (Proceso Unificado de Software) plantea que el rol de arquitecto es el responsable por el desarrollo y mantenimiento de la arquitectura de software, la que incluye como hemos visto las principales decisiones técnicas y las restricciones sobre el diseño e implementación del proyecto.

El rol de arquitecto existe en otras metodologías de desarrollo, con actividades diferentes pero con el mismo objetivo dentro del equipo de desarrollo.

¿Qué características debe cumplir un arquitecto?

“El arquitecto ideal debe ser una persona de letras, matemático, al corriente de estudios históricos, un estudiante diligente de la filosofía, conocido de la música, no ignorante de medicina, entendido en las espuestas de consultoría jurídica, al corriente de astronomía y de cálculos astronómicos” [Ayuda Rational, 2003]
Resumiendo, el arquitecto del software debe tener visión, y una profundidad de la experiencia que permite tomar decisiones rápidamente y hacer el juicio adecuado, crítico en ausencia de la información completa [Ayuda Rational, 2003]

Las principales actividades que realiza el arquitecto son:

- Priorizar los Casos de Uso

- Definir los Casos de Uso como: críticos, secundarios, auxiliares u opcionales, lo que permite definir los módulos, subsistemas y escenarios así como la interacción entre ellos, que permita tomar decisiones hacia donde centrar los esfuerzos de implementación.
- Análisis de la arquitectura: Definir la arquitectura candidata del sistema teniendo en cuenta arquitecturas similares u otros sistemas, definir además los estilos arquitectónicos, patrones de arquitectura, los principales mecanismos de diseño arquitectónico.
- Identificar mecanismos de diseño: Refinar el análisis de la arquitectura teniendo en cuenta las restricciones impuestas por el entorno de implementación.
- Estructurar el modelo de implementación: Permite establecer la estructura en la que va a residir la implementación del sistema.
- Reutilización de elementos de diseño existentes: Permite analizar las interacciones en los diagramas de clases del Análisis para encontrar interfaces, diseño de clases y subsistemas. Refinar la arquitectura e incorporar elementos reutilizables si es posible, identificar problemas comunes a los que se pueda crear soluciones generales o comunes (patrones, o familias de productos).
- Identificar los elementos de diseño: Analizar las interacciones entre las clases de Análisis para identificar los elementos de modelo de diseño arquitectónico.
- Describir la arquitectura en tiempo de ejecución: Analizar requerimientos de concurrencia, identificar los procesos y la comunicación entre dichos procesos, identificar el ciclo de vida de dichos procesos. En este caso no se propondrá en la solución la utilización de la vista de procesos, ya que esta es opcional dependiendo de las características del sistema.

Los principales artefactos a desarrollar son:

- Documento Descripción de la Arquitectura (4 Vistas)
- Modelo de Despliegue
- Modelo de Implementación
- Protocolos
- Interfaces

2.2 Descripción de la Arquitectura

Este documento proporciona comprensión arquitectónica global del sistema, usando un número de vistas arquitectónicas diferentes, se muestra diferentes aspectos del sistema. Este documento permite capturar y tomar decisiones arquitectónicamente significativas que deban ser tomadas en el desarrollo del sistema.

Propósito

Los propósitos que se persiguen la descripción de la arquitectura son:

- Mejor comprensión del sistema.
- Organización del Desarrollo.
- Fomento de la reutilización
- Hacer evolucionar el sistema.

Alcance

Abarca todo el sistema Índices de Precios del consumidor (IPC).

Definiciones, Acrónimos y Abreviaturas

IPC: Índices de Precios al Consumidor

BD: Base de Datos

CU: Caso de Uso

Representación arquitectónica

Este documento presenta la arquitectura como una serie de vistas:

- Vista de Caso de Uso
- Vista Lógica
- Vista de implementación
- Vista de despliegue

Estas vistas están representadas en UML usando Rational Rose 2003.

Objetivos y Restricciones de la Arquitectura

Existen algunos objetivos y restricciones del sistema en cuanto a seguridad, portabilidad y reusabilidad que son significativas para la arquitectura. Ellos son:

- El servidor de la entidad debe de soportar varias conexiones a la vez por lo que debe de tener como mínimo 248 MG de RAM para poder responder a todas las solicitudes que se le puedan hacer al servidor montado. Esta capacidad satisface la necesidad de memoria para el servidor de Base de Datos (SQL Server 2000) que estará en esta misma máquina.
- La capacidad de disco duro en el servidor debe de ser no menos de 10 GB, en el caso de los clientes no menos de 5 GB.
- La red existente en las instalaciones debe de soportar la transacción de paquetes de información de al menos unas 10 máquinas recurrentes.
- Para hacer más fiable la aplicación debe de estar protegida contra fallos de corriente y de conectividad, para lo que se deberá parametrizar los tiempos para realizar copias de seguridad. Implementar las transacciones de paquetes en la red con el protocolo TCP/IP que permite la recuperación de los datos.

- En cuanto a los sistemas operativos tanto los clientes como el servidor deben de tener Windows 2000 o superior.
- Todas las estaciones de trabajo contarán con la instalación del Framework 2.0.
- Todos los componentes estarán encapsulados en cada uno de los paquetes correspondiente a sus casos de uso.

Herramientas de desarrollo

Las herramientas de desarrollo que se utilizarán son:

- Para el modelado Rational Rose 2003 el cual demanda como mínimo 256 MB de RAM y recomendado 512 MB de RAM.
- Para implementación como IDE de desarrollo Visual Studio 2005 el cual demanda como mínimo 512 MB de RAM.
- Como servidor de Base de Datos SQL Server 2000.
- Sistema de control de versiones Subversión.

Estructura del equipo de Desarrollo:

El equipo de desarrollo está formado por estudiantes.

- Un Arquitecto Principal.
- Un analista.
- Un Diseñador.
- Un Programador.
- Un Documentador.
- 2 Ingeniero de pruebas
- 2 Diseñadores de Base de Datos

Restricciones de acuerdo a la estrategia de diseño

- El diseño de las aplicaciones se hará utilizando la Programación Orientada a Objetos (POO). Encapsulación de la lógica por clases.
- Se utilizará las tecnologías que brindan los frameworks definidos para cada una de las capas de la aplicación:
- Para la capa de presentación: Framework 2.0 de .Net
- Para la capa de lógica del negocio: los objetos del negocio, Framework 2.0 de .Net
- 2.4. Para la capa de Acceso a Datos: Tier Developer 4.03

2.2.1 Vista de Casos de Uso

A partir de la vista de Casos de Uso, se puede definir los escenarios o los casos de uso que serán de interés para cada iteración del ciclo de desarrollo. Esta describe los escenarios o casos de uso que tienen significación y que encapsulan la funcionalidad central del sistema. Los casos de uso del sistema arquitectónicamente más significativo de la presente iteración se muestran a continuación:

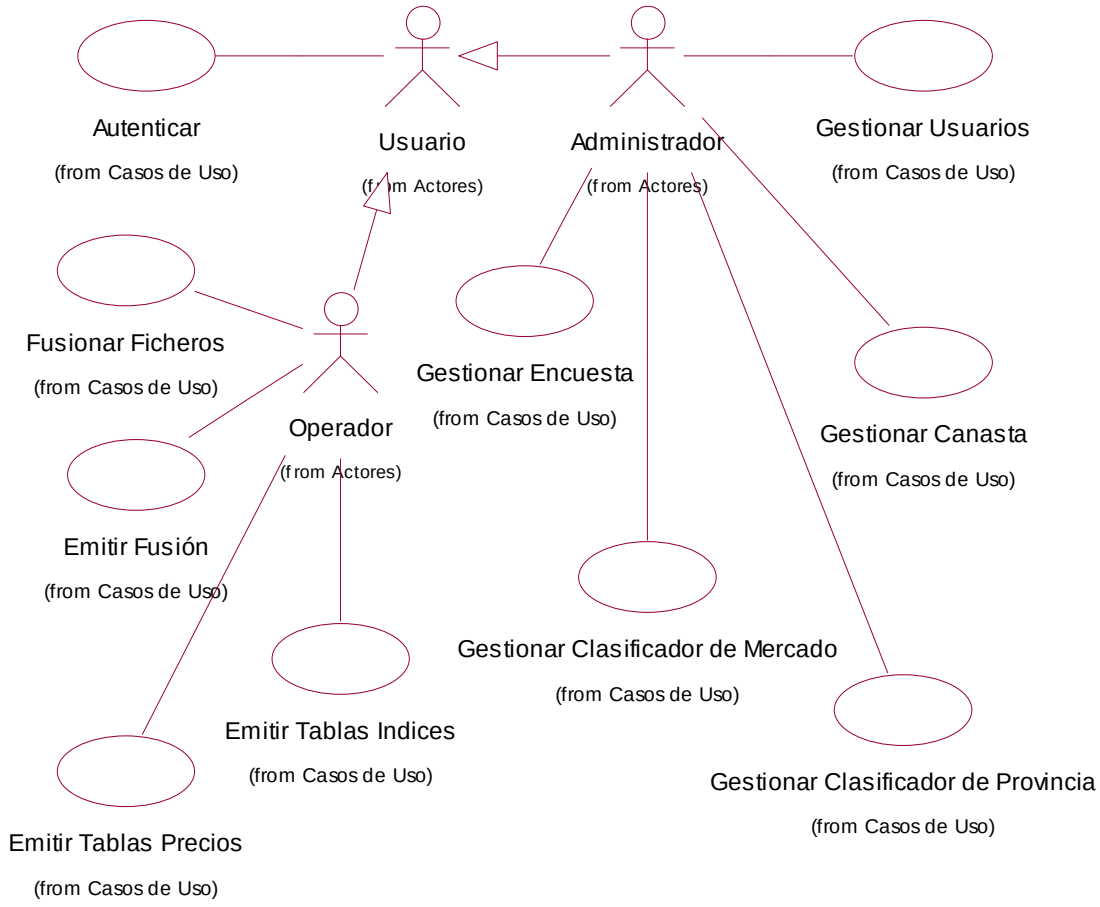

Figure 11 Diagrama de casos de Uso Significativos para la Arquitectura

- Caso de uso Autenticar.

El CU se inicia cuando e l usuario solicita el acceso al sistema. El mismo introduce su nombre de usuario y contraseña. El sistema procesa los datos entrados, permitiendo o no el acceso al sistema en dependencia del rol del usuario. Para el rol de Operador no se tiene acceso a las opciones de administración del menú, para e rol de Administrador no hay límites de acceso. Si a la hora de autenticarse un usuario, los datos introducidos no existen o no corresponden con los almacenados en base de dato (BD), el sistema niega el acceso.

Figure 12 Interfaz de Usuario para la autenticación.

- Gestionar Usuarios

Este CU es realizado por el Administrador y su objetivo es poder controlar los usuarios del sistema. El Administrador puede realizar operaciones con los usuarios para una mejor administración. Las operaciones son: Agregar Usuario, el Administrador puede insertar nuevos usuarios especificando su nombre de usuario, contraseña y rol. La segunda operación es: Modificar Usuario, el Administrador puede realizar cambios en el nombre de usuario, o contraseña, o rol de un usuario cualquiera y salvar los cambios realizados. Operación tercera: Eliminar Usuario, el Administrador puede darle de baja a cualquier usuario del sistema. Necesariamente para la operación de Agregar Usuario, el sistema tiene que verificar que no exista otro usuario con el mismo nombre.

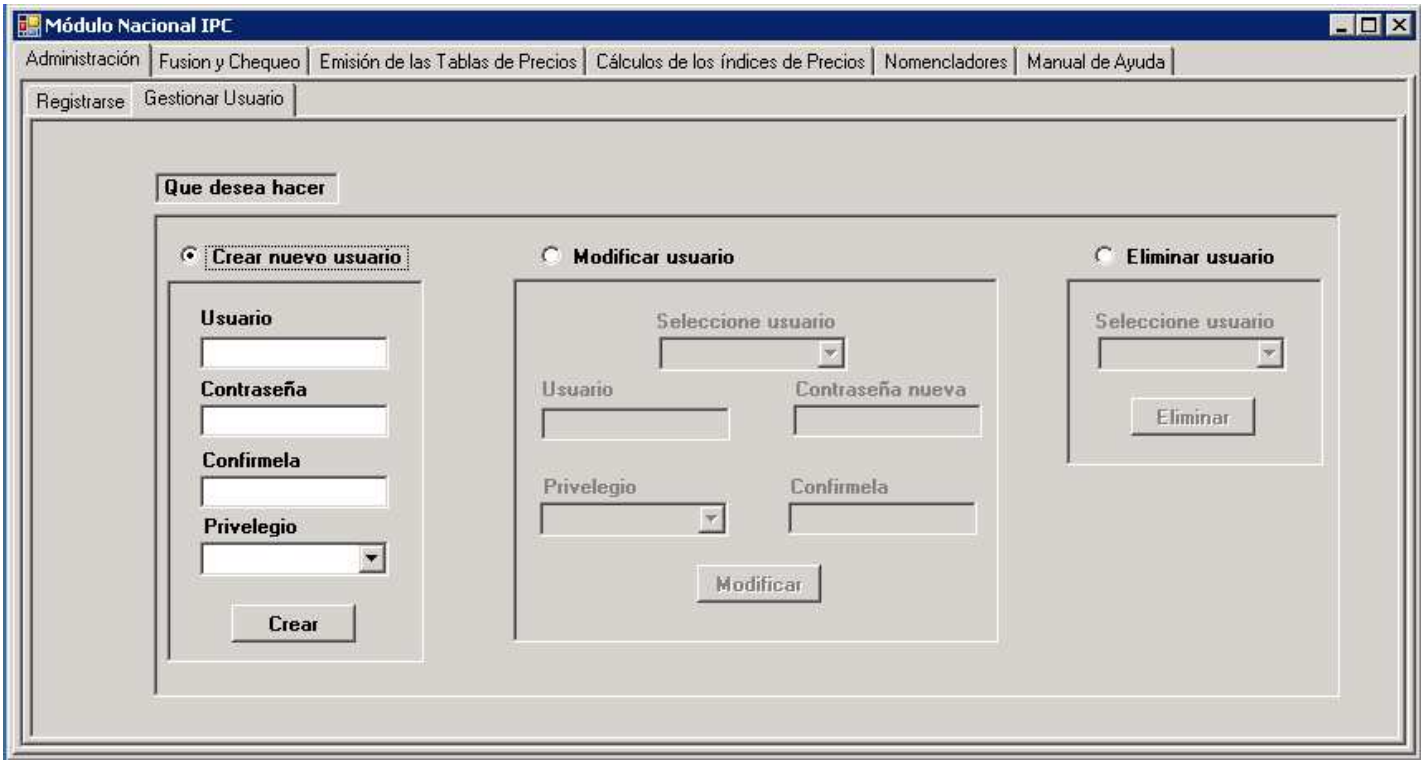

Figure 13 Interfaz de usuario para gestionar los usuarios del sistema.

- Fusión Nacional

Este CU es el encargado de recopilar la información que envían las provincias mensualmente y almacenarla en la BD. Se inicia cuando el usuario hace la petición de realizar una fusión, seleccionando la fecha que desea hacer la fusión. El sistema carga y parsea de un directorio todos los ficheros que fueron enviados en esa fecha, tomando de ellos la información y almacenándola en BD. Si se selecciona una fecha a fusionar y existe información ya fusionada en esa misma fecha, el sistema entonces alerta al Operador de que esa información que existe puede ser reemplazada, dándole opciones al mismo de continuar con la operación o abortar.

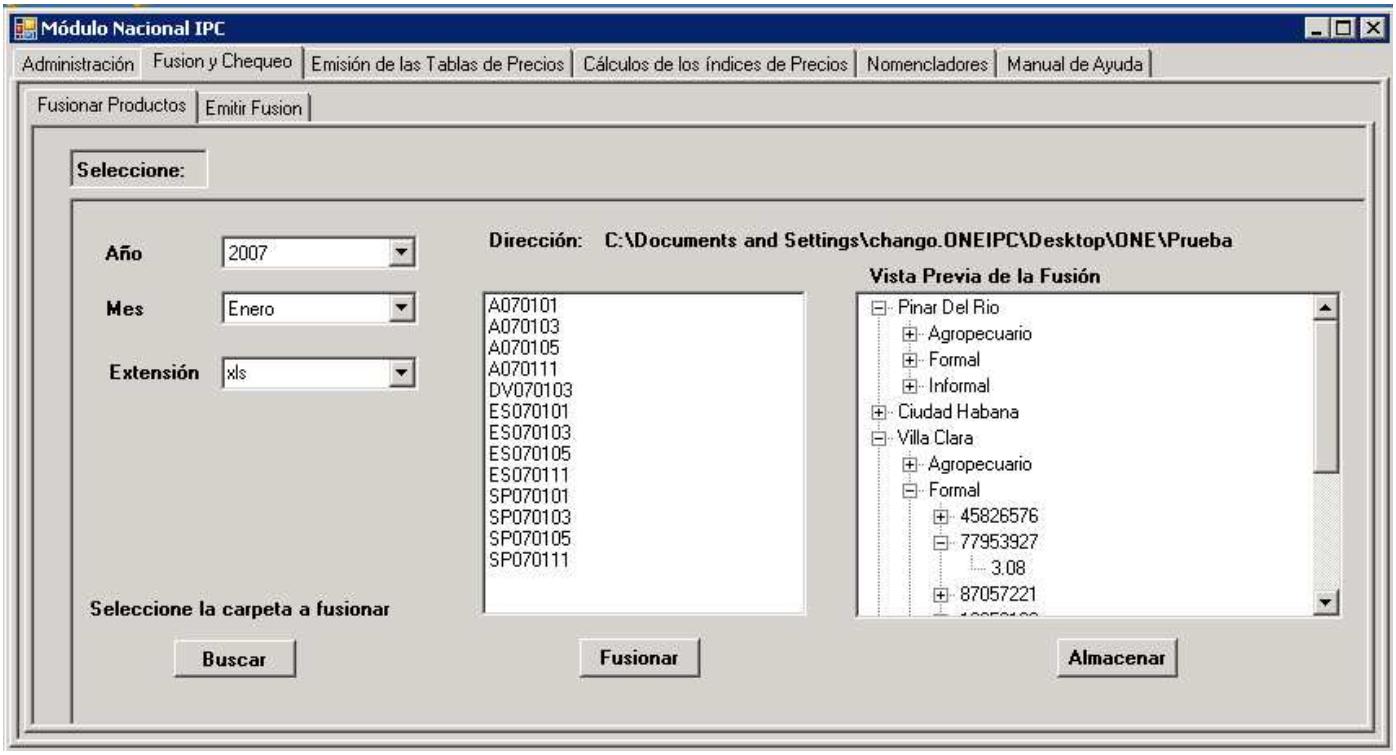

Figure 14 Interfaz de usuario para fusionar los ficheros.

- Emitir Tablas de Precios

Como su nombre lo especifica, este CU se encarga de emitir tablas de precios. El Operador selecciona una fecha, un tipo de mercado, y una región específica para emitir las tablas de precio. El sistema se encarga entonces de recolectar la información correspondiente a esos datos para después procesarla, o sea, calcular los valores que de las variables que se requieren para finalmente proyectar en pantalla las tablas de precios. Al igual que en los casos de uso anteriores, el sistema informa si no existen datos relacionados con la fecha y tipo de mercado seleccionados previamente.

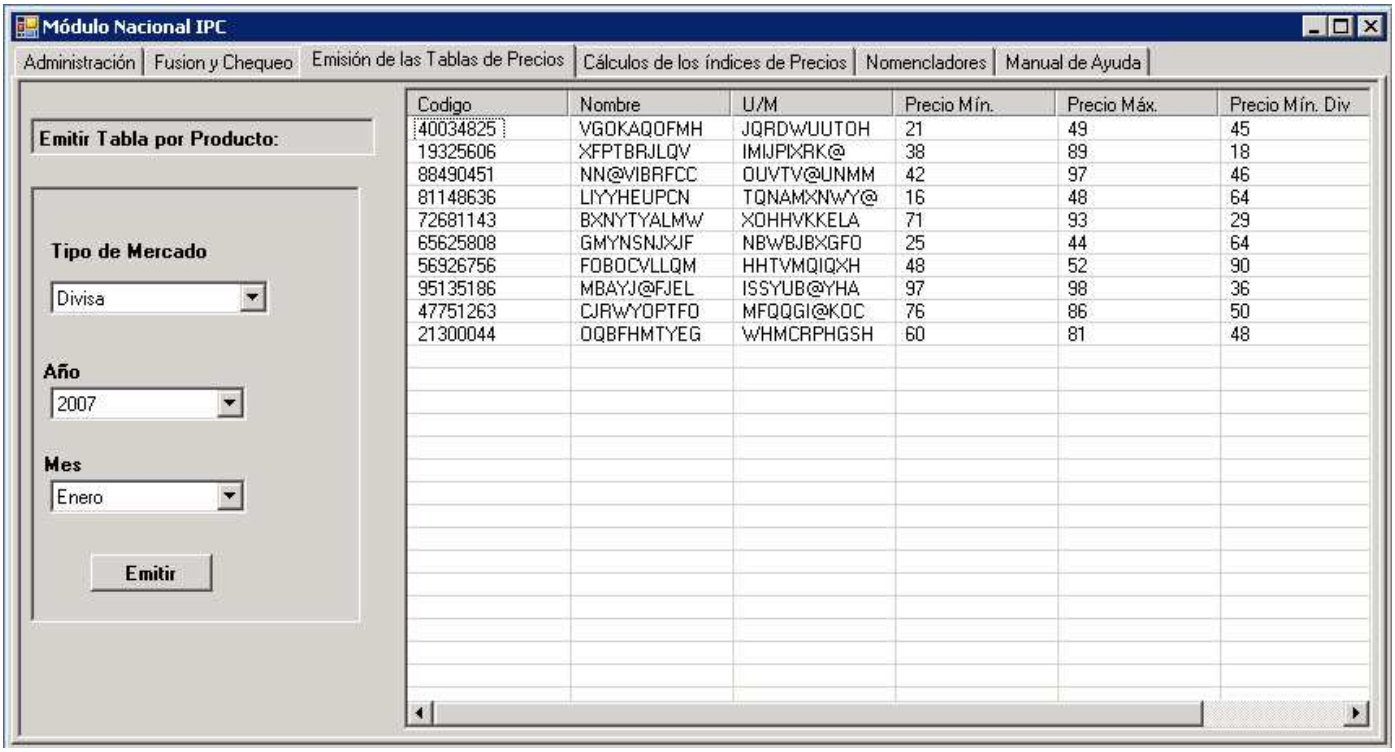

Figure 15 . Interfaz de usuario para emitir las tablas de precios.

- Emitir Tablas de Índices

Este CU es muy similar al descrito anteriormente. Su diferencia radica en el tipo de tabla que se va a emitir en pantalla. El Operador selecciona una fecha, un tipo de mercado, y una región específica para emitir las tablas de índice. El sistema se encarga entonces de recolectar la información correspondiente a esos datos para después procesarla, o sea, calcular los valores de los índices de precios que se requieren para finalmente proyectar en pantalla las tablas de precios. Al igual que en los casos de uso anteriores, el sistema informa si no existen datos relacionados con la fecha y tipo de mercado seleccionados previamente.

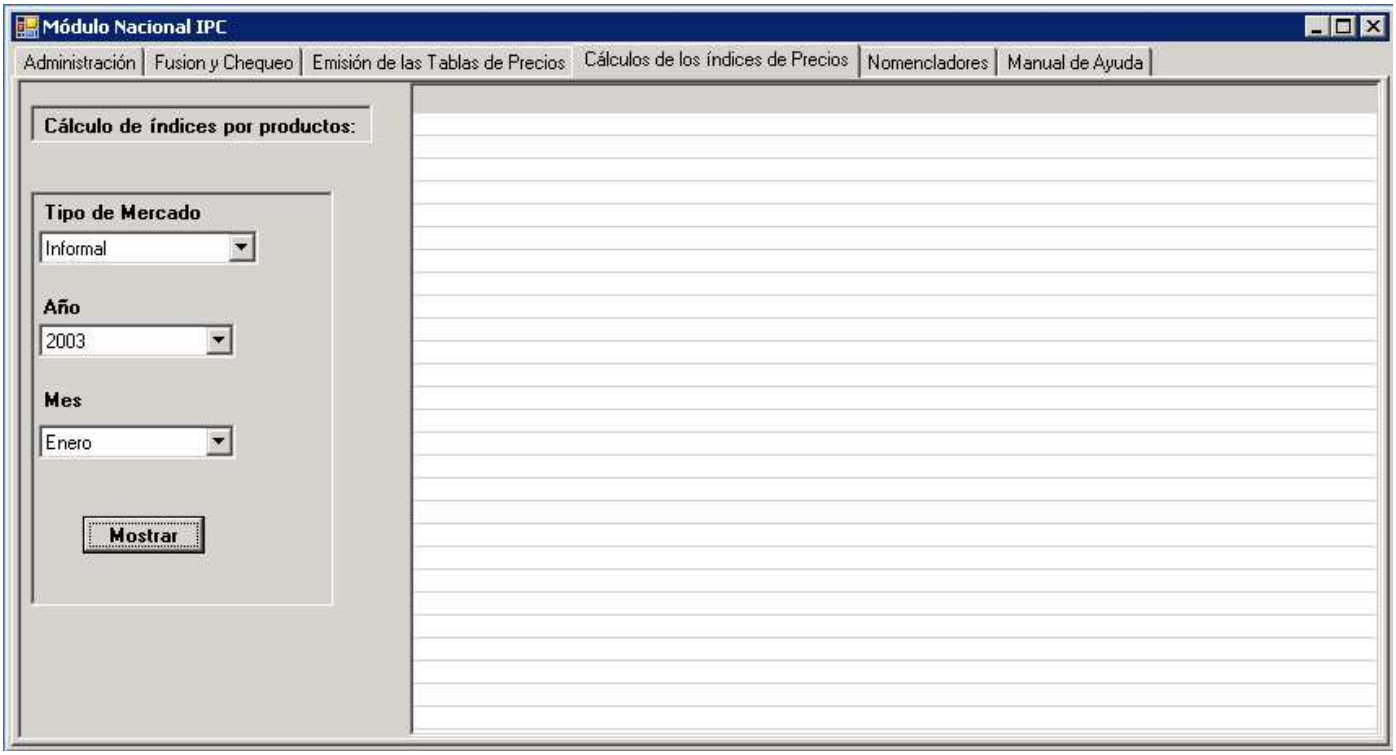

Figure 16 Interfaz de usuario para emitir las tablas de índices.

- Emitir Fusión

Este CU es el encargado de mostrar en pantalla la información referente a las diferentes fusiones que se realizan en el CU Fusionar Ficheros. Esto es, el Operador selecciona la fecha correspondiente a la información que desea ver o imprimir y el sistema se encarga de visualizarle o no esa información a través de una interfaz. Lógicamente se realiza una verificación, si no existe información de la fecha seleccionada, se muestra un mensaje mostrando ese problema.

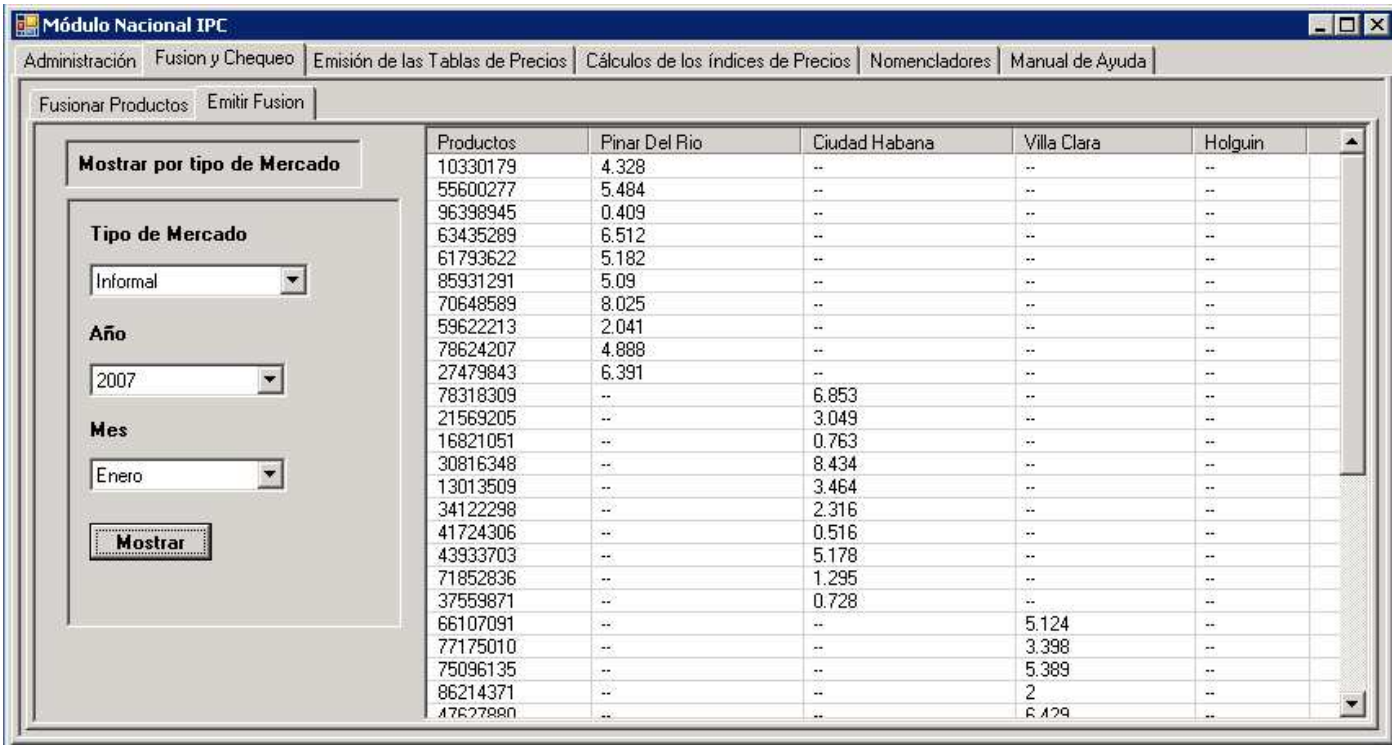

Figure 17 Interfaz de usuario para emitir fusión.

- Gestionar Clasificador de Productos

El Clasificador de Productos es una entidad persistente predefinida que contiene una muestra universal de datos relacionados con Productos. Su uso básico es para nomenclar o validar cada producto que se inserte en BD en cada fusión, y para obtener en algún momento dado el nombre de un producto determinado. Este Clasificador de Productos está expuesto a cambios, por tanto se hace necesario poder manipularlo de alguna forma, y eso es lo que pretende este CU. Las operaciones que se realizan sobre Clasificador son: Eliminar, el Administrador selecciona los datos de un producto dado y los elimina. Esta operación es válida para cuando un producto determinado sale de circulación o deja de existir, por lo que se hace necesario desaparecer sus datos, entiéndase por datos el código, nombre, unidad de medida, etc. Operación Modificar, el Administrador selecciona los datos de un producto determinado y puede modificarlos. Esta operación es válida para cuando existen cambios en la unidad de medida de un

producto, o en el código de un producto, entre otros ejemplos. Este CU representa una nueva funcionalidad con respecto al sistema que usa la ONE.

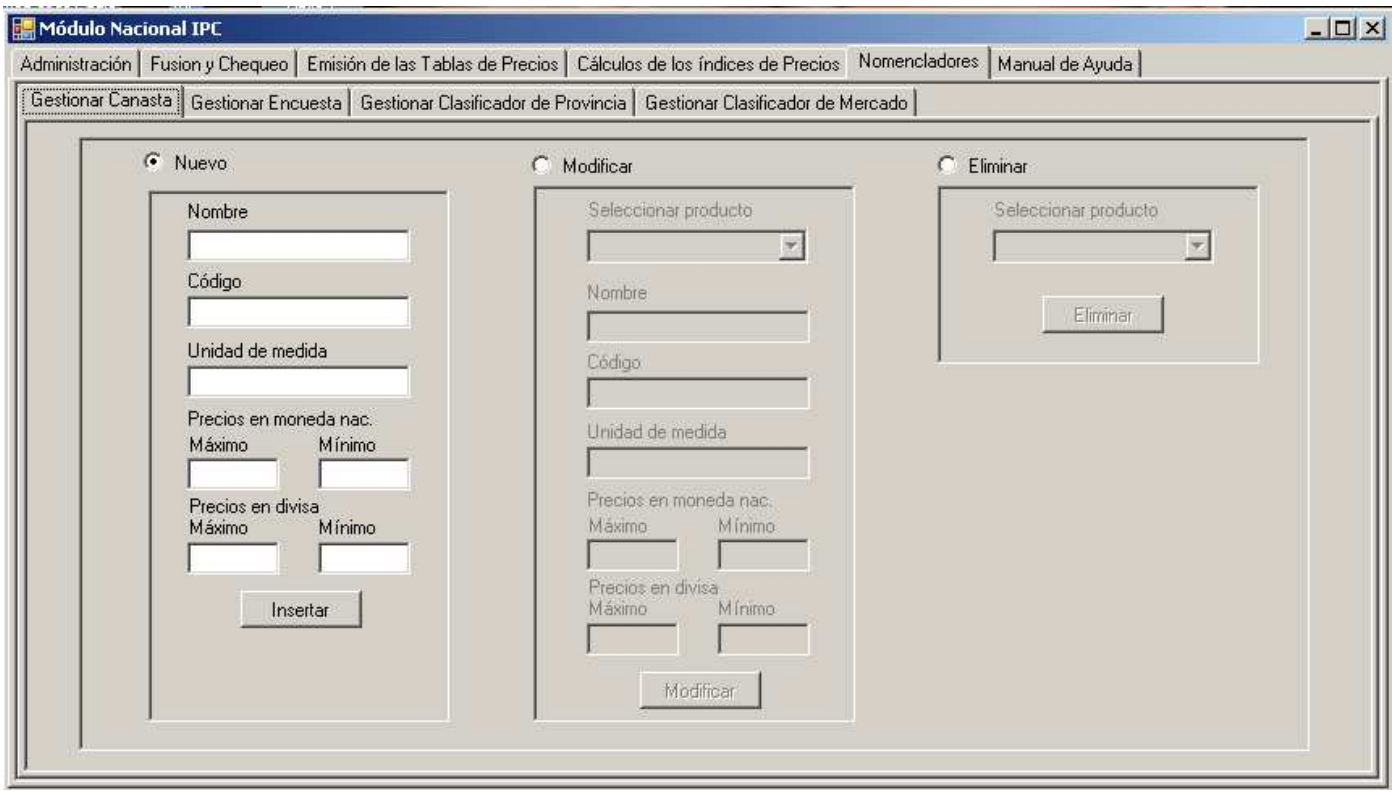

Figure 18 Interfaz de usuario para gestionar los clasificadores de productos

- CU Gestionar Clasificador de Provincia.

El clasificador de provincia es una entidad persistente que contiene el id y el nombre de cada provincia. Esta entidad es la que brinda al sistema la información de cada provincia, por ejemplo, el nombre a la hora de emitir algún reporte. Este caso de uso se basa en poder manipular estos datos con el objetivo de brindar una mayor flexibilidad al sistema. Si en un futuro cambia la división política administrativa, a través del sistema se pueden realizar estos cambios sin tener que reprogramar nada. Las funcionalidades

básicas que brinda este caso de uso son: Eliminar una provincia, insertar una provincia, y modificar los campos de una provincia, o sea, cambiarle el nombre, o el id.

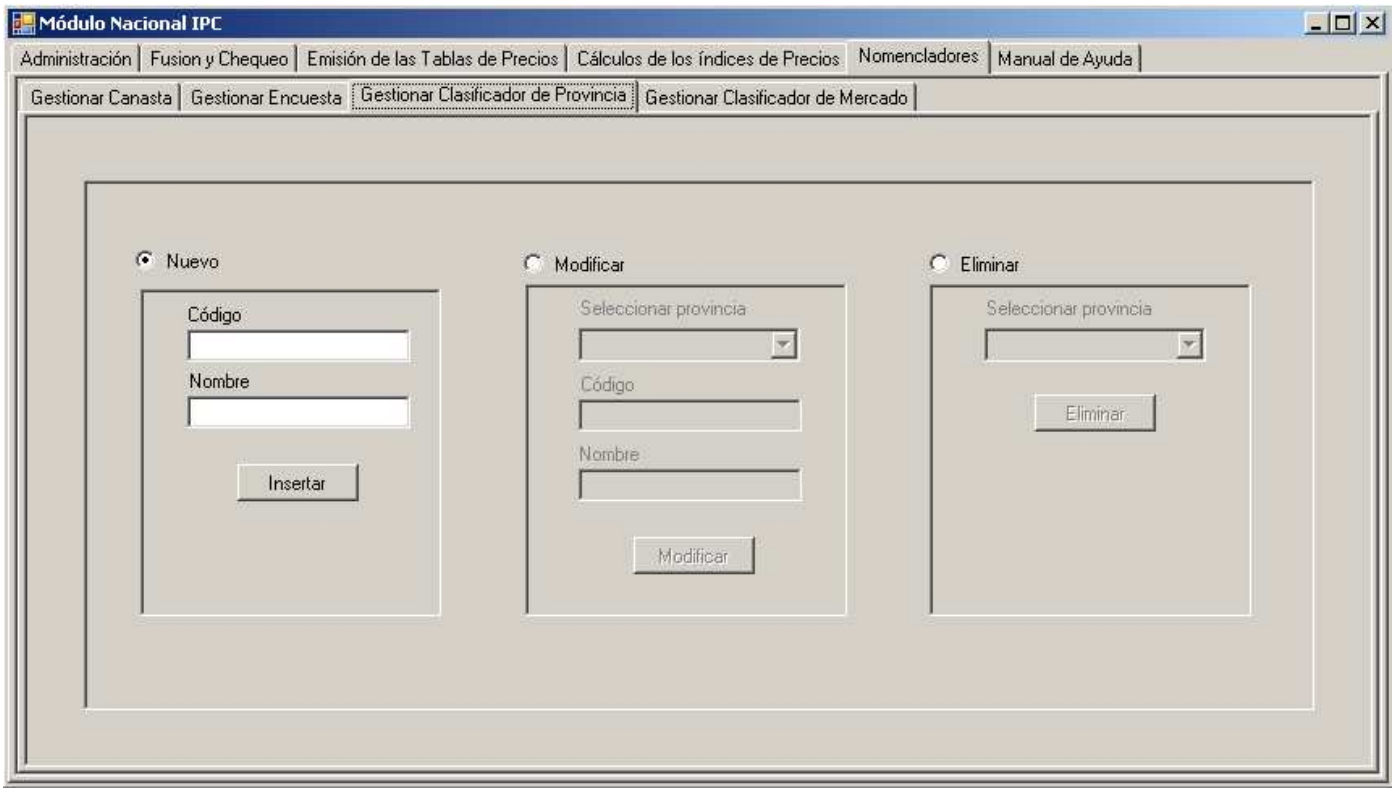

Figure 19 Interfaz de usuario para gestionar los clasificadores de provincias.

- CU Gestionar Encuesta (MOLINA 2007)

La encuesta es una entidad persistente que contiene cuatro precios de cada uno de los mercados para un producto. Estos datos fueron recogidos en el año 1999. En esencia, estos precios recogidos en la encuesta se usan para calcular los índices de precios con respecto al período base. Pero lo fundamental de este caso de uso es poder manipular los datos de la encuesta, con el fin de poder cambiar estos datos en un futuro cuando se realice una encuesta nueva. Las funcionalidades básicas son: Eliminar producto de la encuesta, modificar los campos de un producto de la encuesta, insertar un nuevo producto a la encuesta.

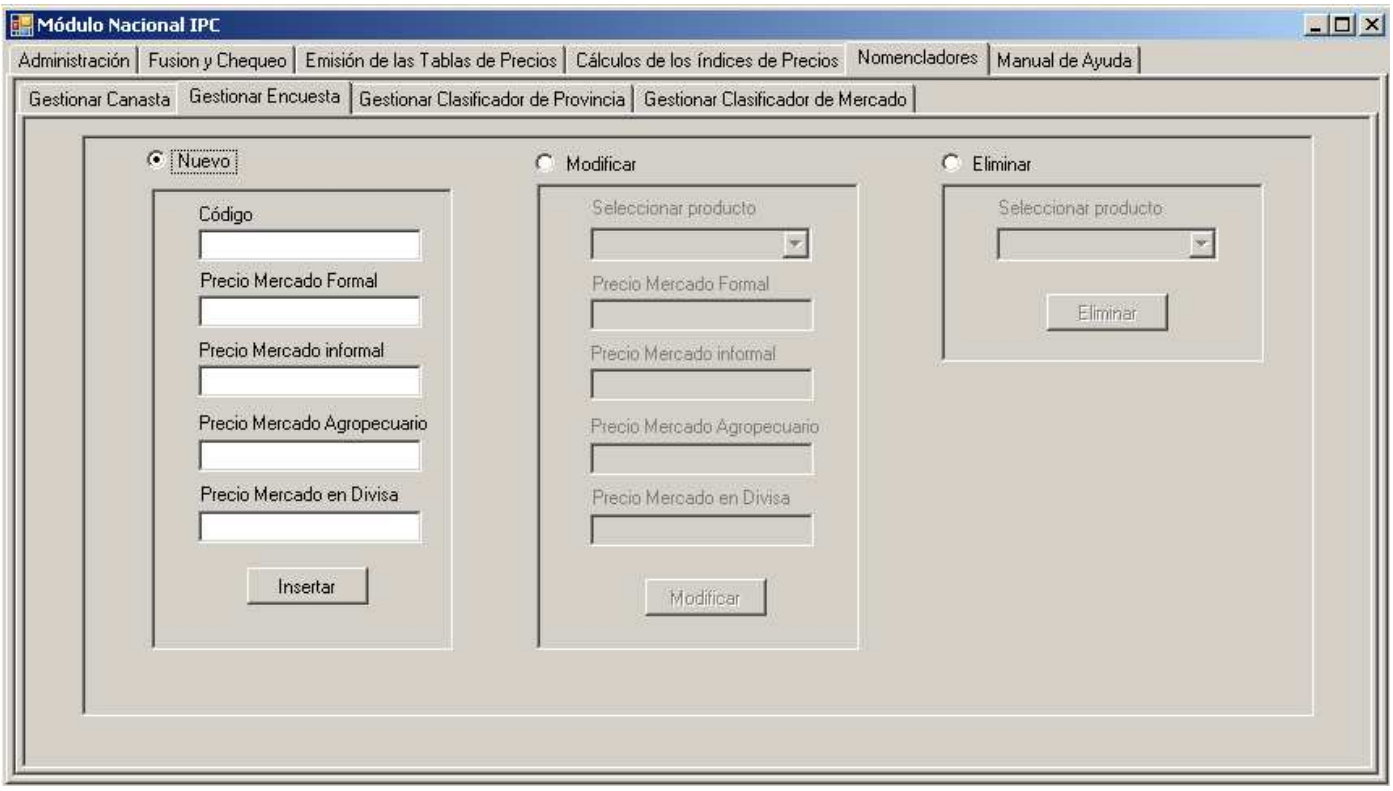

Figure 20 Interfaz de usuario para gestionar la encuesta

- Gestionar Clasificador de Mercado

El clasificador de mercado es una entidad persistente que contiene el id y el tipo de cada mercado. Esta entidad es la que brinda al sistema la información de un mercado, por ejemplo, el tipo de mercado para la hora de parsear un fichero o de emitir un reporte. Este caso de uso se basa en poder manipular estos datos con el objetivo de brindar una mayor flexibilidad al sistema. Si en un futuro cambia alguno de los mercados, a través del sistema se pueden actualizar estos cambios sin tener que reprogramar. Las funcionalidades básicas que brinda este caso de uso son: Eliminar un tipo de mercado, insertar un mercado nuevo, y modificar los campos de un mercado, o sea, cambiarle el tipo o el id.

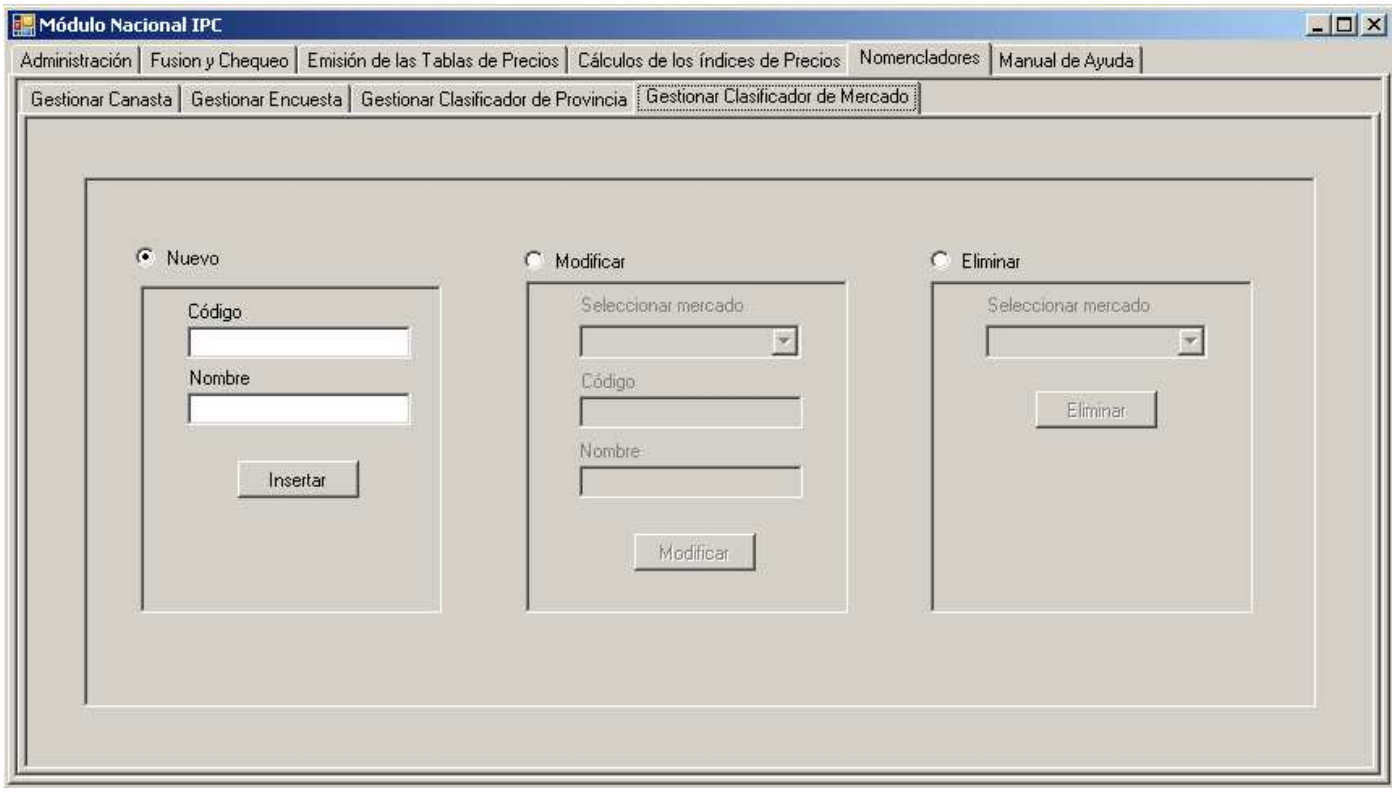

Figure 21 Interfaz de usuario para gestionar los clasificadores de mercados.

2.3 Vista Lógica

La vista lógica tiene gran importancia en el documento de descripción de la arquitectura pues en ella se hace referencia a la distribución lógica de la aplicación en capas, así como la dependencia que existen entre las mismas y por otro lado se encuentra la distribución en subsistemas y el análisis de las interfaces que proporcionan para la comunicación entre ellos.

Distribución en Capas

La vista está formada por cuatro paquetes principales:

Presentación

La capa de presentación contiene los componentes necesarios para habilitar la interacción del usuario con la aplicación. La misma estará formada por componentes de interfaz, como son los formularios Windows

Forms, los componentes de esta capa administran la interactuación con el usuario, muestran los datos al mismo, obtienen sus datos personales e interpretan los eventos generados para actuar en los datos del negocio, cambiar el estado de la interfaz o facilitar la tarea al cliente. Cuando este interactúa con un elemento de la interfaz, se genera un evento que llama al código de una función de control. Ésta, a su vez, llama a componentes empresariales o de negocio los cuales llaman a componentes lógicos de acceso a datos para implementar la acción deseada y recuperar los datos que se han de mostrar.

Aplicación

Contiene las clases controladoras que encapsulan la lógica del negocio.

Lógica de Acceso a Datos:

Encapsula las clases de acceso a datos en tres paquetes fundamentales. El paquete Map proporciona la lógica para el mapeo de las tablas de la base de dato generadas por el Tier Developer. El paquete EnterpriseNonServiceLib que representa al framework que proporciona el Tier Developer para el manejo de la lógica de acceso a datos. Estos componentes abstraen la semántica del almacén de datos a utilizar y la tecnología de acceso a datos y proporcionan una interfaz simple de programación para la recuperación y realización de operaciones con datos. Los componentes lógicos de acceso a datos proporcionan acceso simple a funcionalidad de bases de datos (consultas y operaciones de datos), devolviendo estructuras de datos simples y complejas. Ocultan las idiosincrasias de la invocación y el formato del almacén de datos de los componentes y las interfaces de usuario que las consumen. La implementación de una lógica propia de acceso a datos en los componentes lógicos de acceso a datos permite encapsular toda la lógica de acceso a datos de la aplicación completa en una única ubicación central, lo que facilita el mantenimiento y la extensibilidad de la aplicación. Por último, el paquete DAO, que 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.

Persistentes

Este paquete contendrá las clases que tendrán un tiempo de vida considerablemente largo para la aplicación o sea las clases que son persistentes, será utilizado por la capa de negocio y por la capa de acceso a datos quien se encargara de llevar estas clases al modelo Entidad Relación, es decir al almacén de datos.

La siguiente imagen muestra la distribución de los paquetes más significativos dentro de cada una de las capas definidas para la aplicación y la dependencia entre ellos.

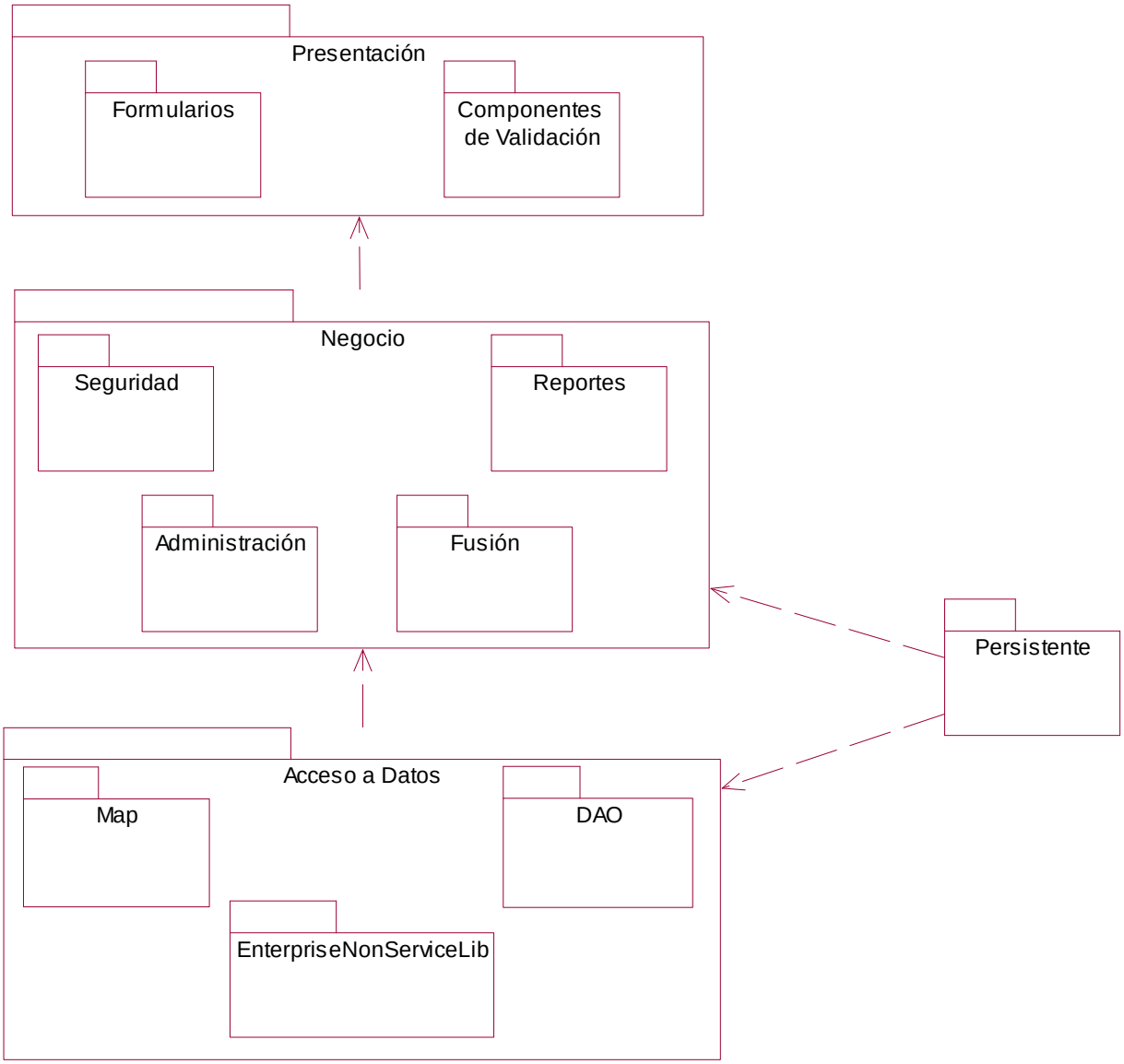

Figure 22 Paquetes Significativos Arquitectónicamente

Subsistemas y Dependencias

Existe una dependencia lógica entre cada Subsistema, como muestra la figura nos encontramos como 4 subsistemas que en su totalidad conforman la aplicación, la ventaja de utilizar esta separación lógica es que una ves que los mismos hayan sido identificados y modelados correctamente se pueden desarrollar en paralelo lo cual acortaría el costo y el tiempo de desarrollo del sistema, cada uno de ellos responden a una organización lógica en capas que permitirá ganar en claridad y que a su ves será capaz de agrupar funcionalidades en cada una de esas capas descritas. Para obtener una arquitectura ejecutable se consideran por cada subsistema los casos críticos o los casos de uso que brindan una funcionalidad básica funcional del sistema, posteriormente se mencionaran por cada subsistema los casos de uso que encapsula que ya han sido descritos en la vista de casos de uso arquitectónicamente significativos. Se presentaran los diagramas de clases incluidos en la realización de cada CU encapsulados en su respectivo Subsistema.

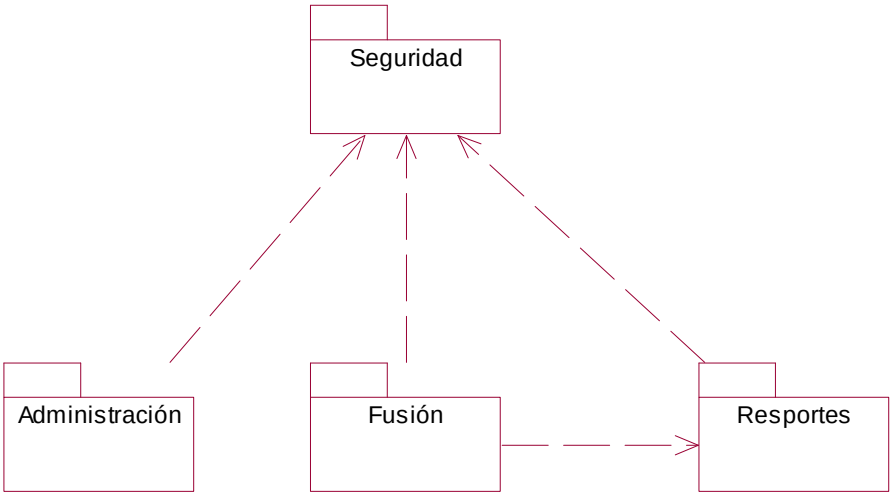

Figure 23 . Relaciones de los subsistemas de diseño

Subsistema Seguridad

Es el subsistema del cual dependerán los demás subsistemas de la aplicación, en este subsistema se hace una agrupación de casos de uso relacionados entre si y que brindan una funcionalidad clara y bien definida, se muestra ademas las principales clases de diseño que participan en la realizacion de dichos casos de uso.

Casos de uso agrupados:

- Autenticar

Si no se cumple con la realización de este caso de uso no se podrá interactuar con las funcionalidades básicas del sistema, que como se podrá apreciar estarán encapsuladas en los demás subsistemas, es por ello que se establece la dependencia que existe entre los demás módulos con el subsistema de Seguridad.

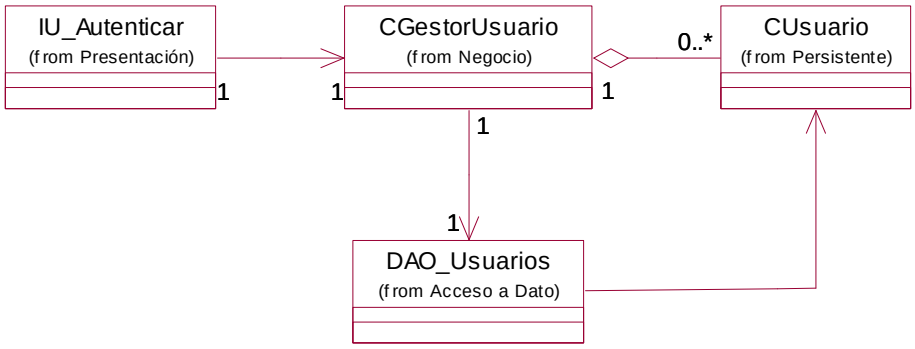

Figure 24 Diagrama de clases de diseño que participan en el Caso de Uso (Autenticar).

- Gestionar Usuarios

Este caso de uso gestiona todo lo relacionado con la administración de los usuarios, ya sean agregar nuevos usuarios, eliminar o modificar datos de usuarios existentes.

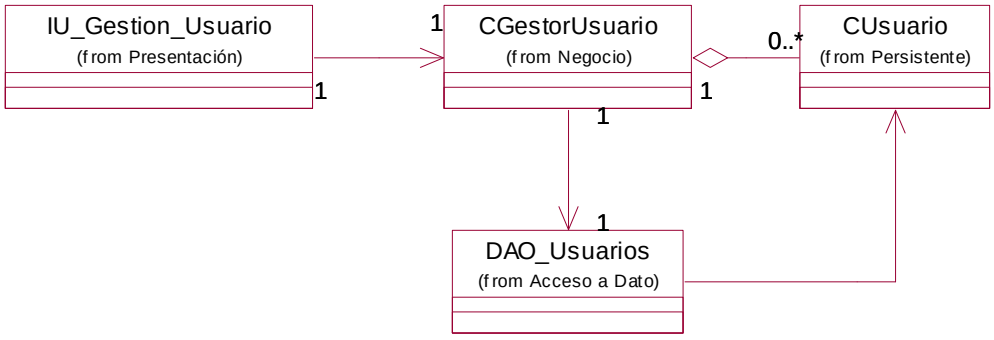

Figure 25 Diagrama de clases de diseño que participan en el Caso de Uso (Gestionar Usuario).

Subsistema Administración

Este subsistema 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 ni 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 , 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.

Aspectos que hacen extensible la arquitectura y que tiene en cuenta este subsistema:

- El subsistema tendrá en cuenta lo relacionados con el aumento o disminución del número de provincias en caso de una nueva distribución política administrativa.
- El subsistema tendrá en cuenta lo relacionado con poder agregar o eliminar productos a un tipo de mercado lo cual no hace rígido la cantidad de productos de los cuales se recogen los datos a nivel provincial y municipal.
- El subsistema será capaz de proveer al usuario funcionalidades relacionadas con el aumento o la disminución de los tipos de mercados . Este es una de las razones por las cuales nuestro país no puede comprar ningún sistema al exterior pues el cálculo de los índices de precios se centra en un solo tipo de moneda y generalmente un tipo de mercado.

Las operaciones que se realizan sobre estos clasificadores son las mismas para los cuatro casos de uso: eliminar, insertar y modificar, entre otras. Para 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.

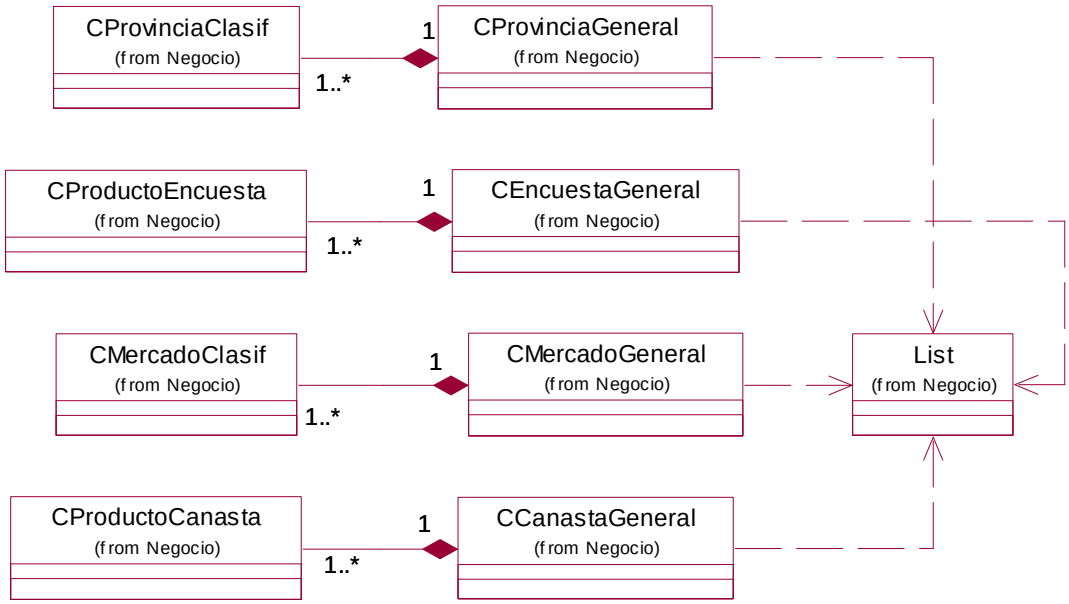

Figure 26 Algunas clases del diseño SD Administración.

Las operaciones que se realizan sobre estos clasificadores son las mismas para los cuatro casos de uso: eliminar, insertar y modificar, entre otras. Para 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.

Casos de Uso que se agrupan en este subsistema:

- Gestionar Clasificador de Provincias.

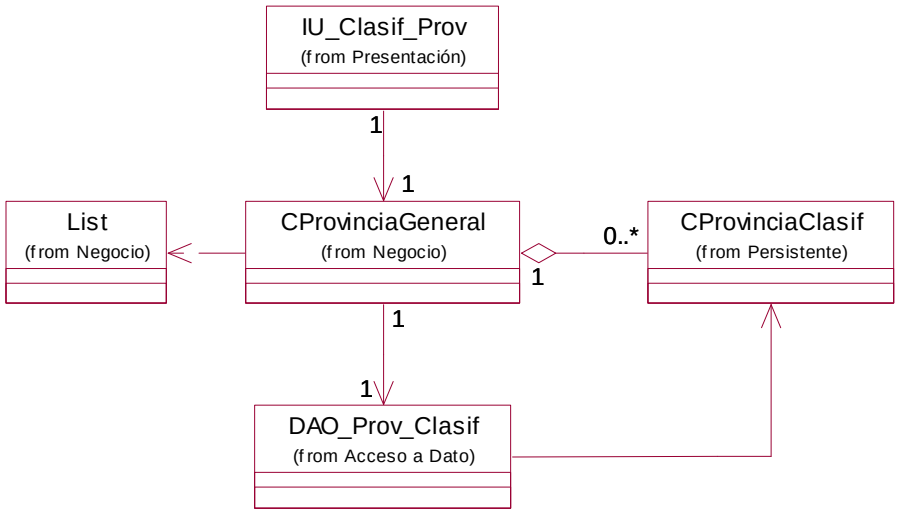

Figure 27 Algunas clases del diseño del CU Gestionar Clasificador de Provincias

- Gestionar Encuesta.

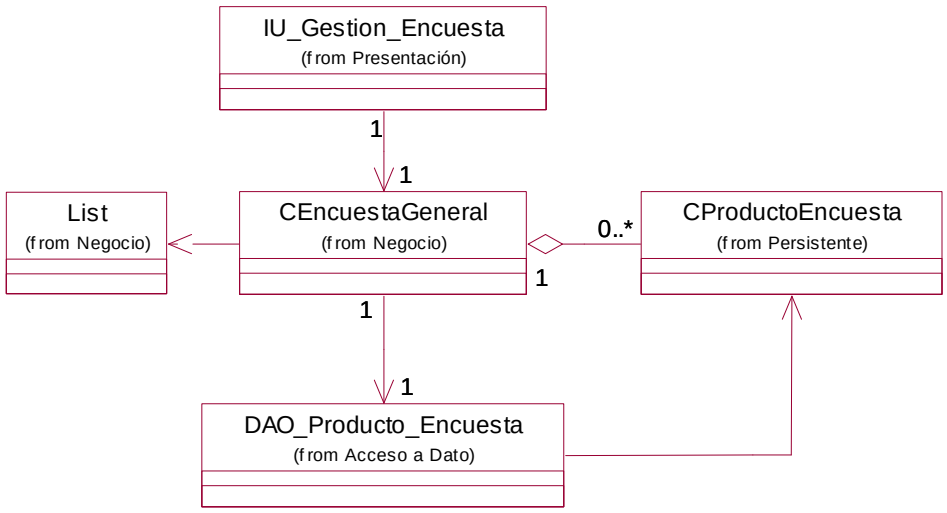

Figure 28 Algunas clases del diseño del CU Gestionar Encuesta

- Gestionar Canasta.

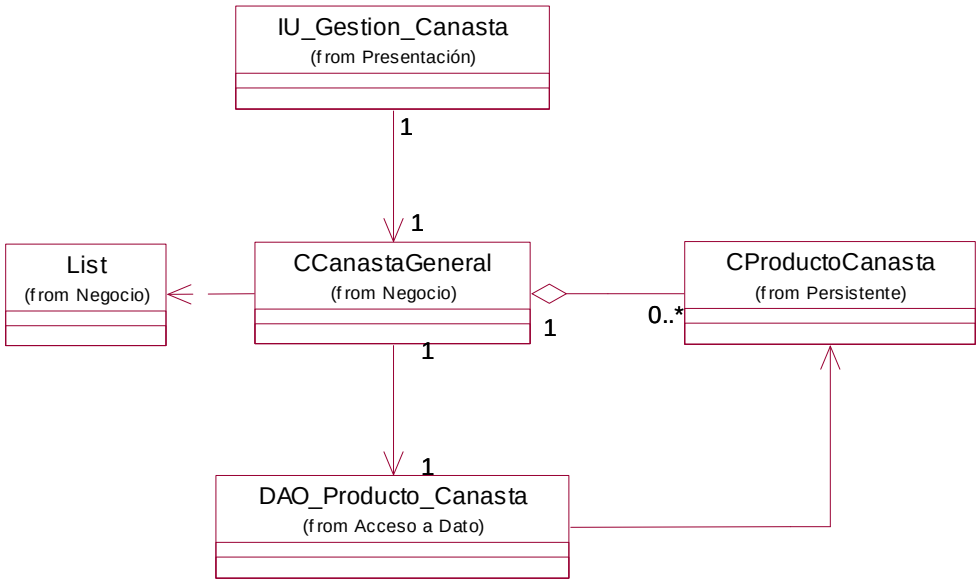

Figure 29 Algunas clases del diseño del CU Gestionar Canasta

- Gestionar Clasificador de Mercado

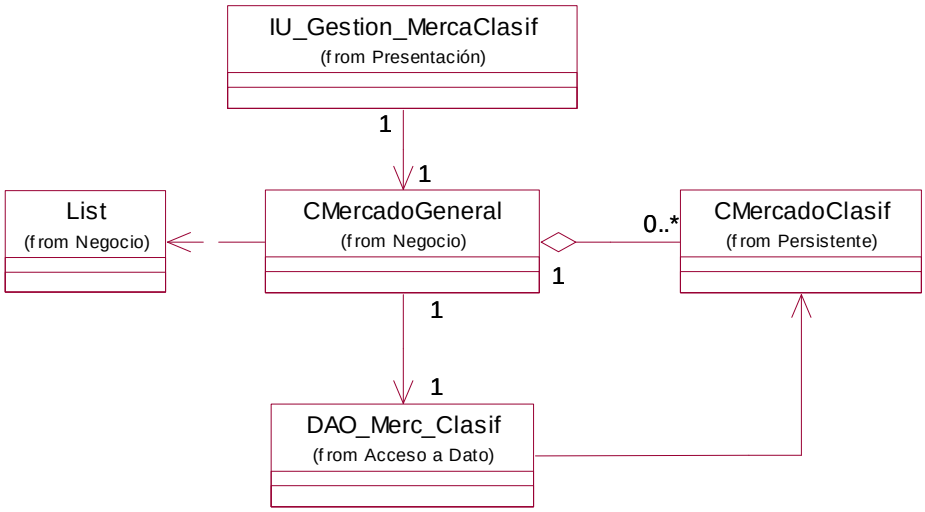

Figure 30 Algunas clases del diseño del CU Gestionar Clasificador de Mercado.

Subsistema 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.

Casos de uso que encapsula el Subsistema:

- Emitir Fusión

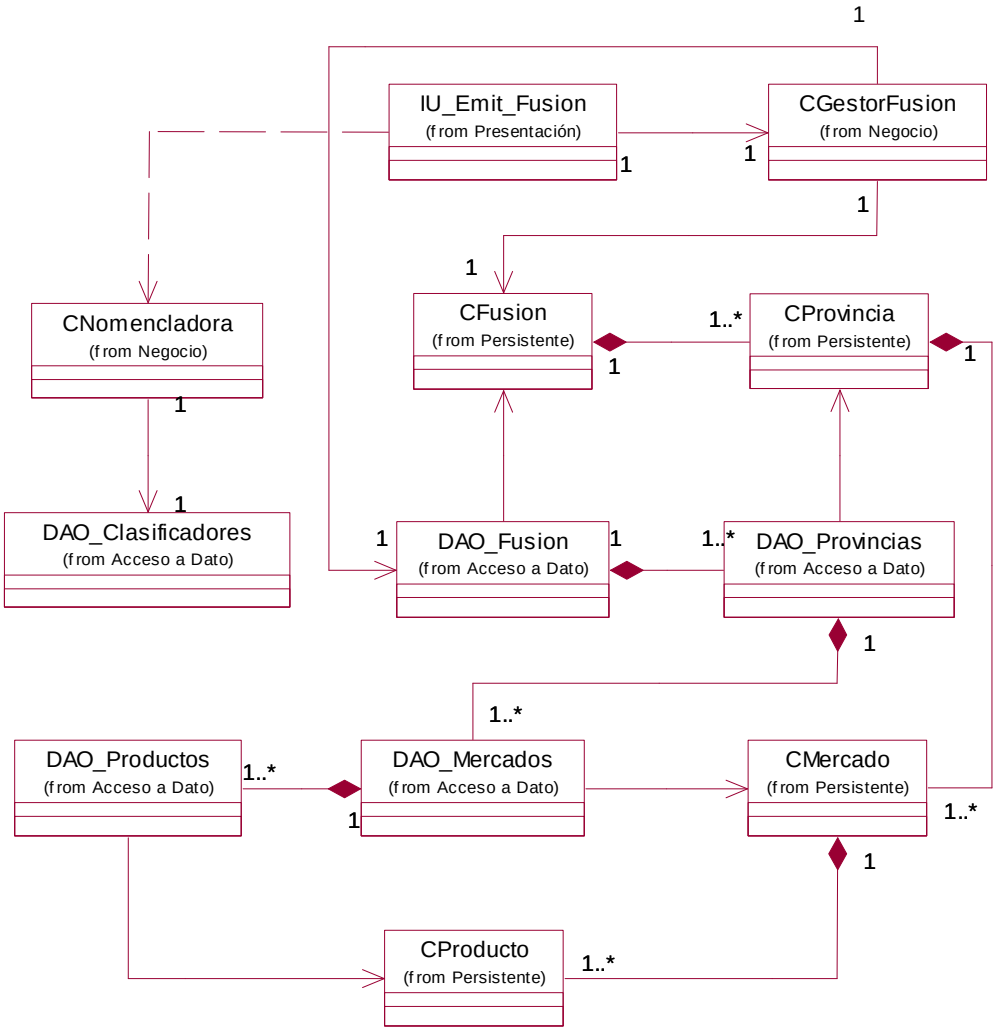

Figure 31 Algunas clases del diseño del CU Emitir Fusión.

- Emitir Tabla de Índices

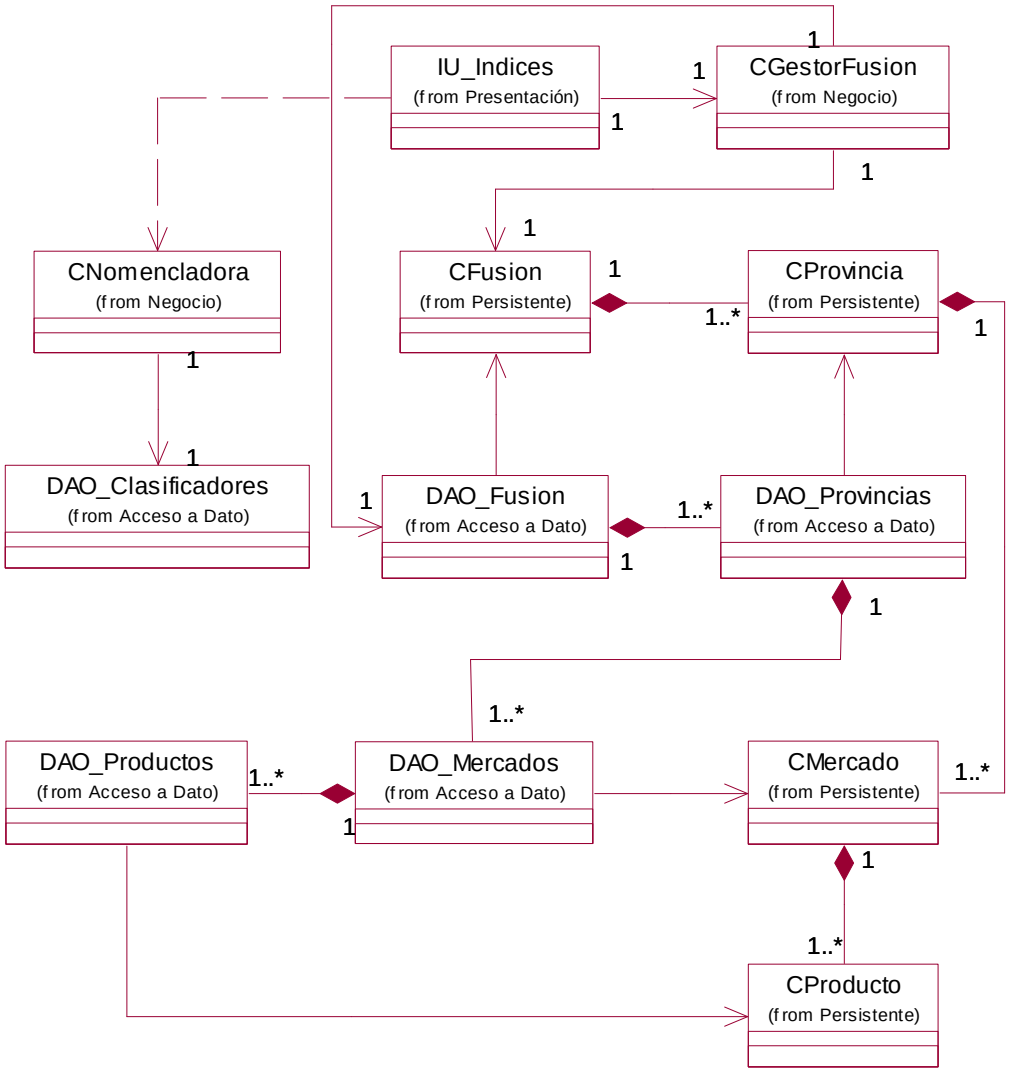

Figure 32 Algunas clases del diseño del CU Emitir Tabla de Índices.

- Emitir Tabla de Precios

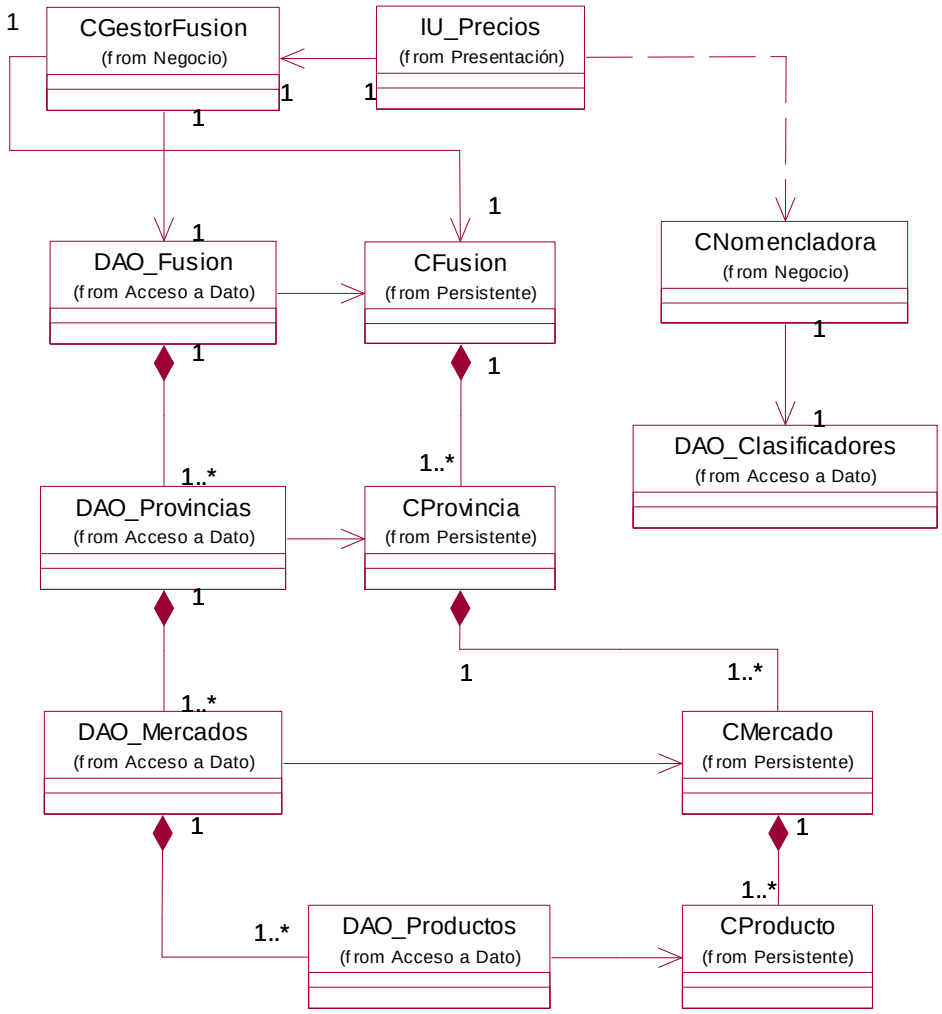

Figure 33 Algunas clases del diseño del CU Emitir Tabla de Precios

Subsistema Fusión

Este subsistema de diseño fue definido con la intensión de englobar la funcionalidad de capturar la información que viene en los ficheros provenientes desde las distintas provincias y almacenar en un almacén de datos esa información. Toda esta operación se logra mediante un caso de uso, el CU 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 de 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.

El proceso de parseo de los ficheros:

Es importante tener en cuenta la importancia de este proceso dentro del sistema y la calidad que debe tener el mismo, un mínimo error en el procesamiento de la información y podría causar resultados posteriores no deseados. Este proceso primeramente se basa 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

Table2 Formato del Fichero

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

Las siglas ES representan a un mercado formal, 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 CFusión tiene un método público llamado Fusionar, el cual va recibiendo cada una de las direcciones donde se encuentra cada fichero. Cada vez que ese método recibe una entrada, se produce la lectura del fichero y se obtiene como resultado una lista de productos, la cual es insertada en su mercado correspondiente y con su respectiva fecha.

A continuación se muestra un diagrama donde se representan las clases que interactúan en este subsistema.

Caso de uso que encapsula el Subsistema:

- Fusionar Ficheros.

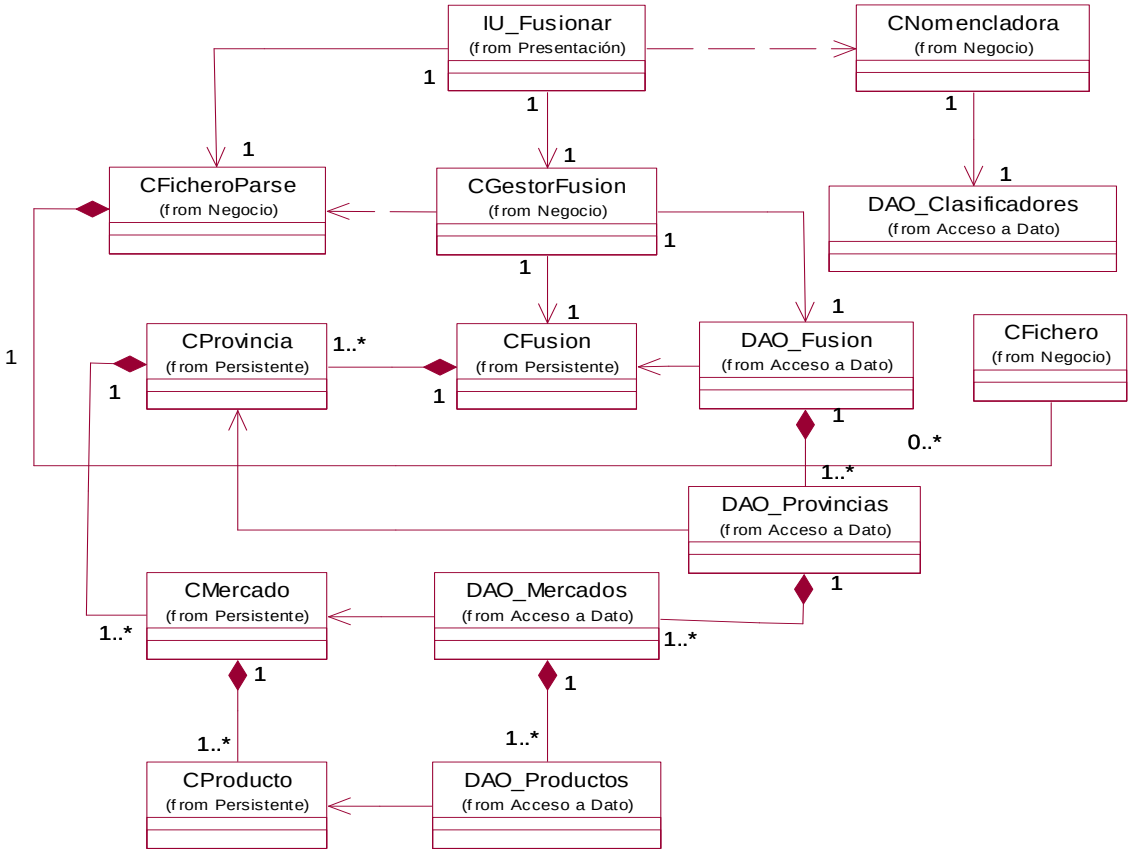

Figure 34 Algunas clases del diseño del CU Fusionar Ficheros

2.4 Vista de Implementación.

En la implementación empezamos con el resultado del diseño e implementamos el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, ficheros de código binario, ejecutables y similares

El flujo de trabajo de diseño se propone crear un plano del modelo de implementación, por lo que sus últimas actividades están vinculadas a la creación del modelo de despliegue. El flujo de trabajo de implementación describe cómo los elementos del modelo del diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el modelo de despliegue.

Los diagramas de despliegue y componentes conforman lo que se conoce como un modelo de implementación al describir los componentes a construir y su organización y dependencia entre nodos físicos en los que funcionará a aplicación.

El flujo de implementación esta fuertemente determinado por el lenguaje de programación.

El mismo tiene traza con el Modelo de Diseño. Se mantendrá la misma distribución en subsistema propuesta lo único que las clases de cada unos de los casos de uso encapsuladas en los subsistemas estarán enfocadas a componentes, esta distribución por componentes se puede hacer de varias maneras, entre ellas se encuentran la distribución de componentes por caso de usos de los subsistemas pero ubicados en cada una de las capas que forman parte del Modelo Lógico o de Diseño, pero esto puede traer consigo que no se especifique bien la relación de componentes dentro de los subsistemas en capas, puesto que cada capa tiene una dependencia total de los componentes de la capa con la que se relaciona, es por eso que la descomposición en componentes solo se realizara teniendo en cuenta los casos de uso agrupados en cada subsistemas, así la dependencia será estricta entre los componentes y quedará mejor definida.

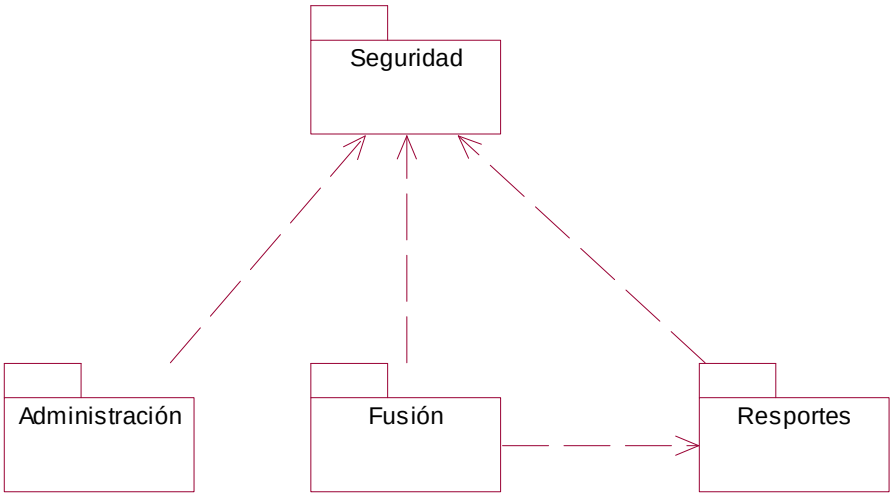

Figure 35 Modelo de Implementación

Subsistema de Implementación Seguridad

El Subsistema de Implementación Seguridad contiene dos casos de uso que tienen peso en la arquitectura Autenticar y Gestionar Usuarios, estos casos de uso contiene sus respectivos diagramas expresando la relación que existe entre cada uno de los componentes que participan en su realización, los mismos ubicados dentro de cada capa definida con anterioridad.

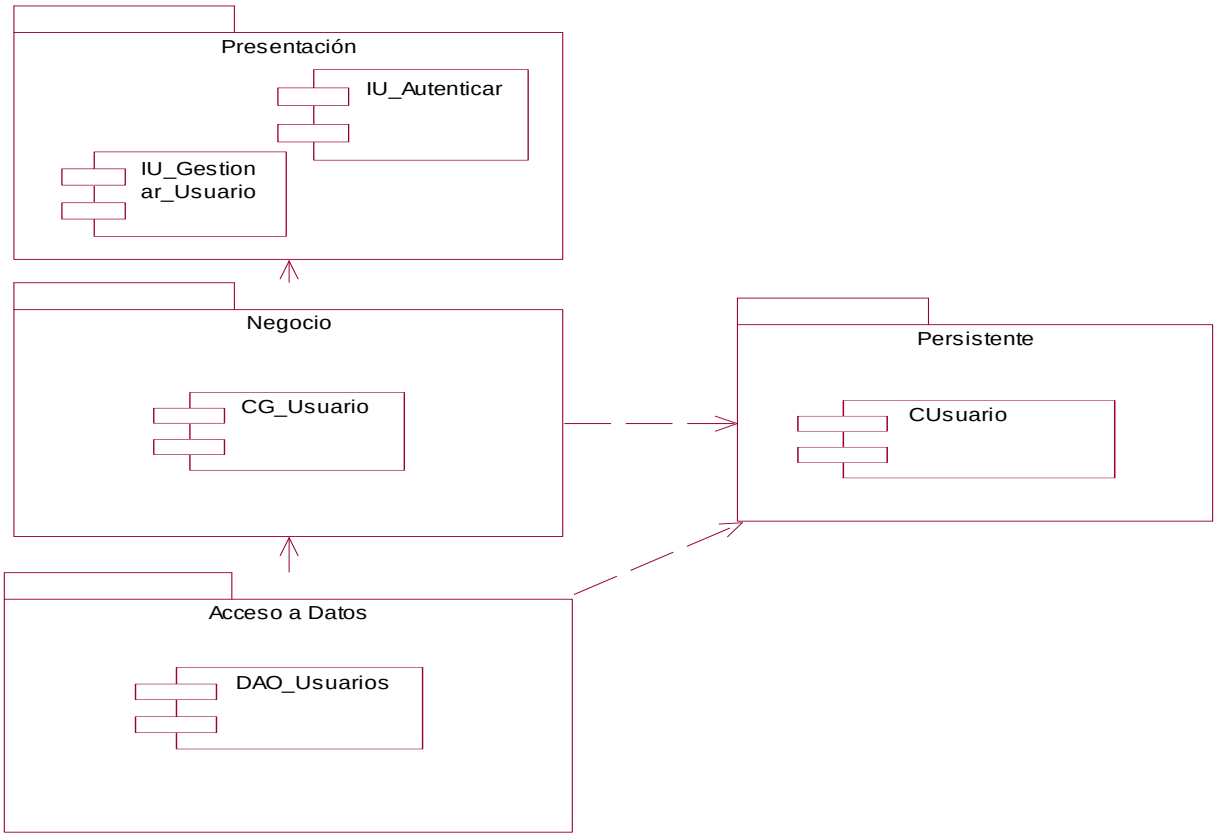

Figure 36 Diagrama Componentes CU Autenticar y Gestionar Usuarios

Subsistema de Implementación Fusión

El Subsistema de Implementación Fusión contiene un caso de uso que tiene peso en la arquitectura, Fusionar, este caso de uso contiene sus respectivos diagramas expresando la relación que existe entre cada uno de los componentes que participan en su realización, los mismos ubicados dentro de cada capa definida con anterioridad.

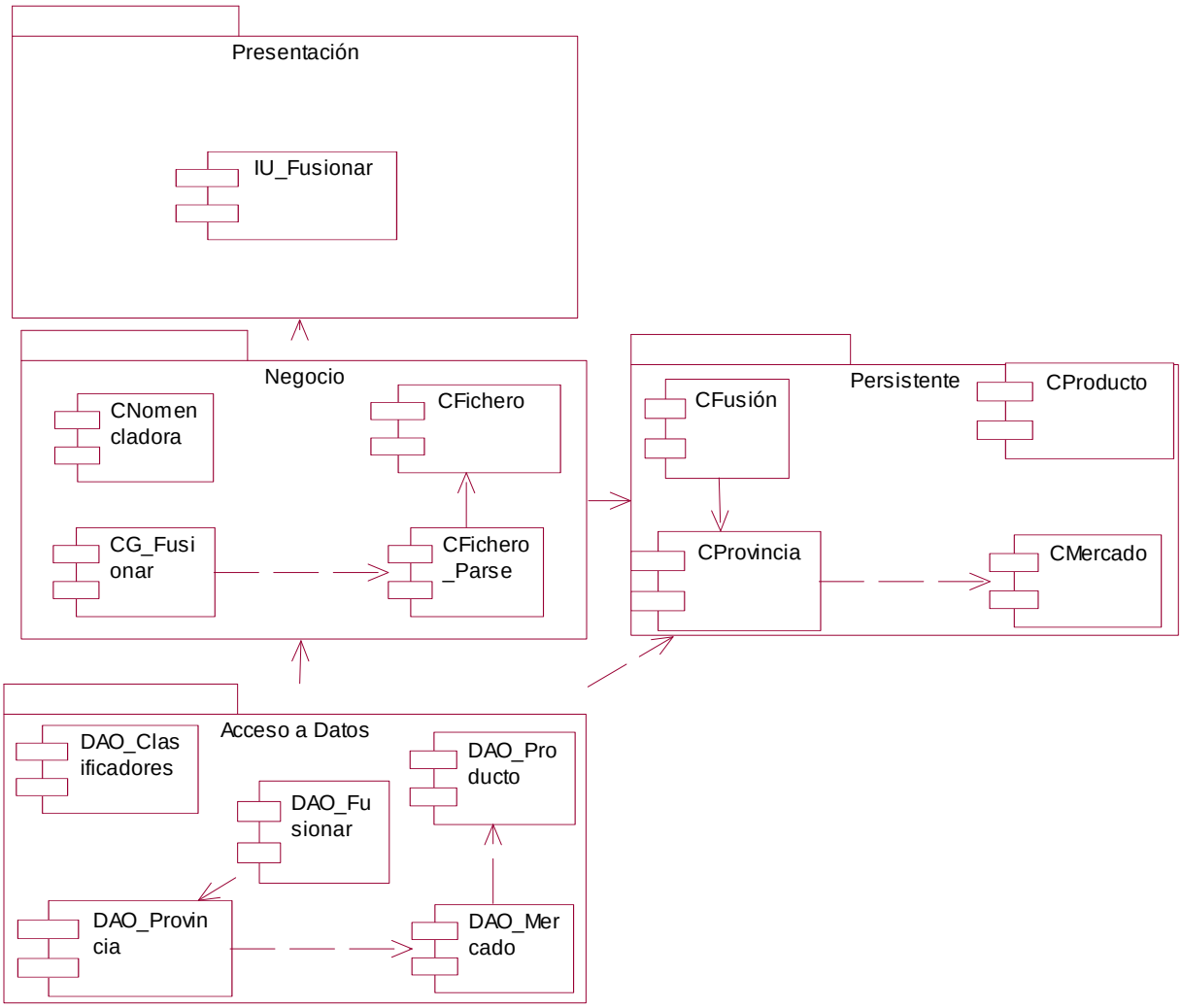

Figure 37 Diagrama componentes CU Fusión

Subsistema de Implementación Administración

El Subsistema de Implementación Administración contiene los casos de uso que tienen peso en la arquitectura, Gestionar Canasta, Gestionar Clasificador de Mercado, Gestionar Clasificador de Provincias, Gestionar Clasificador de Productos. Estos casos de uso contiene sus respectivos diagramas expresando

la relación que existe entre cada uno de los componentes que participan en sus realizaciones, los mismos ubicados dentro de cada capa definida con anterioridad.

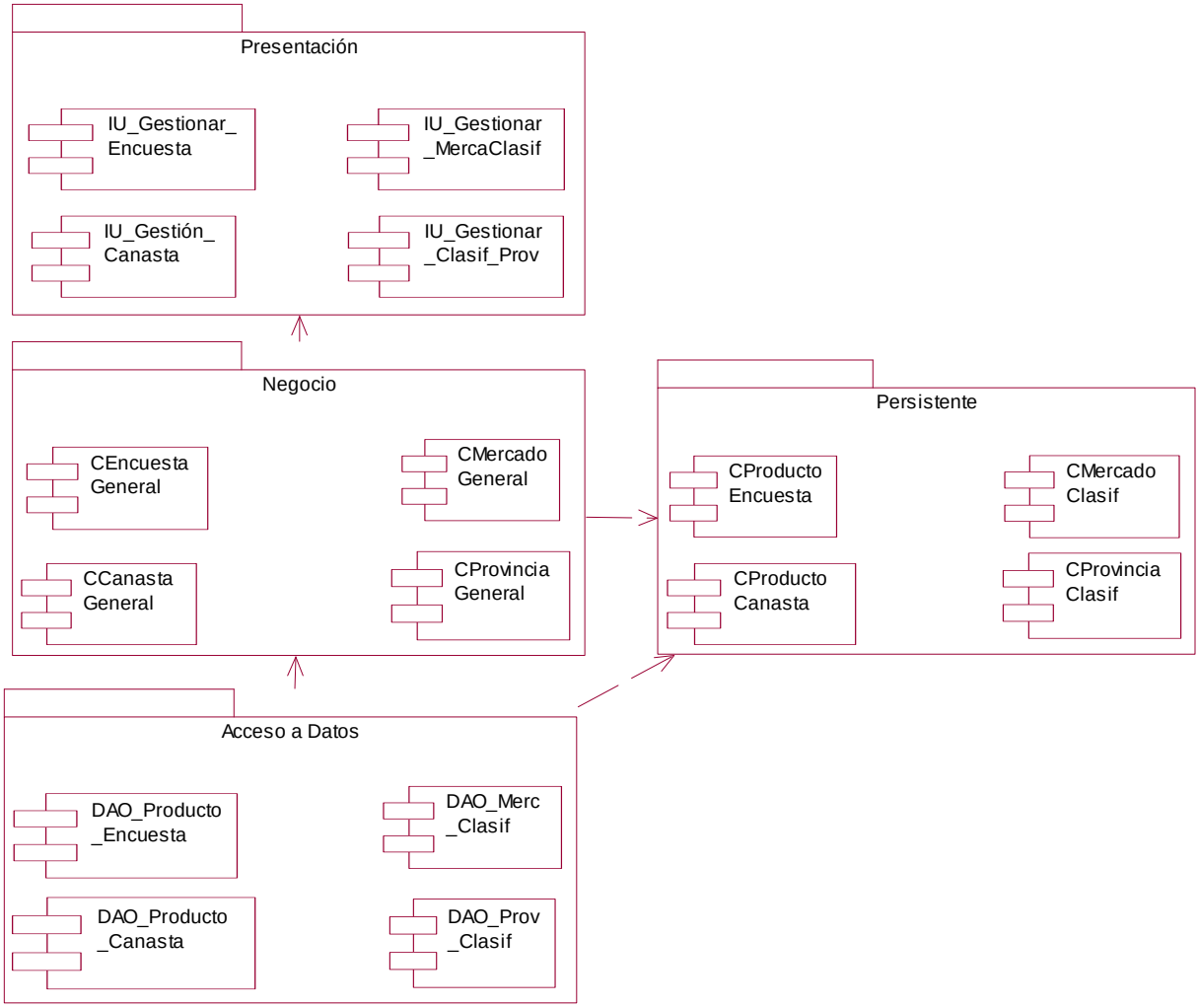

Figure 38 Diagrama componentes CU Gestionar Clasificador de Mercado, de Producto, de Provincia, de Canasta.

Subsistema de Implementación Reportes

El Subsistema de Implementación Reportes contiene los casos de uso que tienen peso en la arquitectura, Emitir Tabla de Índices, Emitir Tabla de Precios, Emitir Fusión. Estos casos de uso

contienen sus respectivo diagramas expresando la relación que existe entre cada uno de los componentes que participan en sus realizaciones, los mismos ubicados dentro de cada capa definida con anterioridad.

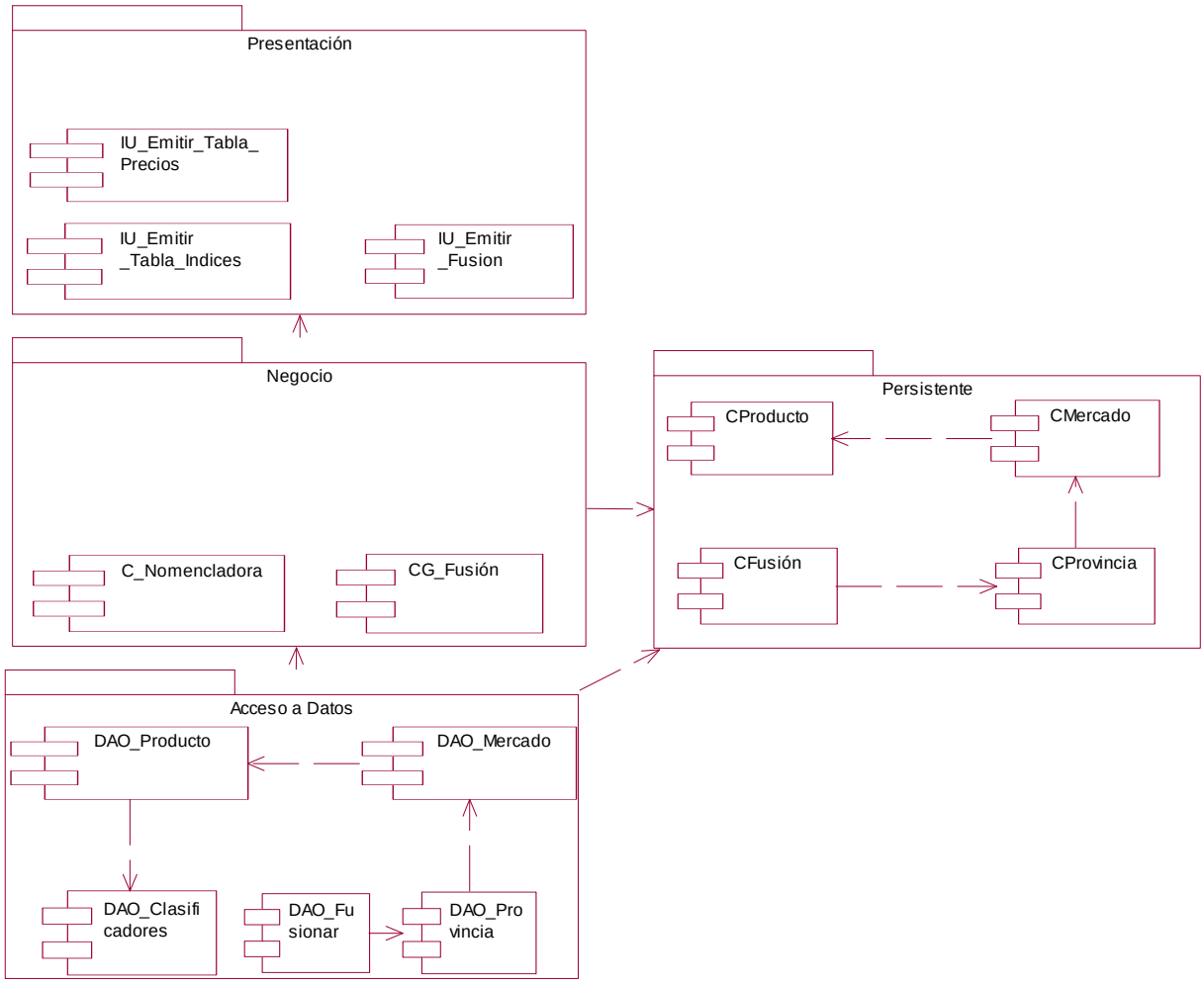

Figure 39 Diagrama componentes CU Emitir Tabla de Precios, Emitir Fusión, Emitir Tabla de Índices.

2.5 Vista de Despliegue

El arquitecto es responsable de considerar los aspectos referentes a la futura distribución física (despliegue) del sistema, llamada además arquitectura de hardware, el hardware es un punto clave para el correcto funcionamiento del sistema, ya que este no es capaz de ejecutarse en el “aire”.

En ocasiones las condiciones del hardware tienen que ser específicas, la configuración de red existente tiene que cumplir ciertos parámetros, hay protocolos que deben estar disponibles. Imaginemos una aplicación cliente que se conecta a una base de datos empleando ADO, si los componentes de acceso a datos de Microsoft (MDAC) no están instalados en la máquina cliente, la petición será inefectiva, porque la máquina cliente no esta capacitada para interpretar el protocolo necesario para dar respuesta a dicha solicitud, por tanto no se podrá “entender” con el servidor de datos.

Imaginemos ahora que el mismo sistema deberá ser capaz de soportar 1 millón de peticiones a la base de datos en un segundo. Si el gestor de datos empleado es un Pentium MMX con 64 Mb de RAM, la aplicación no podrá satisfacer dichos requisitos, ya que el Procesador (computadora), seleccionada como gestor de datos, no fue el correcto. El sistema no funcionará.

Trabajar además con los dispositivos reales es de suma importancia, puesto que una mala selección de los mismos podría acarrear una respuesta no deseada o irrealizable por parte de la aplicación.

La vista de despliegue incluye además una representación de los principales componentes del sistema y la ubicación que tendrán esos componentes dentro de cada nodo.

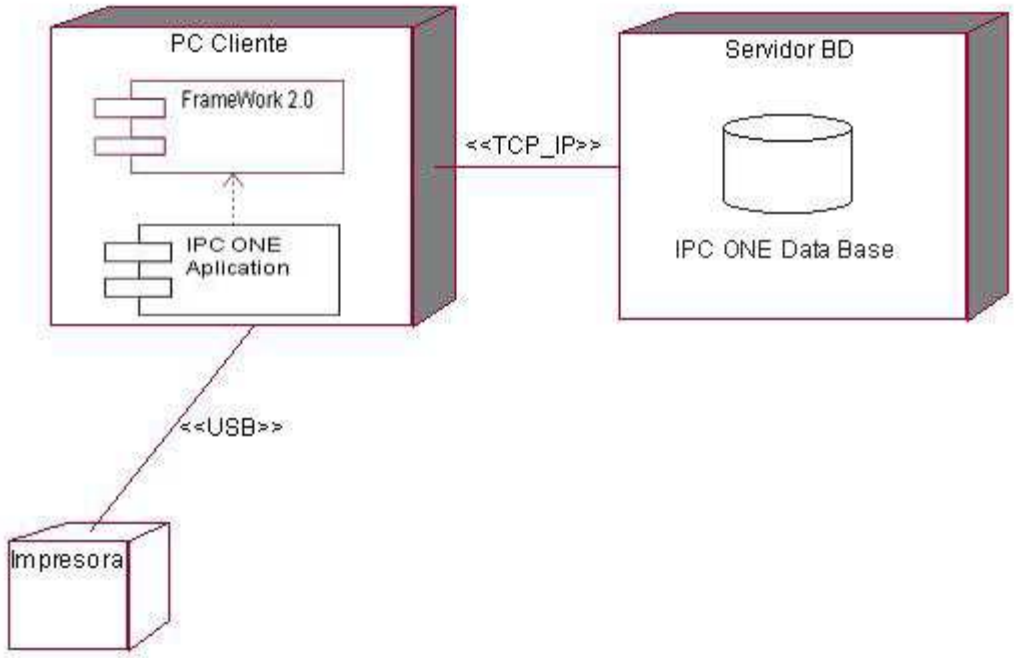

Figure 40 Diagrama de Despliegue.

Como se muestra en la imagen el sistema estará compuesto por dos nodos , PC Cliente que en esencia contendrá el componente aplicación, el cual dependerá de la instalación del componente Framework 2.0, sin la instalación de dicho componente la aplicación no será útil, al realizar el instalador de la aplicación se incluye la instalación de dicho framework , por otra parte se dedica un nodo que contendrá la Base de Datos, de esta forma se delegan tareas específicas a cada nodo y por otra parte el mismo sirve de gestor y almacén de datos a la aplicación cuando esta corra en más de una ubicación física, la conexión entre ambos nodos se realizará mediante el protocolo TCP-IP, y la PC Cliente constará con un dispositivo Impresora que será utilizado en caso de la necesidad de imprimir datos obtenidos de la aplicación como constancia .

Capítulo 3 Calidad de la Arquitectura de Software

Introducción

Todo diseño arquitectonico debe tener encuenta atributos de calidad, los mismos expresarán de forma sustancial las características que tendrá el sistema a implementar pero no lo garantiza. La mayoria de los atributos de calidad se miden por los requisitos no funcionales los cuales definen los escenarios, las tácticas a utilizar y los frameworks de desarrollo, todo en función de satisfacer dichos atributos.

3.1 Atributos de Calidad

Dentro de los atributos de calidad encontramos:

- Funcionamiento
- Disponibilidad
- Modificabilidad
- Seguridad
- Verificabilidad (testeability)
- Gestionabilidad
- Usabilidad

Hacer un adecuado balance entre la mantenibilidad, la interoperabilidad, la portabilidad, la funcionalidad, la seguridad, la disponibilidad, y la reusabilidad, entre otros atributos, es esencial para obtener un sistema que cumpla las expectativas de las distintas partes interesadas. [Bastarrica, 2005]

Estos atributos se pueden dividir en dos grandes grupos los que se pueden verificar en la ejecución del software y los que no, dentro del primer grupo se encuentran el funcionamiento, seguridad, disponibilidad, funcionalidad y usabilidad. Dentro del segundo grupo, y no menos importantes, están la modificabilidad, portabilidad, reusabilidad, integrabilidad y verificabilidad.

3.2 Que debe cumplir la Arquitectura Propuesta:

La solución propuesta debe:

- Satisfacer los requisitos (analistas)
- Ser construible (programadores & diseñadores)
- Ser probada (Administrador de Calidad)
- Ser administrable (administradores)

La descripción de la solución debe:

- Ser evaluable (por otros arquitectos)

El proceso de construcción debe:

- Ser manejable (por el jefe de proyecto)

3.3 ¿Qué permite una solución evaluable?

- Debe permitir evaluar compromisos y opciones.
- Por tanto, debe tener rastreabilidad entre las partes de la solución y del problema.

Proceso manejable:

- Debe ser particionada e indicar dependencias.
- Particiones: unidades naturales de asignación de trabajo.
- Dependencias: la base para definir calendario.

La solución debe satisfacer requisitos

- Debe convencer al analista.

3.4 Atributos de Calidad según la IEEE.

La IEEE std 1061, 1992 plantea como principales atributos de calidad:

- Eficiencia
- Funcionalidad
- Mantenibilidad / Portabilidad
- Confiabilidad
- Usabilidad

Métodos para evaluar la calidad de la arquitectura.

Existen métodos para la evaluación de la arquitectura propuesta entre ellos encontramos:

- SAAM

Este método de evaluación está basado en escenarios y permite evaluar una arquitectura o evaluar y comparar varias.

- ATAM (Architecture Trade Off Analysis Method)

Este método de evaluación obtiene su nombre no solo porque nos dice cuán bien una arquitectura particular satisface las metas de calidad, sino que también provee ideas de cómo esas metas de calidad interactúan entre ellas, como realizan concesiones mutuas (tradeoff) entre ellas.

- ARID(An Evaluation Method for Partial Architectures)

Método conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo.

Como cumple la arquitectura con los atributos de calidad.

Para evaluar la calidad de la arquitectura se explica a continuación cómo la misma cumple con cada uno de de los atributos y luego se establece una escala de valores de 1 a 6 la cual expresa de forma cuantitativa como se cumple cada uno de los atributos y se brinda una valoración final calculando el promedio de valores de cada uno de los parametros de calidad.

- [1 a 2] mal cumplimiento del atributo
- [3 a 4] cumplimiento regular
- [5 a 6] buen cumplimiento

Funcionamiento

La prupuesta de arquitectura cumple con este atributo de calidad puesto que la misma centra su desarrollo en dos caracteristicas fundamentales.

- Se basa en un proceso que considera los casos de uso como un elemento de vital importancia. Esto da una medida de cómo se consibe un sistema que realmente responda a las necesidades del cliente, las cuales quedan expresadas en los casos de uso.
- Se basa en un proceso iterativo e incremental. Esto da una medida de cómo la arquitectura propone un prototipo del sistema funsional escojiendo las funsionalidades basicas o casos de uso criticos y los prioriza en las primeras iteraciones dentro del cilco de vida.

Mediante estas dos caracteristicas queda demostrado porque se cumple que la arquitectura propuesta va dirigida a obtener un sistema funsional.

Se considera que tiene un valor de 5 dentro de la escala.

Eficiencia

La arquitectura considera la posibilidad de no tener que duplicar informacion, ya que el volumen de datos que se envia desde las provincias a la ONE referente al valor de los precios dentro de cada mercado

establecido por cada una de las provincias el sistema lo almacena en una base de datos y luego se podran manipular los mismos desde dicho almacen de datos sin necesidad de acudir nuevamente a los ficheros, todo este proceso se realiza mediante la fusión de los ficheros referente a cada provincia en un mes determinado. Por otra parte la utilización de recursos es mínima puesto que la arquitectura destina los componentes principales de la aplicación en dos nodos, una contendrá la aplicación y el otro nodo la Base de Datos. De esta manera se dividen de forma bien definida las tareas específicas a cada uno de los sistemas de computo.

Se considera que tiene un valor de 4 dentro de la escala.

Flexibilidad

La arquitectura considera la flexibilidad del sistema agregando funsionalidades como la posibilidad de poder agregar nuevos productos a la canasta básica, además considera la necesidad de que en caso de una nueva distribución político administrativa la ONE tendra la posibilidad de poder agregar nuevas provincias con sus respectivos clasificadores, entiendase por clasificadores el hecho de asignar los mercados que contendrá y los productos en dichos mercados.

Por otra parte debido a la versatilidad de la economía cubana se considera la posibilidad de que la Entidad necesite crear nuevos tipos de mercados dentro de la infraestructura.

Se considera que tiene un valor de 5 dentro de la escala.

Modificabilidad

La arquitectura considera la modificabilidad del sistema. La misma se evidencia dentro de la arquitectura en capas propuesta, las cuales ofrecen interfaces de comunicación que agrupan funsionalidades bien definidas, esto trae consigo que modificar una capa no tenga consecuencias sobre el funcionamiento de otra.

Se considera que tiene un valor de 4 dentro de la escala.

Análisis de los resultados:

Luego del análisis de el valor que tiene cada uno de los atributos dentro de la escala prupuesta se obtiene un promedio de la calidad de la arquitectura de valor 4.5 por lo que se considera que cumple con los parámetros de calidad analizados.

Valoración Final de la Solución Propuesta

La solución propuesta se basa en el la elaboración y análisis de dos documentos fundamentales propuestos por la metodología RUP, estos son la Línea Base de la Arquitectura y Documento de Descripción de la Arquitectura.

La utilización del documento Línea Base de la Arquitectura constituye un apoyo al documento Descripción de la Arquitectura propuesta por la metodología de desarrollo RUP (Proceso Unificado de Software), el ubicar toda la información que se recoge en la Línea Base en el mismo documento de descripción, haría de este último un documento muy denso y que no pudiera recoger toda la información y la especificación que tiene de esta forma. La organización de la Línea Base está constituida por los aspectos que se ha entendido son los más importantes para la descripción. El documento Descripción de la Arquitectura propone la utilización de la 4 de las vistas propuestas por la metodología de desarrollo RUP (Proceso Unificado de Desarrollo), hay que resaltar que el análisis de cada uno de los escenarios especificados y contemplados en este documento se pueden utilizar para la descripción del sistema.
Atendiendo a todos estos aspectos se considera que la arquitectura propuesta responde a los principales puntos a medir, la misma satisface los requisitos de los clientes, es construible por parte de los desarrolladores, puede ser probada y es administrable. A partir de estos señalamientos se propone un conjunto de recomendaciones.

Conclusiones

La propuesta de solución dada abarca dos documentos de obligada construcción, y constante refinamiento por parte del arquitecto de software del sistema. Se hizo un estudio detallado de los principales aspectos de la descripción arquitectónica. Se definió los principales estilos arquitectónicos, sus componentes, configuraciones y restricciones, impuestas por los requisitos de los clientes. Dada las restricciones impuestas por algunos requisitos no funcionales del cliente y en función de los principales atributos de calidad de la Arquitectura de Software se definieron las principales tecnología y herramientas a utilizar en el desarrollo del sistema, y se configuró cada uno de los puestos de trabajo por roles en el equipo de desarrollo. Con la Arquitectura de Software propuesta cumple con: satisface los requisitos de los clientes, es “construible” por los desarrolladores, es “verificable” puede ser probada por los revisores y es “administrable” por los administradores del sistema.

Recomendaciones

1. El refinamiento constante de la arquitectura propuesta, durante el ciclo de desarrollo.
2. El análisis de las dependencias entre los subsistemas de implementación para evitar que existan retrasos en alguno de ellos y que se vea afectado en cuanto a tiempo la entrega de cada una de las actividades de implementación.
3. La aplicación de cualquiera de los métodos de evaluación de la arquitectura, con el objetivo de identificar las principales debilidades, tanto en la descripción arquitectónica como en el propio diseño arquitectónico.
4. Migrar hacia Software Libre y garantizar asi que el sistema sea multiplataforma.

Referencias

[Shaw, Garlan, 1996] Shaw, M. y. G., D. (1996). Software Architecture. Perspectives of an Emerging Discipline, Prentice Hall.

[Fayad et al, 1999]Fayad, M., Schmidt, D., y Johnson, R. E. (1999). Application Frameworks., Wiley & Sons. 3.

[Szyperski, 2000] Szyperski, C. (2000). Components and Architecture. Software Development Online: 10.

[Brown et al, 1998] Brown, A. W. y. W., K. C. (1998) The Current State of CBSE. IEEE Software

[Bastarrica, 2005] Bastarrica, M. C. (2005) Atributos de Calidad y Arquitectura del Software. Volume, 4

[Booch, 1999] Booch, G. (1999). The Unified Modeling Language User Guide, Addison Wesley.

Bibliografía

Ivar Jacobson, G. B., J. Rumbaugh (2000). El proceso de desarrollo unificado de software. Madrid, Pearson Educación. S.A.

Larman, C. (2004). UML y Patrones. La Habana, Félix Varela.

Pfleeger, S. L. (2002). Ingeniería de Software: Teoría y Práctica. Madrid, Prentice-Hall.

Pressman, R. (2001). Ingeniería del Software: Un enfoque práctico. Madrid, McGraw Hill.

Rolando Alfredo, S. C. (2002). EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN

CIENTIFICA. Ciudad de la Habana.

Society, S. E. S. C. o. t. I. C. (2000). "IEEE Recommended Practice for Architectural Description

of Software-Intensive Systems."

[Colectivo Autores, 1999] Autores, C. d. (1999) Guía de Iniciación al Lenguaje JAVA. Junta de

Castilla y Leon

Bass, L., Clements, P., y Kazman, R. (1998) Software Architecture in Practice. Addison Wesley.

Bastarrica, M. C. (2005) Atributos de Calidad y Arquitectura del Software. Volume, 4

Booch, G. (1999). The Unified Modeling Language User Guide, Addison Wesley.

Bosch., J. (2000). Design & Use of Software Architectures". Addison Wesley.

Box, D. (1998). Essential COM, Addison Wesley.

Brown, A. W. y. W., K. C. (1998) The Current State of CBSE. IEEE Software

Carlos Reynoso, N. K. (2004) Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. UNIVERSIDAD DE BUENOS AIRES

CCITT/ITU-T (1988). Redes de Comunicacion de Datos. Interconexion de Sistemas Abiertos: Modelo y Notacion.

Clements, P. (1996). A Survey of Architecture Description Languages. Proceedings of the International Workshop on Software Specification and Design.

Deepak Alur, J. C. a. D. M. (2001). Core J2EE Patterns: Best Practices and Design Strategies. P. H. S. M. Press.

Gamma, E. (2002). Patrones de Diseño, Addison-Wesley.

Gómez, S. P. (2006) Swing Univ. Politécnica de Madrid

Hernán Astudillo, S. H. (1998) Understanding the Architect's Job. Software Architecture Workshop of OOPSLA'98

IEEE-1471 (2000) Recommended Practice for. Architectural Description of Software-Intensive Systems